11. Write a function that prints a dictionary where the keys are numbers between 1 and 5 and the values are cubes of the keys.
12. Consider a tuple t1=(1, 2, 5, 7, 9, 2, 4, 6, 8, 10). WAP to perform following operations:
    a) Print half the values of the tuple in one line and the other half in the next line.
    b) Print another tuple whose values are even numbers in the given tuple.
    c) Concatenate a tuple t2=(11,13,15) with t1.
    d) Return maximum and minimum value from this tuple
13. WAP to accept a name from a user. Raise and handle appropriate exception(s) if the text entered by the user contains digits and/or special characters.

Essential/recommended readings

1. Taneja, S., Kumar, N. *Python Programming- A modular Approach*, 1st edition, Pearson Education India, 2018.
2. Balaguruswamy E. *Introduction to Computing and Problem Solving using Python,* 2nd edition, McGraw Hill Education, 2018.

Suggestive readings

(i) Brown, Martin C. *Python: The Complete Reference*, 2nd edition, McGraw Hill Education, 2018.

(ii) Guttag, J.V. *Introduction to computation and programming using Python,* 2nd edition, MIT Press, 2016.

Note: Examination scheme and mode shall be as prescribed by the Examination Branch, University of Delhi, from time to time.

DISCIPLINE SPECIFIC CORE COURSE – 2 (DSC-2): Computer System Architecture

Credit distribution, Eligibility and Prerequisites of the Course

Course title & Code	Credits	Credit distribution of the course			Eligibility criteria	Pre-requisite of the course (if any)
		Lecture	Tutorial	Practical/ Practice		
DSC 02 Computer System Architecture	4	3	0	1	Pass in Class XII	Pass in Class XII

Learning Objectives

This course introduces the students to the fundamental concepts of digital computer organization, design and architecture. It aims to develop a basic understanding of the building blocks of the computer system and highlights how these blocks are organized

together to architect a digital computer system.

Learning outcomes

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

- Design and Simplify Combinational and sequential circuits using basic building blocks.
- Represent data in binary form, convert numeric data between different number systems and perform arithmetic operations in binary.
- Explain instruction cycle, pipelining and interrupts.
- Explain data communication between CPU, memory and I/O devices.
- Simulate the design of a basic computer using a software tool.

SYLLABUS OF DSC- 2

UNIT – I (2 Weeks)
Digital Logic Circuits: Logic Gates, Truth Tables, Boolean Algebra, Digital Circuits, Combinational Circuits, Introduction to Sequential Circuits, Circuit Simplification using Karnaugh Map, Don't Care Conditions, Flip-Flops, Characteristic Tables, Excitation Table.

UNIT – II (3 Weeks)
Digital Components (Fundamental building blocks): Designing of combinational circuits- Half Adder, Full Adder, Decoders, Encoders, Multiplexers, Registers and Memory (RAM , ROM and their types) , Arithmetic Microoperations, Binary Adder, Binary Adder-Subtractor.

UNIT – III (2 Weeks)
Data Representation and Basic Computer Arithmetic: Number System, r and (r-1)'s Complements, data representation and arithmetic operations.

UNIT – IV (3 Weeks)
Basic Computer Organization and Design: Bus organization, Microprogrammed vs Hardwired Control , Instruction Codes, Instruction Format, Instruction Cycle, Instruction pipelining, Memory Reference, Register Reference and Input Output Instructions, Program Interrupt and Interrupt Cycle.

UNIT – V (2 Weeks)
Processors: General register organization, Stack Organization, Addressing Modes, Overview of Reduced Instruction Set Computer (RISC) , Complex Instruction Set Computer (CISC), Multicore processor and Graphics Processing Unit (GPU)

UNIT – VI (3 Weeks)
Memory and Input-Output Organization: Memory hierarchy (main, cache and auxiliary memory), Input-Output Interface, Modes of Transfer: Programmed I/O, Interrupt initiated I/O, Direct memory access.

Practical component (if any) -

(Use Simulator – CPU Sim 3.6.9 or any higher version for the implementation)

1. Create a machine based on the following architecture:

Registers

IR	DR	AC	AR	PC	I	E
16 bits	16 bits	16 bits	12 bits	12 bits	1 bit	1 bit

emory 4096 words 6 bits per word	Instruction format 15 12 11 0

Opcode	Address

Basic Computer Instructions

Memory Reference			Register Reference	
Symbol	Hex		Symbol	Hex
AND	0xxx	Direct Addressing	CLA	7800
ADD	1xxx		CLE	7400
LDA	2xxx		CMA	7200
STA	3xxx		CME	7100
BUN	4xxx		CIR	7080
BSA	5xxx		CIL	7040
ISZ	6xxx		INC	7020
AND_I	8xxx	Indirect Addressing	SPA	7010
ADD_I	9xxx		SNA	7008
LDA_I	Axxx		SZA	7004
STA_I	Bxxx		SZE	7002
BUN_I	Cxxx		HLT	7001
BSA_I	Dxxx		INP	F800

ISZ_I	Exxx		OUT	F400

Refer to Chapter-5 of reference 1 for description of instructions.

Design the register set, memory and the instruction set. Use this machine for the assignments of this section.

2. Create a Fetch routine of the instruction cycle.
3. Write an assembly program to simulate ADD operation on two user-entered numbers.
4. Write an assembly program to simulate SUBTRACT operation on two user-entered numbers.
5. Write an assembly program to simulate the following logical operations on two user- entered numbers.
   i. AND
   ii. OR
   iii. NOT
   iv. XOR
   v. NOR
   vi. NAND
6. Write an assembly program for simulating following memory-reference instructions.
   i. ADD
   ii. LDA
   iii. STA
   iv. BUN
   v. ISZ
7. Write an assembly language program to simulate the machine for following register reference instructions and determine the contents of AC, E, PC, AR and IR registers in decimal after the execution:
   i. CLA
   ii. CMA
   iii. CME
   iv. HLT
8. Write an assembly language program to simulate the machine for following register reference instructions and determine the contents of AC, E, PC, AR and IR registers in decimal after the execution:

i. INC

ii. SPA

iii. SNA

iv. SZE

9. Write an assembly language program to simulate the machine for following register reference instructions and determine the contents of AC, E, PC, AR and IR registers in decimal after the execution:

i. CIR

ii. CIL

10. Write an assembly program that reads in integers and adds them together; until a negative non-zero number is read in. Then it outputs the sum (not including the last number).

11. Write an assembly program that reads in integers and adds them together; until zero is read in. Then it outputs the sum.

Essential/recommended readings

1. David A. Patterson and John L. Hennessy. "*Computer Organization and Design : The Hardware/Software interface*", 5th edition, Elsevier, 2012.
2. Mano, M. *Computer System Architecture*, 3rd edition, Pearson Education, 1993.

Suggestive readings (if any)

(i) Mano, M. *Digital Design*, Pearson Education Asia, 1995.

(ii) Null, L., & Lobur, J. *The Essentials of Computer Organization and Architecture*, 5th edition, (Reprint) Jones and Bartlett Learning, 2018.

(iii) Stallings, W. *Computer Organization and Architecture Designing for Performance* 8th edition, Prentice Hall of India, 2010.

DISCIPLINE SPECIFIC CORE COURSE– 3 (DSC-3): Mathematics for computing

Credit distribution, Eligibility and Pre-requisites of the Course

Course title & Code	Credits	Credit distribution of the course			Eligibility criteria	Pre-requisite of the course (if any)
		Lecture	Tutorial	Practical/ Practice		
DSC 03 Mathematics for computing	**4**	**3**	**0**	**1**	Pass in Class XII	Pass in Class XII

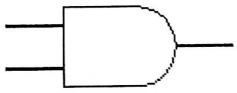
Lecture 1.

logic gate (AND, OR, XOR, NOT, NAND, NOR and XNOR)

A logic gate is an elementary building block of a digital circuit. Most logic gates have two inputs and one output. At any given moment, every terminal is in one of the two binaryconditions *low* (0) or *high* (1), represented by different voltage levels. The logic state of a terminal can, and generally does, change often, as the circuit processes data. In most logic gates, the low state is approximately zero volts (0 V), while the high state is approximately five volts positive (+5 V).

There are seven basic logic gates: AND, OR, XOR, NOT, NAND, NOR, and XNOR.

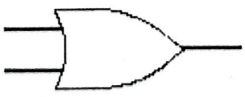
The *AND gate* is so named because, if 0 is called "false" and 1 is called "true," the gate acts in the same way as the logical "and" operator. The following illustration and table show the circuit symbol and logic combinations for an AND gate. (In the symbol, the input terminals are at left and the output terminal is at right.) The output is "true" when both inputs are "true." Otherwise, the output is "false."



AND gate

Input 1	Input 2	Output
	1	
1		
1	1	1

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."



OR gate

Input 1	Input 2	Output
	1	1
1		1
1	1	1

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." Another way of looking at this circuit is to observe that the output is 1 if the inputs are different, but 0 if the inputs are the same.

XOR gate

Input 1	Input 2	Output
	1	1
1		1
1	1	

A logical *inverter* , sometimes called a *NOT gate* to differentiate it from other types of electronic inverter devices, has only one input. It reverses the logic state.

Inverter or NOT gate

Input	Output
1	
	1

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

NAND gate

Input 1	Input 2	Output
		1
	1	1
1		1
1	1	

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

NOR gate

Input 1	Input 2	Output
		1
	1	
1		
1	1	

The *XNOR (exclusive-NOR) gate* is a combination XOR gate followed by an inverter. Its output is "true" if the inputs are the same, and"false" if the inputs are different.

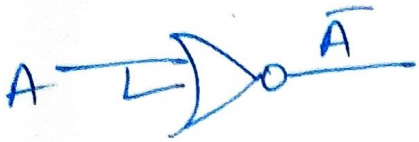
XNOR gate

Input 1	Input 2	Output
		1
	1	
1		
1	1	1

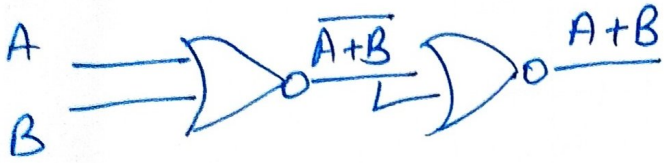
Using combinations of logic gates, complex operations can be performed. In theory, there is no limit to the number of gates that can be arrayed together in a single device. But in practice, there is a limit to the number of gates that can be packed into a given physical space. Arrays of logic gates are found in digital integrated circuits (ICs). As IC technology advances, the required physical volume for each individual logic gate decreases and digital devices of the same or smaller size become capable of performing ever-more-complicated operations at ever-increasing speeds.

Construction of other gates using NOR gate

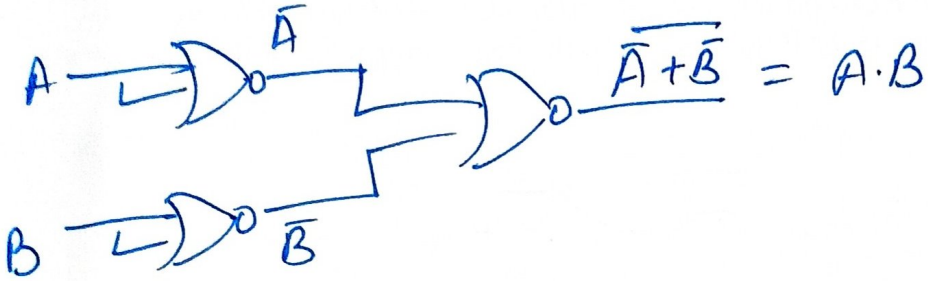
NOT



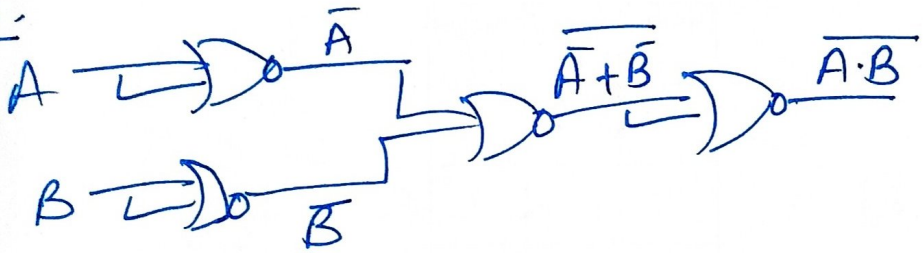
OR



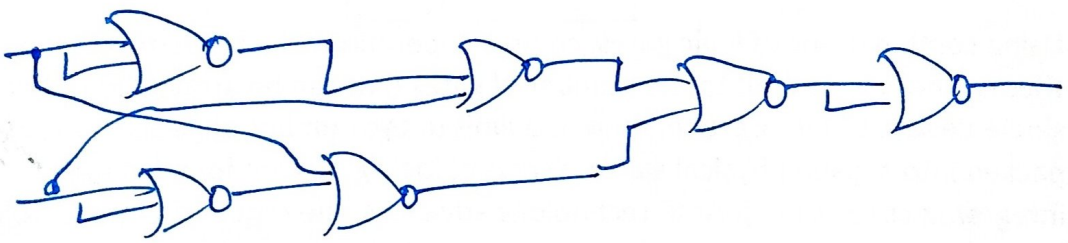
AND



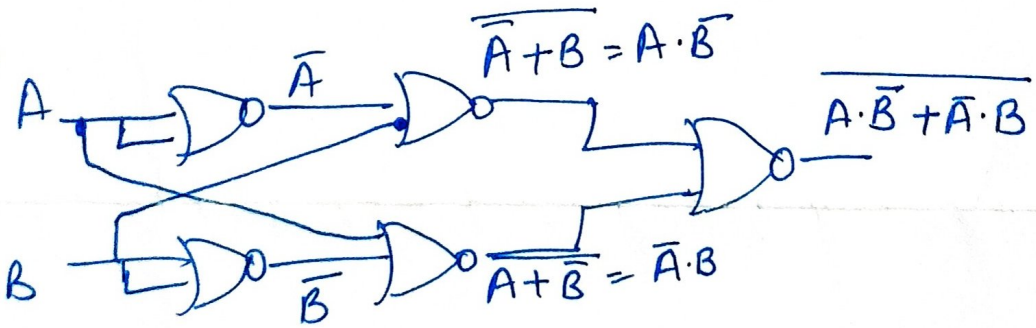
NAND



XOR



XNOR

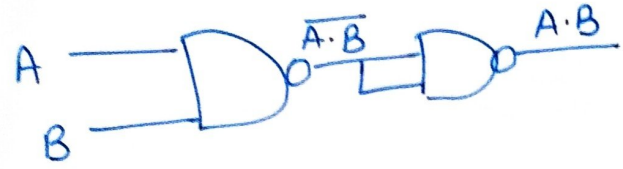


Input 1	Input 2	Output
		1
	1	
1		

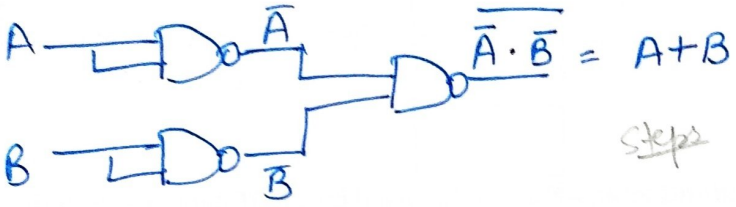
Construction of other gates using NAND gate

NOT

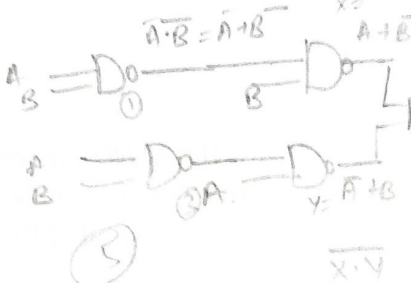
AND



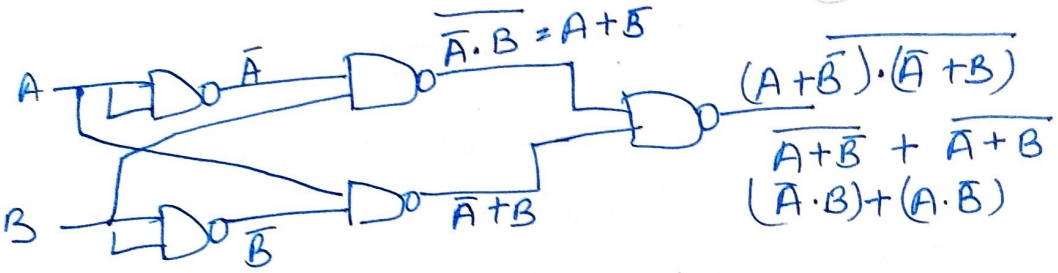
OR



Steps

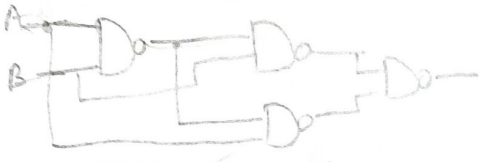


XOR

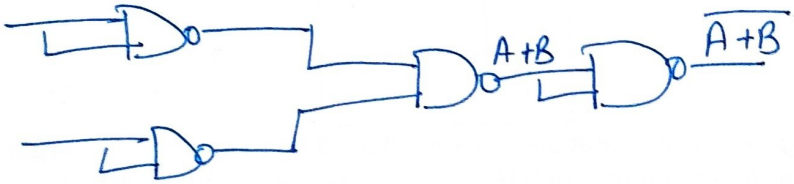


$\overline{A+\bar{B}} + \overline{\bar{A}+B}$

$(\bar{A} \cdot B) + (A \cdot \bar{B})$



NOR



XNOR