ਜਗਤ ਗੁਰੂ ਨਾਨਕ ਦੇਵ
ਪੰਜਾਬ ਸਟੇਟ ਓਪਨ ਯੂਨੀਵਰਸਿਟੀ
ਪਟਿਆਲਾ

JAGAT GURU NANAK DEV

PUNJAB STATE OPEN UNIVERSITY, PATIALA

(Established by Act No. 19 of 2019 of the Legislature of State of Punjab)

The Motto of the University

(SEWA)

SKILL ENHANCEMENT EMPLOYABILITY WISDOM ACCESSIBILITY



Bachelor of Computer Applications (BCA)
Course Name: Computer System Architecture
Course Code : BCA 3 03T

ADDRESS: C/28, THE LOWER MALL, PATIALA-147001
WEBSITE: www.psou.ac.in

SELF-INSTRUCTIONAL STUDY MATERIAL FOR JGND PSOU, ALL COPYRIGHTS WITH JGND PSOU, PATIALA

JAGAT GURU NANAK DEV
PUNJAB STATE OPEN UNIVERSITY PATIALA
(Established by Act No.19 of 2019 of Legislature of the State of Punjab)

PROGRAMME COORDINATOR :
Dr. Monika Pathak
Assistant Professor, School of Sciences and Emerging Technologies
Jagat Guru Nanak Dev Punjab State Open University, Patiala

PROGRAMME CO-COORDINATOR :
Dr. Gaurav Dhiman
Assistant Professor, School of Sciences and Emerging Technologies
Jagat Guru Nanak Dev Punjab State Open University, Patiala

COURSE COORDINATOR :
Dr. Karan Sukhija
Assistant Professor, School of Sciences and Emerging Technologies
Jagat Guru Nanak Dev Punjab State Open University, Patiala

JAGAT GURU NANAK DEV
PUNJAB STATE OPEN UNIVERSITY PATIALA
(Established by Act No.19 of 2019 of Legislature of the State of Punjab)

PREFACE

Jagat Guru Nanak Dev Punjab State Open University, Patiala was established in Decembas 2019 by Act 19 of the Legislature of State of Punjab. It is the first and only Open Universit of the State, entrusted with the responsibility of making higher education accessible to all especially to those sections of society who do not have the means, time or opportunity to pursue regular education.

In keeping with the nature of an Open University, this University provides a flexible education system to suit every need. The time given to complete a programme is double the duration of a regular mode programme. Well-designed study material has been prepared in consultation with experts in their respective fields.

The University offers programmes which have been designed to provide relevant, skill-based and employability-enhancing education. The study material provided in this booklet is self instructional, with self-assessment exercises, and recommendations for further readings. The syllabus has been divided in sections, and provided as units for simplification.

The Learner Support Centres/Study Centres are located in the Government and Government aided colleges of Punjab, to enable students to make use of reading facilities, and for curriculum-based counselling and practicals. We, at the University, welcome you to be a part of this institution of knowledge.

Prof. G. S. Batra,
Dean Academic Affairs

BCA-3-03T: Computer System Architecture

Total Marks: 100
External Marks: 70
Internal Marks: 30
Credits: 4
Pass Percentage: 35%

INSTRUCTIONS FOR THE PAPER SETTER/EXAMINER

1. The syllabus prescribed should be strictly adhered to.
2. The question paper will consist of three sections: A, B, and C. Sections A and B will have four questions from the respective sections of the syllabus and will carry 10 marks each. The candidates will attempt two questions from each section.
3. Section C will have fifteen short answer questions covering the entire syllabus. Each question will carry 3 marks. Candidates will attempt any ten questions from this section.
4. The examiner shall give a clear instruction to the candidates to attempt questions only at one place and only once. Second or subsequent attempts, unless the earlier ones have been crossed out, shall not be evaluated.
5. The duration of each paper will be three hours.

INSTRUCTIONS FOR THE CANDIDATES

Candidates are required to attempt any two questions each from the sections A and B of the question paper and any ten short q questions from Section C. They have to attempt questions only at one place and only once. Second or subsequent attempts, unless the earlier ones have been crossed out, shall not be evaluated.

Course: Computer System Architecture	
Course Code: BCA-3-03T	
Course Outcomes (COs) After the completion of this course, the students will be able to:	
CO1	Explain the organization of basic computer , its design and the design of control unit.
CO2	Demonstrate the working of central processing unit and RISC and CISC Architecture.
CO3	Describe the operations and language of the register transfer, micro operations and input- output organization.
CO4	Understand the organization of memory and memory management hardware.
CO5	Elaborate advanced concepts of computer architecture, Parallel Processing, inter processor communication and synchronization.

Section A

Unit I: Basics of Data Representation- Number System, Conversions of Number Systems, 1's and 2's Complements, fixed and floating point representation, character representation, addition, subtraction, magnitude comparison.

Unit II: **Introduction to Boolean algebra** - Logic gates, Boolean algebra, K-Maps, Sum of Products, Product of Sums.
Unit III: Combinational circuits and Sequential Circuits: decoders, multiplexors, Encoders, DE-multiplexers Half Adders, Full Adders, Flip Flops, registers, counters and memory units.
Unit IV: Basic Computer Organization and Design- Computer Architecture, Structure, Computer registers, Common Bus Systems, Arithmetic, Logical, Shift Micro-operations, and Design of ALU.

Section B

Unit V: Timing and Control Unit-Instruction cycle, Memory reference instructions, Register reference instructions, Input-output instructions, Design of Timing and Control Unit.
Unit VI: Design of Central Processing Unit: Register organization, stack organization, Register Organization, one address instructions, two address instructions, and three address instructions. Instruction formats, addressing modes.
Unit VII: Input-Output Organization: I/O interfaces, Data transfer schemes. I/O control mechanisms - Program controlled, Interrupt controlled and DMA controller.
Unit VIII: **Memory Unit**: Memory hierarchy, High-speed memories, Organization of a Cache memory unit, Virtual memory, Memory management.

Reference Books

- Mano, Morris M., "Computer System Architecture", 3rd ed., Prentice Hall, 2007
- Hayes, J.P., "Computer Architecture and Organization", McGraw Hill, 1998
- Hennessy, J.L., Patterson, D.A, and Goldberg, D., "Computer Architecture a Quantitative

  Approach", Pearson Education Asia, 2005
- Leigh, W.E. and Ali, D.L., "System Architecture: software and hardware concepts", South Western Publishing Co., 2000

Bachelor of Computer Applications (BCA)

COMPUTER SYSTEM ARCHITECTURE

UNIT I: INTRODUCTION TO NUMBER SYSTEM

STRUCTURE

1.0 Objectives

1.1 Introduction

1.2 Types of Number System

1.3 Conversion of Number System

1.4 Representation of Signed Binary Numbers

- **1.4.1.Sign-magnitude**
- **1.4.2.One's Complement**
- **1.4.3.Two's Complement**

1.5.Addition and Subtraction using 1's complement

- **1.5.1.Subtraction using 1's complement**
- **1.5.2.Binary Addition using 2's Complement**
- **1.5.3 Subtraction using 1's complement**
- **1.5.4.Subtraction using 2's complement**

1.6.Fixed and Floating point representation

1.7.Binary Addition and Substraction

1.8.Chracter Code Repersention

1.9 Summary

1.10 Practice Questions

1.0 OBJECTIVES

- Understnading Number System, Conversions of Number Systems, 1's and 2's Complements,
- Understnading addition, subtraction and magnitude comparison.

1.1 INTRODUCTION

In Digital System, the number system is used to represent in information in number system has different base value and the index value. In which, the most common of them are the decimal number system,binary number system,,octal number system and hexadecimal number system. In these number systems, the base or radix of the number depends on the total number of the digital used in number system.Suppose, if the in number system is binary that there are only two numbers 0 and 1.The number 0 represent the lower state of the value or 1 represent the highest value. The digital value in the number system can be calculated by using digit value,index value, and finally through the base value.

1.2 Types of Number System

In the digital computer system, there are four types of number systems used for representing information.

1. Binary Number System
2. Decimal Number System
3. Octal Number System
4. Hexadecimal Number System

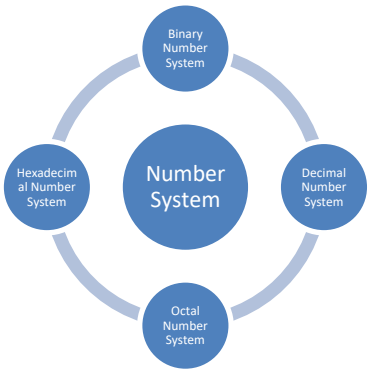


Diagram of Number System

1. Binary Number System: Generally,a binary number system is used to represent the basic structure of digital computer. It contains only two digits, either 0 or 1. In digital electronics 0 means off and 1 means on value or we can say 0 is the absence of electronic pulse and 1 means presence of electronic pulse. Each digit is known as single value of bit.the base value of binary

number system is 2.

Basic Characteristics of Binary numbers:

1.It contain only two values, either 0 or 1.

2.It is also known as base 2 number system.

3. The position of a digit represents the 0 power of the base(2). Example: 2^0

2.Decimal Number System: In digital electronics, decimal number system contains 0 to 9 digits and its base value is 10. In which 0 is the minimum value of the digit, and 9 is to represent the maximum value.

3.Octal Number System: In digital system,octal number system uses the digital value from (0 to 7). In which 0 is the minimum value and 7 is the maximum value. There are only eight bit value is possible. It contain only three bits to represent the octal number system.

Basic Characteristics of octal number system:

1. Octal number system carries the value from 0 to 7.
2. The base value of octal number system is 8.
3. The position of a digit represents the 0 power of the base(8). Example: 8^0

Number	Octal Number
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

4.Hexadecimal Number System: It is the last way to represent the number in the digital number system called hexadecimal number system.It contain the value from 0 to 15 and the base address of the value is 16 means it identifies 16 symbols(0 to 9 and A to F), where A is represent 10 value and B is 11 and so on. There are only 4 bits are required to represent hexadecimal number system.

Basic Characteristics of hexadecimal number system:

1.It has to repersent ten digits from 0 to 9 and 6 letters from A to F.

2. In hexadecimal letters from A to F defines numbers from 10 to 15.

3.In hexadecimal number, the position of a digit represents the 0 power of the base(16).

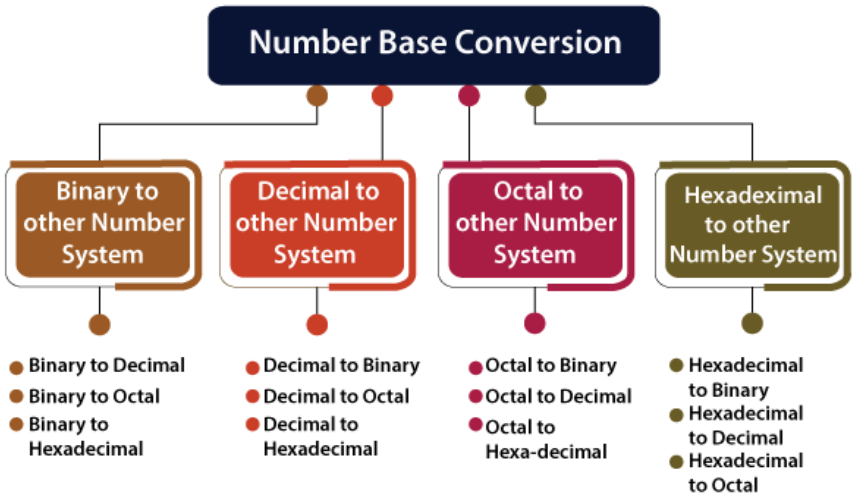
Binary Number	Hexadecimal Number
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Table of Hexadecimal Code

1.3 CONVERSION OF NUMBER SYSTEM

In Digital Electronics,various types of number systems such as binary,digital,octal, and hexadecimal and the number system can be converted from one number system to another number system like

1. Binary to other Number Systems.
2. Decimal to other Number Systems.
3. Octal to other Number Systems.
4. Hexadecimal to other Number Systems.



1.3.1 Binary to Other Number System: In digital computer system,there are three possible conversion through binary number system and these are given below:

a.Binary to Decimal Number System
b.Binary to Octal Number System
c.Binary to Hexadecimal Number System

a.Binary to Decimal Number System: It is very simple process of conversion.The process starts from multiplying the bits of binary number system with its corresponding positional weights after that,add all those products.

Example: $(10110.001)_2$

We multiplied each bit of $(10110.001)_2$ with its respective positional weight, and last we add the products of all the bits with its weight.

$\mathbf{(10110.001)_2}=(1\times2^4)+(0\times2^3)+(1\times2^2)+(1\times2^1)+(0\times2^0)+(0\times2^{-1})+(0\times2^{-2})+(1\times2^{-3})$

$\mathbf{(10110.001)_2}=(1\times16)+(0\times8)+(1\times4)+(1\times2)+(0\times1)+(0\times1/2)+(0\times1/4)+(1\times1/8)$

$(10110.001)_2$=16+0+4+2+0+0+0+0.125
$(10110.001)_2$=$(22.125)_{10}$Answer.

b.Binary to Octal Number System: The base value of octal number system is 8.In a binary number system, the group of three bits is equal to one octal digit. In digital computer system, there are three steps to convert a binary number system into octal number system and these steps are given below:

1.In first step, make a group of three bits on both the sides.If there will be one or two bits left to make a group then add required number of bits.

2.In the second step, write the octal digits corresponding to each group.

Example: $(111110101011.0011)_2$

1. Firstly, we make group of three bits on both sides of the binary point.

111 110 101 011.001 1

On the right side of the binary point, the last group has only one bit. To make it a complete group of three bits, we added two zeros on the extreme side.

111 110 101 011.001 100

2. Then, we wrote the octal digits, which correspond to each group.

$(111110101011.0011)_2=(7653.14)_8$Answer.

c. Binary to Hexadecimal Number System: The base value of hexadecimal number system is 16. In digital system, the group of four bits is equal to one hexadecimal number system. In digital computer system, there are two steps to convert a binary number system into hexadecimal number system and these steps are given below:

1.In the first step, make a group of four bits on both the sides.If there will be any number of bits are left then add required number of bits.

2.In the second step, write the octal digits corresponding to each group.

Example: $(10110101011.0011)_2$

1. Firstly, we make pairs of four bits on both sides of the binary point.

111 1010 1011.0011

On the left side of the binary point, the first group has three bits. To make it a complete group of four bits, add one zero on the extreme side.

0111 1010 1011.0011

2. Then, we write the hexadecimal digits, which correspond to each group.

$(011110101011.0011)_2=(7AB.3)_{16}$Answer.

1.3.2. Decimal to Other Number System

In digital computer system,there are three possible conversion through decimal number system and these are given below:

a. Decimal to Binary Number System
b.Decimal to Octal Number System
c.Decimal to Hexadecimal Number System

a.Decimal to Binary Number System: In digital computer system,the decimal number can be be represented in interger or floating point integer. When the decimal number is floating point integer, then it will be converted in both part(integer value and fractional value).

In digital system, there are the two basic ways to represent decimal number into a similar number of any base.

1.In the first step, division operation is used.

2.In another step,multiplication method is used .

1.Using Division method : In this method ,decimal number is repeatedly divided by 2 until we get the remainder to be 0..After that the result can be calculated from bottom to top.The first remainder is known as LSB(Least Significant Bit) and the last remainder is known as MSB(Most Significant Bit).

Example: 262_{10}

Solution:Steps to solve

1. Divide the Given number by 2
2. For the next iteration , get the quotient value.
3. Get the remainder which is used to determine the binary number

4. Repeat the steps until the iteration gets a 0 as quotient.

	262		
2	131	-	0
2	65	-	1
2	32	-	1
2	16	-	0
2	8	-	0
2	4	-	0
2	2	-	0
2	1	-	0

b.Using Multiplication Method: In this method,decimal fraction can be converted into binary by repition value by 2 ,until we get the fraction production is 0.After that the result can be calculated from top to bottom.

b.Decimal to Octal Number System: This method is similar to decimal to binary conversion.In which,the base value or radix value is 8.Therefore, the value is divided by 8,when we want to convert integer value to octal number system or we want use multiplication method to convert fractional decimal to octal.

Example:

$(425)_{10}$ = (___?___)$_8$

Solution:

$(425)_{10}$=()$_8$

8	425		
8	53	1	↑
8	6	5	↑
	0	6	↑

∴$(425)_{10}$=(651)$_8$**Answer.**

c. Decimal to Hexadecimal Number System: In this method ,the conversion of decimal to hexa is similar to decimal to bimary and decimal to octal number system. In which, the base value is 16,therefore, the division method is similar to integer value and multiplication by 16 for fraction

value.

Example:

$(1423)_{10}$ = (___?___)$_{16}$

Solution:

$(1423)_{10}$=()$_{16}$

16	1423		
16	88	F	↑
16	5	8	↑
	0	5	↑

$\therefore(1423)_{10}=(58F)_{16}$**Answer.**

1.3.3 Octal to Other Number System

In digital computer system,there are three possible conversion through Octal number system and these are given below:

a. Octal to Decimal Number System
b.Octal to Binary Number System
c.Octal to Hexadecimal Number System

a.Octal to Decimal conversion: Octal Number is similar to binary to decimal conversion,just change the weights assigned to each octal number.

Example:126_8 to decimal.

Octal Number--→1 2 6

Weights-→ $(1x8^2) + (2x8^1) + (6 \times 8^0)$

$(8^2 = 64)$, $(8^1 = 8)$ and $(8^0 = 1)$, this gives a multiplier value for each column.

Multiply the digit in each column by the column multiplier value for that column to give:

1x64 = 64 2x8 =16 6x1 = 6

Then simply add these results to give the decimal value.

$64 + 16 + 6 = 86_{10}$

Therefore $126_8 = 86_{10}$ Answer.

Example:

Octal Number--→7 3 5

Weights---------→8^2 8^1 8^0

$5*8^0=5$

$3*8^1=24$

$7*8^2=448$

$5+24+448=(477)_{10}$ Answer.

Example:$(0.24)_8$

Octal Number--→ .24

Weights----→ 8^{-1} 8^{-2}

$(2*8^{-1})=2*1/8=1/4$ $(4*8)^{-2}=4*1/8^2=4/64=1/16$

$(0.24)_8=(0.3125)_{10}$Answer.

b. Octal to Binary Number System: Octal number is one of the number systems which has value of base is $8(2^3)$, that means there only 8 symbols − 0, 1, 2, 3, 4, 5, 6, and 7,therefore eachoctal digit can represented by a 3-bit binary number.

Octal Symbol	Binary equivalent
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Table of 3-bit Octal Equivalent Binary code

Example:$(46)_8=(\ )_2$

Octal Number----------→4 6

3-Bit Octal Number----------→100 110

$(46)_8=(100110)_2$Answer.

Example: Octal Number----------→5 6 7 . 1 1 6

3-Bit Octal Number----------→ 101 110 111 001 001 110

$(567.116)_8=(101110111.001001110)_2$Answer.

c. Octal to Hexadecimal Number System: Octal to hexadecimal is similar to hexadecimal to octal number.But by the following rules we have to convert easily from octal to hexadecimal conversion:

1. Firstly, we will find the binary equivalent of 3-bit number and arrange in group combination. .
2. Next, convert binary number to its hexadecimal equivalent making 4-bit group combination.

Example 1: $(152.25)_8$

Step 1:

We write the three-bit binary digit for 1, 5, 2, and 5.

$(152.25)_8$=$(001101010.010101)_2$

So, the binary number of the octal number 152.25 is **$(001101010.010101)_2$**

Step 2:

1. Now, we make pairs of four bits on both sides of the binary point.

0 0110 1010.0101 01

On the left side, the first pair has only one digit, and on the right side, the last pair has only two-digit. To make them complete pairs of four bits, add zeros on extreme sides.

0000 0110 1010.0101 0100

2. Now, we write the hexadecimal digits, which correspond to each pair.

$(0000\ 0110\ 1010.0101\ 0100)_2=(6A.54)_{16}$ Answer.

Example:$(4\ 5\ 6\ 7\ .\ 2\ 6\ 6)_8$
$(\ 100\ 101\ 110\ 111\ 010\ 110\ 110\ )$
$(4567.266)_8=($ **$100101110111.010110110)_{16}$ Answer.**

1.3.3 Hexadecimal to Other Number System

In digital computer system,there are three possible conversion through Octal number system and these are given below:

a. Hexadecimal to Decimal Number System
b.Hexadecimal to Binary Number System
c. Hexadecimal to Octal Number System

a.Hexadecimal to Decimal Number System: In this method, all the conversion is similar to binary to decimal and octal to decimal. Before conversion we have to know about the equvilant of hexadecimal into decimal conversion and hexadecimal into binary conversion.

Example: (A10)16 = ( )10

Solution:
(A10)16=()10
A10

=A×162+1×161+0×160

=10×256+1×16+0×1

=2576

∴(A10)16=(2576)10

b. Hexadecimal to Binary Number System: In this method,the base value is 16,therefore each hexadecimal value can be converted into binary equivalent just replacing each hexadecimal digit with 4-bit binary equivalent.

Example:

1.(283)16 = ( )2

Solution:
(283)16=()2

2	8	3
0010	1000	0011

$\therefore(283)_{16}=(1010000011)_2$

c.Hexadecimal to Octal Number System: In this method,when converting from hexadecimal to octal,it is often easier to first convert the hexadecimal into binary and then from binary to octal number system.

Example: 1. (951)16 = ( )8

Solution:
(951)16=()8

First convert hexadecimal to binary
(951)16=()2

9	5	1
1001	0101	0001

∴(951)16=(100101010001)2

Now convert binary to octal
(100101010001)16=()8

100	101	010	001
4	5	2	1

$\therefore(100101010001)_2=(4521)_8$

$\therefore(951)_{16}=(4521)_8$**Answer.**

1.4 Representation of Signed Binary Numbers:

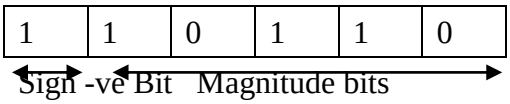
It can be represented by following three forms:

1.Sign-magnitude 2.One's Complement 3.Two's Complement

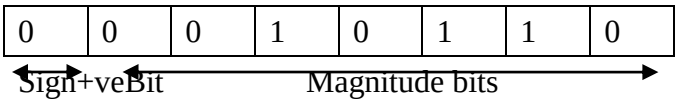
1.4.1 Sign-magnitude: In number representation technique sign-magnitude is represented by one signbit(0 or 1) and magnitude bits.In which '0' means positive(+ve) value and '1' means negative number(-ve) value.Addtionally, magnitude bits gives the value of signed number.

Example:110110 repersents $(-22)_{10}$

In above example,-ve value represent the 1 value and other 5 values represent the magnitude.



Example: (+22)=(10110)



1's compliment and 2's compliment: In digital computer system one compliment and two compliment are two important concepts.These two concepts lies under signed binary number.

1.4.2 One's Compliment

One's Compliment of any binary number can be obtained by inverting all the 1's by 0's and all the 0's by 1's.

Example:

Binary Number	**One's Complement**
101101 ----------→	010010
10011001----------→	01100110
0001010----------→	11101010

One;s complememt form is also used for representing singed number.

Example : $(0011)_2=(-3)_{10}$

Covert into One's Complement

$(1100)_2=(-3)_{10}$

After convert one complement, the negative value(-) is converted into (+) value and we get signed number of one's complement.

Example: $(0000)_2=(+0)_{10}$

Covert into One's complement

$(11111)_2=(-0)_{10}$

After convert one complement, the negative value(-) is converted into (+) value and we get signed number of one's complement.

1.4.3.Two's Complement

Two compliment is obtained by adding 1 to the 1's compliment of the binary number.The best part of the two's compliment is that it includes signed bit automatically.

Example:

101101 Binary Number

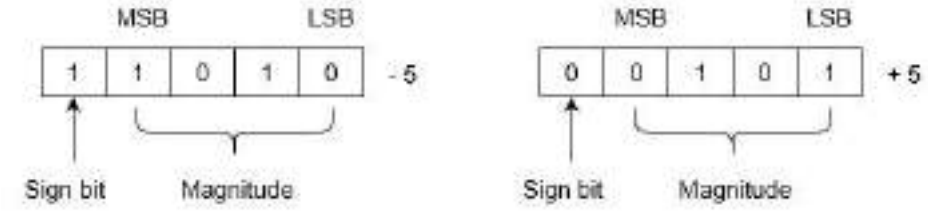
010010 One's Complememt

+1

010011 Two's Complement

Signed number are calculated similar to 1's Complement to 2's Complement.

Example: Let we are using 5 bits register. The representation of -5 and +5 will be as follows:



In above example,+5 is represented as it is represented in positive sign magnitude method. -5 is represented in negative sign magnitude.

Steps for represented -5 using the following steps:

(i) +5 = 0 0101

(ii) Take 1's complement of 0 0101 and that is 1 1010. MSB is 1 which indicates that number is negative.

MSB is always 1 in case of negative numbers.

1.5. Addition and Subtraction using one's complement

In binary representation ,we have to discuss the arithemetic operations on one's complement and two's complement .

1.5.1.Addition using 1's complement

There are three different cases possible when we add two binary numbers which are as follows:

Case I: When addition of the positive number with a negative number when the positive value has greater magnitude.

In this case,firstly we calculate the 1's complement of the given negative value and sum up with the given positive number.If we get the end-around carry 1, then it will added to the LSB.

Example:+ 1110 and - 1101

Solution:

		+		1		1		1		0	⇒	0 1		1		1	0
-	1	1	0	1	⇒	1 0		0		1	0	(taking		1's		complement)	
			0 0					0				0					0
					1												carry
			0 0					0				0					1

Answer. + 0001.

Example:+ 1101 and - 1011

(Assume that the representation is in a signed 5-bit register).

Solution:

+ 1 1 0 1 ⇒ 0 1 1 0 1

- 1 0 1 1 ⇒ 1 0 1 0 0 (taking 1's complement)

0 0 0 0 1

1 carry

0 0 0 1 0

Answer. + 0010.

Case 2: When addition of the positive number with a negative number in this case the negative number has a higher magnitude.

In this case,firstly calculate the 1's complement of the negative value and sum up with positive number but there will be non end-carry.After that,the sum is obtained by taking 1's complement of the magnitude bits of the result to get the final result.

Example: + 1010 and - 1100

Solution:

+ 1 0 1 0 ⇒ 0 1 0 1 0

- 1 1 0 0 ⇒ 1 0 0 1 1 (1's complement)

1 1 1 0 1

Answer. – 0010.

Example:+ 0011 and - 1101.

Solution:

+ 0 0 1 1 ⇒ 0 0 0 1 1

- 1 1 0 1 ⇒ 1 0 0 1 0 (1's complement)

1 0 1 0 1

Answer. – 1010.

Case 3: When addition of two negative numbers.

When the addition of two negative numbers 1's complement.In which, end-around carry will always considered.After that, which get added to the LSB, and for getting the final result, we take the 1's complement of the result.

Example: -1010 and -0101

Solution:

- 1 0 1 0 ⇒ 1 0 1 0 1 (1's complement)

- 0 1 0 1 ⇒ 1 1 0 1 0 (1's complement)

0 1 1 1 1

1 carry

1 0 0 0 0

1's complement of the magnitude bits of sum is 1111 and the sign bit is 1.

Answer. -1111.

Example:-0110 and -0111.

Solution:

- 0 1 1 0 ⇒ 1 1 0 0 1 (1's complement)

- 0 1 1 1 ⇒ 1 1 0 0 0 (1's complement)

1 0 0 0 1

1 carry

1 0 0 1 0

1's complement of 0010 is 1101 and the sign bit is 1.

Answer.- 1101.

1.5.2 Subtraction using 1’s complement

There are some steps to subtract two binary numbers using I’s complement.

1.Firstly, find the 1's complement of the subtrahend.

2.After that, add the complement number with the minuend.

3.If got a carry, add the carry to its LSB, otherwise it take 1's complement of the result which will be negative.

Example 1: 10101 - 00111

We take 1's complement of subtrahend 00111, which comes out 11000. Now, sum them. So,

10101+11000 =1 01101.

In the above result, we get the carry bit 1, so add this to the LSB of a given result,

i.e., 01101+1=01110

Example 2: 10101 - 10111

We take 1's complement of subtrahend 10111, which comes out 01000. Now, add both of the numbers. So,

10101+01000 =11101.

In the above result, we didn't get the carry bit. So calculate the 1's complement of the result, i.e., 00010, which is the negative number.

1.5.3.Binary Addition using 2’s Complement

Binary addition becomes easier when negative numbers are expressed using 2’s complement.

Case I: When the positive number has a greater magnitude while addition of the positive number with a negative number.

In the above case,find the 2’s complement of the given negative number and sum up with given positive number.If the carry value will be 1 then a positive number and carry value will be discared and left bits are the final result.

Example:(i) -1011 and -0101

Solution:

+ 1 0 1 1 ⇒ 0 1 0 1 1

- 0 1 0 1 ⇒ 1 1 0 1 1 (2's complement)

(Carry 1 discarded) 0 0 1 1 0

Answer. + 0110.

(ii) + 0111 and – 0011.

Solution:

+ 0 1 1 1 ⇒ 0 0 1 1 1

- 0 0 1 1 ⇒ 1 1 1 0 1

(Carry 1 discarded) 0 0 1 0 0

Answer.+ 0100.

Case II: When negative number has a higher magnitude.

Firstly, add a positive value with the 2's complement value of the negative number.If there is no carry value is found than take the 2's complement ang get the final result.

Example:(i) + 0 0 1 1 and - 0 1 0 1

Solution:

+ 0 0 1 1 ⇒ 0 0 0 1 1

- 0 1 0 1 ⇒ 1 1 0 1 1 (2's complement)

1 1 1 1 0

2’s complement of 1110 is (0001 + 0001) or 0010.

Answer. - 0010.

(ii) + 0 1 0 0 and - 0 1 1 1

Solution:

+ 0 1 0 0 ⇒ 0 0 1 0 0

- 0 1 1 1 ⇒ 1 1 0 0 1 (2’s complement)

1 1 1 0 1

2’s complement of 1101 is 0011.

Answer. – 0011.

Case III: When both the numbers are negative.

When two negative numbers are added a carry will be generated from the sign bit which will be discarded. 2’s complement of the magnitude bits of the operation will be the final sum.

(i) – 0011 and – 0101

Solution:

- 0 0 1 1 ⇒ 1 1 1 0 1 (2’s complement)

- 0 1 0 1 ⇒ 1 1 0 1 1 (2’s complement)

(Carry 1 discarded) 1 1 0 0 0

2’s complement of 1000 is (0111 + 0001) or 1000.

Answer. – 1000.

(ii) -0111 and – 0010.

Solution:

- 0 1 1 1 ⇒ 1 1 0 0 1 (2’s complement)

- 0 0 1 0 ⇒ 1 1 1 1 0 (2’s complement)

(Carry 1 discarded) 1 0 1 1 1

2’s complement of 0111 is 1001.

Answer. – 1001.

1.5.4. Subtraction using 2's complement:

There are some steps to subtract two binary numbers using I’s complement.

1.Firstly, find the 2's complement of the subtrahend.

2.After that,add the complement number with the minuend.

3.If we get the carry by adding both the numbers, then we discard this carry and the result is positive else take 2's complement of the result which will be negative.

Example 1: 10101 - 00111

2's complement of subtrahend 00111, which is 11001. Now, sum them. So,

10101+11001 =1 01110.

After solve it, the carry bit 1. So we discard this carry bit and get the final result and a positive number.

Example 2: 10101 - 10111

2's complement of subtrahend 10111, which comes out 01001. Now, additionof both of the numbers. So,

10101+01001 =11110.

After solve it, we didn't get the carry bit. So calculate the 2's complement of the result, i.e., 00010. It is the negative number and the final answer.

1.6.FIXED AND FLOATING POINT REPERSENTATION

In computer system,data is represented by binary bits 0's and 1's because computer understand only machine language either 0's and 1's.In Computer system bits stored in memory registers so, scientists have designed a real number representation method in 8-bit,16-bit and 32-bits combinations.There are two approaches that are developed to stored real numbers and these methods are:

1.Fixed point number 2.Floating point number

1.Fixed Point Number:

In computing, fixed-point number representation is a real data type for a number. By the use of fixed number representation, data is converted into binary form, and then data is processed, stored and used by the system.

Fixed point representation of data



Sign bit -The fixed-point numbers in binary uses a sign bit. A positive number has a sign bit 0,while a negative number has a sign bit 1.

Integral Part – The integral part is of different lengths at different places. It depends on the register's size, like in an 8-bit register, integral part is 4 bits.

Fractional part –Fractional part is similar to integral part.It takes different lengths at different places

8 bits = 1Sign bit + 4 bits(integral) + 3bits (fractional part)

16 bits = 1Sign bit + 9 bits(integral) +6 bits (fractional part)

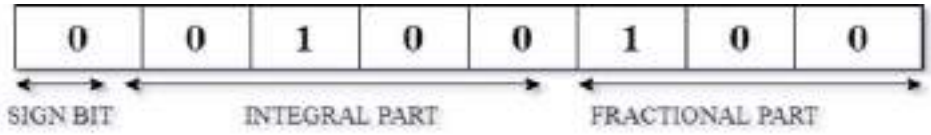
32 bits = 1Sign bit + 15 bits(integral) + 9 bits (fractional part)

Example:Number is 4.5

Step 1:- Convert the number into binary form.

4.5 = 100.1

Step 2:- Represent binary number in Fixed point notation



The smallest negative number in fixed-point representation.

1	**1**	**1**	**1**	**1**	**1**	**1**	**1**

Smallest negative number = -15.875

The largest number in fixed-point representation.

0	**1**	**1**	**1**	**1**	**1**	**1**	**1**

Larger number = +15.875

2.Floating-point Representation

Computer system uses floating-point numbers representation to convert input data into binary form. The binary form number is converted into '*scientific notation*,' and then this scientific notation is converted into floating-point representation.

The floating-point notation has two types of notation

1. Scientific notation
2. Normalized notation

Scientific notation – Method of representing binary numbers into $a \times b^e$ form. Scientific notation is further converted into floating-point notation because floating-point notation only accepts scientific notation.

For example:

Number = 376.423 (its not scientific notation)

Number in scientific = 36.4423×10^1 or 3.64423×10^2

a x b^e
a= any real number
b = base
e = exponent integer

For example: 32.625×10^3

$1101.101 * 2^{101}$

where 1101.101 is the mantissa part.

2^{101} = It is the base part where we need not explicitly represent radix or base because the binary base is always 2.

1.7.BINARY ADDITION AND SUBSTRACTION

Binary Addition: In Number System, the binary addition and substation is similar to the decimal number system. When we may add two numbers suppose 12 and 10, the result is 22.

The rules for binary addition are:

0 + 0 = 0 Sum of 0 with Carry of 0

0 + 1 = 1 Sum of 1 with Carry of 0
1 + 0 = 1 Sum of 1 with Carry of 0
1 + 1 = 10 Sum of 0 with Carry of 1
1 + 1 + 1 = 11 Sum of 1 with Carry of 1

Example:

10+10=100

carry 1 1 0
\+ 1 0
1 0 0

Example:

0011010 + 001100 = 00100110

1 1 carry
0 0 1 1 0 1 0 = 26_{10}
+ 0 0 0 1 1 0 0 = 12_{10}
0 1 0 0 1 1 0 = 38_{10}

Binary Subtraction: In Binary Subtraction, this is primary technique. In this method, ensure that the subtracting value is larger to smaller otherwise this method won't work properly. If minuend is smaller than the subtrahend than subtraction is possible.

The rules for binary addition are:

0 - 0 = 0 Subtract 0 with Borrow 0
0 - 1 = 1 Subtract 0 with Borrow 1
1 - 0 = 1 Subtract 1 with Borrow 0
1 - 1 = 10 Subtract 0 with Borrow 0

Case	A - B	Subtract	Borrow
1	0 - 0	0	0
2	1 - 0	1	0
3	1 - 1	0	0
4	0 - 1	0	1

Table of Binary Subtraction

Example: 1 1 0 1 Minuend

- 0 1 0 1Subtrahend

1 0 0 0

Borrow

Example: 1 0 10 1

-0 1 1 0

0 0 1 1

1.8.CHRACTER CODE REPERSENTION

We know that digital circuits and computer system understand binary numbers i.e 0 and 1 but in real environment we also practice on character codes such as Alphanumeric codes or ASCII codes.Latest computer system are used for transferring the data in form of names,letters and other symbols.So, it has deal with the letters and symbols for conveying intelligently information with the use of "Alpha-Numeric Codes".

ASCII Code:

ASCII codes are represented as American Standard Code for Information Interchange and is pronounced as 'ask-ee'.It is a seven bit code based on the English alphabets and these codes are introduced in 1967.Since, then ASCII codes are being modified and updated.It contains 128 characters because ASCII code is 7 bit code and it represent 2^7=128.In digital format,a total number of 95 printable characters.In which, 26 upper case letters(A-Z),26 lower case letters(a-z),10 numerals(0-9), and 33 special characters which contains mathematical symbols,punnchuation marks and some space characters.So, in latest systems ASCII code is development of 8bit code and it can be represent as 2^8=256 characters.

DEC	OCT	HEX	BIN	Symbol	Description

0	0	0	0	NUL	Null char
1	1	1	1	SOH	Start of Heading
2	2	2	10	STX	Start of Text
3	3	3	11	ETX	End of Text
4	4	4	100	EOT	End of Transmission
5	5	5	101	ENQ	Enquiry
6	6	6	110	ACK	Acknowledgment
7	7	7	111	BEL	Bell
8	10	8	1000	BS	Back Space
9	11	9	1001	HT	Horizontal Tab
10	12	0A	1010	LF	Line Feed
11	13	0B	1011	VT	Vertical Tab
12	14	0C	1100	FF	Form Feed
13	15	0D	1101	CR	Carriage Return
14	16	0E	1110	SO	Shift Out / X-On
15	17	0F	1111	SI	Shift In / X-Off
16	20	10	10000	DLE	Data Line Escape
17	21	11	10001	DC1	Device Control 1 (oft. XON)
18	22	12	10010	DC2	Device Control 2
19	23	13	10011	DC3	Device Control 3 (oft. XOFF)

20	24	14	10100	DC4	Device Control 4
21	25	15	10101	NAK	Negative Acknowledgement
22	26	16	10110	SYN	Synchronous Idle
23	27	17	10111	ETB	End of Transmit Block
24	30	18	11000	CAN	Cancel
25	31	19	11001	EM	End of Medium
26	32	1A	11010	SUB	Substitute
27	33	1B	11011	ESC	Escape
28	34	1C	11100	FS	File Separator
29	35	1D	11101	GS	Group Separator
30	36	1E	11110	RS	Record Separator
31	37	1F	11111	US	Unit Separator
32	40	20	100000		Space
33	41	21	100001	!	Exclamation mark
34	42	22	100010	“	Double quotes (or speech marks)
35	43	23	100011	#	Number
36	44	24	100100	$	Dollar
37	45	25	100101	%	Procenttecken
38	46	26	100110	&	Ampersand
39	47	27	100111	‘	Single quote

40	50	28	101000	(	Open parenthesis (or open bracket)
41	51	29	101001	)	Close parenthesis (or close bracket)
42	52	2A	101010	*	Asterisk
43	53	2B	101011	+	Plus
44	54	2C	101100	,	Comma
45	55	2D	101101	–	Hyphen
46	56	2E	101110	.	Period, dot or full stop
47	57	2F	101111	/	Slash or divide
48	60	30	110000	0	Zero
49	61	31	110001	1	One
50	62	32	110010	2	Two
51	63	33	110011	3	Three
52	64	34	110100	4	Four
53	65	35	110101	5	Five
54	66	36	110110	6	Six
55	67	37	110111	7	Seven
56	70	38	111000	8	Eight
57	71	39	111001	9	Nine
58	72	3A	111010	:	Colon
59	73	3B	111011	;	Semicolon

60	74	3C	111100	<	Less than (or open angled bracket)
61	75	3D	111101	=	Equals
62	76	3E	111110	>	Greater than (or close angled bracket)
63	77	3F	111111	?	Question mark
64	100	40	1000000	@	At symbol
65	101	41	1000001	A	Uppercase A
66	102	42	1000010	B	Uppercase B
67	103	43	1000011	C	Uppercase C
68	104	44	1000100	D	Uppercase D
69	105	45	1000101	E	Uppercase E
70	106	46	1000110	F	Uppercase F
71	107	47	1000111	G	Uppercase G
72	110	48	1001000	H	Uppercase H
73	111	49	1001001	I	Uppercase I
74	112	4A	1001010	J	Uppercase J
75	113	4B	1001011	K	Uppercase K
76	114	4C	1001100	L	Uppercase L
77	115	4D	1001101	M	Uppercase M
78	116	4E	1001110	N	Uppercase N
79	117	4F	1001111	O	Uppercase O

80	120	50	1010000	P	Uppercase P
81	121	51	1010001	Q	Uppercase Q
82	122	52	1010010	R	Uppercase R
83	123	53	1010011	S	Uppercase S
84	124	54	1010100	T	Uppercase T
85	125	55	1010101	U	Uppercase U
86	126	56	1010110	V	Uppercase V
87	127	57	1010111	W	Uppercase W
88	130	58	1011000	X	Uppercase X
89	131	59	1011001	Y	Uppercase Y
90	132	5A	1011010	Z	Uppercase Z
91	133	5B	1011011	[	Opening bracket
92	134	5C	1011100	\	Backslash
93	135	5D	1011101	]	Closing bracket
94	136	5E	1011110	^	Caret – circumflex
95	137	5F	1011111	_	Underscore
96	140	60	1100000	`	Grave accent
97	141	61	1100001	a	Lowercase a
98	142	62	1100010	b	Lowercase b
99	143	63	1100011	c	Lowercase c
100	144	64	1100100	d	Lowercase d

101	145	65	1100101	e	Lowercase e
102	146	66	1100110	f	Lowercase f
103	147	67	1100111	g	Lowercase g
104	150	68	1101000	h	Lowercase h
105	151	69	1101001	i	Lowercase i
106	152	6A	1101010	j	Lowercase j
107	153	6B	1101011	k	Lowercase k
108	154	6C	1101100	l	Lowercase l
109	155	6D	1101101	m	Lowercase m
110	156	6E	1101110	n	Lowercase n
111	157	6F	1101111	o	Lowercase o
112	160	70	1110000	p	Lowercase p
113	161	71	1110001	q	Lowercase q
114	162	72	1110010	r	Lowercase r
115	163	73	1110011	s	Lowercase s
116	164	74	1110100	t	Lowercase t
117	165	75	1110101	u	Lowercase u
118	166	76	1110110	v	Lowercase v
119	167	77	1110111	w	Lowercase w
120	170	78	1111000	x	Lowercase x
121	171	79	1111001	y	Lowercase y

122	172	7A	1111010	z	Lowercase z
123	173	7B	1111011	{	Opening brace
124	174	7C	1111100	|	Vertical bar
125	175	7D	1111101	}	Closing brace
126	176	7E	1111110	~	Equivalency sign – tilde
127	177	7F	1111111		Delete

According to table, we can see that 0-9 numbers represents binary numbers with 0011 prefix value. In the same way,upper case represents 0101 0000 to 0101 1010 and lower case represents by 0111 0000 to 0111 1010.

1.9.SUMMARY

1.https://www.javatpoint.com
2.http://gpmeham.edu.in
3.https://www.math-only-math.com
4.https://atozmath.com
5.Morris Mano of Computer System Architechture.
6.Aharon Yadin of Computer System Architechture.
7.Sajjan Singh and Gurpreet Sandhu of Computer System Architechture.

1.10.PRACTICE QUESTIONS

Key Exercise of Number System	
Solve Decimal to Binary	**Solve Decimal to Octal**
1.Convert $(27)_{10}$ to $(\ )_2$	1.Convert$(127)_{10}$ to$(\ )_8$
2.Convert $(39)_{10}$ to $(\ )_2$	2.Convert$(89.1625)_{10}$ to $(\ )_8$
3.Convert $(0.625)_{10}$ to $(\ )_2$	3.Convert$(0.5625)_{10}$ to $(\ )_8$
4.Convert $(25.15625)_{10}$ to $(\ )_2$	4.Convert$(469)_{10}$ to $(\ )_8$
5.Convert $(423.625)_{10}$ to $(\ )_2$	5.Convert$(287.684)_{10}$ to $(\ )_8$
Solve Decimal to Hexadecimal	**Solve Binary to Decimal**
1.Convert$(829)_{10}$ to$(\ )_{16}$	1.Convert $(11011)_2$ to $(\ )_{10}$
2.Convert$(778.7625)_{10}$ to $(\ )_{16}$	2.Convert $(10101.1101)_2$ to $(\ )_{10}$
3.Convert$(1268)_{10}$ to $(\ )_{16}$	3.Convert $(101101)_2$ to $(\ )_{10}$
4.Convert$(46)_{10}$ to $(\ )_{16}$	4.Convert $(1001.0101)_2$ to $(\ )_{10}$
5.Convert$(231.89)_{10}$ to $(\ )_{16}$	5.Convert $(0000.1111)_2$ to $(\ )_{10}$

Solve Binary to Octal 1.Convert $(101111)_2$ to ( )$_8$ 2.Convert $(11010111)_2$ to ( )$_8$ 3.Convert $(1000100.100110011)_2$ to ( )$_8$ 4.Convert $(1010101101.1110111)_2$ to ( )$_8$ 5.Convert $(0.001110)_2$ to ( )$_8$	**Solve Binary to Hexadecimal** 1.Convert $(101111)_2$ to ( )$_{16}$ 2.Convert $(11010111)_2$ to ( )$_{16}$ 3.Convert $(1000100.100110011)_2$ to ( )$_{16}$ 4.Convert $(1010101101.1110111)_2$ to ( )$_{16}$ 5.Convert $(0.001110)_2$ to ( )$_{16}$
Solve Octal to Decimal 1.Convert $(735)_8$ to ( )$_{10}$ 2.Convert $(0.24)_8$ to ( )$_{10}$ 3.Convert $(146.51)_8$ to ( )$_{10}$ 4.Convert $(265.78)_8$ to ( )$_{10}$ 5.Convert $(987)_8$ to ( )$_{10}$	**Solve Octal to Binary** 1.Convert $(46)_8$ to ( )$_2$ 2.Convert $(407)_8$ to ( )$_2$ 3.Convert $(1256)_8$ to ( )$_2$ 4.Convert $(567.116)_8$ to ( )$_2$ 5.Convert $(700.0356)_8$ to ( )$_2$
Solve Octal to Hexadecimal 1.Convert $(3562)_8$ to ( )$_{16}$ 2.Convert $(4567.266)_8$ to ( )$_{16}$ 3.Convert $(1256)_8$ to ( )$_{16}$ 4.Convert $(567.116)_8$ to ( )$_{16}$ 5.Convert $(750.0456)_8$ to ( )$_{16}$	**Solve Hexadecimal to Decimal** 1.Convert $(2A5B)_{16}$ to ( )$_{10}$ 2.Convert $(0.F1C)_{16}$ to ( )$_{10}$ 3.Convert $(2D.2D)_{16}$ to ( )$_{10}$ 4.Convert $(EEB)_{16}$ to ( )$_{10}$ 5.Convert $(A72.BF8)_{16}$ to ( )$_{10}$
Solve Hexadecimal to Binary 1.Convert $(A6C)_{16}$ to ( )$_2$ 2.Convert $(9A.1A)_{16}$ to ( )$_2$ 3.Convert $(F26.1BO)_{16}$ to ( )$_2$ 4.Convert $(29A6)_{16}$ to ( )$_2$ 5.Convert $(DC5A.3E4)_{16}$ to ( )$_2$	**Solve Hexadecimal to Octal** 1.Convert $(27A9)_{16}$ to ( )$_8$ 2.Convert $(85C.BD3)_{16}$ to ( )$_8$ 3.Convert $(F26.1B)_{16}$ to ( )$_8$ 4.Convert $(29A6)_{16}$ to ( )$_8$ 5.Convert $(DC5A.3E4)_{16}$ to ( )$_8$
Key Exercise of Binary Addition and Subtraction and 1's and 2's Complement	
Solve 1's Complement 1.00110101 2.11011001 3.111101010 4.11101000 5.110001000	**Solve 2's Complement** 1.10101110 2.01101000 3.11011001 4.11101000 5.110001111
Key Exercise of Sign-Magnitude	
Solve decimal equivalent of following Sign-Magnitude 1.001011	**Solve decimal equivalent of following Sign-Magnitude** 1.00010111

2.00001111	2.1010101

REFRENCES

https://www.javatpoint.com/conversion-of-number-system-in-digital-electronics
http://gpmeham.edu.in/wp-content/uploads/2019/04/Electrical-Engg-Digital-Electronics-4th-sem.pdf
https://www.math-only-math.com/binary-addition-using-1s-complement.html
https://atozmath.com/example/NumToBaseConv.aspx?he=e&b1=2&b2=16

Bachelor of Computer Applications (BCA)

COMPUTER SYSTEM ARCHITECTURE

UNIT II: INTRODUCTION TO BOOLEAN ALGEBRA

STRUCTURE

2.0 Objective

2.1 Introduction

2.2 Boolean Algebra

- **2.2.1. Laws of Boolean Algebra**
- **2.2.2. De Morgan's Theorems**

2.3 Logic gates

2.4 Sum of Product and Product of Sums Forms

2.5 K maps

2.6 Summary

2.7. Practice Questions

2.0 OBJECTIVE

Understanding Logic gates, Boolean algebra, K-Maps, Sum of Products, Product of Sums**.**

2.1 INTRODUCTION

In this unit Boolean algebra, rules of Boolean algebra, Logic gates , SOP and POS forms Of Boolean expression are discussed. This unit also discuss the Simplification of Boolean expression using K- Map.

2.2. BOOLEAN ALGEBRA

Boolean Algebra is a set B={a,b,c,……} containing at least two distinct elements on which two binary operations namely logical addition called OR and logical multiplication called AND operation along with a unary operation NOT are defined. It is used to analyse and simplify the digital circuits . It uses only the binary number 0 and 1 .Boolean Algebra was invented by George Boole in 1854.

Rules in Boolean Algebra :In Boolean Algebra variable used can have only two values 0 and 1.0 for low and 1 for high. complement of a variable is represented by a bar over the variable. The complement of 0 is 1 and 1 is 0 .Oring of two variable is represented by a + sign between them. logical Anding of two or more variables is represented by writing a dot between such as A.B.C.

2.2.1 Laws of Boolean Algebra

There are following type of laws in Boolean algebra.

1. Idempotent Law: It is an important law it states that

a) A.A=A

b) A+A=A

2. Commutative law: Commutative law states that changing the sequence of the variables does not have any effect on the output of a logic circuit.

a) A+B=B+A

b) A.B=B.A

3. Associative law : This law states that the order in which the logic operations are performed is irrelevant as their effect is the same.

a) A.(B.C)=(A.B).C

b) A+(B+C)=(A+B)+C

4. Distributive law

Distributive law states the following condition.

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

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

5. AND law

These laws use the AND operation. Therefore they are called as AND laws.

(i) $A.0 = 0$ (ii) $A.1 = A$

(iii) $A.A = A$ (iv) $A.\overline{A} = 0$

6. OR law

These laws use the OR operation. Therefore they are called as OR laws.

(i) $A + 0 = A$ (ii) $A + 1 = 1$

(iii) $A + A = A$ (iv) $A + \overline{A} = 1$

7. Inversion law

This law uses the NOT operation. The inversion law states that double inversion of a variable results in the original variable itself.

(A')'=A

8. Identity law

If 0 is additive identity and 1 is the multiplicative identity of Boolean algebra.

(i) A+1=1=1+A
(ii) A.0=0.A=0

9. Absorption Law

(i) A+(A.B)=A

(ii) A.(A+B)=A

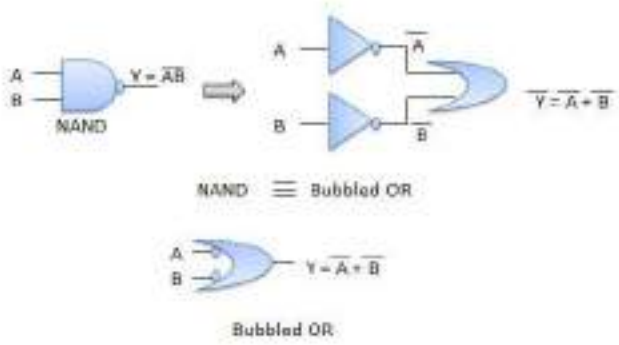
2.2.2 De Morgan's Theorems

De Morgan has suggested two theorems which are extremely useful in Boolean Algebra. The two theorems are discussed below.

De Morgan Theorem 1:

(A.B)'=A'+B'

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.



De Morgan Theorem 1 Diagram

Table showing verification of the De Morgan's first theorem –

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

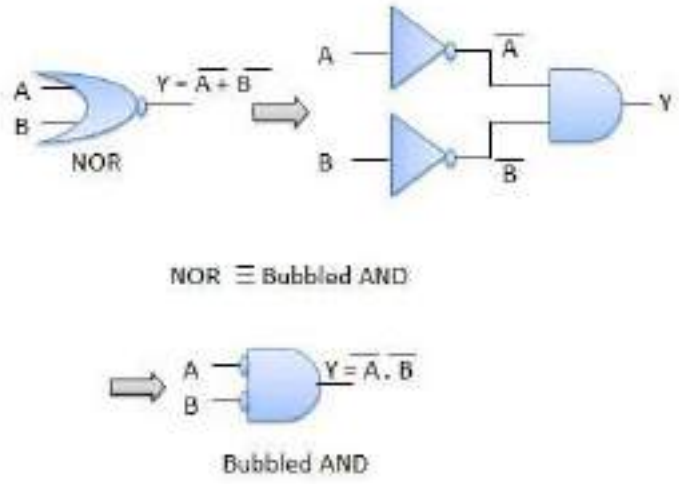
De Morgan Theorem 1 Verification Table

De Morgan Theorem 2:

The LHS of this theorem represents a NOR gate with inputs A and B, whereas the RHS represents an AND gate with inverted inputs.

(A+B)'=A'.B'

This AND gate is called as Bubbled AND.



De Morgan Theorem 2 Diagram

Table showing verification of the De Morgan's second theorem –

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

De Morgan Theorem 2 Verification Table

2.3 INTRODUCTION OF LOGIC GATES

Logic gates gate is an elementary building block of a digital circuit.Most logic gates have two inputs and one output at any given moment. Every input is in one of the two conditions boundary conditions low or high represented by the two different voltage levels.

A variety of logic gates are commonly used in digital computer system .Each gate has a distinct graphic symbol and its operation can be described by means of an algebraic expression.The input output relationship of binary variables for each gate can be represented in tabular form by the truth table.There are three basic logic gates AND ,OR,NOT

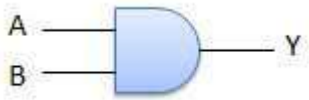
1. AND Gate

And gate is so named because , if 0 is “false” and 1 is called “true” the gate act is the same way as the logical and operation. The following illustration and table shows the circuit symbol and

logic combinations for an AND gate. The output is true when both inputs are true otherwise the output is false.

The algebraic operation symbol of the AND gate function is same as multiplication symbol of ordinary arithmetic .We can use either a dot between the variables or concatenate the variables without an operation symbol between them. According to the truth table, if any of the input is zero then output is also 0.

Logic Diagram



Truth table

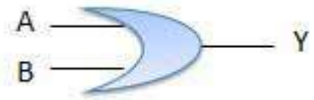
Inputs		Output
A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

2. OR Gate

The OR gate gets its name from the fact that it behaves after the fashion of the logical inclusive "OR". The output is "true" if either or both of the inputs are "True" .If both inputs are "false" then the output is "false". The truth table and the logic diagram is shown in figure below.

Its operation symbol is + similar to arithmetic addition it is if any of the input value is 1 the output is also equal to 1.

Logic Diagram



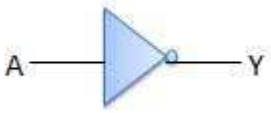
Truth table

Inputs		Output
A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

3.NOT Gate

A logical inverter sometimes called a “NOT” gate has only one input. It reverses the logic state. The algebraic equation used for logic complement is either a prime or a bar over the variable symbol. The triangle symbol also shows the “NOT” gate. It is used to complement the input if input is zero it becomes 1 and if it is 1 it becomes zero. The logic diagram and the truth table are shown in figure below

Logic Diagram



Truth table

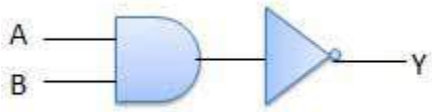
Inputs	Output
A	B
0	1
1	0

3. NAND Gate

The NAND gate operate as an AND gate followed by and NOT gate .It acts in the manner of the logical operation “AND” followed by a negation. The output is “false” if both the inputs are “true”. Otherwise, the output is “true”.

The logic symbol and truth table as shown in figure below

Logic Diagram





Truth table

Inputs		Output
A	B	$\overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

4. Nor Gate

The NOR gate is a combination of OR gate followed by an inverter. Its output is “true” if both inputs are “false”. Otherwise, the output is “false”.

The logic symbol and truth table as shown in figure below.

Logic gate



Truth table

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

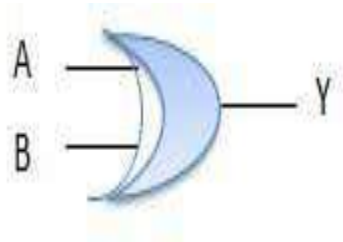
Both NAND gate and NOR gate are sometimes called **Universal gates** as any logical circuit can be drawn using only NAND gates or using only NOR gates. Both NAND gates and NOR gates may have more than two inputs and the output is always the complement of the AND or OR function respectively.

6.XOR and XNOR

The XOR (Exclusive OR) gate acts in the same way as the logical “either/ or”.The output is “true” if either, but not both, of the inputs are “true”.The output is “false” if both inputs are “false” or if both inputs are “true”.

The XNOR (exclusive NOR) gate is a combination of XOR gate followed by an inverter. Its output is true if the inputs are the same and “false” if the inputs at different. The truth table logical expression is shown in figure below.

Logic gate **Truth table**

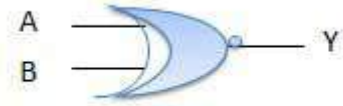


Inputs		Output
A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

7. XNOR GATE

XNOR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. The exclusive-NOR gate is abbreviated as EX-NOR gate or sometime as X-NOR gate. It has n input (n >= 2) and one output.

LOGIC GATE TRUTH TABLE



Inputs		Output
A	B	A ⊙ B
0	0	1
0	1	0
1	0	0
1	1	1

2.4 <u>SUM OF PRODUCT FORM(SOP) AND PRODUCT OF SUMS FORM(POP)</u>

A Boolean expression can be represented in either of the following forms

1. Sum of product form abbreviated as SOP form

2. Product of sums form usually abbreviated as POS form

1. SOP Form

SOP form of a Boolean expression which consists of sum of products of various literals either in product or complemented form is called sum of product form of the expression.

For example the following Boolean expression is in sum of product form

X.Y'+YX

The first term is X and Y' which is product of two Boolean variable hence it is product on the second term is Y into X is also a product term.

2.POS Form

POS form of Boolean expression is consisted of product of sums of various letters either indirect or complemented form is called product of sums form of the expression .

For example the following expression is in POS form

(X+Y)+(X+Y+Z')

Minterm :- A product is in sum of product form of a Boolean expression representing a Boolean function of and variable is called a Minterm if it compromises of exactly and literals each literal appearing only once either in direct or complemented form.

For example

F(X,Y,Z)=X.Y.Z+X'Y+YZ

Maxterm

A Sum term in product of sums (POS) form of a Boolean expression representing a Boolean function of n variables is called a Maxterm if it consists of exactly n literals, each literal appearing only once, either in direct are complemented form.

Consider the following function

F(X,Y,Z)=(X+Y+Z).(X+Y)

Standard order there are two canonical forms sum of product form and product of maxterms for stating any Boolean expression canonical form of a Boolean expression is any Boolean expression is said to be in canonical form or normal form

1. If each term in the expression consists of exactly and variables are literals

2. A literal either in direct or complemented form appears only once and only once in each term

Function is in SOP form is canonical SOP form if each of the product terms contain each and every variable or its complement each term in canonical SOP form is called a minterm.

If function is in POS form or canonical POS form if each of the sometimes contains each and every variable or its complement each term in canonical POS is called a maxterm.

Simplification of Boolean Expressions

Boolean Expression :-A Boolean expression like an algebraic expression consists of Boolean variables, constants connected by suitable Boolean operators +,. along with parenthesis.

Example :

(i) A+B.C+A'C+AB
(ii) (A+B').(B+C')

Simplification of boolean expression can be done using

(i) Boolean algebra
(ii) Karnaugh Map

Boolean rules for Simplification

Boolean algebra rules are used to simplify Boolean expression .For example

Expression	Rule(s) Used
AB(A + B)(B + B)	Original Expression
AB(A + B)	Complement law, Identity law.
(A + B)(A + B)	De Morgan's Law
A + BB	Distributive law. This step uses the fact that or distributes over and. It can look a bit strange since addition does not distribute over multiplication.
A	Complement, Identity.

Expression	Rule(s) Used
(A + C)(AD + AD) + AC + C	Original Expression
(A + C)A(D + D) + AC + C	Distributive.
(A + C)A + AC + C	Complement, Identity.
A((A + C) + C) + C	Commutative, Distributive.
A(A + C) + C	Associative, Idempotent.
AA + AC + C	Distributive.
A + (A + *T*)C	Idempotent, Identity, Distributive.
A + C	Identity, twice.

2.5 INTRODUCTION OF KARNAUGH MAP

The process of reducing a Boolean function to the minimal form by using the laws of Boolean algebra is a tedious one. A better and more elegant way to reduce a Boolean function to the minimal form is with the help of graphical method called Karnaugh Map method. It is a systematic method to reduce a switching function to the minimal form. **K- Map** provides an alternative technique for representing Truth Tables.

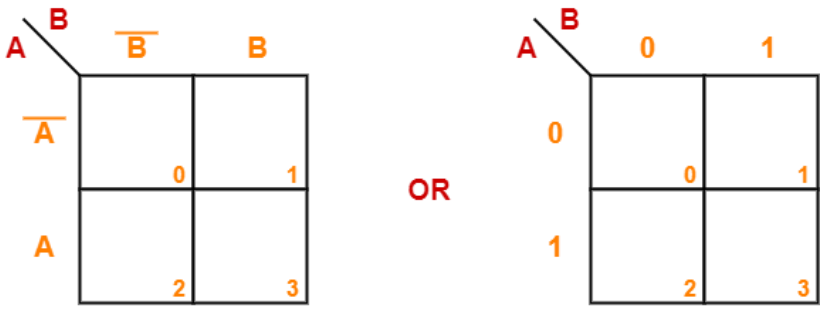
Types of K Map

1. 2 variable K -Map
2. 3 Variable k -Map
3. 4 Variable K- Map

1.Two Variable K Map-

- Two variable K Map is drawn for a boolean expression consisting of two variables.
- The number of cells present in two variable K Map = 2^2 = 4 cells.
- So, for a boolean function consisting of two variables, we draw a 2 x 2 K Map.

Two variable K Map may be represented as-



Two Variable K Map

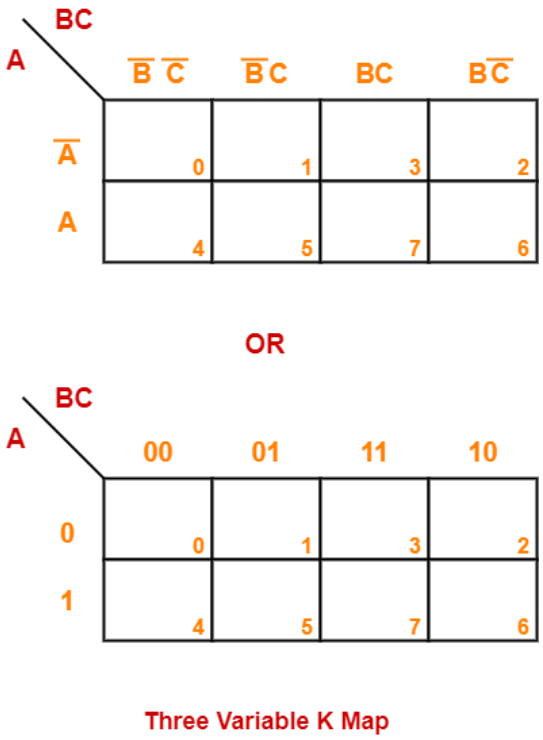
Here, A and B are the two variables of the given boolean function.

2.Three Variable K Map

Three variable K Map is drawn for a boolean expression consisting of three variables.

- The number of cells present in three variable K Map = 2^3 = 8 cells.
- So, for a boolean function consisting of three variables, we draw a 2 x 4 K Map.

Three variable K Map may be represented as-



Three Variable K Map

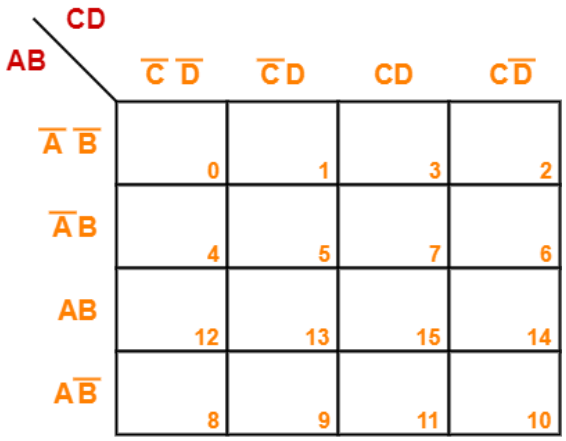
Here, A, B and C are the three variables of the given boolean function.

3.Four Variable K Map

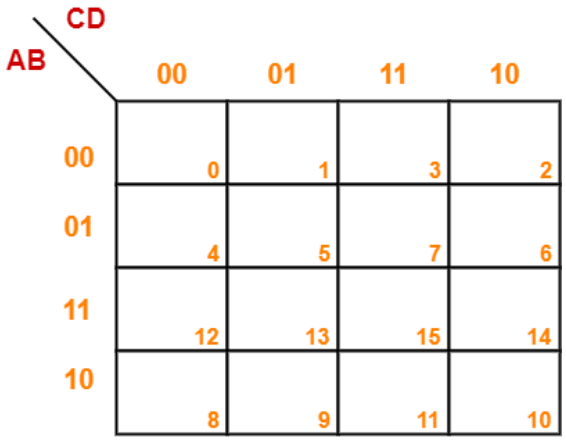
Four variable K Map is drawn for a boolean expression consisting of four variables.

- The number of cells present in four variable K Map = 2^4 = 16 cells.
- So, for a boolean function consisting of four variables, we draw a 4 x 4 K Map.

Four variable K Map may be represented as-



OR



Four Variable K Map

Here, A, B, C and D are the four variables of the given boolean function.

Reducing a Boolean function to the Minimal form using K-Map

Once the K-map has been drawn to represent the given Boolean function the next job is to reduce this function to the minimal form. A function is said to be in minimal form if

(i) It is represented by the minimum number of literals.
(ii) It has a minimum number of terms.

To reduce the Boolean function to the minimal form, follow the following rules

There are the following steps to find the minterm solution or K-map:

Step 1:

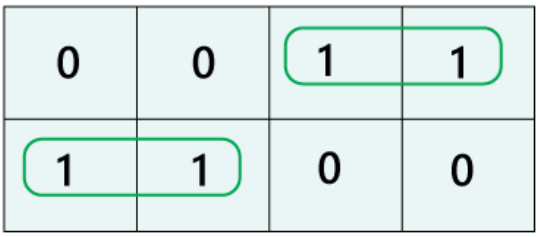
Firstly, we define the given expression in its canonical form.

Step 2:

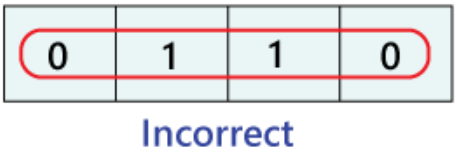
Next, we create the K-map by entering 1 to each product-term into the K-map cell and fill the remaining cells with zeros.

Step 3:

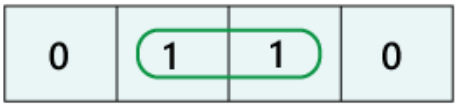
Next, we form the groups by considering each one in the K-map.



Notice that each group should have the largest number of 'ones'. A group cannot contain an empty cell or cell that contains 0.



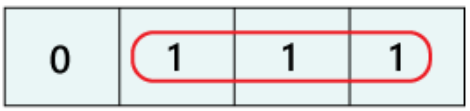
Incorrect



Correct

In a group, there is a total of 2^n number of ones. Here, n=0, 1, 2, …n.

Example: 2^0=1, 2^1=2, 2^2=4, 2^3=8, or 2^4=16.

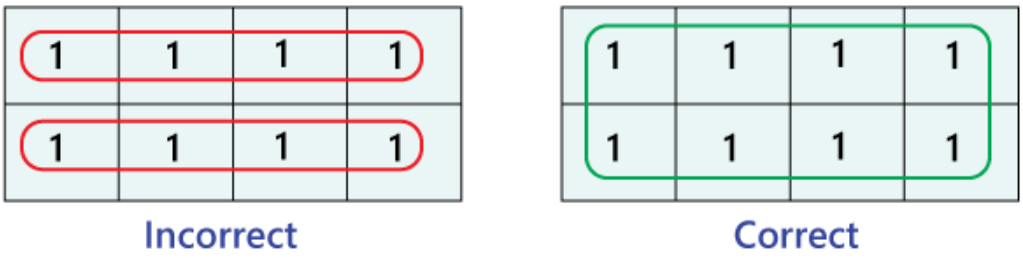


Incorrect

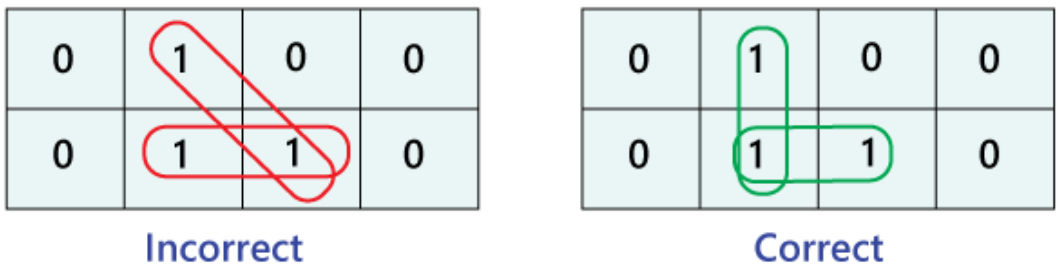


Correct

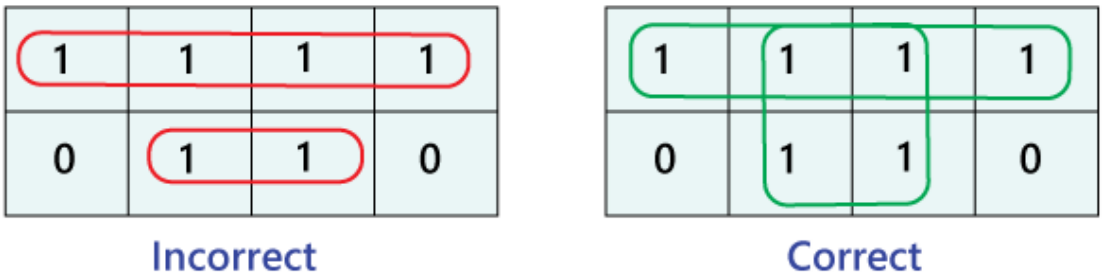
We group the number of ones in the decreasing order. First, we have to try to make the group of eight, then for four, after that two and lastly for 1.



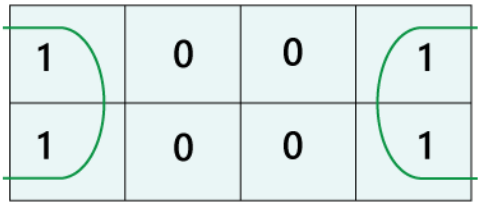
In horizontally or vertically manner, the groups of ones are formed in shape of rectangle and square. We cannot perform the diagonal grouping in K-map.



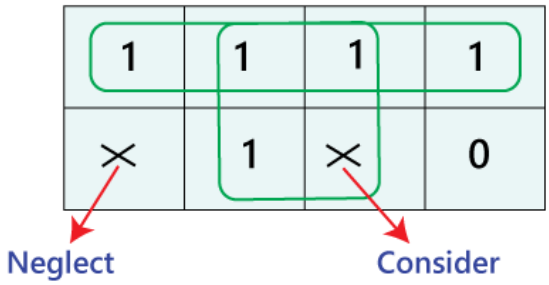
The elements in one group can also be used in different groups only when the size of the group is increased.



The elements located at the edges of the table are considered to be adjacent. So, we can group these elements.



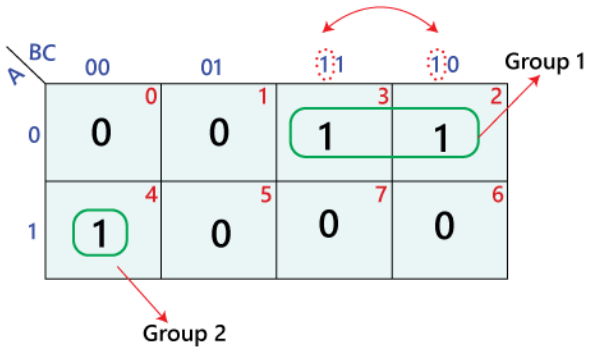
We can consider the 'don't care condition' only when they aid in increasing the group-size. Otherwise, 'don't care' elements are discarded.



Step 4:

In the next step, we find the boolean expression for each group. By looking at the common variables in cell-labeling, we define the groups in terms of input variables. In the below example, there is a total of two groups, i.e., group 1 and group 2, with two and one number of 'ones'.

In the first group, the ones are present in the row for which the value of A is 0. Thus, they contain the complement of variable A. Remaining two 'ones' are present in adjacent columns. In thes e columns, only B term in common is the product term corresponding to the group as A'B. Just like group 1, in group 2, the one's are present in a row for which the value of A is 1. So, the corresponding variables of this column are B'C'. The overall product term of this group is AB'C'.



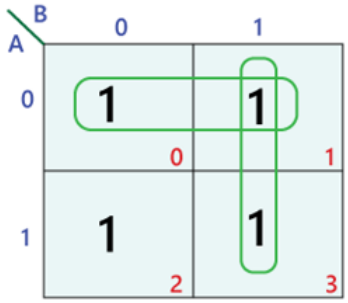
Step 5:

Lastly, we find the boolean expression for the Output. To find the simplified boolean expression in the SOP form, we combine the product-terms of all individual groups. So the simplified expression of the above k-map is as follows:

A'+AB'C'

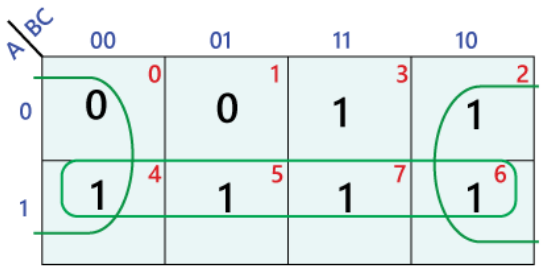
Let's take some examples of 2-variable, 3-variable, 4-variable, and 5-variable K-map examples.

Example 1: Y=A'B' + A'B+AB



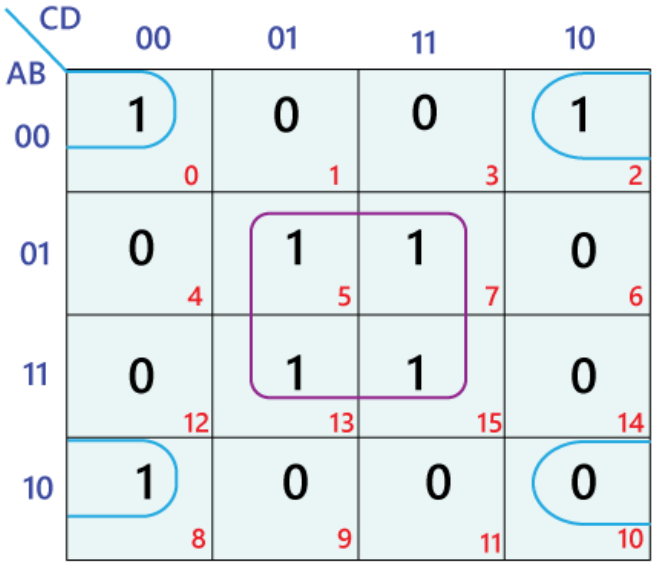
Simplified expression: Y=A'+B

Example 2: Y=A'B'C'+A' BC'+AB' C'+AB'C+ABC'+ABC



Simplified expression: Y=A+C'

Example 3: Y=A'B'C' D'+A' B' CD'+A' BCD'+A' BCD+AB' C' D'+ABCD'+ABCD



Simplified expression: y=bd+b'd'

Problems Based on Karnaugh Map

Problem-01:

Minimize the following boolean function-

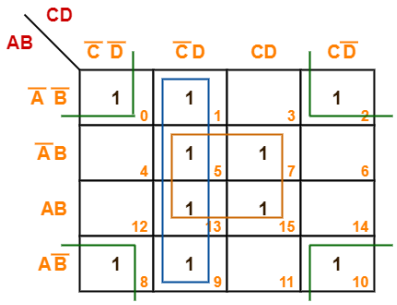
F(A, B, C, D) = Σm(0, 1, 2, 5, 7, 8, 9, 10, 13, 15)

Solution-

Since the given boolean expression has 4 variables, so we draw a 4 x 4 K Map.

- We fill the cells of K Map in accordance with the given boolean function.
- Then, we form the groups in accordance with the above rules.

Then, we have-



Now,

F(A, B, C, D)

= (A'B + AB)(C'D + CD) + (A'B' + A'B + AB + AB')C'D + (A'B' + AB')(C'D' + CD')

= BD + C'D + B'D'

Thus, minimized boolean expression is-

F(A, B, C, D) = BD + C'D + B'D'

Problem-02:

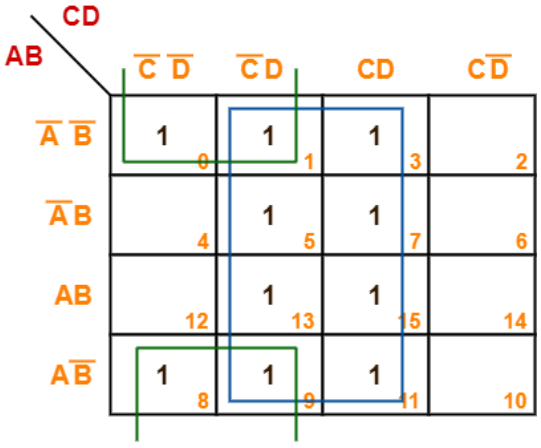
Minimize the following boolean function-

F(A, B, C, D) = Σm(0, 1, 3, 5, 7, 8, 9, 11, 13, 15)

Solution-

- Since the given boolean expression has 4 variables, so we draw a 4 x 4 K Map.
- We fill the cells of K Map in accordance with the given boolean function.
- Then, we form the groups in accordance with the above rules.

Then, we have-



Now,

F(A, B, C, D)

= (A'B' + A'B + AB + AB')(C'D + CD) + (A'B' + AB')(C'D' + C'D)

= D + B'C'

Thus, minimized boolean expression is-

F(A, B, C, D) = B'C' + D

Problem-03

Minimize the following boolean function-

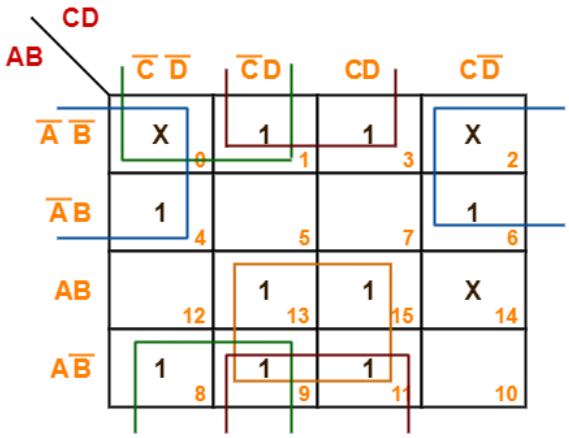
F(A, B, C, D) = Σm(1, 3, 4, 6, 8, 9, 11, 13, 15) + Σd(0, 2, 14)

Solution

Since the given boolean expression has 4 variables, so we draw a 4 x 4 K Map.

- We fill the cells of K Map in accordance with the given boolean function.
- Then, we form the groups in accordance with the above rules.

Then, we have-



Now,

F(A, B, C, D)

= (AB + AB')(C'D + CD) + (A'B' + AB')(C'D + CD) + (A'B' + AB')(C'D' + C'D) + (A'B' + A'B)(C'D' + CD')

= AD + B'D + B'C' + A'D'

Thus, minimized boolean expression is-

F(A, B, C, D) = AD + B'D + B'C' + A'D'

Problem-04:

Minimize the following boolean function-

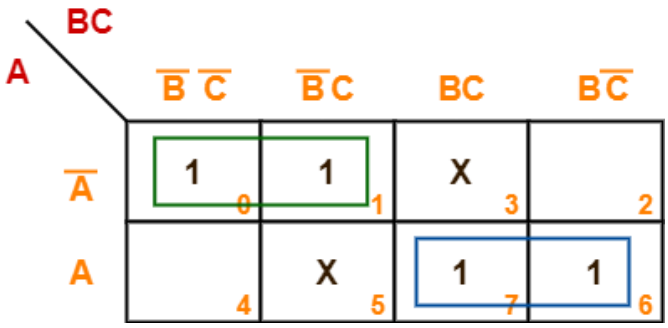
F(A, B, C) = Σm(0, 1, 6, 7) + Σd(3, 5)

Solution-

Since the given boolean expression has 3 variables, so we draw a 2 x 4 K Map.

- We fill the cells of K Map in accordance with the given boolean function.
- Then, we form the groups in accordance with the above rules.

Then, we have-



Now,

F(A, B, C)

= A'(B'C' + B'C) + A(BC + BC')

= A'B' + AB

Thus, minimized boolean expression is-

F(A, B, C) = AB + A'B'

2.6 SUMMARY

1.https://www.javatpoint.com

2.http://gpmeham.edu.in

3.https://www.math-only-math.com

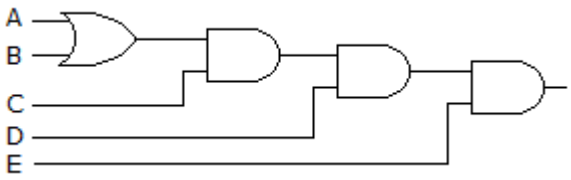
4.https://atozmath.com

5.Morris Mano of Computer System Architechture.

6.Aharon Yadin of Computer System Architechture.

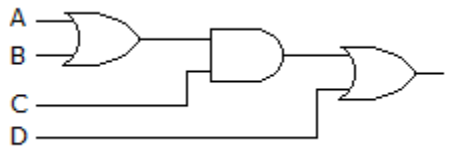
7.Sajjan Singh and Gurpreet Sandhu of Computer System Architechture.

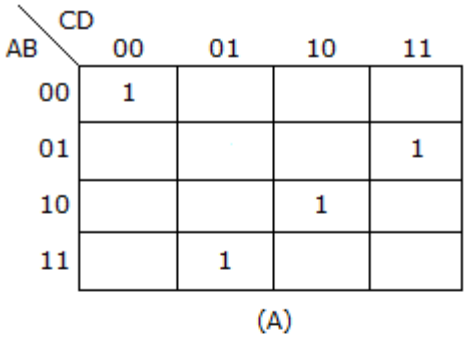
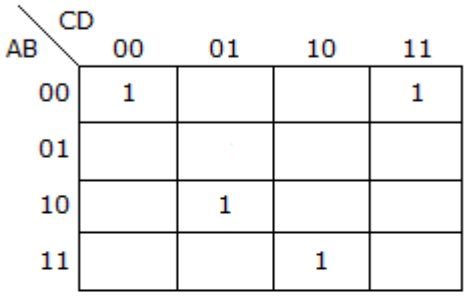
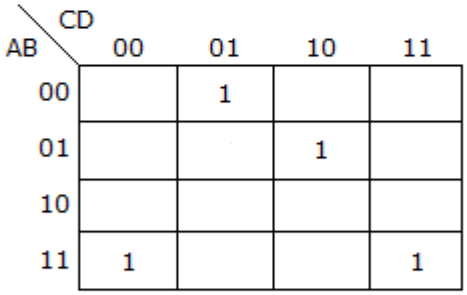
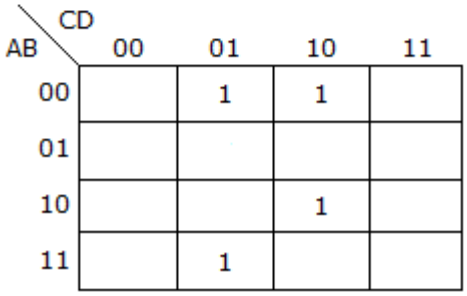
2.7 PRACTICE QUESTIONS

<table>
<tr><th colspan="2">Key Exercise of DeMorgan's theorem</th></tr>
<tr><td>1.Convert the following SOP expression to an equivalent POS expression.
$A\,B\,C + A\,\overline{B}\,\overline{C} + A\,\overline{B}\,C + A\,B\,\overline{C} + \overline{A}\,\overline{B}\,C$
A. $(\overline{A} + \overline{B} + \overline{C})(\underline{A} + B + \overline{C})(\overline{A} + B + C)$
B. $(A + B + C)(A + \overline{B} + C)(A + \overline{B} + \overline{C})$
C. $(\overline{A} + \overline{B} + \overline{C})(A + \overline{B} + C)(A + \overline{B} + C)$
D. $(A + B + C)(\overline{A} + B + \overline{C})(A + \overline{B} + C)$</td><td>2.Derive the Boolean expression for the logic circuit shown below:


A. C(A + B)DE
B. $\left[C(A + B)D + \overline{E}\right]$
C. $\left[\left[C(A + B)D\right]\overline{E}\right]$
D. ABCDE</td></tr>
<tr><td>3.Which of the following expressions is in the sum-of-products (SOP) form?
A. (A + B)(C + D)
B. (A)B(CD)
C. AB(CD)
D. AB + CD</td><td>4.One of De Morgan's theorems states that $\overline{X + Y} = \overline{X} \cdot \overline{Y}$. Simply stated, this means that logically there is no difference between:
A. a NOR and an AND gate with inverted inputs
B. a NAND and an OR gate with inverted inputs
C. an AND and a NOR gate with inverted inputs
D. a NOR and a NAND gate with inverted inputs</td></tr>
<tr><td>5.Applying DeMorgan's theorem to the expression $\overline{ABC}$, we get _____.
A. $\overline{A} + \overline{B} + \overline{C}$
B. $\overline{A + B + C}$</td><td>6.An AND gate with schematic "bubbles" on its inputs performs the same function as a(n)_____ gate.
A. NOT</td></tr>
</table>

C. $A + \overline{B} + C\overline{C}$ **D.** $A(B + C)$	**B.** OR **C.** NOR **D.** NAND
7.For the SOP expression $A\,\overline{B}\,C + \overline{A}\,B\,C + A\,B\,\overline{C}$, how many 1s are in the truth table's output column? **A.** 1 **B.** 2 **C.** 3 **D.** 5	8.A truth table for the SOP expression $A\,B\,\overline{C} + A\,\overline{B}\,C + \overline{A}\,\overline{B}\,C$ has how many input combinations? **A.** 1 **B.** 2 **C.** 4 **D.** 8
9.How many gates would be required to implement the following Boolean expression before simplification? XY + X(X + Z) + Y(X + Z) **A.** 1 **B.** 2 **C.** 4 **D.** 5	10.Determine the values of A, B, C, and D that make the product term $\overline{A}B\overline{C}D$ equal to 1. **A.** A = 0, B = 1, C = 0, D = 1 **B.** A = 0, B = 0, C = 0, D = 1 **C.** A = 1, B = 1, C = 1, D = 1 **D.** A = 0, B = 0, C = 1, D = 0
11.Which Boolean algebra property allows us to group operands in an expression in any order without affecting the results of the operation [for example, A + B = B + A]? **A.** associative **B.** commutative **C.** Boolean **D.** distributive	12.Applying DeMorgan's theorem to the expression $\overline{\overline{(X + Y)} + \overline{Z}}$, we get _____ **A.** $(X + Y)Z$ **B.** $(\overline{X} + \overline{Y})Z$ **C.** $(X + Y)\overline{Z}$ **D.** $(\overline{X} + \overline{Y})\overline{Z}$
13.Use Boolean algebra to find the most simplified SOP expression	14.Converting the Boolean expression LM + M(NO + PQ) to SOP form, we get _____.

for F = ABD + CD + ACD + ABC + ABCD. **A.** F = ABD + ABC + CD **B.** F = CD + AD **C.** F = BC + AB **D.** F = AC + AD	**A.** LM + MNOPQ **B.** L + MNO + MPQ **C.** LM + M + NO + MPQ **D.** LM + MNO + MPQ
15.A Karnaugh map is a systematic way of reducing which type of expression? **A.** product-of-sums **B.** exclusive NOR **C.** sum-of-products **D.** those with overbars	16.Applying the distributive law to the expression $A(B + \overline{C} + D)$, we get _____. **A.** $AB + AC + AD$ **B.** $ABCD$ **C.** $A + B + C + D$ **D.** $AB + A\overline{C} + AD$
17.Derive the Boolean expression for the logic circuit shown below:  **A.** $CA + CB + CD$ **B.** $C(A + B)\overline{D}$ **C.** $C(A + B) + D$ **D.** $CA + CB + D$	18.Mapping the standard SOP expression $A\,B\,C\,\overline{D} + \overline{A}\,\overline{B}\,C\,D + A\,\overline{B}\,\overline{C}\,D + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$, we get

	 (A)  (B)  (C)  (D)
19.Applying DeMorgan's theorem to the expression $\overline{(W + X + Y)Z}$, we get _____. **A.** $\overline{W}\,\overline{X}\,\overline{Y}\,\overline{Z}$	20.Determine the binary values of the variables for which the following standard POS expression is equal to 0.

B. $\overline{WXY}\,Z$	**A.** (0 + 1 + 0)(1 + 0 + 1)
C. $WXY\overline{Z}$	**B.** (1 + 1 + 1)(0 + 0 + 0)
D. $\overline{W}\,\overline{X}\,\overline{Y} + \overline{Z}$	**C.** (0 + 0 + 0)(1 + 0 + 1)
	D. (1 + 1 + 0)(1 + 0 + 0)

Bachelor of Computer Applications (BCA)

COMPUTER SYSTEM ARCHITECTURE

UNIT III: COMBINATIONAL CIRCUITS AND SEQUENTIAL CIRCUITS

STRUCTURE

3.0 Objective

3.1 Decoders

3.2 Encoders

3.3 Multiplexer

3.4 De-multiplexer

3.5 Half adders

3.6 Full adders

3.7 Flip-flops

3.8 Counters

3.9 Memory units

3.10 Practice Questions

3.0 OBJECTIVES

- To understand the basic concept of Decoders and Encoders
- To design the block diagram, truth table and logic diagram for MUX and De-MUX
- To design the block diagram of Half Adders and Full Adders using truth table and logic diagrams.
- To describe Flip- Flops and their types.
- To discuss Synchronous and Asynchronous Counters
- To understand the concept of memory units like RAM and ROM.

3.1 DECODERS

A decoder is a combinational circuit which converts n lines of input into 2^n lines of output. For example if we have 2 inputs then there will be 4 outputs. Similarly for 3 inputs, 8 outputs, Decoder performs the reverse operation of the **Encoder**. At a time, only one input line is activated for simplicity. The produced 2^N-bit output code is equivalent to the binary information. 2 to 4 decoder is shown below in the block diagram.

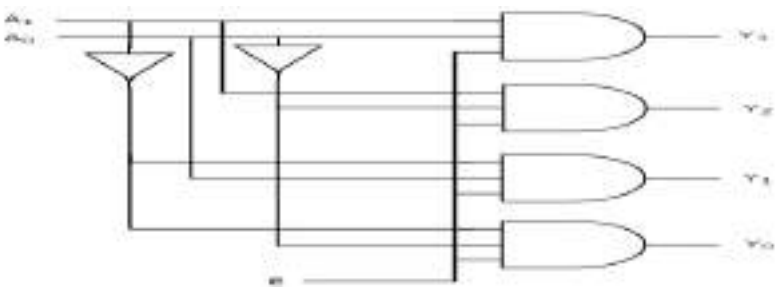


In 2 to 4 decoder, one E (enable) and 2 inputs A_1 and A_0 are used, a decoder will be active only when E is one, and then there will be four outputs like Y_0, Y_1, Y_2 and Y_3.

Enable	INPUTS		OUTPUTS			
E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

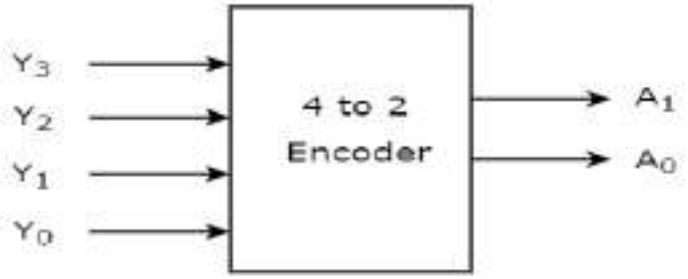
The Boolean function for Y0 is represented as:

$Y_0=E.A_1'A_0'$, $Y_1=EA_1'A_0$, $Y_2=EA_1A_0'$, $Y_3=A_1A_0$ and the logic diagram is shown in the figure below for 2 to 4 decoder.



3.2 ENCODERS

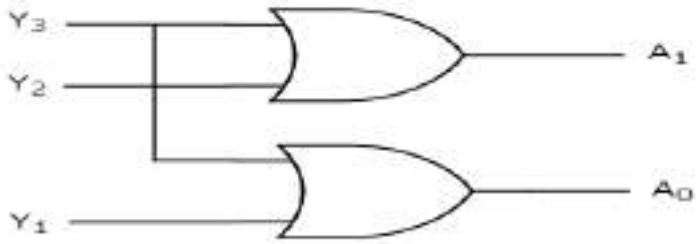
An encoder is a combinational circuit that converts binary information in the form of a 2^N input lines into N output lines, which represent N bit code for the input. For simple encoders, it is assumed that only one input line is active at a time. For example if we have 4 inputs there will be 2 outputs, similarly for 8 inputs there will be 3 outputs. 4 to 2 encoder is shown below in the block diagram.



Four inputs Y_0, Y_1, Y_2 and Y_3 and two output's A_0 and A_1 is shown in the table below.

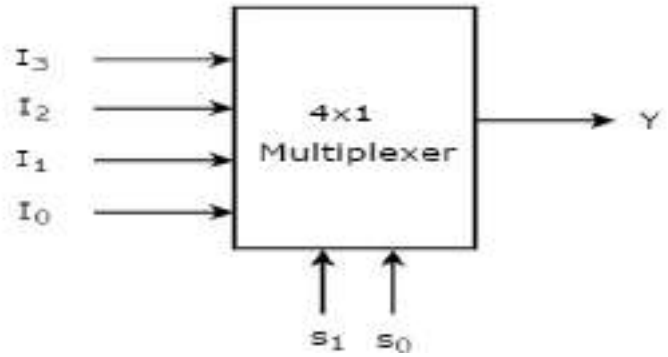
Inputs				Outputs	
Y_3	Y_2	Y_1	Y_0	A_1	A_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

The Boolean expression is evaluated from table as: $A_1= Y_3+Y_2$, $A_0= Y_3+Y_1$. The circuit diagram is shown below for 4 to 2 encoder.



3.3 MULTIPLEXER

Multiplexer is also a combinational circuit that can have maximum of 2^n data inputs, 'n' selection lines with single output line. Any one of these data inputs will be active based on the values of selection lines. 4x1 Multiplexer has four data inputs I_3, I_2, I_1 and I_0, two selection lines S_1 and S_0 and one output Y. The block diagram of 4x1 Multiplexer is shown in the following figure.

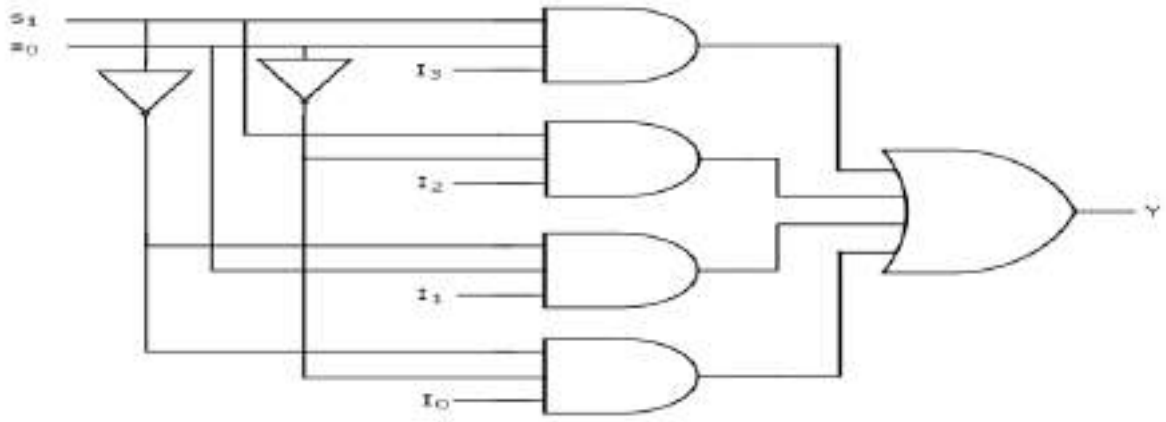


One of these 4 inputs will be connected to the output based on the combination of inputs present at these two selection lines. **Truth table** of 4x1 Multiplexer is shown below.

Selection Lines		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

From Truth table, we can directly write the **Boolean function** for output, Y as

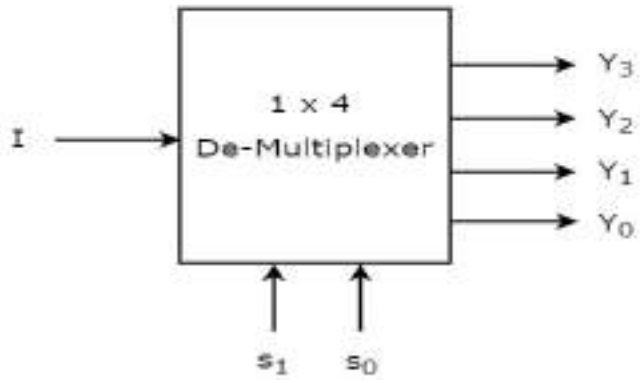
$$Y = S_1'S_0'I_0 + S_1'S_0I_1 + S_1S_0'I_2 + S_1S_0I_3$$



3.4 DE-MULTIPLEXER

It is also a combinational circuit that accomplishes the opposite operation of Multiplexer. It has single input, 'n' selection lines and maximum of 2^n outputs. The input will be connected to one of these outputs based on the values of selection lines. As there are 'n' selection lines, there will be 2^n possible combinations of zeros and ones. So, each combination can select only one output. For example 1X4 De-Multiplexer block diagram is shown in the block diagram which has one input and four outputs with two selection lines

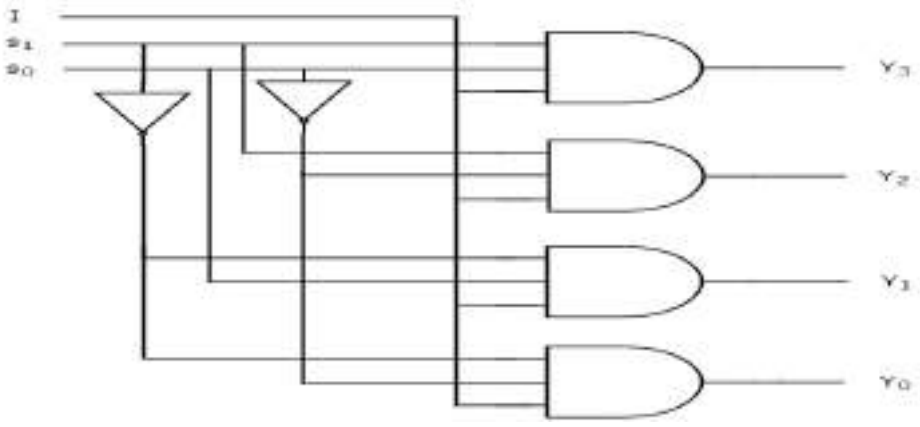
It has one input I, two selection lines, s_1 and s_0 and four outputs Y_3, Y_2, Y_1 and Y_0. The block diagram of 1x4 De-Multiplexer is shown in the following figure.



The single input 'I' is being connected to one of the four outputs, Y_3 to Y_0 based on the values of two selection lines s_1 and s0 as shown in the Truth table. The logic function is generated using as $Y_3=S_1\ S_0\ I$, $Y_2=S_1S_0'\ I$, $Y_1=S_1'S_0I$, $Y_0=S_1\ 'S_0'\ I$, or it may be written using 0 as S_0' and 1 as S_0 .

Selection Inputs		Outputs			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

The logic diagram for DEMUX is shown below:



3.5 HALF ADDERS

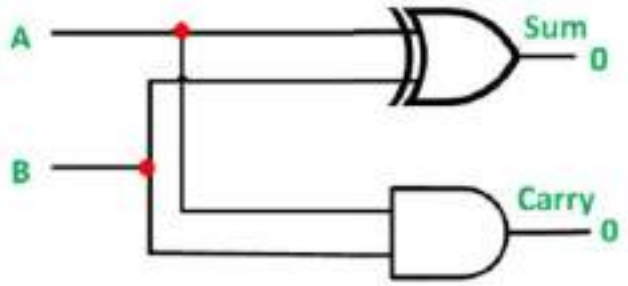
Half Adders is a combinational circuit which can add maximum two inputs and provide one output as SUM and other output as Carry. For example in the block diagram A and B are inputs and two outputs are Carry and Sum.



The truth table of Half Adders is shown below with two inputs A and B. Sum is performed after adding A and B like 0+0=0, 0+1=1, 1+0=1 and 1+1 is 0 with carry 1.

Inputs		Outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

The output of Sum is A Ex-or B, Carry is A And B which is represented in the logic diagram shown below:



3.6 FULL ADDERS

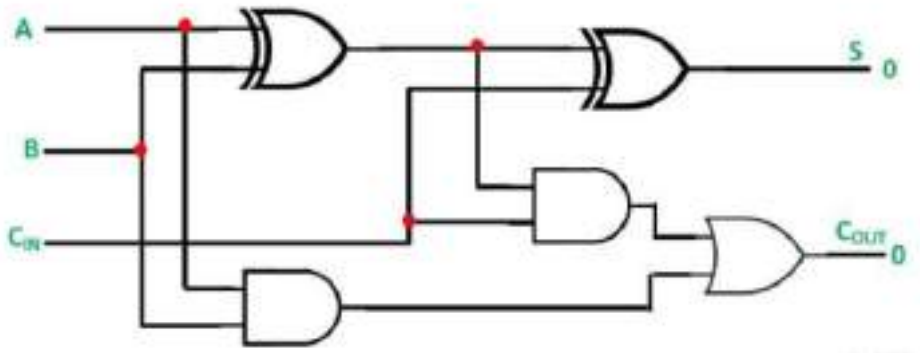
The full adder is a combinational circuit which can add more than two bits at the same time. For example in the block diagram there are three inputs A, B and a carry input C_{IN} and two outputs C_{OUT} and SUM.



The truth table for full adder is shown below like if A=0, B=, Cin=0, S will become 0 and Cout also be 0. Similarly, other inputs will be evaluated.

Inputs			Outputs	
A	B	CIN	COUT	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

The output of S will be Ex-Or operation between A, b and CIN, Cout is the AND operation between AB+BCin+ACin.



3.7 FLIP-FLOPS

Flip flop is formed using logic gates, which are made of transistors. Flip flops are considered as the basic building blocks in the memory of electronic devices. Each flip flop is capable to store one bit of data. Flip – flops have two stable states and therefore they are also considered

as bistable multi-vibrators. The two stable states of flip-flops are High (logic 1) and Low (logic 0).The term flip – flop actually means that they can switch between the states under the impact of a control signal (clock or enable). In other words, they can 'flip' to one state and 'flop' back to other state.

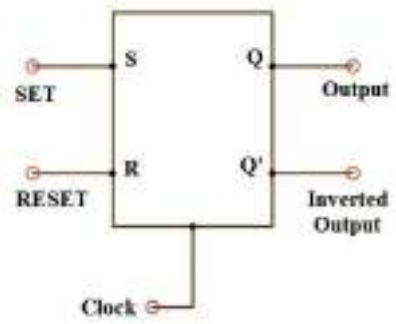
Types of flip flops

As per the operation of flip-flops, they are basically divided into following four types:

- S-R flip flop
- D flip flop
- J-K flip flop
- T flip flop

3.7.1 S-R Flip Flop

The S-R flip-flop is the basic flip-flop among all other flip-flops. All the other flip flops are developed after SR-flip-flop. S-R stands for SET and RESET. This can also be termed as RS flip-flop. The only difference is RS is inverted SR flip-flop.

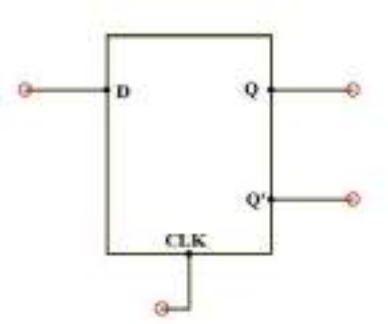


S-R Flip flop

S	R	Q	Q'	State
1	0	1	0	Q is set to 1
1	1	1	0	No change
0	1	0	1	Q' is set to 1
1	1	0	1	No change
0	0	1	1	Invalid

3.7.2 D flip flop

In the SR flip flop, sometimes an uncertain state occurred, which can be eluded by using D flip flop. D stands for "Data" in D flip-flop and is basically created from SR flip flop. The two inputs (S &R) of the clocked SR flip flop are connected to an inverter. It is one of the most widely used flip – flops. It has a clock signal (Clk) as one input and Data (D) as other. There are two outputs which complement each other. The symbol of D flip – flop is shown below.



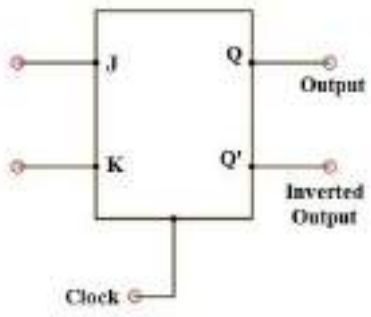
D Flip-Flop

Clock	D	Q	Q'	State
0	0	Q	Q'	No change
1	0	0	1	Reset Q to 0
1	1	1	1	Sets Q to 1

Truth Table of D Flip flop

3.7.3 J-K Flip Flop

JK flip – flop is named after Jack Kilby, an electrical engineer who developed IC. A JK flip – flop is a variation of SR flip – flop. In this, the J input is similar to the set input of SR flip – flop and the K input is similar to the reset input of SR flip – flop. The condition J = K = 1 which is not allowed in SR flip – flop (S = R = 1) is interpreted as a toggle command. The JK flip flop has two data inputs J and K, one clock signal input (CLK) and two outputs Q and Q'.



JK Flip flop

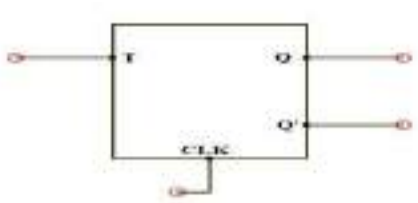
Clock	J	K	Q	Q'	State
1	0	0	Q	Q'	No change
1	0	1	0	1	Reset Q to 0
1	1	0	1	0	Sets Q to 1
1	1	1	-	-	Toggles

Truth Table of JK flip flop

3.7.4 T Flip Flop

T flip flop is also known as "Toggle Flip – flop". Toggle is to change the output to complement the previous state in the presence of clock input signal. The T flip flop has T input, one clock signal input (CLK) and two outputs Q and Q'.

T	Q	Q'
0	0	0



T Flip flop

1	0	1
0	1	0
1	1	0

Truth Table of T Flip flop

3.8 COUNTERS

A distinctive type of sequential circuit used to count the pulse is known as a counter. In other words, a group of flip flops where the clock signal is applied is known as counters. The counter is one of the broadest applications of the flip flop. Based on the clock pulse, the output of the counter contains a predefined state. The number of the pulse can be counted using the output of the counter. The truth table is represented as follows:

Clock	Counter output		State number	Decimal counter output
	Q_B	Q_A		
Initially	0	0	-	0
1st	0	1	1	1
2nd	1	0	2	2
3rd	1	1	3	3
4th	0	0	4	0

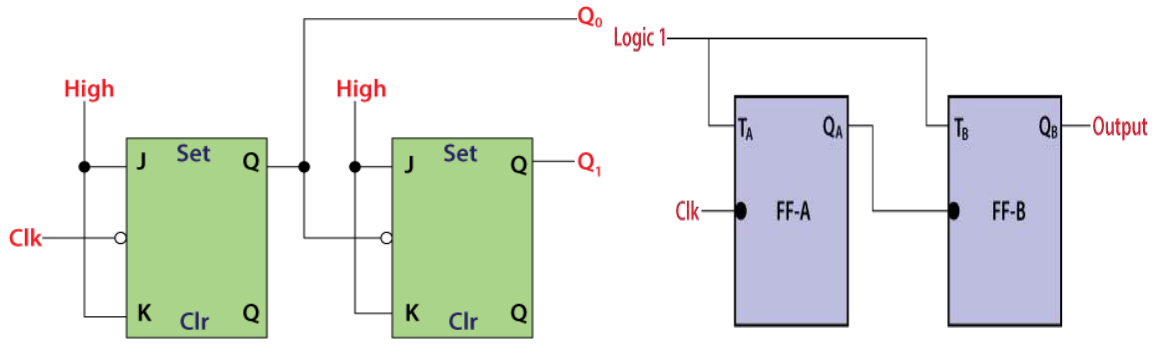
There are the two types of counters:

- Asynchronous Counters
- Synchronous Counters

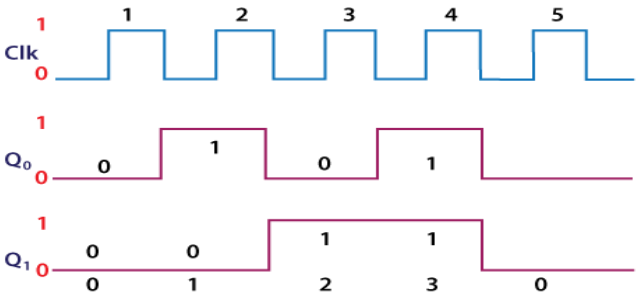
3.8.1 Asynchronous or ripple counters

An asynchronous counter is also termed as the ripple counter. Separately from the T flip flop, JK flip flop can also be used in a 2-bit asynchronous counter by setting both of the inputs to 1 permanently. The external clock pass to the clock input of the first flip flop, i.e., FF-A and its output, i.e., is passed to clock input of the next flip flop, i.e., FF-B. The diagram of the 2-bit Asynchronous counter in which we used two T flip-flops is shown below:

Block Diagram



Signal Diagram



Operation

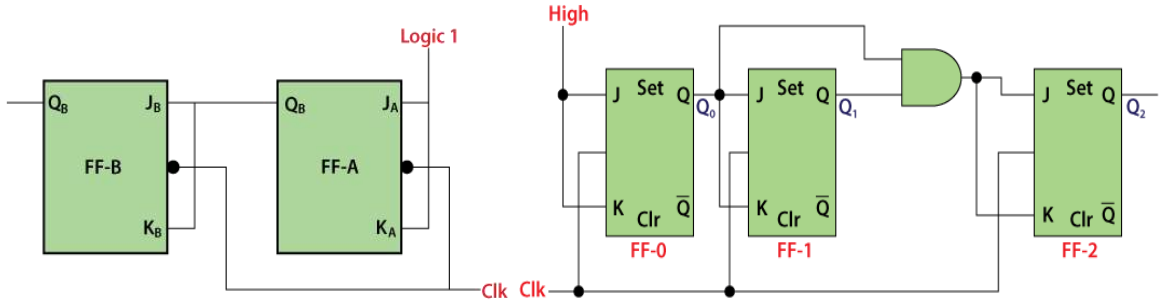
	Condition	Operation
1.	When both the flip flops are in reset condition	The outputs of both flip flops, i.e., **Q_A Q_B, will be 0.**
2.	When the first negative clock edge passes	The first flip flop will toggle, and the output of this flip flop will change from 0 to 1. The output of this flip flop will be taken by the clock input of the next flip flop. This output will be taken as a positive edge clock by the second flip flop. This input will not change the second flip flops output state because it is the negative edge triggered flip flop. **So, Q_A = 1 and Q_B = 0**
3.	When the second negative clock edge is	The first flip flop will toggle again, and the output of this flip flop will change from 1 to 0. This output will be taken

	applied	as a negative edge clock by the second flip flop. This input will change the second flip flop's output state because it is the negative edge triggered flip flop. **So, Q_A = 0 and Q_B = 1.**
4.	When the third negative clock edge is applied	The first flip flop will toggle again, and the output of this flip flop will change from 0 to 1. This output will be taken as a positive edge clock by the second flip flop. This input will not change the second flip flop's output state because it is the negative edge triggered flip flop. **So, Q_A = 1 and Q_B = 1**
5.	When the fourth negative clock edge is applied	The first flip flop will toggle again, and the output of this flip flop will change from 1 to 0. This output will be taken as a negative edge clock by the second flip flop. This input will change the output state of the second flip flop. **So, Q_A = 0 and Q_B = 0**

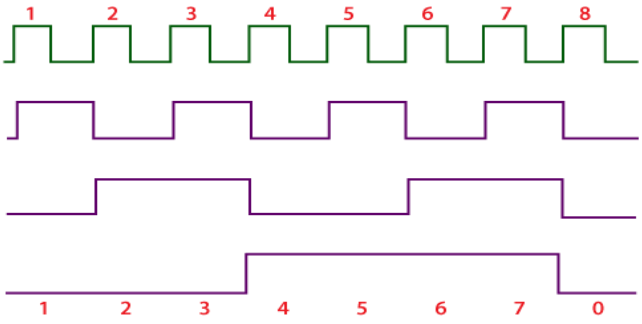
3.8.2 Synchronous counters

In the asynchronous counter, the present counter's output passes to the input of the next counter. So, the counters are connected like a chain. The drawback of these counters is that it creates the counting delay, and the propagation delay during the counting stage. The synchronous counter is designed to remove this drawback. In the synchronous counter, the same clock pulse is passed to the clock input of all the flip flops. The clock signals produced by all the flip flops are the same as each other. The diagram of a 2-bit synchronous counter is shown below in which the inputs of the first flip flop, i.e., FF-A, are set to 1. So, the first flip flop will work as a toggle flip-flop. The output of the first flip flop is passed to both the inputs of the next JK flip flop.

Logical Diagram



Signal Diagram



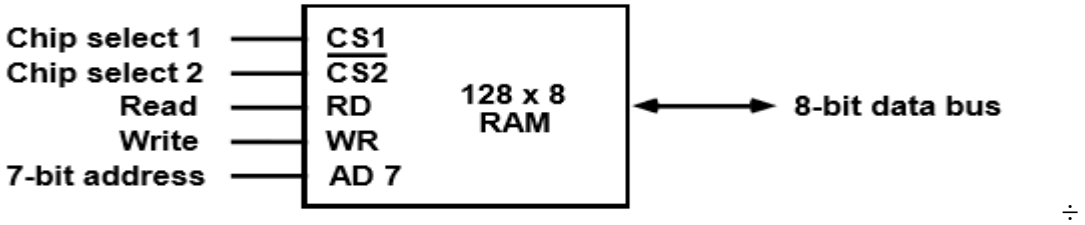
Operation

	Condition	Operation
1.	When both the flip flops are in reset condition	The outputs of both flip flops, i.e., Q_A Q_B, will be 0. So, $\mathbf{Q_A = 0}$ and $\mathbf{Q_B = 0}$
2.	When the first negative clock edge passes	The first flip flop will be toggled, and the output of this flip flop will be changed from 0 to 1. When the first negative clock edge is passed, the output of the first flip flop will be 0. The clock input of the first flip flop and both of its inputs will set to 0. In this way, the state of the second flip flop will remain the same. So, $\mathbf{Q_A = 1}$ and $\mathbf{Q_B = 0}$
3.	When the second negative clock edge is applied	The first flip flop will be toggled again, and the output of this flip flop will be changed from 1 to 0. When the second negative clock edge is passed, the output of the first flip flop will be 1. The clock input of the first flip flop and both of its inputs will set to 1. In this way, the state of the second flip flop will change from 0 to 1.

		So, Q_A = **0** and Q_B = **1**
4.	When the third negative clock edge is applied	The first flip flop will toggle from 0 to 1, but at this instance, both the inputs and the clock input set to 0. Hence, the outputs will remain the same as before. So, Q_A = **1** and Q_B = **1**
5.	When the fourth negative clock edge is applied	The first flip flop will toggle from 1 to 0. At this instance, the inputs and the clock input of the second flip flop set to 1. Hence, the outputs will change from 1 to 0. So, Q_A = **0** and Q_B = **0**

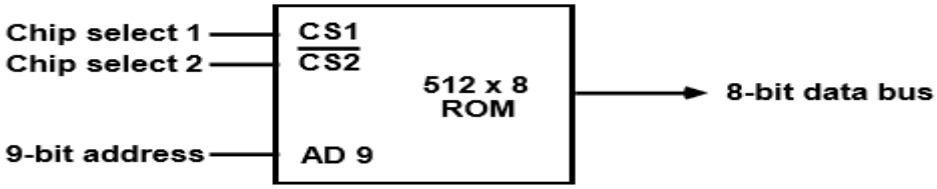
3.9 MEMORY UNITS

Main memory is divided between RAM and ROM, Where RAM size for example is shown in the block diagram is 128X8, 7 address lines and 8 data lines are used to represent this memory. CS1 and CS2 are chip select signals, RD and WR are read and write pins.



÷

ROM size for example is taken as 512X8, indicates 9 address lines and 8 data lines. CS1 and CS2 are chip select signals.



3.10 PRACTICE QUESTIONS

Q1: Differentiate between Encoders and Decoders.

Q2. What are the applications of Multiplexer and DE multiplexer?

Q3. Design Full adder using two Half adders.

Q4. Design the block diagram to convert from S-R flip flop to J-K

Q5. What are the applications of Flip Flops?

Q6. Differentiate between Synchronous and Asynchronous counters.

Q7. How many address lines are required to construct a 1024X8 bytes of RAM?

Bachelor of Computer Applications (BCA)

COMPUTER SYSTEM ARCHITECTURE

UNIT IV: BASIC COMPUTER ORGANIZATION AND DESIGN

STRUCTURE

4.0 Objectives

4.1 Computer Architecture and Structure

4.2 Computer Registers

4.3 Common Bus Systems

4.4 Arithmetic Micro-operations

4.5 Logic Micro-operations

4.6 Shift Micro-operations

4.7 Design of Arithmetic and Logic Unit

4.8 Summary

4.0 OBJECTIVES

- To study about the basics of Computer Architecture and Structure.
- To understand the concepts of Registers and their values based on Memory Unit.
- To know about the Common bus system for transferring data between memory and other registers.
- To describe arithmetic micro-operations through Arithmetic Circuit
- To study about the use of logical and shift micro-operations using examples.
- To design the ALU using various arithmetic, logical and shift micro-operations.

4.1 COMPUTER ARCHITECTURE AND STRUCTURE

Computer Architecture: Computer architecture is a set of rules and methods dealing how software and hardware technology standards interact to form a computer system. The architecture of a system is separately specified components in terms to its structure.

There are different types of computer architectures:

- **Von-Neumann Architecture:** Von-Neumann also known as Princeton architecture, is based on stored program computer concept, where we have single memory for read/write operations for instructions and data.
- **Harvard Architecture:** In the Harvard Architecture, the memory is split into two parts – one for data and another for programs. Data and instructions can be accessed from different memory locations.
- **Instruction Set Architecture:** An Instruction Set Architecture (ISA) is an abstract model of a computer. It describes the design of a computer in terms of the basic operations it must support. An ISA may be classified in a number of different ways. A common classification is by architectural complexity – Complex Instruction Set Computer (CISC) and Reduced Instruction Set Computer (RISC).

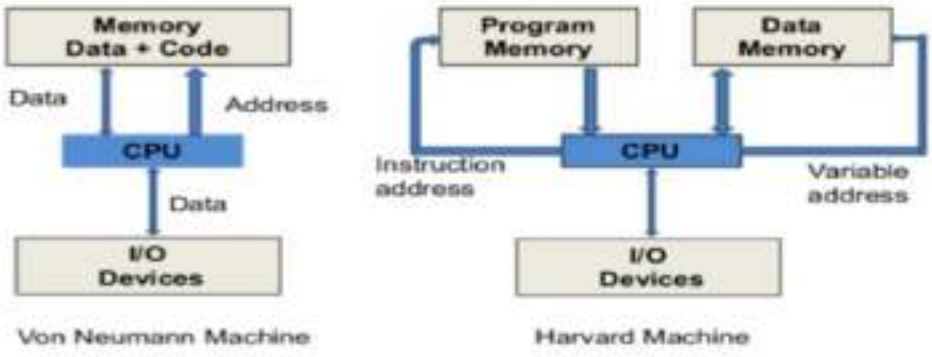


Figure 1: Von-Neumann and Harvard Architecture

Computer Organization: Computer Organization is concerned with how the various components of computer hardware operate and they are interconnected to implement the architectural specification.

In Computer Organization, there are different types of mechanisms:

- General register organization
- Stack organization
- Single accumulator organization

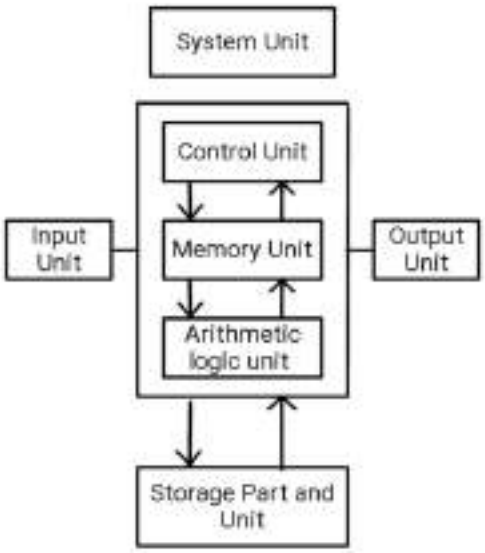


Figure 2: Basic Organization of Computer System

4.1.1 Difference between computer architecture and computer organization

Following are some of the important differences between Computer Architecture and Computer Organization:

Table 1: Difference between Computer Architecture and Organization

Sr. No.	Computer Architecture	Computer Organization
1	It is concerned with the way hardware components are connected and what the computer does.	It is concerned with structure and behavior of computer system and how it does it.
2	It helps us to understand the functional behavior of computer system.	It deals with the structural relationship of computer system.
3	All the high-level design issue are handled by the computer architecture.	All the low-level design issue are handled by the computer organization.
4	Computer Architecture is designed first.	Computer Organization is started after finalizing the computer architecture.
5	Architecture involves Logic (Instruction Sets, Addressing modes, Data types, Cache optimization.	Organization involves Physical components (Circuit design, Adders, Signals, and Peripherals).

4.2 COMPUTER REGISTERS

Computer Instructions are stored in the memory and memory is set of registers. Registers are used for temporary storage whereas memory is used for permanent storage. Registers are of various types like Data Register, Address Register, Accumulator, Instruction Register, Program Counter, Temporary Register, Input Register and output Register. For example the memory unit has size **4096X16,** means 4096 words and each word size is of 16 bit. 12 bits are used to specify 4096 words whereas 16 bits are used to specify data bits. So, the address bus for this example will be of 12 bit and data bus will be of 16 bit. Address bus is always unidirectional and Data Bus is always bi-directional. For the above example, various registers have been defined in the table below:

Data Register (DR) holds the operand of the memory and it is of 16 bit because data bus is of 16 bit. Similarly, Temporary Register (TR) is also of 16 bits as it is also used to operate on the data. Accumulator (AC) is used as processor register and it is used to perform all Arithmetic and logical operations and result will further be stored into AC. AC also operates on data and size of AC is of 16 bits. Instruction Register (IR) is used to store the instruction which is being fetched and executed and it is also of 16 bits due to data bus size. Address Register (AR) is used to hold the address, so its size depends on the address bus. Program Counter (PC) is a location counter used to fetch the address of next instruction from the memory, when the address is fetched, its value is being incremented by 1. The size of PC is of 12 bits. INPR an OUTR are sued to read and write the data from input-output devices. As ASCII character set is used to read and write, So, INPR and OUTR are of 8 bits.

Table 2: Description and Size of each Register

DR	16	Data Register	Holds memory operand
AR	12	Address Register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction Register	Holds instruction code
PC	12	Program Counter	Holds address of instruction
TR	16	Temporary Register	Holds temporary data
INPR	8	Input Register	Holds input character
OUTR	8	Output Register	Holds output character

4.3 COMMON BUS SYSTEM

The basic computer has a memory unit of size 4096x16 (example), 8 registers, and a control unit. Wires or buses must be offered to transfer information from one register to another and between memory and registers. The number of wires required will be more, if different connections are made between the output of each register and the inputs of another registers. So, the best way is to use more efficient scheme for transferring information using common bus system which is shown below in the Figure 3. The outputs of each registers and memory are connected via the common bus system. The specific output that has been selected for each bus lines at any given time is calculated from the binary value of three selection variables S2, S1, S0. For example, DR is 011 and decimal value is 3 and for selecting memory it is 111 or decimal equivalent 7. The 16-bit outputs of DR are placed on the common bus lines when S2 S1 S0 = 0 1 1. The particular register whose LD (load) input is enabled act as a destination register and can receive the data from the data bus during the next clock pulse transition. When the contents of AR (12 bits) or PC (12 bits) are applied to the 16-bit common bus, the four most significant bits are set to 0's. When AR or PC receives information from the data bus, only the 12 least significant bits are transferred into the register. The input register INPR and the output register OUTR have 8 bits each and communicate with the eight least significant bits in the bus.

INPR is connected to provide the information to the bus but OUTR can only take information from the common bus. As INPR receives character from an input device and further it is being transferred to AC whereas OUTR receives a character from AC and provides the output it to an output device [2]. There is no data transfer from OUTR to any of seven registers. The bus lines are connected to the inputs of six registers and the memory. Five registers have three control inputs: LD (load), INR (increment), and CLR (clear)

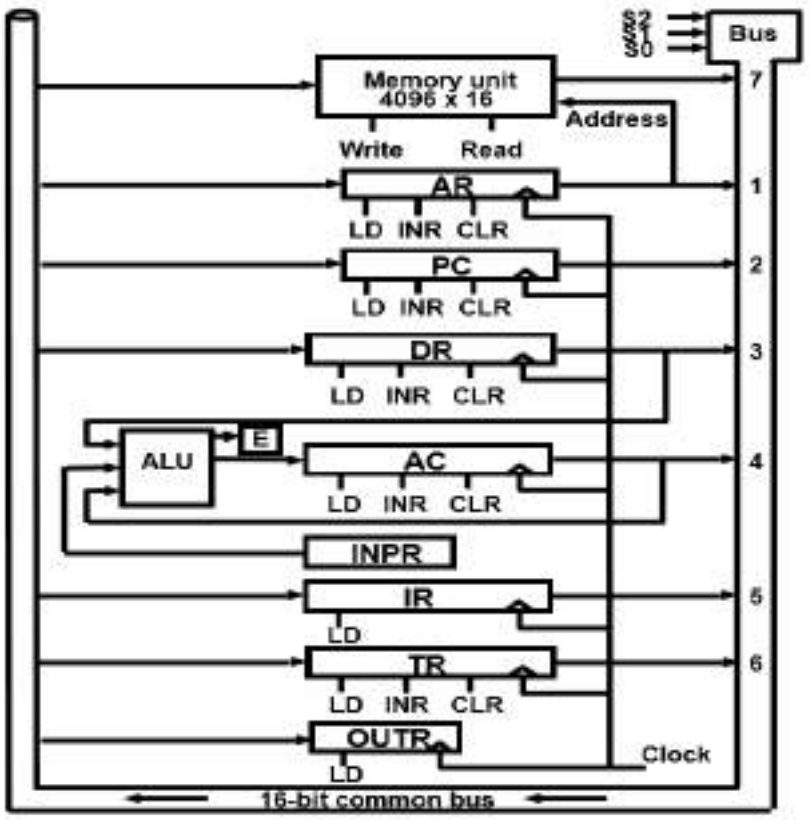


Figure 3: Common Bus System [1]

4.4 ARITHMETIC MICRO OPERATIONS

Arithmetic micro-operations dealt with arithmetic operations like addition, subtraction, increment, decrement, add with carry, subtract with borrow etc. For example as shown in the Table 3 below, there are eight arithmetic micro-operations, so three selection (S1, S0, Cin) lines are required. For example, at first input 0, 0, 0 addition microopration is performed, similarly, other micro-operations will be performed on other inputs.

Table 3: Arithmetic Micro-operations List

S1	S0	Cin	Y	Output	Microoperation
0	0	0	B	D = A + B	Add
0	0	1	B	D = A + B + 1	Add with carry
0	1	0	B'	D = A + B'	Subtract with borrow
0	1	1	B'	D = A + B'+ 1	Subtract
1	0	0	0	D = A	Transfer A
1	0	1	0	D = A + 1	Increment A
1	1	0	1	D = A - 1	Decrement A
1	1	1	1	D = A	Transfer A

Arithmetic circuit for 4 bit micro-operations is being shown below in the Figure 4. Firstly, 4 MUX and 4 Full Adders are required to perform 4 bit operations. For the first input, S1, S0, Cin (0, 0, 0), first input A is given as it is to Full Adder (FA) and second input B is multiplexed through 4X1 MUX. As B is required as an input for A+B micro-operations, so, B is multiplexed

at 0, 0. For Selection lines (0,0), but at Cin=1, A+B+1 (Add with carry) operation will be performed. For next selection line (0, 1), B' is required , so the output will be A+B'. If Cin is 1 with (0, 1) selection lines, A+B'+1 will be performed. If the selection line is (1, 0), then only A output is required, so 0 is multiplexed and if Cin=1, then A+1 will be the output. For selection line (1, 1) with Cin=0, the output decrement is required, so 1 is multiplexed on each MUX to do the operation like A+1111=A-1. But if the Cin=1 then again A-1+1=A will be performed.

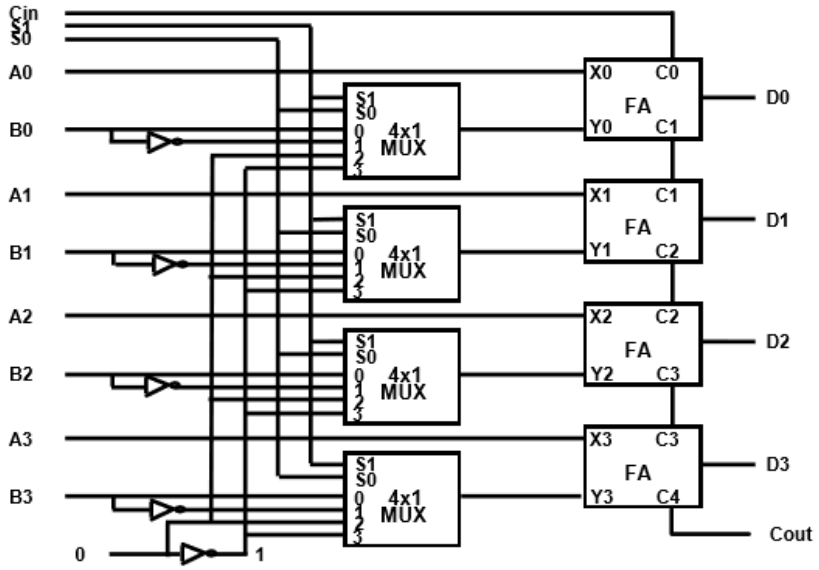


Figure 4: Design of Arithmetic Circuit [1]

4.5 LOGIC MICRO OPERATIONS

Logic micro-operations are used to perform binary operations on string of nits using logical gates, these operations consider each bit of the registers separately and treat them as binary variables. There are four basic logical operations as shown below in the Table. To design the circuit of 4 logical gates, two selection lines are required, like for S1, S0 (0,0), AND operation is performed, for (0,1), OR operation is performed, for (1,0), XOR opeartion is performed and at (1,1) Compliment operation is perfomred that revrese the input 1 to 0 and 0 to 1.

Table 4: List pf Logic Micro-operations

S_1	S_0	Output	μ-operation
0	0	$F = A \wedge B$	AND
0	1	$F = A \vee B$	OR
1	0	$F = A \oplus B$	XOR
1	1	F = A'	Complement

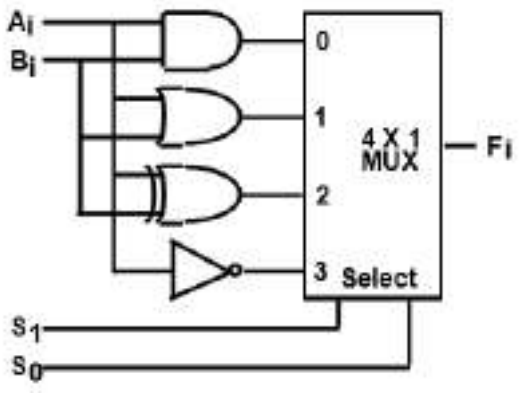


Figure 5: Design of Logic Unit

4.6 SHIFT MICRO OPERATIONS

Shift micro-operations are the micro-operations that are used to transfer serial information. These are also used along with arithmetic micro-operations, logical micro-operation, and other data-processing operations.

There are three types of shifts micro-operations:

- Logical:
- Circular
- Arithmetic

Logical Microperations transfers the 0 zero through the serial input when it is used for shl logical shift-left as shown in Figure 6 and shr for shift-right describes in Figure 7. For example, **10110011,** during shl operation, the output will be: **01100110,** as 0 is placed in the missing digit of MSB. For logical shift right, the output will be: **01011001.**

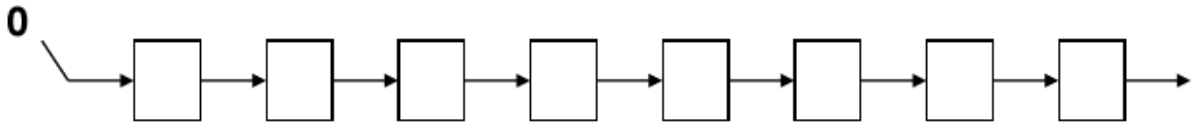


Figure 6: Logical- SHL operation

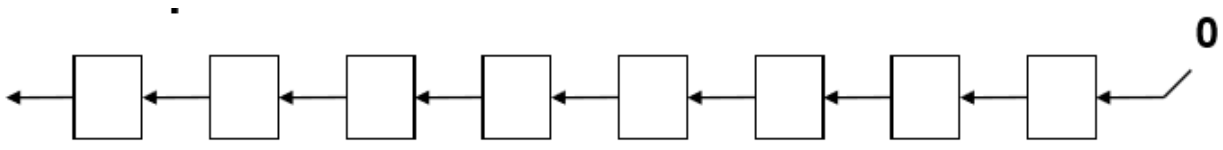


Figure 7: Logical- SHR operation

Circular Shift Left and Right Operations: During CIL (Circular shift left) shown in Figure 9 and CIR(Circular Shift Right) shown in Figure 8, MSB becomes LSB and LSB becomes MSB. For the above example, the output for CIL is: 01100111, for CIR: 11011001

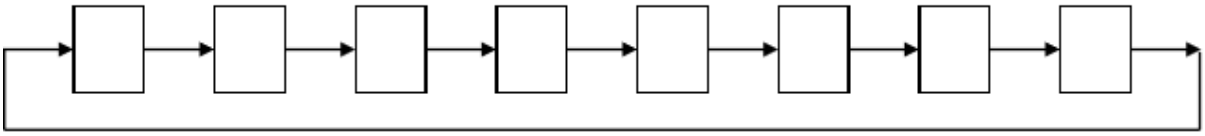


Figure 8: CIR operation

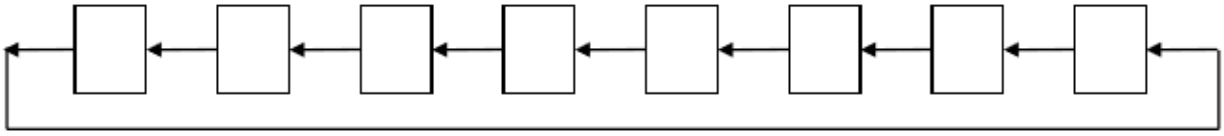


Figure 9: CIL operation

Arithmetic Shift Left and Right Micro-operations: ASR (Arithmetic Shift Right) is shown in Figure 10 that sign bit remains the same and moved to the next bit also. Arithmetic micro-operations are used for signed numbers. For example: 10110011, the output will be for ASR: 11011001, similarly, for ASL shown in Figure 11, the output is same as LSL (logical shift left), only the change of sign bit during ASL operation.

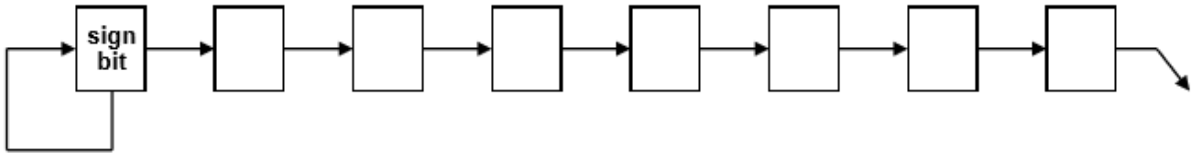


Figure 10: ASR operation

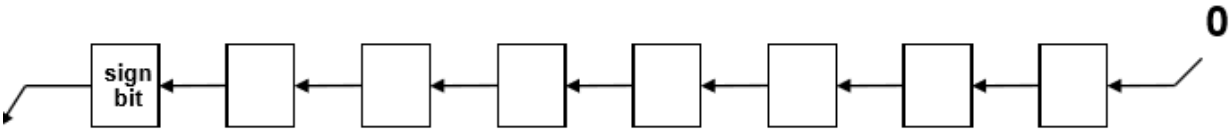


Figure 11: ASL operation

The sign bit is checked by using EX-OR operation between last two bits, if last two bits are same the sign will be positive otherwise, sign will be negative. If the sign is different there will be overflow and V=1, if the sign is same, then no overflow and V=0 as shown in the Figure 12 below.

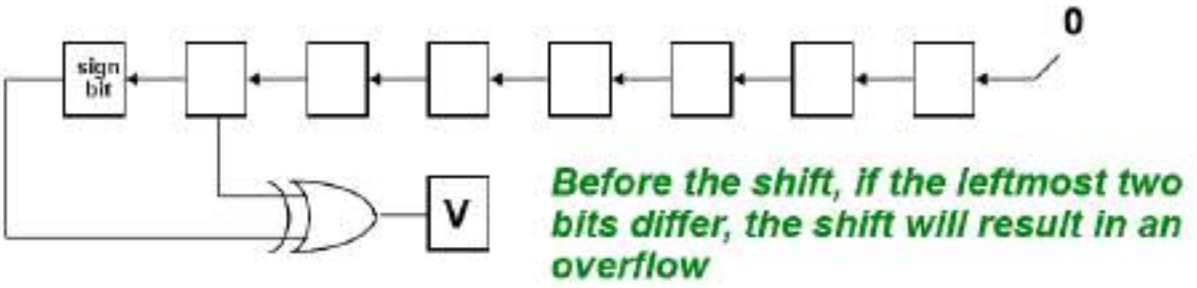


Figure 12: Overflow Flag during ASL operation

4.7 DESIGN OF ALU

ALU is used to perform all types of micro-operations like arithmetic, logical and shift. For designing 4-bit ALU, the operations shown in the Table 5 are being implemented by designing ALU. 8 micro-operations are same as arithmetic circuit and 4 are logical micro-operations and 2 are shift left and shift right.

Table 5: ALU micro-operations

S3	S2	S1	S0	Cin	Operation	Function
0	0	0	0	0	F = A+B	Addition
0	0	0	0	1	F = A +B+1	Add with carry
0	0	0	1	0	F = A + B'	Subtract with borrow
0	0	0	1	1	F = A + B + 1	Subtraction
0	0	1	0	0	F = A	Transfer A
0	0	1	0	1	F = A + 1	Increment A
0	0	1	1	0	F = A - 1	Decrement A
0	0	1	1	1	F = A	TransferA
0	1	0	0	X	F = A $\wedge$ B	AND
0	1	0	1	X	F = A $\vee$ B	OR
0	1	1	0	X	F = A $\oplus$ B	XOR
0	1	1	1	X	F = A'	Complement A
1	0	X	X	X	F = shr A	Shift right A into F
1	1	X	X	X	F = shl A	Shift left A into F

For performing all the operations shown in the table [3], the design of ALU is presented in the Figure 13 below. 4X1 MUX is required to select any one from four micro-operations like arithmetic, logical, shift right and shift left. If S2, S3 (0, 0) then Arithmetic Circuit will be selected, if it is (0,1), Logic Circuit will be implemented, for (1,0), shift right operation, for (0,1), shift left operation will be implemented. Arithmetic Circuit have one Carry input which is passed to the other Full Adders already explained **in the Figure 4: Logic Circuit also implemented same as shown above in the Figure 5.**

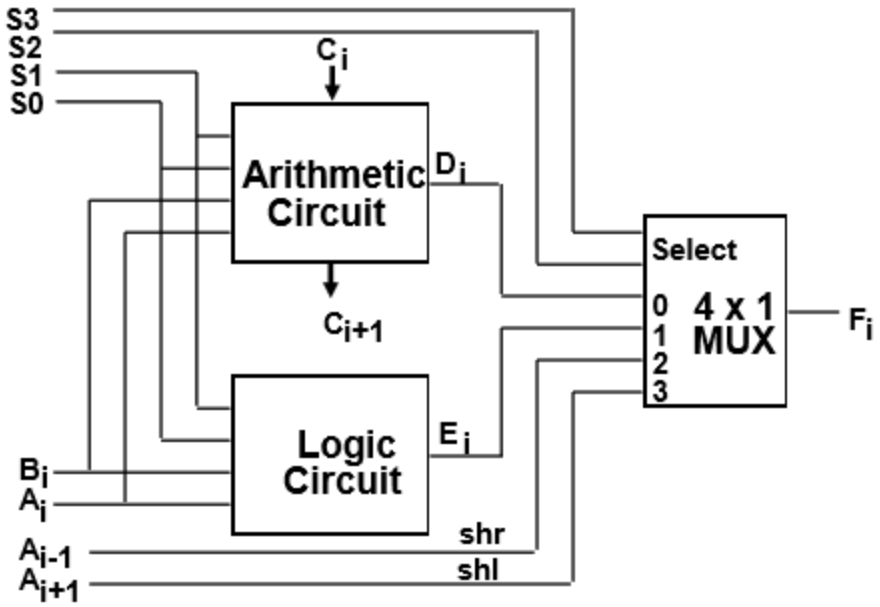


Figure 13: Design of ALU

4.8 SUMMARY

- Basics of Computer Architecture have been discussed, Difference between Computer Structure and Architecture is also discussed.
- Registers are used for temporary storage and Memory is used for permanent storage, Size of registers depends on the Memory Size.
- Types of Registers like PC, AR, DR, TR, IR, INPR and OUTR have also been described
- Common Bus system is used to transfer the data between various registers and Memory and registers.
- Arithmetic micro-operations like addition, subtraction, increment, decrement, add with carry, subtract with borrow has also been discussed.
- Logic micro-operations are like AND, OR, NOT, EX-OR has been discussed and designed through 4X1 Mux.
- Shift micro-operations are of three types like logical, arithmetic and circular, it has also been discussed using numerical example.
- ALU is designed by using arithmetic, logical and shift micro-operations.

Some Objective Type Questions

Q1: The memory size is 4K*16, how many bits there will be in Program Counter, Accumulator.

PC___

AC___

a) 12,15
b) 12, 16
c) 16,12
d) None of these

Q2: Binary addition of (3) and (-7), produces Carry flag= _____, Sign Flag_____

Q3: R= 10001010, determine the value of R after performing arithmetic shift right operation._____

Q4. Write the output of 8 bit no. 1001001 after performing CIL and CIR.

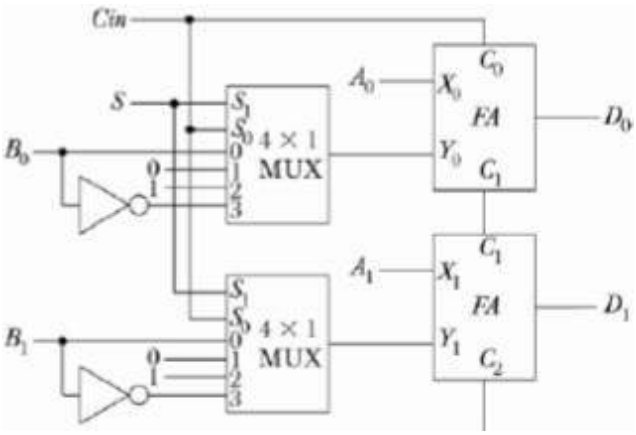
Q5. A Computer has a memory of size 2Kx32. The instruction has four parts indirect bit, opcode, register part specify 32 registers and address part. How many bits there are in address field_____, opcode_____register part_____,mode field_____

a. 12, 11, 5, 32
b. 11, 15, 5, 1
c. 32, 5, 16, 11
d. None of these

Questions for Practice

Q1. Design 4 bit Decrementer using 4 Full Adders.

Q2. Design the truth table for the following diagram and Write the value of S and Cin (in binary only) for the following Arithmetic Circuit:



Q3. Design the logic circuit by taking 16 micro-operations

Q4. Design 4 bit Shifter circuit for performing Logical Shift lLeft and Right operations.

Q5: Design an adder/subtractor circuit with carry input C_{in} and two inputs A and B. When C_{in}=0, the circuit performs A + B. When C_{in}=1, the circuit performs A – B, Verify the circuit using A=5 and B=2.

References:

1. M. Morris Mano, Computer system architecture (3. Ed.). Pearson Education 1993, ISBN 978-0-13-175563-5, pp. I-XIII, 1-524.

2. M. Morris Mano: Pracniques: simulation of Boolean functions in a decimal computer. Commun. ACM 8(1): 39-40 (1965)

3. Https: file:///C:/Users/Dr.%20Mukesh%20Kumar/Downloads/4_bit_ALU.pdf

Bachelor of Computer Applications (BCA)

COMPUTER SYSTEM ARCHITECTURE

UNIT V: TIME AND CONTROL UNIT

STRUCTURE

5.0 Objective

5.1 Introduction

5.2 Instruction cycle

5.3 Memory reference instructions

5.4 Register reference instructions

5.5 Input-output instructions

5.6 Design of Timing and Control Unit

5.7 Summary

5.0 OBJECTIVE

- To understand the Instruction cycle
- To Know Memory reference instructions, Register reference instructions and Input-output instructions,
- To understand the Design of Timing and Control Unit

5.1 INTRODUCTION

Computer instructions are a set of machine language instructions that a particular processor understands and executes. A computer performs tasks on the basis of the instruction provided.

Timing

The events on a bus are coordinated with the help of timing signals provided by a **clock**. These timing signals are managed by the control unit.

.**Clock**

The 'clock' provides timing signals to control components and operations of the CPU. The clock defines regular time intervals, called cycles. To perform any basic operation, the CPU divides the action into sequence of basic steps. Each basic step is completed in one clock cycle. The classification of buses can be done based on the clock, which is of two categories: Synchronous bus and Asynchronous bus.

Figure 5.1 shows the timing diagrams for synchronous read operation. The 'clock' is the system clock. This clock is used to synchronize the operations of the system. The CPU places the address on the address bus at the beginning of the clock cycle. During this time, the address is decoded and data is accessed from the memory. The memory places the data on the data bus. The Read operation is activated during this time and the processor reads the data from the data bus, and stores it in one of the registers.

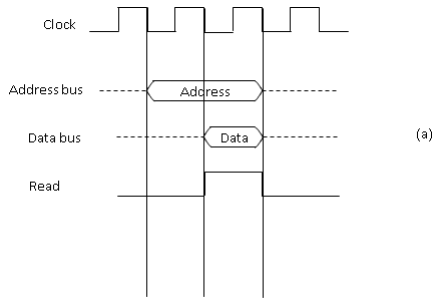


Figure 5.1 timing diagrams for synchronous read operation

An instruction comprises of groups called fields. These fields include:

- The Operation code (Opcode) field which specifies the operation to be performed.
- The Address field which contains the location of the operand, i.e., register or memory location.
- The Mode field which specifies how the operand will be located.

The Timing and Control Unit unit of the microprocessor issues necessary timing and control signals for the execution of instructions. It generates three types of signals namely status, control and timing signals required for the operation of memory and I/O devices. It has three control signals. It controls all external and internal circuits. It operates with reference to clock signal. It synchronizes all the data transfers.

Timing signal in computer is generated by electrical pulses that are generated in the processor or in external devices in order to synchronize computer operations. The main timing signal comes from the computer's clock, which provides a frequency that can be divided into many slower cycles. Other timing signals may come from a timesharing or real-time clock.

The control unit of the central processing unit regulates and integrates the operations of the computer. It selects and retrieves instructions from the main memory in proper sequence and interprets them so as to activate the other functional elements of the system at the appropriate moment…

5.2 INSTRUCTION CYCLE

A program residing in the memory unit of the computer consists of a sequence of instructions. The program is executed in the computer by going through a cycle for each instruction. Each instruction cycle in turn is subdivided into a sequence of subcycles or phases The instruction cycle (also known as the fetch–decode–execute cycle, or simply the fetch-execute cycle) is the cycle that the central processing unit (CPU) follows from boot-up until the computer has shut down in order to process instructions.

The processing involved in the execution of a single instruction is termed as Instruction Cycle.

The organization of the CPU is based on the functions it performs which are to fetch an instruction, decode it and execute it. They are explained as follows:

1. Fetch Instruction: CPU reads instructions from memory

2. Change Program Counter (PC)

3. Interpret instruction: Determine the type of instruction fetched.

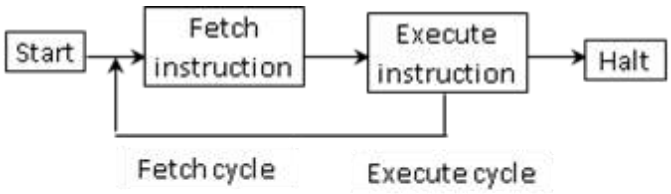
4. Decode Instruction: Instruction is decoded to determine what action is required.

5. Fetch data: If the instruction uses data, fetch it from memory or I/O devices

6. Execute instruction: The execution of the instruction may require arithmetic or logical operations on data

7. Write data: The results are stored in the memory location.

To do these operations, CPU needs to store some data temporarily in the CPU registers



Basic Instruction cycle

In the beginning of each instruction cycle, the processor fetches the instruction from the memory. The processor contains a special register called a Program Counter (PC), which holds the address of next instruction to be executed. The address of this instruction is in the Program Counter (PC) which holds the address of the next instruction to be fetched. Now, initially the address of the very first instruction ('i' here) must be entered into PC. After the instruction is fetched, the PC is incremented. The fetched instruction is loaded in the instruction register (IR). This instruction is interpreted by the decoder and the required action is performed. Another special register is Accumulator (AC), which is directly connected to the ALU and holds one of the operands for the operation to be performed

Now, instruction is fetched from the address, decoded and finally executed by the processor. Similarly, other instructions are also executed one by one.

To illustrate how an instruction is executed, consider the addition of data stored in 16-bit memory locations with addresses 100 and 101. The instruction is 16 bits: 4 bits for the opcode, 12 bits for the operand. The program adds the contents of the memory word at address 830 to the contents of the memory word stored at address 831. The results are stored at address 831. The implementation requires three fetch and three execute cycles, as described below:

1. PC contains address 100, which is the address of the first instruction. The instruction is loaded in the IR.
2. The first 4 bits of the IR are for the opcode which indicates that AC is to be loaded. The remaining 12 bits hold the address, 830 in this case.
3. The PC is incremented.
4. The contents of AC (location 830) and the contents of 831 are added. The result is stored in AC.
5. PC is incremented and the next instruction is fetched.
6. The contents of AC are stored in location 831.

As can be seen from the above mentioned 6 steps for the addition of two numbers, three instruction cycles are used. Each of this instruction cycle consists of a ‘fetch’ and ‘execute’ cycle. The instructions used are quite simple. More complex instructions may lead to lesser number of instructions used for the same operation. For example, the execution cycle may involve more than one reference to the memory. Or in some case, the reference might be to the I/O instead to the memory. Some exchanges of data/operand may be between processor or memory or I/O devices, or just internal processor operations like between registers and ALU. These exchanges depend on the read and write operations.

5.3 MEMORY REFERENCE INSTRUCTIONS

The external memories store the programs and data required by the processors.

Memory Reference refer to memory address as an operand. The other operand is always accumulator. It specifies 12-bit address, 3-bit opcode (other than 111) and 1-bit addressing mode for direct and indirect addressing

In order to specify the microoperations needed for the execution of each instruction, it is necessary that the function that they are intended to perform be defined precisely. Some instructions have an ambiguous description. This is because the explanation of an instruction explanation.

Following Table lists the seven memory-reference instructions

TABLE Memory-Reference Instructions

Symbol	Operation decoder	Symbolic description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR], \quad E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC, \quad PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1,$ If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$

AND to AC

This is an instruction that perform the AND logic operation on pairs of bits in AC and the memory word specified by the effective address. The result of the operation is transferred to AC .

The microoperations that execute this instruction are:

AC <- AC /\ DR, SC <--- 0

ADD to AC

This instruction adds the content of the memory word specified by the effective address to the value of AC. The sum is transferred into AC and the output carry C_{out} is transferred to the E (extended accumulator) flip-flop. The microoperations needed to execute this instruction are

DR← M[AR]

AC← AC + DR, E← C_{out} , SC ← 0

LDA: Load to AC

This instruction transfers the memory word specified by the effective address to AC. The microoperations needed to execute this instruction are

DR←M [AR]
AC←DR , ← 0

STA: Store AC

This instruction stores the content of AC into the memory word specified by the effective address. Since the output of AC is applied to the bus and the data input of memory is connected to the bus, we can execute this instruction with one microoperation:

M [AR] ← AC, SC ← 0

BUN: Branch Unconditionally

This instruction transfers the program to the instruction specified by the effective address. that PC holds the address of the instruction to be read from memory in the next instruction cycle. PC is incremented at time T1 to prepare it for the address of the next instruction in the program sequence. The BUN instruction allows the programmer to specify an instruction out of sequence and we say that the program branches (or jumps) unconditionally. The instruction is executed with one microoperation:

PC ← AR, SC ← 0

The effective address from AR is transferred through the common bus to PC .

BSA: Branch and Save Return Address

This instruction is useful for branching to a portion of the program called a subroutine or procedure. When executed, the BSA instruction stores the address of the next instruction in sequence (which is available in PC) into a memory location specified by the effective address. The effective address plus one is then transferred to PC to serve as the address of the first instruction in the subroutine. This operation was specified in the following register transfer:

M[AR] <-- PC, PC <-- AR + I

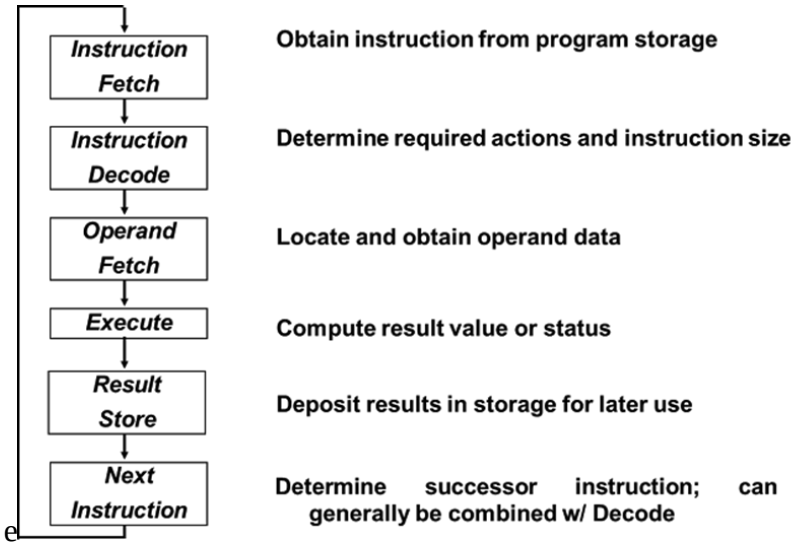
ISZ: Increment and Skip if Zero

This instruction increments the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1. The programmer usually stores a negative number (in 2's complement) in the memory word. As this negative number is repeatedly incremented by one, it eventually reaches the value of zero. At that time PC is incremented by one in order to skip the next instruction in the program.

Since it is not possible to increment a word inside the memory, it is necessary to read the word into DR, increment DR, and store the word back into memory. This is done with the following sequence of microoperations:

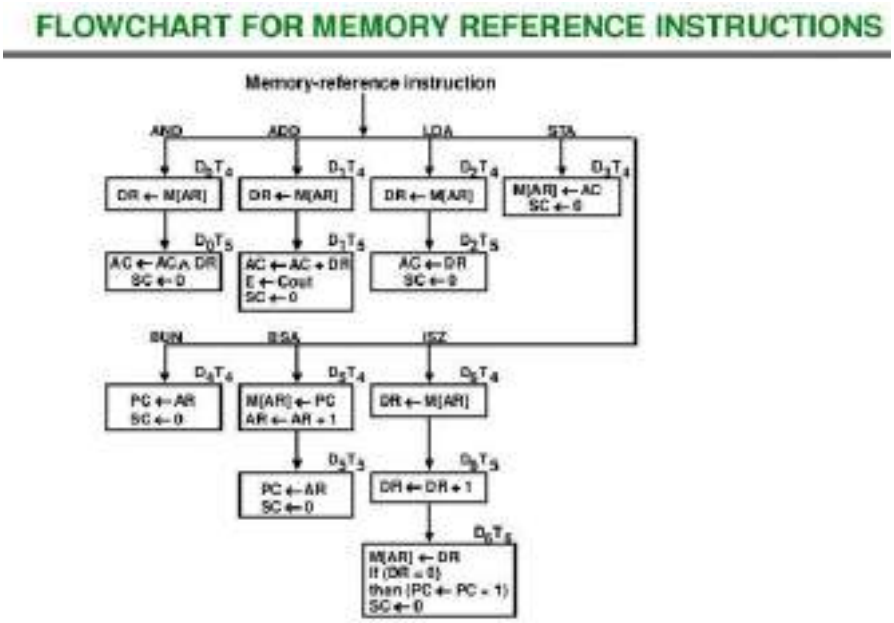
DR <-- M [AR]
DR <-- DR + 1

M [AR] <-- DR,
if (DR = 0) then (PC ← PC + 1), SC ← 0



Control Flowchart

A flowchart showing all microoperations for the execution of the seven memory-reference instructions is shown in Fig. 5-11. The control functions are indicated on top of each box.



5.4 REGISTER REFERENCE INSTRUCTIONS

Register Reference instructions perform operations on registers rather than memory addresses. The IR(14 – 12) is 111 (differentiates it from memory reference) and IR(15) is 0 (differentiates it from input/output instructions). The rest 12 bits specify register operation

TABLE Execution of Register-Reference Instructions

$D_7I'T_3 = r$ (common to all register-reference instructions)
$IR(i) = B_i$ [bit in $IR(0–11)$ that specifies the operation]

	r:	$SC \leftarrow 0$	Clear SC
CLA	rB_{11}:	$AC \leftarrow 0$	Clear AC
CLE	rB_{10}:	$E \leftarrow 0$	Clear E
CMA	rB_9:	$AC \leftarrow \overline{AC}$	Complement AC
CME	rB_8:	$E \leftarrow \overline{E}$	Complement E
CIR	rB_7:	$AC \leftarrow$ shr AC, $AC(15) \leftarrow E$, $E \leftarrow AC(0)$	Circulate right
CIL	rB_6:	$AC \leftarrow$ shl AC, $AC(0) \leftarrow E$, $E \leftarrow AC(15)$	Circulate left
INC	rB_5:	$AC \leftarrow AC + 1$	Increment AC
SPA	rB_4:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$	Skip if positive
SNA	rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	Skip if negative
SZA	rB_2:	If $(AC = 0)$ then $PC \leftarrow PC + 1)$	Skip if AC zero
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$	Skip if E zero
HLT	rB_0:	$S \leftarrow 0$ (S is a start–stop flip-flop)	Halt computer

The first seven register-reference instructions perform clear, complement, circular shift, and increment microoperations on the AC or E registers. The next four instructions cause a skip of the next instruction in sequence when a stated condition is satisfied. The skipping of the instruction is achieved by incrementing PC once again (in addition, it is being incremented during the fetch phase at time T1). The condition control statements must be recognized as part of the control conditions. The AC is positive when the sign bit in AC(IS) = 0; it is negative when AC(IS) = I. The content of AC is zero (AC = 0) if all the flip-flops of the register are zero. The HLT instruction clears a start-stop flip-flop S and stops the sequence counter from counting. To restore the operation of the computer, the start-stop flip-flop must be set manually.

5.5 INPUT-OUTPUT INSTRUCTION

Input/Output Instructions are for communication between computer and outside environment. The IR(14 – 12) is 111 (differentiates it from memory reference) and IR(15) is 1 (differentiates it from register reference instructions). The rest 12 bits specify I/O

An instruction which is used to print something on the screen or any other output device is known as output instruction. Similarly, an instruction which is used to input information from the user is known as input instruction

The I/O devices are given specific addresses. The processor similarly views the I/O operations as memory operations. It concerns commands that include the address for the device.

The I/O instructions are needed for the following objectives −

- It is used to analyzing flag bits.
- It can transfer data to or from the AC register.
- It is used for controlling interrupts.

The I/O instructions carry the opcode as 1111. They are recognized through the control when $D_7 = 1$ and I=1. The operation to be executed is determined by the different remaining bits.

There are various I/O Instructions which are shown in the table −

There are various I/O Instructions which are shown in the table

Symbol	Description
INP	The INP instruction address the information from the INPR to AC which has 8 low order bits. It also clears the input flag to 0
OUT	It can send the 8 low order bits from AC into output register OUTPR. It also clears the output flag to 0.
SKI	These are the status flags. They skip the next instructions when flag = 1, They are primarily branching instructions.
SKO	It is similar to SKI.
ION	Enables (set) interrupt.
IOF	Disables (clear) interrupt.

5.6 DESIGN OF TIMING AND CONTROL UNIT

The timing for all registers in the basic computer is controlled by a master clock generator. The control signals are generated in the control unit and provide control inputs for the multiplexers in the common bus, control inputs in processor registers, and microoperations for the accumulator

The clock pulses are applied to all flip-flops and registers in the system, including the flip-flops and registers in the control unit. The clock pulses do not change the state of a register unless the register is enabled by a control signal. The control signals are generated in the control unit and provide control inputs for the multiplexers in the common bus, control inputs in processor registers, and microoperations for the accumulator.

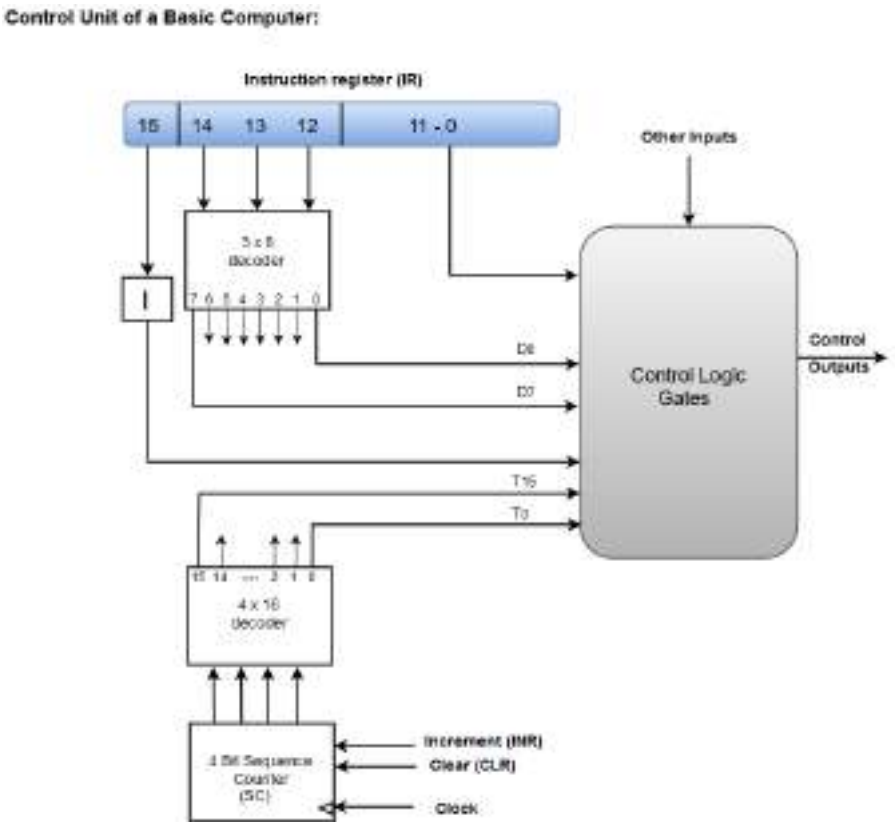
There are two major types of control organization:

1. Hardwired control
2. Microprogrammed control
3. **Hardwired controlzzz**

The Hardwired Control organization involves the control logic to be implemented with gates, flip-flops, decoders, and other digital circuits.

- A Hard-wired Control consists of two decoders, a sequence counter, and a number of logic gates.
- An instruction fetched from the memory unit is placed in the instruction register (IR).
- The component of an instruction register includes; I bit, the operation code, and bits 0 through 11.
- The operation code in bits 12 through 14 are coded with a 3 x 8 decoder.
- The outputs of the decoder are designated by the symbols D0 through D7.
- The operation code at bit 15 is transferred to a flip-flop designated by the symbol I.
- The operation codes from Bits 0 through 11 are applied to the control logic gates.
- The Sequence counter (SC) can count in binary from 0 through 15.

The following figure shows the block diagram of a Hardwired Control organization.

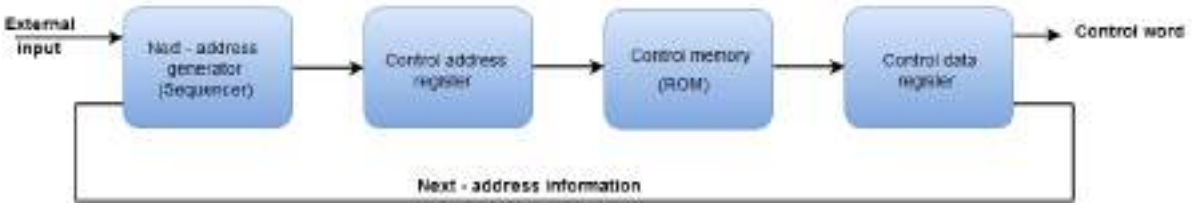


Micro-programmed Control

The Microprogrammed Control organization is implemented by using the programming approach. In Microprogrammed Control, the micro-operations are performed by executing a program consisting of micro-instructions.

The following image shows the block diagram of a Microprogrammed Control organization.

Microprogrammed Control Unit of a Basic Computer:



- The Control memory address register specifies the address of the micro-instruction.
- The Control memory is assumed to be a ROM, within which all control information is permanently stored.
- The control register holds the microinstruction fetched from the memory.
- The micro-instruction contains a control word that specifies one or more micro-operations for the data processor.
- While the micro-operations are being executed, the next address is computed in the next address generator circuit and then transferred into the control address register to read the next microinstruction.
- The next address generator is often referred to as a micro-program sequencer, as it determines the address sequence that is read from control memory.

5.7 SUMMARY

Timing signal in computer architecture are Electrical pulses generated in the processor or in external devices in order to synchronize computer operations. The main timing signal comes from the computer's clock, which provides a frequency that can be divided into many slower cycles. Other timing signals may come from a timesharing or real-time clock.

The function of timing and control unit is to provide timing and control signal to the microprocessor to perform the various operation. It has three control signals. It controls all external and internal circuits. It operates with reference to clock signal. It synchronizes all the data transfers.

This unit of the microprocessor issues necessary timing and control signals for the execution of instructions. It generates three types of signals namely status, control and timing signals required for the operation of memory and I/O devices.

A control unit works by receiving input information that it converts into control signals, which are then sent to the central processor. The computer's processor then tells the attached hardware what operations to carry out.

CPU time (or process time) is the amount of time for which a central processing unit (CPU) was used for processing instructions of a computer program or operating system, as opposed

to elapsed time, which includes for example, waiting for input/output (I/O) operations or entering low-power (idle) mode.

The difference between main memory and control memory is that the main objective of the Control unit is used in the central processing unit to controlling the operation whereas the Memory unit in the central processing unit is used for storing the information in the computer system. The control unit does not holds the final result whereas memory unit do this task.

Questions

1. What is the function of timing and control unit?
2. What are timing and control signals?
3. What is timing in CPU?
4. What is the purpose of control unit in a computer?
5. What is the difference between main memory and control memory?

Bachelor of Computer Applications (BCA)

COMPUTER SYSTEM ARCHITECTURE

UNIT VI: DESIGN OF CENTRAL PROCESSING UNIT

STRUCTURE

6.0 Objectives

6.1 Register Organization

6.2 Stack Organization

6.3 Instruction Organization

6.3.1 Three address instructions

6.3.2 Two address instructions

6.3.3 One address instructions

6.4 Instruction formats

6.5 Addressing Modes

6.6 Summary

6.0 OBJECTIVES

- Understanding Register organization
- Understanding stack organization,
- Understanding Register instructions and addressing modes.

6.1 REGISTER ORGANIZATION

Registers are the smaller and the fastes*t* accessible memory units in the central processing unit (CPU). According to memory hierarchy, the registers in the processor, function a level above the main memory and cache memory. The registers used by the central unit are also called as processor registers.

A register can hold the instruction, address location, or operands. Sometimes, the instruction has register as a part of itself.

Register organization is the arrangement of the registers in the processor. The processor designers decide the organization of the registers in a processor. Different processors may have different register organization. Depending on the roles played by the registers they can be categorized into two types, user-visible register and control and status register. In a register organized computer, the CPU has the logic needed to execute its particular instruction set and is divided into data paths and control unit. Instructions are read, operations are performed, and the results are stored in general-purpose registers (GPR). The general-purpose register file is a set of registers of that particular CPU. Each register has a name, or an ID, or number (used synonymously). The register file allows an instruction to access the registers in any order by specifying the register number of the register to be accessed. This is similar to specifying the address of the memory location in the memory system in any order. Reading the contents of a general-purpose register does not change their contents.

Within the CPU, there are a set of registers that serve two functions:

User-visible Registers: These registers store the data from the main memory that will be used during execution. Thus, they minimize the main memory references.

Control and Status Registers: These are used by the control unit to control the operation of the CPU, and by operating system programs to control the execution of the program.

User-Visible Registers

User-visible registers are referenced by means of machine language that the CPU executes. They are categorized as under:

- General Purpose registers
- Data register
- Address register
- Flag registers

- User Visible Registers

    - General Purpose Register
    - Data Register
    - Address Register
    - Condition Codes
- Control and Status Registers
    - Program Counter
    - Instruction Register
    - Memory Address Register
    - Memory Buffer Register

General Purpose registers

The general-purpose register file (GPR) allows an instruction to access the registers in any order. Each register has a name (or number), and the data stored in that register.

These registers can be assigned a variety of functions. Any GPR can contain the operand for any opcode Some GPRs are used for addressing functions. Some of these registers are dedicated registers for floating point and stack operation. Some special registers are Index register, Stack Pointer, and Segment Pointer.

Data Registers

Data registers may be used only to hold data and cannot be used for calculations.

Address Registers

Address registers hold the address of the instruction. They may act as general-purpose register when they are used for a particular addressing mode.

Flag registers

Flag registers are a special category of registers which are partially visible to the users.

They hold the **condition codes,** which are also referred to as flag registers. The Condition codes are the bits set by the CPU hardware as a result of operations. For example, an arithmetic operation may produce a positive, negative, zero, or an overflow result. The result of the operation is stored in the flag register, and the condition of the result, which can be of the form of a code, is also set

Control and Status registers

The control and status registers are employed to control the operation of the CPU. Most of the registers are not visible to the user. These registers may have names that are specific to the organization of the CPU. The following four registers are essential for execution of an instruction:

Program Counter (PC): Contains the address of the next instruction to be fetched.

Instruction register (IR): Contains the most recently fetched instruction.

Address register (AR): Contains address of a location in memory.

Data register (DR): Contains a word of data that has been most recently read from memory, or written to memory.

These registers are used for the movement of data between the CPU and main memory. Some commonly used registers with their commonly used attributes are described in Figure 6.1

Register Symbol	**Register (Size Bits)**	**Register Name**	**Function**
PC	12	Program Counter	Holds address of next instruction
TR	16	Temporary Register	Holds Temporary data
IR	16	Instruction Register	Holds Instruction Code
AC	16	Accumulator	Processor Register
AR	16	Memory Address Register	Holds address for memory
DR	12	Data Register	Holds Memory Operand
INPR	8	I/P Register	Holds I/P Character
OUTR	8	O/P Register	Holds O/P character

As pointed out in Figure, the data register (DR) holds the operand read from the memory. The accumulator register (AC) is a general purpose processing register. The instruction read from memory is placed in the instruction register (IR). The temporary register (TR) is used for holding temporary data during processing. The memory address register (AR) has 12 bits of a memory address. The program counter (PC) holds the address of the next instruction to be used from the memory after current instruction is executed. The input register (INPR) receives an 8-bit character from an input device. The output register (OUTR) holds an 8-bit character for a output device.

The information of the status of the results in the CPU is provided by a register known as Program Status Word (PSW). PSW contains condition codes and other status information. Common fields or flags include the following:

Sign: Holds sign bit of the result of the last arithmetic operation.

Zero: It is set when the result is 0.

Carry: Set if addition result has a carry, or a subtraction needs a borrow.

Equal: If logical compare result is equality.

Overflow: Used to indicate arithmetic overflow.

Interrupt enable/disable: Used to enable or disable interrupts.

6.2 STACK ORGANIZATION

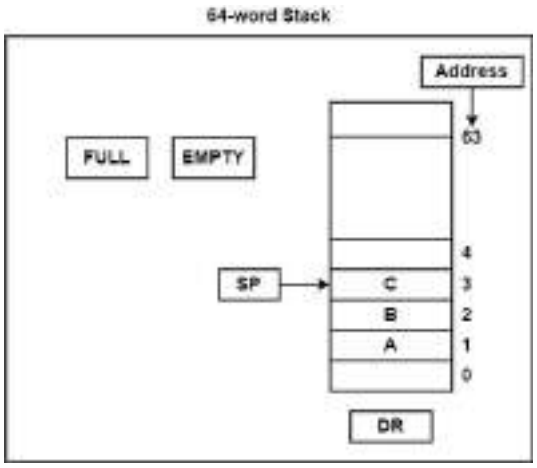
A stack is a memory unit with an address register. This register stores the address of top of stack. This is known as Stack Pointer (SP). The stack pointer continually holds the address of the element that is located at the top of the stack. It can insert an element into or delete an element from the stack. The insertion operation is known as push operation and the deletion operation is known as pop operation. In a computer stack, these operations are simulated by incrementing or decrementing the SP register

Note: Stack based CPU organisation uses zero address instruction.

Register Stack

The stack can be arranged as a set of memory words or registers. Consider a 64-word register stack arranged as displayed in the figure below. The stack pointer register includes a binary number, which is the address of the element present at the top of the stack. The three-element A, B, and C are located in the stack.

The element C is at the top of the stack and the stack pointer holds the address of C that is 3. The top element is popped from the stack through reading memory word at address 3 and decrementing the stack pointer by 1. Then, B is at the top of the stack and the SP holds the address of B that is 2. It can insert a new word, the stack is pushed by incrementing the stack pointer by 1 and inserting a word in that incremented location.



The stack pointer includes 6 bits, because $2^6 = 64$, and the SP cannot exceed 63 (111111 in binary). This is because if 63 is incremented by 1, the result is 0(111111 + 1 = 1000000). SP holds only the six least significant bits. If 000000 is decremented by 1 thus the result is 111111.

Therefore, when the stack is full, the one-bit register ‘FULL’ is set to 1. If the stack is null, then the one-bit register ‘EMTY’ is set to 1. The data register DR holds the binary information which is composed into or readout of the stack.

The push operation is executed as follows −

SP←SP + 1	It can increment stack pointer
K[SP] ← DR	It can write element on top of the stack
If (SP = 0) then (FULL ← 1)	Check if stack is full
EMTY ← 0	Mark the stack not empty

The main two operations that are performed on the operators of the stack are **Push** and **Pop**. These two operations are performed from one end only.

Push –
This operation results in inserting one operand at the top of the stack and it decrease the stack pointer register.

Pop –
This operation results in deleting one operand from the top of the stack and it increase the stack pointer register.

PDP-11, Intel’s 8085 and HP 3000 are some of the examples of the stack organized computers.

6.3 INSTRUCTION ORGANIZATION

A computer performs a task based on the instruction provided. Instruction in computers comprises groups called fields. These fields contain different information as for computers everything is in 0 and 1 so each field has different significance based on which a CPU decides what to perform. The most common fields are:

- Operation field specifies the operation to be performed like addition.
- Address field which contains the location of the operand, i.e., register or memory location.
- Mode field which specifies how operand is to be founded.

Copying data from one place to another is the most fundamental of all operations. Here, copying means creating an identical bit pattern as the original. There are two reasons that data may be copied from one location to another. The first is the assignment of values to variables, such as A = B. The value in memory location B is copied to memory location A. The second purpose is to make the data available for its efficient use. For this reason, variables are copied from memory locations to registers.

The operation of adding two numbers is a fundamental capability in any computer. The statement C = A + B in a high-level language program is a command to the computer to add the current values of the two variables called A and B, and to assign the sum to a third variable C. When the program containing this statement is compiled, the three variables A, B, and C are assigned to distinct locations in the memory. The contents of these locations represent the values of the three variables. Hence, the above high-level language statement requires the following action to take place in the computer.

[C]← [A] + [B]

To carry out this action, the contents of memory locations A and B are fetched from the memory and transferred into the processor where their sum is computed. This result is then sent back to the memory and stored in location C. This operation can be accomplished in various ways.

6.3.1 Three address instruction

First assume that this action is to be accomplished by a single machine instruction of three operands — A, B and C. This **three-address** instruction can be represented symbolically as:

Add A, B, C

Operands A and B are called the **source** operands, C is called the **destination** operand, and Add is the operation to be performed on the operands. A general three-address instruction has the format:

Operation Source 1, Source 2, Destination

A 3-address instruction is too large to fit in one word for a reasonable word length. Thus, a format that allows multiple words to be used for a single instruction would be needed to represent an instruction of this type.

6.3.2 Two-address instruction

A two-address instruction has only two operands. A **two-address** instruction has the form

Operation Source, Destination

An Add instruction of this type is:

```
Add   A, B
```

The above statement performs the operation [B] ← [A] + [B]. When the sum is calculated, the result is sent to the memory and stored in location B, replacing the original contents of this location. This means that operand B is both a source and a destination.

We had defined three- and two-address instructions. However, even two-address instructions will not normally fit into one word for usual word lengths and address sizes.

6.3.2 One-address instruction

One-address instructions specify only one memory operand. When a second operand is needed, as in the case of an Add instruction, it is understood implicitly to be a unique location. A processor register, usually called the accumulator, may be used for this purpose. Thus, the **one-address** instruction is:

```
Add   A
```

Which means the following:
Add the contents of memory location A to the contents of the accumulator register and place the sum back into the accumulator. Other more commonly used one-address instructions are:

```
Load   A
```

and

```
Store   A
```

The Load instruction copies the contents of memory location A into the accumulator and the Store instruction copies the contents of the accumulator into memory location A. Using only one-address instructions, the operation [C] ← [A] + [B] can be performed by executing the sequence of instructions:

```
Load   A
Add    B
Store  C
```

Note that the operand specified in the instruction may be a source or a destination, depending on the instruction. In the Load instruction, address A specifies the source operand and the destination operand is accumulator, which is implied. Address C specifies the destination location in the STORE instruction and here the accumulator is the implied source.

6.4 Instruction Formats

Basic Formats

The information within the computer is of two types: instruction, or data. Data is further subdivided into numerical and non-numerical form. Numerical data is further of two types: fixed point, and floating point.

Fixed Point Numbers

Integers are fixed point numbers which have an implied binary point at the right hand side. The binary fixed point number can be represented as:

$b_a b_b b_c$ ….. b_k

where each b_i is either 0 or 1 and binary point is present on the right side of the number.

Floating point representation

Fractions are represented as floating point numbers. A floating point number consists of two parts: Mantissa M and Exponent E, which denote the number M x B^E, where E is the base.

Non-numeric data

Non-numeric data usually takes the form of variable-length character strings encoded in one or several standard codes, such as ASCII code.

Word length

Information is represented in a computer by means of binary words. A 'word' is a unit of information of some fixed length 'n'. A word with n bits can represent information up to 2^n different things. For example, with n=4, the 10 decimal digits can be encoded as under:

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

Bytes

Bytes are 8-bit words. They are used to encode alphanumeric characters or symbols, which are generally the standard symbols on the keyboard. A byte can store two decimal digits with no wasted space. Most computers have the 8-bit byte as the smallest addressable unit of information.

Within a computer, several word sizes are used to load and store various instructions. Example load and store need long address fields. Precision of a number word is determined by its length.

6.5 ADDRESSING MODES

In general, a program operates on data that resides in computer's memory. The data can be organized in various ways. Programmers use organizations called data structures to represent data used in computation. The different ways in which the location of an operand is specified in an instruction is referred to as 'addressing modes'.

The address field of a typical instruction is relatively small. The aim is to reference a large number of memory locations in main memory or virtual memory. To achieve this, a variety of addressing methods are employed. They involve trade-offs between address range and number of memory references. Some of the common addressing techniques are as under:

1. Immediate addressing
2. Direct addressing
3. Indirect addressing
4. Register addressing
5. Register Indirect addressing
6. Displacement addressing
7. Stack addressing

These modes are explained with the help of figures for the address and the method of accessing the memory. The notations used are as follows: A – Contents of address field in the instruction; R – Contents of address field that refer to a register; EA – Effective address of the memory location containing the referenced operand; (X) – Contents of location X. For the control unit to determine which addressing mode is being used, one or more bits in the instruction format can be used as the mode field. The effective address will either be a main memory address, virtual memory address or a register.

1. Immediate addressing

Immediate addressing is the simplest form of addressing. In this form of addressing, the operand is actually present in the instruction.

OPERAND = A

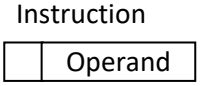


Figure 6.1 Immediate Addressing

This mode is used to set initial values of variables, or to define constants. The number is set as two's complement and th e sign bit. Advantage of this mode is that only the instruction-fetch is required and no memory reference is needed, thus saving one instruction cycle. Disadvantage is that the size of the number is restricted to the size of the address field.

2. Direct addressing

This is also a very simple form of addressing. In direct addressing, the address field contains the effective address of the operand, as shown in Figure 6.2 .

EA = A

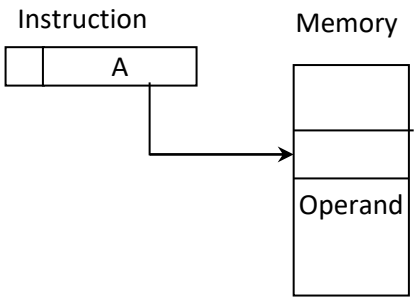


Figure 6.2 Direct Addressing

This scheme requires only one memory reference. The limitation is that it makes available only limited address spaces and is usually less than the word length.

3. Indirect addressing

In indirect addressing, the address field refers to address of a word in memory, which in turn contains the full address of the operand. This is demonstrated in Figure 6.3.

EA = [A]

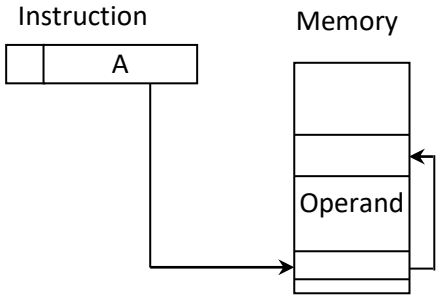


Figure 6.3 Indirect Addressing

The advantage of this mode is that for word length of N, address space of 2^N is available. The disadvantage is that two memory references are needed to fetch the operand. One reference is needed to get the address of the operand and the second reference gets its value.

4. Register Addressing

Register addressing is similar to direct addressing. The difference is that the address field refers to a register rather than a memory location. This form of addressing is exhibit in Figure 6.4.

EA = R

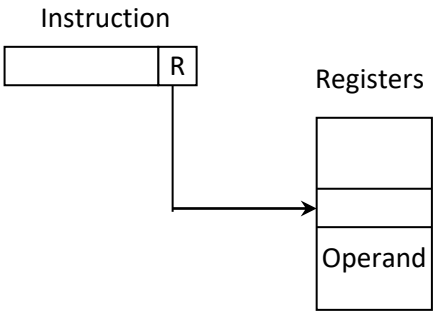


Figure 6.4 Register Addressing

The advantage of register addressing is that only small address field is needed in the instruction. This means that no memory reference is required. Thus, register addressing is very fast. The disadvantage of this mode is that address space very limited.

5. Register Indirect Addressing

Register indirect addressing is similar to indirect addressing but address field refers to a register rather than a memory location. Figure 6.5 shows this form of addressing.

EA = [R]

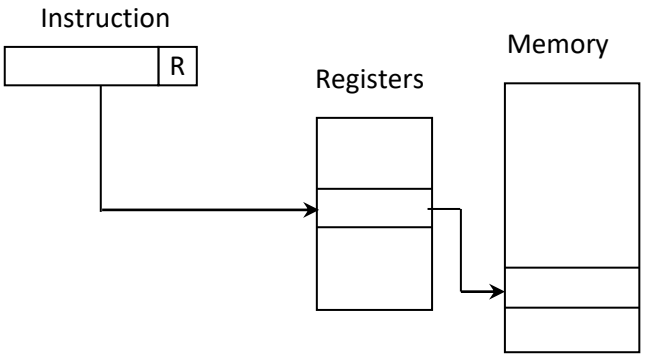


Figure 6.5 Register Indirect Addressing

The advantage of this form of addressing is that though it is the same as indirect addressing, but use one less memory reference. The disadvantage is that it has to access the registers twice.

6. Relative Addressing

This form of addressing combines the capabilities of direct addressing and register indirect addressing. The effective address is given as:

EA = A + [R]

The implied register is the program counter. Current instruction address is added to the address field to produce effective address. The effective address is a displacement relative to the address of the instruction.

Base Register Addressing

The referenced register contains a memory address, and the address field contains a displacement from that address. The register reference may be explicit or implicit. This kind of addressing is quite convenient to implementation segmentations. In some implementations, a single segment-base register is used. In other usage, a programmer may choose a register to hold the base address of the segment.

Index Addressing

In this form of memory reference, the address field references a main memory address, and the referenced register contains a positive displacement from that address. Figure 6.6 depicts Index addressing. This usage is just the opposite of 'base-register' addressing. Here the address field is considered to be a memory address. The method of calculating the effective address is the same as base register addressing.

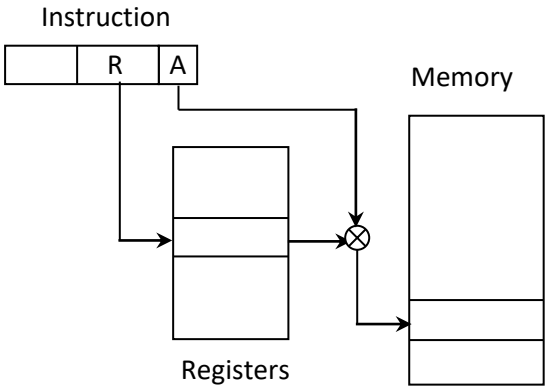


Figure 6.6 Index Addressing

Indexing is used in iterative operations. Index registers are used for such iterative tasks. Here the index register is just incremented or decremented.

EA = EA +(R)

R ← (R) + 1

First the contents of the address field are used to access a memory location containing a direct address. The value of the index register is then added to the register value.

8. Stack addressing

The stack is a reserved block of locations. The stack pointer is maintained in a register. Stack addressing is depicted in Figure 6.7. Stack addressing is similar to register indirect addressing. Stack Pointer maintained in a register

EA = [R]

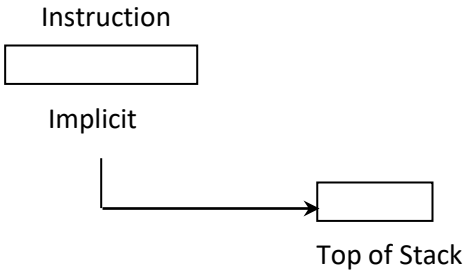


Figure 6.7 Stack Addressing

Stack mode of addressing is a form of implied addressing. The machine instructions do not include a memory reference. The top of the stack is the implied address.

Bachelor of Computer Applications (BCA)

COMPUTER SYSTEM ARCHITECTURE

UNIT VII: INPUT-OUTPUT ORGANIZATION

STRUCTURE

7.0 Objectives

7.1 I/O interfaces

7.2 Data transfer schemes

7.3 I/O control mechanisms

7.3.1 Program controlled

7.3.2 Interrupt controlled

7.3.3 DMA controller

7.4 Summary

7.0 OBJECTIVES

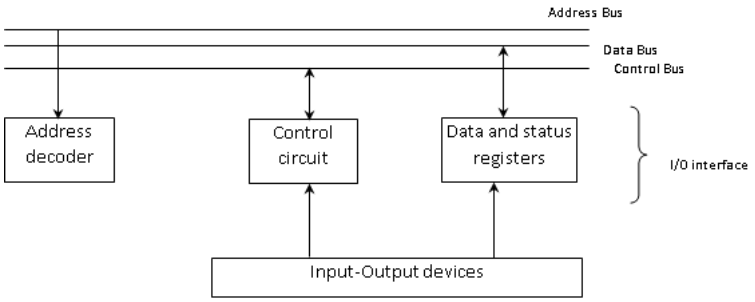
- Understanding I/O interfaces and data transfer schemes.
- Understanding I/O control mechanisms - Program controlled, Interrupt controlled and DMA controller

7.1 I/O INTERFACES

The I/O subsystem of a computer provides an efficient mode of communication between the central system and the outside environment. It handles all the input- output operations of the computer system. Input or output devices that are connected to computer are called peripheral devices

Each I/O device assigned a unique set of address

Device recognizes this address and responds to the commands given by the CPU



Block diagram: Hardware required to connect I/O devices

Interface is a shared boundary between two separate components of the computer system which can be used to attach two or more components to the system for communication purposes. There are two types of interface

1. CPU Inteface
2. I/O Interface

In Input-Output Interface, peripherals connected to a computer need special communication links for interfacing with CPU. In computer system, there are special hardware components between the CPU and peripherals to control or manage the input-output transfers. These components are called input-output interface units because they provide communication links between processor bus and peripherals. They provide a method for transferring information between internal system and input-output devices.

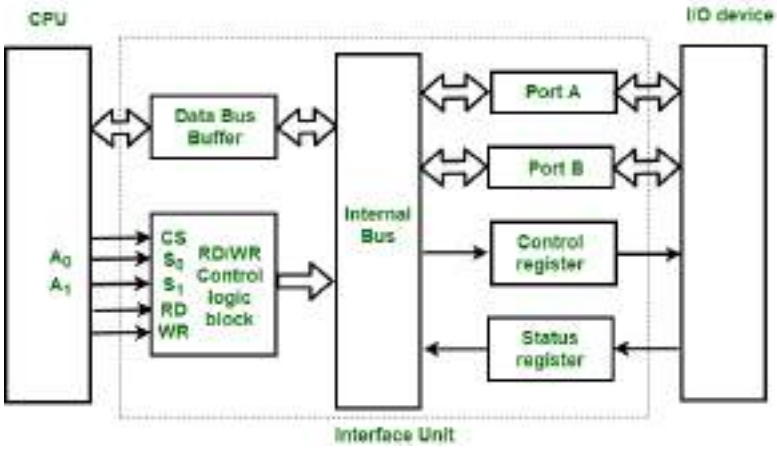
The block diagram of an Input-Output Interface unit contain the following blocks :

1. Data Bus Buffer

2. Read/Write Control Logic

3. Port A, Port B register

4. Control and Status register



Data Bus Buffer :
The bus buffer use bi-directional data bus to communicate with CPU. All control word data and status information between interface unit and CPU are transferred through data bus.

Port A and Port B :

Port A and Port B are used to transfer data between Input-Output device and Interface Unit. Each port consist of bi-directional data input buffer and bi-directional data output buffer. Interface unit connect directly with an input device and output disk or with device that require both input and output through Port A and Port B i.e. modem, external hard-drive, magnetic disk.

Control and Status Register
CPU gives control information to control register on basis of control information. Interface unit control input and output operation between CPU and input-output device. Bits which are present in status register are used for checking of status conditions. Status register indicate status of data register, port A, port B and also record error that may be occur during transfer of data.

Read/Write Control Logic :
This block generates necessary control signals for overall device operations. All commands from CPU are accepted through this block. It also allow status of interface unit to be transferred onto data bus through this block accept CS, read and write control signal from system bus and S_0, S_1 from system address bus. Read and Write signal are used to define direction of data transfer over data bus.

Read Operation: CPU <---- I/O device

Write Operation: CPU ----> I/O device

The read signal direct data transfer from interface unit to CPU and write signal direct data transfer from CPU to interface unit through data bus.

Address bus is used to select to interface unit. Two least significant lines of address bus (A_0 , A_1) are connected to select lines S_0, S_1. This two select input lines are used to select any one of four registers in interface unit. The selection of interface unit is according to the

following criteria :

7.2 DATA TRANSFER SCHEMES

Data transfer schemes can provide an efficient means of transmitting data between the processing unit and the I/O devices. In a computer, the data transfer happens between any of these combinations CPU and memory, CPU and I/O devices, and memory and I/O devices.

A computer is interfaced with many devices of different speeds. Therefore, I/O devices may not be ready to transfer data as soon as the microprocessor issues the instruction for this purpose. Many data transfer schemes have been developed to solve this problem.

Classification of Data Transfer Schemes

The data transfer schemes have been broadly classified into two categories

Programmed Data Transfer Schemes

In a programmed data transfer scheme, data transfer takes place between the CPU and I/O device under the control of a program that resides in the memory. In this scheme, the program is executed by the CPU. This scheme is used when a limited extent of information is to be transferred.

The three important types of programmed data transfer schemes are −

- **Synchronous Data Transfer Scheme** − This type of programmed data transfer scheme is used when the processor and the I/O devices match in speed. Some suitable instructions such as IN and OUT are used for ‘to and from’ data transfer of I/O devices.
- **Asynchronous Data Transfer Scheme** − This type of programmed data transfer scheme is used when the speeds of I/O devices and the microprocessor do not match and also when the timing characteristics of the I/O devices are not predictable.
- **Interrupt Driven Data Transfer Scheme** − In this programmed data transfer scheme, the processor enables the I/O devices and then continues to execute its original program instead of wasting time checking the status of the I/O devices. When the I/O devices are ready to send and receive data, then the processor is informed through a specific control line called the ‘Interrupt line’.
- **DMA Data Transfer Scheme**

In DMA data transfer, data is directly transferred from the memory to the I/O device or vice versa without going through the microprocessor. This scheme is used when there is a requirement to send bulk data. Transferring bulk data using a microprocessor consumes more time. Therefore, the microprocessor performs the data transfer between an I/O device and memory using this DMA technique.

For a DMA transfer, I/O devices must also contain electronic circuitry to generate control signals. But most I/O devices are not equipped with such facilities. Hence, to solve this problem, manufacturers have developed a single-chip programmable DMA controller to interface I/O devices with the microprocessor for DMA transfer.

7.3 I/O CONTROL MECHANISMS

There are three basic I/O mechanisms that computer systems can use to communicate with peripheral devices: memory-mapped input/output, I/O-mapped input/output, and direct memory access (DMA). ... Each I/O mechanism has its own set of advantages and disadvantages, which we will discuss in the following sections.

7.3.1 Programmed I/O

Programmed I/O is a method used in all computers for controlling I/O operations. It is most useful in small, low-speed systems where the cost of hardware has to be kept minimum. In this method of I/O, all I/O operations are executed under the direct control of the CPU. The data transfer is generally between two programmable registers. The I/O device does not have a direct access to the main memory. A data transfer from an I/O device to main memory requires CPU to execute many I/O instructions.

In programmed I/O, the CPU, the memory and the I/O devices communicate through the system bus. The address lines that are used to select the memory locations can also be used to select the I/O devices which are connected to I/O ports that have addresses. The CPU does not make any difference between the addresses of memory locations and the addresses of I/O ports. Programmed I/O is further of two types: Memory mapped I/O and I/O Mapped I/O

In Memory Mapped I/O, the same bus is used to address both memory and I/O devices. A part of the main memory addressable space is assigned to I/O ports. The instructions that are used for addressing the memory and for addressing the I/O ports are the same, except that in the former case, address of memory is referenced whereas in the latter case, address of an I/O port is referenced. The usual load and store instructions are used to transfer data words to or from the I/O ports and no special I/O instructions are needed. Figure 7.14(a) shows the memory structure for memory-mapped I/O. The control lines READ and WRITE are activated by the CPU while processing the memory instruction.

In I/O mapped I/O, a special class of CPU instructions are used for performing I/O. These instructions are IN and OUT instructions which can read and write a single byte to an I/O device. I/O devices have a separate address space from general memory, either provided by an extra I/O pin on the CPU's physical interface, or an entire bus is dedicated to I/O. In the I/O mapped I/O, the addresses assigned to memory and I/O ports are separate. In other words, the address spaces of memory and I/O devices are different. The structure is shown in Figure 7.14(b). The Read M and Write M control lines are activated by memory referencing instruction which does not affect the I/O device.

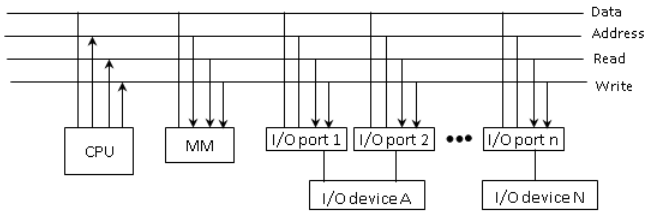


Figure 7.14(a) Memory-Mapped I/O (Shared Memory and I/O Address Space

The CPU executes separate I/O instructions to activate Read I/O and Write I/O lines, which causes a word to be transferred between the addressed I/O port and the CPU.

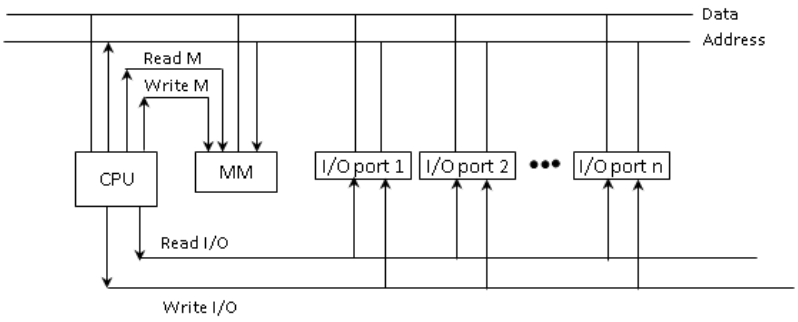


Figure 7.14(b) I/O Mapped I/O (Separate Memory and Address Space

Thus, just two I/O instructions are needed to implement I/O mapped I/O. The instruction IN causes a word to be transferred from the I/O port to one of the registers of the CPU. The instruction OUT transfers a word from a register of the CPU to the I/O port. When the CPU executes the instruction IN or OUT, the I/O port is expected to be ready to respond to the instruction. The CPU can be programmed to test the status of the device. The status of the I/O device is checked by performing the following steps:

Limitations of Programmed I/O:

The limitation of this method is that when processor issues command to the I/O module, it must wait until the I/O operation is complete. As processor is faster than I/O module, lot of time is wasted. The CPU takes time to test and service the I/O device, which limits the speed of data transfer. Thus, the performance of the entire system is affected. The time that is spent by the CPU in testing status of I/O device can be better utilized in other tasks.

7.3.2 Interrupt Initiated I/O

Interrupts are provided to improve the efficiency of the processor as most external devices are much slower than the processor. Thus, whenever a device requests for I/O, the CPU may initiate an interrupt so that the already executing job can wait for the time till the I/O is completed. An example is the data being transferred by the processor to the printer.

In the methods of I/O discussed above, the processor had to wait for I/O module to start read/write of data. The wait state can be reduced or eliminated if processor issues an I/O command to a module without introducing an idle state. For such an event to happen, the I/O module will interrupt the processor to request service when it is ready to exchange data with the processor. The processor executes the data transfer and then resumes its normal

processing. The interrupt signal is generated in case of exceptional events that cause CPU to transfer control temporarily from its current program to another program. This program is known as ‘interrupt handler’ that performs the program to the event that needs service. Interrupts can be internal or external. I/O interrupts are external interrupts where the CPU initiates or terminates an I/O request. Interrupts are also produced by hardware or software error detection.

The interrupt is provided to the program running in the CPU by activating a control line called ‘Interrupt Request’, which connects the interrupt source to the CPU. A register in the CPU stores the interrupt indicator that the CPU tests at regular intervals, usually at the end of every instruction cycle.

When two or more interrupt requests are present at the same time, priorities must be assigned to the interrupts. The interrupt with the highest priority is selected for handling. The following events occur:

1. CPU identifies source of interrupt
2. CPU obtains memory address of the required interrupt handler.
3. Save status of CPU registers and PC in a subroutine call.
4. Load address of the interrupt handler in the PC.
5. Execute the routine till a return instruction is reached.
6. When return instruction encountered, control is transferred to the interrupted program.

If an interrupt is masked, the CPU ignores such interrupts. High priority interrupts are serviced before the low priority interrupts.

Interrupt Selection

The techniques used for selection of an interrupt are polling, daisy chaining and independent requests.

Vectored Interrupts

‘Vectored interrupt’ is the kind of interrupt where the interrupting device supplies the CPU with the starting address or the ‘interrupt vector’ of the program. In this case, the interrupt request from a particular device causes a direct hardware implementation to the correct interrupt-handling program. When multiple requests for interrupts are received, each interrupt line generates a unique fixed request. The address made available by each request is used to modify the contents of the PC. Interrupt requests are received in an interrupt register. If the interrupt register has b bits, it can give priority to b devices, where bit b_i will enable or disable the line connecting device i. Figure 7.15 depicts a multiple-interrupt line arrangement using Vectored Interrupt

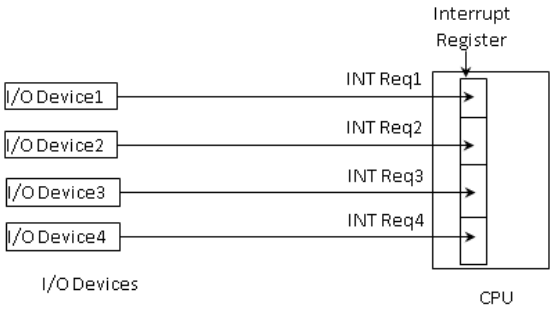


Figure 7.15 Vectored Interrupt Architecture

The program control is transferred using Vectored Interrupts with the arrangement shown in Figure 7.15. When an interrupt request from I/O port i is accepted then the address is generated from the priority encoder and used by the PC. As soon as interrupt occurs, many events start both in the hardware and software. The events are: Device issues an interrupt signal to the processor, Processor finishes execution of current instruction before responding to the interrupt, Processor sends an acknowledgement signal to the device that issued the interrupt, and the control is transferred to the interrupt routine after saving information of program status word, PC pushed onto the stack and needed to resume the current program at the point of interrupt. After this, PC is loaded with the interrupt service routine

The process of interrupt initiated I/O is that the I/O module receives a Read command from the processor. The I/O module then reads the data in from the concerned peripheral device. The data is input in the data register of the module and at the same time an interrupt signal is sent over the control line. The module then waits until the processor requests for the data. When the request is made, module places the data on the data bus, and then is ready for another I/O operation. Thus, for the input operation, the processor just has to issue a Read command, and then continue doing whatever it was doing. When the interrupt occurs, the status of all the registers is saved and the I/O is processed.

A tradeoff between programmed I/O and Interrupt initiated I/O is termed as DMA, and is used to transfer the data over the I/O bus with the least intervention of the CPU.

7.3.3 DMA Controller

Methods of transferring data between the peripheral device and the rest of the system require a large overhead, as well as the CPU time. The CPU affects the rate of I/O transfer in two ways: First, a delay occurs while the I/O device is waiting to be tested. Second, Programmed I/O transfers data through the CPU rather than passing the data directly to the I/O device.
A solution is not to bring the CPU in the data transfer and have the system memory communicate directly with I/O device.
Direct Memory Access (DMA) refers to a method of movement of data where a peripheral device transfers information directly to or from memory, without the CPU being required to perform the transfer. DMA is a system that can control the memory without using the CPU, allowing certain hardware subsystems within a computer to access system memory for

reading and/or writing independently of the CPU. Direct memory access (DMA) channels are system pathways used by many devices to transfer information directly to and from memory. Examples of hardware systems that use DMA include disk drive controllers, graphic cards, network cards and sound cards. Computers that have DMA channels can transfer data to and from devices much more quickly than computers without a DMA channel. This is useful for making quick backups and for real-time applications.

Principle

DMA is an essential feature of all modern computers as it allows devices to transfer data without subjecting the CPU to spend time waiting for certain events to happen. Otherwise, the CPU would have to copy each piece of data from the source to the destination. A DMA transfer essentially copies a block of memory from one device to another. While the CPU initiates the transfer, it does not execute the transfer itself. The transfer is performed by a DMA controller which is typically part of the motherboard.

A typical DMA request is used to copy a block of memory from system RAM to or from a buffer on the device. Such an operation does not stop the processor to perform other tasks. DMA transfers are essential to high performance systems

DMA module takes over control of the system to transfer data to and from memory. It uses the bus when the processor does not need it, or it suspends operation temporarily.

The process of DMA is summarized below:

1. Know whether read or write is requested, using read/write control line between processor and DMA module
2. Know the address of the I/O device issuing request and communicate on the data line
3. The starting address of memory location to read or write, communicated on data lines and stored by the DMA module in the IODR
4. The number of words to be read or written, which is stored in the data count register DCR

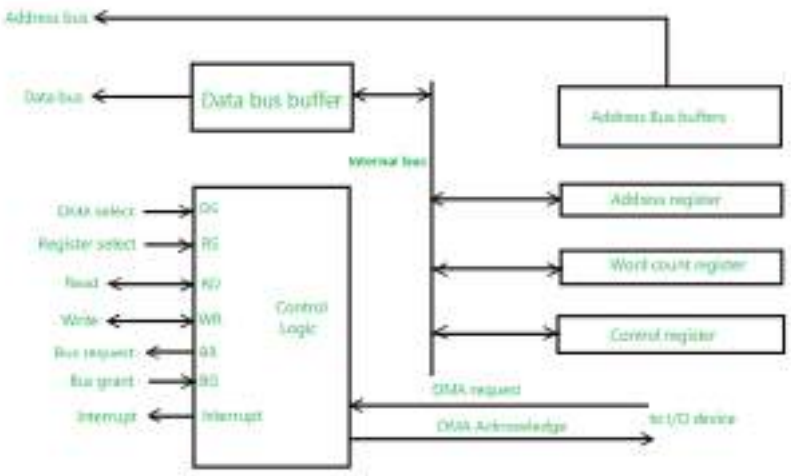
DMA Controller is a hardware device that allows I/O devices to directly access memory with less participation of the processor. DMA controller needs the same old circuits of an interface to communicate with the CPU and Input/Output devices.

Figure below shows the block diagram of the DMA controller. The unit communicates with the CPU through data bus and control lines. Through the use of the address bus and allowing the DMA and RS register to select inputs, the register within the DMA is chosen by the CPU. RD and WR are two-way inputs. When BG (bus grant) input is 0, the CPU can communicate with DMA registers. When BG (bus grant) input is 1, the CPU has relinquished the buses and DMA can communicate directly with the memory.

The DMA controller has three registers as follows.

- Address register – It contains the address to specify the desired location in memory.
- Word count register – It contains the number of words to be transferred.
- Control register – It specifies the transfer mode.

All registers in the DMA appear to the CPU as I/O interface registers. Therefore, the CPU can both read and write into the DMA registers under program control via the data bus.



Block Diagram of DMA controller

Explanation :

The CPU initializes the DMA by sending the given information through the data bus.

- The starting address of the memory block where the data is available (to read) or where data are to be stored (to write).
- It also sends word count which is the number of words in the memory block to be read or write.
- Control to define the mode of transfer such as read or write.
- A control to begin the DMA transfer.

DMA transfers overcome the problem of occupying the CPU for the entire time it's performing a transfer. The CPU initiates the transfer, then it executes other ops while the transfer is in progress, finally it receives an interrupt from the DMA controller when the transfer is done Hardware using DMA: disk drives, graphics cards, network cards, sound cards DMA can lead to cache coherency problems If a CPU has a cache and external memory, then the data the DMA controller has access to (stored in RAM) may not be updated with the correct data stored

The objective of DMA is to move functionality from the CPU to peripherals because:

- Peripherals use less current than the CPU;
- Performing operations directly between peripherals allows the CPU to shut down, saving system power;
- Minimal software requirements and CPU cycles

7.4 SUMMARY

The main purpose of the I/O interfaces is to transmit and receive data; however, the portion designated as an I/O interface may contain additional resources, such as voltage translators, registers, impedances, and buffers. Peripherals connected to a computer need special communication links for interfacing with CPU. In computer system, there are special hardware components between the CPU and peripherals to control or manage the input-output transfers. These components are called input-output interface units because they provide communication links between processor bus and peripherals.

Modes of I/O Data Transfer

Data transfer between the central unit and I/O devices can be handled in generally three types of modes which are: Programmed I/O, Interrupt Initiated I/O, Direct Memory Access

Programmed I/O instructions are the result of I/O instructions written in computer program. Each data item transfer is initiated by the instruction in the program. Usually, the program controls data transfer to and from CPU and peripheral. Transferring data under programmed I/O requires constant monitoring of the peripherals by the CPU. In the programmed I/O method the CPU stays in the program loop until the I/O unit indicates that it is ready for data transfer. This is time consuming process because it keeps the processor busy needlessly.

This problem can be overcome by using interrupt initiated I/O. In this when the interface determines that the peripheral is ready for data transfer, it generates an interrupt. After receiving the interrupt signal, the CPU stops the task which it is processing and service the I/O transfer and then returns back to its previous processing task. In the discussion on computer architecture and the role of the Central Processing Unit a brief description was given on how the CPU may transfer data to or from a number of external (other than memory) devices. The operation treated the I/O system for reading and writing in the same manner as memory, using address, data lines and WR RD control lines. This requires CPU intervention and is costly in "time". Direct Memory Access--the ability of an I/O subsystem to transfer data to and from a memory subsystem without processor intervention. DMA Controller--a device that can control data transfers between an I/O subsystem and a memory subsystem in the same manner that a processor can control such transfers.

Using Direct Memory Access solves many problems. Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This technique is known as DMA. In this, the interface transfer data to and from the memory through memory bus. A DMA controller manages to transfer data between peripherals and memory unit. The DMA controller can issue commands to the memory that behave exactly like the commands issued by the CPU. The DMA controller in a sense is a second processor in the system but is dedicated to an I/O function. The DMA controller as shown below connects one or more I/O ports directly to memory, where the I/O data stream passes through the DMA controller faster and more efficiently than through the processor as the DMA channel is specialised to the data transfer task

The DMA adds one more level of complexity to the I/O interface because a DMA controller has independent access to memory. One set of wires (bus) can carry at most one transaction at a time. If the DMA and a microprocessor share the signal wire to memory there must be a mechanism to arbitrate which shall have access to memory when both attempt to at the same time.

Bachelor of Computer Applications (BCA)

COMPUTER SYSTEM ARCHITECTURE

UNIT VIII: MEMORY UNIT

STRUCTURE

8.0 Objective

8.1 Memory Hierarch: An Introduction

8.2 High-speed Memories

8.3 Organization of a Cache Memory Unit

8.3.1 Characteristics of Cache Memories

8.3.2 Cache Performance

8.3.3 Mapping Data in Memory to a Location in Cache

8.4 Virtual Memory

8.4.1 Virtual Address Space

8.4.2 Physical Address Space

8.5 Memory Management

8.5 Memory Management

8.6 Summary

8.0 Objectives

- To understand Memory hierarchy and High-speed Memories
- Organization of a Cache Memory Unit, Virtual Memory and Memory Management

8.1 MEMORY HIERARCH: AN INTRODUCTION

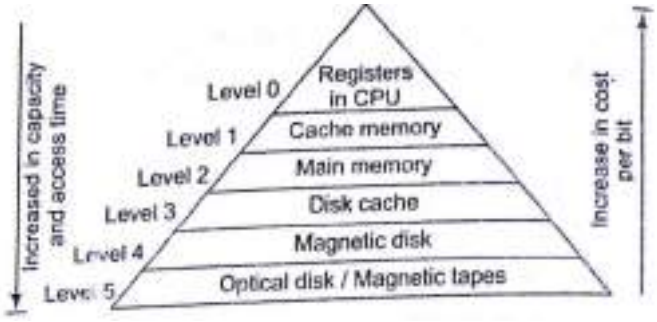
The memory system of a computer comprises of various kinds of memory where programs and data are stored. Various types of memories are used in the computer system. Ideally, the memory should be fast, large, and inexpensive. But it is not possible to meet all these requirements simultaneously. To solve this problem, many different structures and organizations are used, which will be discussed in the following sections.

The basic unit of memory is the binary digit or bit. The bit may contain 0 or 1. The programs and data are stored in the memory, which are transferred to the CPU for the execution. To increase the performance of the computer system, the processor speed has to match with the rate of information transfer from the memory. The aim is to provide high bandwidth of the memory at reasonable cost.

In computer architecture, the memory hierarchy separates computer storage into a hierarchy based on response time. Since response time, complexity, and capacity are related, the levels may also be distinguished by their performance and controlling technologies.

Memory Hierarchy is an enhancement to organize the memory such that it can minimize the access time. The Memory Hierarchy was developed based on a program behavior known as locality of references.

The various types of memories in the computer have different speeds and costs, arranged as levels in a hierarchy. The goal in designing an n-level memory hierarchy is to achieve a performance close to that of the fastest memory, and cost per bit of memory should be close to the cheapest memory. The hierarchy is structured so that the memories at level i are ‘higher’ than the memories at level i+1. The word ‘higher’ means that the performance attributes should be higher. The speed is inversely proportional to capacity and cost. The hierarchy of memory in a general computer is as shown below:



Memory Hierarchy

Memories in a hierarchy can be classified on the basis of method of access and time of access, which are explained below:

i. Accessing method

The method of access of a memory divides the memories into three basic classes:

- **Random-Access Memory (RAM)**

The access time t_a of a RAM word is independent of its location

- **Sequential-Access Memory (SAM)**

Static random-access memory (static RAM or SRAM) is a type of random-access memory (RAM) that uses latching circuitry (flip-flop) to store each bit. SRAM is volatile memory; data is lost when power is removed. The information in SAMs is accessed serially or sequentially. Examples are shift register memory like a first-in-first-out (FIFO) buffer, charged-coupled-devices (CCDs) and magnetic bubble memories. A direct-access storage device (DASD) is a secondary storage device in which each physical record has a discrete location and a unique address Term DASD is a shorthand describing hard disk drives, magnetic drums, and data cells. DASDs are rotational devices made of magnetic materials where any block of information can be accessed directly. They are accessed through special interfaces called channels.

ii. Speed or Access Time

The memory hierarchy is generally organized so that the highest level has the fastest speed and the lowest level has the slowest speed, as is shown in Figure 6.1. On the basis of access time, the memory is classified as under:

- **Primary Memory**

This Memory is made of RAMs

- **Secondary Memory**

Secondary Memory is made of DASDs or SAMs. The three most common DASDs are drums, fixed-head disks and movable-arm disks.

8.2 HIGH-SPEED MEMORIES

Caches are high speed memories realized using SRAM technology that stores a small subset of the data in the main memory and that the CPU can access directly and with minimal delay. Data is transferred in and out of the cache as and when required by the CPU. Caches nowadays are included on the processor chip itself.

Since processor chip size cannot be increased beyond a certain size cache size on processor chips are usually very limited. All high performance processors have some form of on chip cache memory. This cache can again be divided into separate instruction and data caches. This is called a Split cache. Having a combined cache offers greater hit rate as it offers flexibility in mapping new information in the cache. Split caches on the other hand make it possible to simultaneously access both the caches. This leads to increased performance but needs more complicated circuitry for the parallel access.

High Performance Processors have also included multiple levels of cache. L1 cache is present on chip and L2 is implemented using SRAM technology and is external to the processor. However, L2 cache in a much smaller size can be included on chip as well thus giving multi-level on chip cache.

If two levels of cache are used and as L1 is more close to the processor it is important to design the L1 cache in such way that promotes faster access by the processor. This is because its access time has a large effect on the clock rate of the processor. A cache even if it is on chip cannot be accessed at the same speed as the CPU registers. The cache is much bigger and complex. Accesses to cache can be speeded up by accessing multiple words at the same time and then transferring it one by one to the processor for execution. This technique is used in many processors. The second level and subsequent levels of cache are now designed to be much slower than the L1 cache but have to be big enough to have a high hit rate.

8.3 ORGANIZATION OF A CACHE MEMORY UNIT

Cache memory, also called cache, is supplementary memory system that temporarily stores frequently used instructions and data for quicker processing by the central processing unit (CPU) of a computer. Cache holds a copy of only the most frequently used information or program codes stored in the main memory. The term cache refers to fast intermediate memory within a large memory system. They provide the CPU with fast, single-cycle access to the external memory. Cache acts as a buffer between CPU and main memory.

8.3.1 Characteristics of Cache Memories

In the memory hierarchy, it is seen that the speed of main memory is very less as compared to the speed of the CPU's registers. The speed of cache is quite close to the speed of the CPU. Thus, by the use of cache memory, the processor does not have to spend much time in waiting to access instructions and data from memory.

Cache Memory is a special very high-speed memory. It is used to speed up and synchronize with high-speed CPU. Cache memory is costlier than main memory or disk memory but economical than CPU registers. Cache memory is an extremely fast memory type that acts as a buffer between RAM and the CPU. It holds frequently requested data and instructions so that they are immediately available to the CPU when needed. Cache memory is used to reduce the average time to access data from the Main memory. The cache is a smaller and faster memory which stores copies of the data from frequently used main memory locations. There are various different independent caches in a CPU, which store instructions and data.

The cache is effective because of its property of '**locality of reference'**. This principle is based on the fact that instructions in localized areas of the program are executed repeatedly during some time period, and the rest of the program is accessed relatively infrequently. The active program is placed in the cache memory and the total execution time is reduced significantly. The arrangement of the cache memory is shown in Figure 8.1.

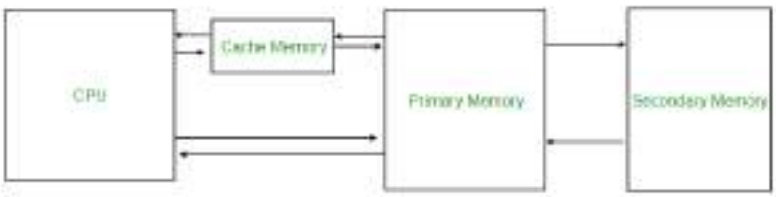


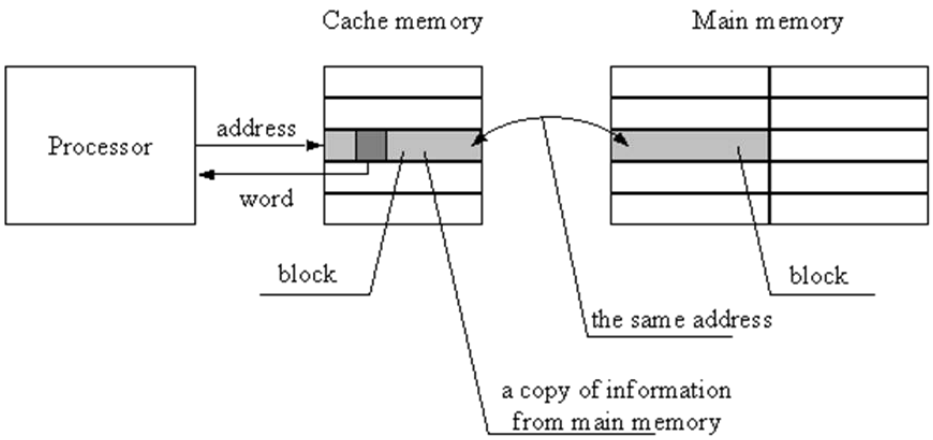
Figure 8.1 Cache memory arrangement

8.3.2 Cache Performance

When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache.

- If the processor finds that the memory location is in the cache, a **cache hit** has occurred and data is read from cache
- If the processor **does not** find the memory location in the cache, **cache miss** has occurred. For a cache miss, the cache allocates a new entry and copies in data from main memory, then the request is fulfilled from the contents of the cache.

If there is a cache memory in a computer system, then at each access to a main memory address in order to fetch data or instructions, processor hardware sends the address first to the cache memory. The cache control unit checks if the requested information resides in the cache. If so, we have a "hit" and the requested information is fetched from the cache. The actions concerned with a read with a hit are shown in the figure below.



'Locality of reference' implies that whenever an information item (instruction or data) is first needed, it should be brought in cache and remain there till it is needed again. Also, instead of fetching just one item from the main memory to the cache, several items that reside at adjacent addresses should be also brought in at the same time.

Example

When an application is running, it may cache certain data in the system memory, or RAM. ... For example, if you are working on a video project, the video editor may load specific video clips and audio tracks from the hard drive into RAM.

Cache memory temporarily stores information, data and programs that are commonly used by the CPU. When data is required, the CPU will automatically turn to cache memory in search of faster data access.

8.3.3 Mapping Data in Memory to a Location in Cache

Cache is close to CPU and faster than main memory. But at the same time is smaller than main memory. The cache organization is about mapping data in memory to a location in cache.

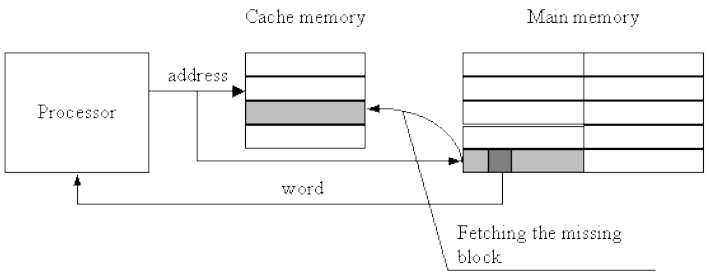
One way to go about this mapping is to consider last few bits of long memory address to find small cache address, and place them at the found address.

But the problem with this approach is, loss of the information about high order bits and no way to find out the lower order bits belong to which higher order bits.

The solution is to use Tag

To handle above problem, more information is stored in cache to tell which block of memory is stored in cache. We store additional information as Tag.

If the requested information does not reside in the cache, we have a "miss" and the necessary information is fetched from the main memory to the cache and to the requesting processor unit. The information is not copied in the cache as single words but as a larger block of a fixed volume. Together with information block, a part of the address of the beginning of the block is always copied into the cache. This part of the address is next used at readout during identification of the proper information block. The actions executed in a cache memory on "miss" are shown below.



Explanation:

Assume a single level of cache memory below. If there are two cache levels, then on "miss" at the first level, the address is transferred in a hardwired way to the cache at the second level.

If at this level a "hit" happens, the block that contains the requested word is fetched from the second level cache to the first level cache. If a "miss" occurs also at the second cache level, the blocks containing the requested word are fetched to the cache memories at both levels. The size of the cache block at the first level is from 8 to several tens of bytes (a number must be a power of 2). The size of the block in the second level cache is many times larger than the size of the block at the first level.

Cache Block

A set of contiguous address location of same size is referred to by the term ‘block’ or ‘cache lines’, which are of the same length. A cache typically contains 4 to 64 consecutive bytes. Lines are numbered consecutively starting from 0. Thus, with 32 byte line size, line 0 is 0 to 31 bytes, line 1 is bytes 32 to 63 and so on. At any time, some lines are in the cache. When memory is referenced, the cache controller circuit checks if the referenced word is in the cache or not. In case of a hit, the data is transferred from the cache. In case of a miss, a replacement policy is used. When a read request is received from the processor, the contents of a block of memory words containing the location specified are transferred into the cache one word at a time. After this, when the program references any of the locations in this block, these contents are read directly from the cache. As the size of the cache memory is small, the number of blocks in the cache is small as compared to the total number of blocks of the main memory. A mapping function is used for the correspondence between the main memory blocks and those in the cache.

The memory words in a cache are stored in a cache data memory. The main memory is divided into set of blocks and the contents of the data memory are the sets of main memory blocks. The data memory is grouped into small pages called blocks or lines. Each cache block is marked with its block address, which is referred to as **tag**. The tag specifies to what part of memory space the block belongs. The collection of tag addresses currently assigned to the cache are stored in a special memory called the **cache tag memory** or **directory**. The structure of cache is shown in Figure 8.2. For the cache to improve performance of a computer, the time required to check the tag addresses and access the cache’s data memory must be less than the time required to access the main memory. For example, if main memory using DRAM has access time t_{AD} = 50 ns, cache’s data memory in SRAM technology has

access time t_{AS} = 10 ns, then the organization of the cache should be such that matching of tag addresses should be extremely fast.

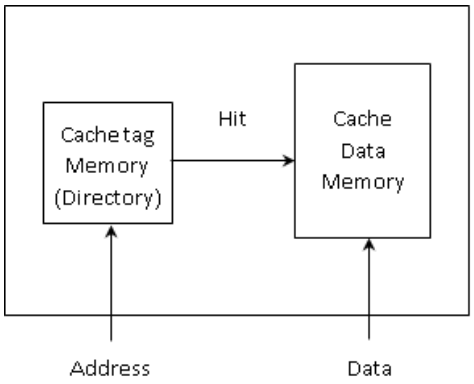


Figure 8.2 Structure of Cache

8.4 VIRTUAL MEMORY

Virtual memory is a technique of memory management that is used to allow slow memory to be used as a level of memory system. It also provides protection between programs running on the same system so that one program cannot modify another program's data.

Virtual Memory is a storage scheme that provides user an illusion of having a very big main memory. This is done by treating a part of secondary memory as the main memory.

In this scheme, user can load the bigger size processes than the available main memory by having the illusion that the memory is available to load the process.

Instead of loading one big process in the main memory, the Operating System loads the different parts of more than one process in the main memory. By doing this, the degree of multiprogramming is increased and therefore, the CPU utilization will also be increased. In modern world, virtual memory has become quite common these days. In this scheme, whenever some pages need to be loaded in the main memory for the execution and the memory is not available for those many pages, then in that case, instead of stopping the pages from entering in the main memory, the OS search for the RAM area that are least used in the recent times or that are not referenced and copy that into the secondary memory to make the space for the new pages in the main memory.

Since all this procedure happens automatically, therefore it makes the computer feel like it is having the unlimited RAM.

The concept of virtual memory was developed to reduce the requirements of having a large main memory. The hard disks or other magnetic media forms the bottom layer of the memory system. DRAMs or core memory form the main memory. The address space of a program is divided into pages, which are adjacent pages of data, and are stored on the magnetic media. When a page of data is referenced, the system copies it into the main memory and is used for the execution of the program. This may require another page of data to be copied from the main memory to the magnetic disk.

8.4.1 Virtual address space

Each program has its own virtual address space, which is a set of addresses that programs use for load and store operations.

8.4.2 Physical address space

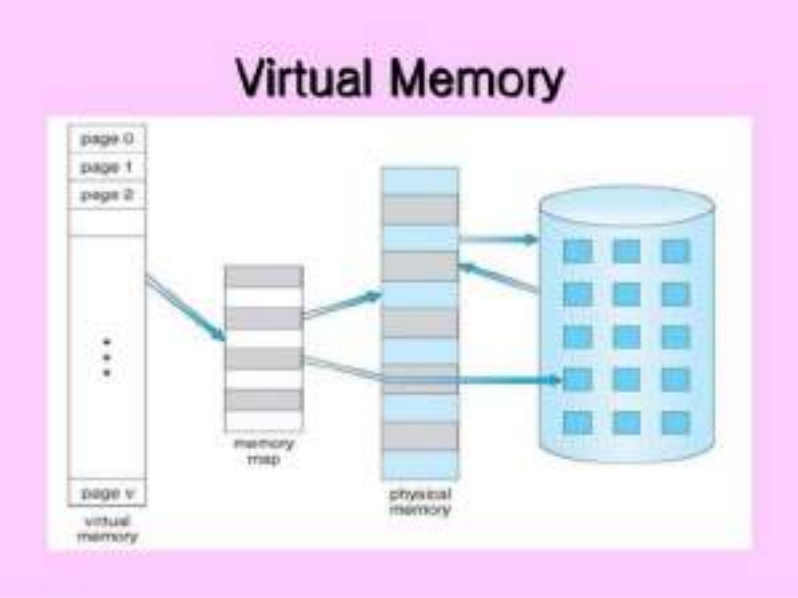
The physical address space is a set of addresses used to reference locations in the main memory.

The terms **virtual address** and the **physical (or real, used synonymously) address** are used to describe addresses in the virtual address space and the physical address space.

The arrangement of virtual memory is shown in Figure 8.4.1

A page, memory page, or virtual page is a fixed-length contiguous block of virtual memory, described by a single entry in the page table. Similarly, a page frame is the smallest fixed-length contiguous block of physical memory into which memory pages are mapped by the operating system. Paging uses fixed size pages to move between main memory and secondary storage. Paging uses page tables to map the logical addresses to physical addresses.

Virtual memory gives the illusion that the main memory is much larger than it actually is. When a program references a virtual address, it cannot tell whether the virtual address was resident in the main memory of the computer or whether it had to be fetched from the magnetic memory.



For example, the user might try to load their email in their browser window while also running a word processing software, a shift scheduling software and a content management system at the same time.

8.5 MEMORY MANAGEMENT

Virtual memory is a feature of an operating system that enables a computer to be able to compensate shortages of physical memory by transferring pages of data from random access memory to disk storage. .This process allows for RAM to be freed up so that a computer can complete the task.

Various models are used for memory management for allocating pages to processes. The two policies in use are for allocating pages to active processes: fixed and variable partitioning policies are used to manage the allocation of memory pages.

In fixed allocation policy, the partition of memory allocated process is fixed for the life-time of the process. In the variable allocation policy, the partition varies dynamically during the lifetime of the process and the requirements of the process. Various paging algorithms are used for fixed and variable management policies.

There are some common policies used for page-replacement memory policies for fixed-space schemes:

1. Least recently used (LRU)

In case of page fault, this algorithm replaces the page in Z(t) with the largest backward distance:

$Q(z) = y$ if and only if $b_i(y) = \max [b_i(x)]$

2. Least frequently used (LFU)

This algorithm replaces the page in Z(t) that has been referenced the least number of

Demand Paging is a popular method of virtual memory management. In demand paging, the pages of a process which are least used, get stored in the secondary memory.

A page is copied to the main memory when its demand is made or page fault occurs. There are various page replacement algorithms which are used to determine the pages which will be replaced. We will discuss each one of them later in detail.

times.

1. First-in first-out (FIFO)

This algorithm replaces the page in Z(t) that has been in memory for the longest time.

2. Last-in first-out (LIFO)

This algorithm replaces the page in Z(t) that has been in memory for the shortest time.

3. Random (RAND)

This algorithm chooses a page in Z(t) at random for replacement.

LRU is the most popular algorithm. A dynamic list, known as a LRU stack is linked with it. This stack arranges the referenced pages from top to bottom by decreasing order of frequency of reference. At the page replacement time, LRU policy chooses the lowest ranked page in the stack.

Advantages of Virtual Memory

1. The degree of Multiprogramming will be increased.
2. User can run large application with less real RAM.
3. There is no need to buy more memory RAMs.

Disadvantages of Virtual Memory

1. The system becomes slower since swapping takes time.

2. It takes more time in switching between applications.
3. The user will have the lesser hard disk space for its use.

8.6 SUMMARY

Memory of a computer system is a hierarchy of memory of various technologies which are organized to provide best performance in terms of speed and cost. The characteristics of memory, which are access time and accessing methods are used as benchmarks for constructing primary memory, secondary memory and cache memory. The various kinds of memory devices are used according to their storage capacity and acceptable level of performance. In Random-Access-Memories, it takes the same amount of time to access any storage location, and they can be accessed in any order. The SRAM retains its state as long as power is applied to it. The charge in the capacitor of a DRAM cell is maintained for tens of milliseconds. In serial access memories, access mechanism is shared by the storage locations.