RASHTRASANT TUKADOJI MAHARAJ NAGPUR UNIVERSITY, NAGPUR

FOUR YEAR BACHELOR OF TECHNOLOGY (B.Tech.) DEGREE COURSE

SEMESTER: THIRD (C.B.C.S)

BRANCH: INFORMATION TECHNOLOGY

Subject Name: Digital Electronics & Fundamental of Microprocessor **Subject code: BTIT303P**

Load	Practical	Credits	College Assessment Marks	University Evaluation	Total Marks
2Hrs (Practical)	2	1	25	25	50

Aim: To prepare a syllabus of subject Digital Electronics & Fundamental of Microprocessor Practical for the students of Information Technology branch.

Prerequisite(s): Nil

Course Objectives: (02-04)

1	To provide the students the knowledge regarding the various types of devices used combinational and Sequential Circuits.
2	To provide the students the knowledge regarding the various types of Flip-flops and Counter in sequential circuits.
3	The course will provide students with fundamentals concepts of microprocessor 8085.

Course Outcome:(4)

At the end of this course students are able to:

CO1	Apply the basic concept of logic gates and their use in combinational and sequential circuits.
CO2	Use and implements Universal logic gates.
CO3	Design and implement basic circuits required in computer system.
CO4	Develop and execute assembly language programs.

Note:

1. Practicals are based on Digital Electronics and Fundamentals of Microprocessor syllabus (subject code: BEIT303T)
2. There should be at the most two Practicals per unit
3. Minimum ten Practicals have to be performed
4. Do not include study experiments

K. D. K. COLLEGE OF ENGINEERING, NAGPUR

IMPARTING TECHNICAL EDUCATION SINCE 1984

NBA Accredited - Five programs:(Civil Engg., Mechanical Engg., Electrical Engg. Comp. Sc. & Engg., Electronics & Telecomm. Engg.) Grade A, NAAC Accredited

---

Department of Information Technology
Semester: III

Name of Subject: Digital Electronics & Fundamental of Microprocessor

Subject Code: BTIT303P
Course Outcome:

At the end of this course students are able to:

CO1	Apply the basic concept of logic gates and their use in combinational and sequential circuits.
CO2	Use and implements Universal logic gates.
CO3	Design and implement basic circuits required in computer system.
CO4	Develop and execute assembly language programs.

LIST OF PRACTICALS

1	To verify the basic logic gates using diodes and transistors.
2	To study and verify Dem organ's theorem.
3	To verify Boolean expression using logic gates.
4	Implementation of ex-or and ex-nor gate using nand gate only.
5	Implementation of ex-or and ex-nor gate using nor gate only
6	To construct half adder, half subtractor, full adder, full subtractor using and, or, not and ex-or gates and verify the truth table.
7	To study& perform r-s & clocked r-s flip-flop, d & clocked d flip flop.
8	To study & perform j-k flip flop
9	To verify the principle of mod-n counter.
10	To study architecture of 8085.
11	Write an assembly language program to multiply two 8 bit data.
12	Write an assembly language program to multiply two 8 bit numbers stored and store the result in memory.

Prof. S.S. Ganorkar
Subject Teacher

Dr.S.P. Khandait
Head of the Department

PRACTICAL NO 1

Aim: To Study basic Logic Gates and verify its truth table.

Theory: AND, OR and NOT gates are basic gates. XOR and XNOR are universal gates. Basically logic gates are electronic circuits because they are made up of number of electronic devices and components. Inputs and outputs of logic gates can occur only in two levels. These two levels are term HIGH and LOW, or TRUE and FALSE, or ON AND off, OR SIMPLY 1 AND 0. A table which lists all possible combinations of input variables and the corresponding outputs is called a „truth table‟. It shows how the logic circuit‟s output responds to various combinations of logic levels at the inputs.

AND GATE: **-** An AND gate has two or more inputs but only one output. The output assumes the logic 1 state only when each one of its inputs is at logic 1 state. The output assumes logic 0 state even if one of its input is at logic 0 state. AND gate is also called an „all or nothing‟ gate. The logic symbol & truth table of two input AND gate are shown in figure 1.a & 1.b respectively. The symbol for AND operation is „.‟. With input variables A & B the Boolean expression for output can be written as;

$$X = A.B$$

NOT GATE A NOT gate is also known an inverter, has only one input and only one output. It is a device whose output is always the complement of its input. That is the output of a not gate assumes the logic 1 state when its input is in logic 0 state and assumes the logic 0 state when its input is in logic 1 state. The logic symbol & truth table of NOT gate are shown in figure 1.e & 1.f respectively. The symbol for NOT operation is A (bar).

Program: (Students are required to draw symbols and its truth table)

RESULT: Thus we have studied basic Logic Gates and its truth table**. CONCLUSION:**

COURSE OUTCOME	**PROGRAM OUTCOME Attained**
CO1	**PO(1,2,3,4,5,6,7,8,9,10,11,12)**

Viva Questions:

What are logic gates?

Name the basic logic gates.

What is the difference between a digital and an analog circuit?

What is the significance of logic gates in digital circuits? What is a truth table

PRACTICAL NO 2

Aim: To verify the Boolean expression using logic gates.
Apparatus: Logic gates trainer kit, logic gates / ICs, patch cords.
THEORY:

Boolean Algebra Operations

The basic operations of Boolean algebra are as follows:

- Conjunction or AND operation
- Disjunction or OR operation
- Negation or Not operation

Laws of Boolean Algebra

There are six types of Boolean algebra laws. They are:

- Commutative law
- Associative law
- Distributive law
- AND law
- OR law
- Inversion law

Those six laws are explained in detail here.

Commutative Law

Any binary operation which satisfies the following expression is referred to as a commutative operation. Commutative law states that changing the sequence of the variables does not have any effect on the output of a logic circuit.

 A. B = B. A A
+ B = B + A

Associative Law

It states that the order in which the logic operations are performed is irrelevant as their effect is the same.

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

Distributive Law

Distributive law states the following conditions:

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

AND Law

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

- A .0 = 0
- A . 1 = A
- A. A = A
-

OR Law

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

- A + 0 = A
- A + 1 = 1
- A + A = A
-

Inversion Law

In Boolean algebra, the inversion law states that double inversion of variable results in the original variable itself.

Program: (Students are required to develop expression for the given problem)

RESULT: Thus we have studied and verify the Boolean expression using logic gates
CONCLUSION:

COURSE OUTCOME	PROGRAM OUTCOME Attained
CO1	PO(1,2,3,4,5,6,7,8,9,10,11,12)

Viva Questions:

What is Boolean algebra, and why is it important in digital circuits? How do logic gates help in verifying Boolean expressions?
What is the significance of De Morgan's Theorems in digital design?
Can you explain the process of simplification using Boolean laws? What are the universal gates? Why are they called universal?

PRACTICAL NO 3

Aim: To study and verify DeMorgan's Theorm.

OBJECTIVES: - To Verify De-Morgan's laws Required Components and Equipment: -

- **IC numbers 7400, 7402, 7404, 7408,743 BreadBoard.**
- **Connecting Wires**

Theory:

Boolean algebra has postulates and identities. We can often use these laws to reduce expressions or put expressions in to a more desirable form. One of these laws is the De- Morgan's law. De-Morgan's law has two conditions, or conversely, there are two laws called De-Morgan's Laws.

First Condition or First law: The compliment of the product of two variables is equal to the sum of the compliment of eachvariable. Thus according to De-Morgan's laws or De-Morgan's theorem if A and B are the two variablesor Boolean numbers. Then accordingly

$$\overline{A.B} = \overline{A} + \overline{B}$$

Second Condition or Second law: The compliment of the sum of two variables is equal to the product of the compliment of eachvariable. Thus according to De Morgan's theorem if A and B are the two variables then.

$$\overline{A + B} = \overline{A} . \overline{B}$$

PROCEDURE: -

1 Patch the left hand side circuit for the first condition of De-Morgan's Law on the Digital Electronics trainer. Connect the inputs to the input switches and output to the LED and Verify the truth table for all the combinations

2. Similarly, patch the right hand side circuit for the first condition of De-Morgan's Law on the Digital electronics trainer. Connect the inputs to the input switches and output to the LED and verify the truth table for all the combinations. Both the truth tables should be similar.

3. Repeat steps 1 and 2 for the second condition of De-Morgan's Law. 4. Get it checked by the instructor.

RESULT: Thus we have studied and and verify DeMorgan's Theorm.

CONCLUSION:

COURSE OUTCOME	PROGRAM OUTCOME Attained
CO1	PO(1,2,3,4,5,6,7,8,9,10,11,12)

Viva Questions:

What is De Morgan's Theorem?

Why is De Morgan's Theorem important in digital logic design?

How does De Morgan's Theorem help in simplifying Boolean expressions? What is the duality principle in Boolean algebra?

Can De Morgan's Theorem be applied to more than two variables?

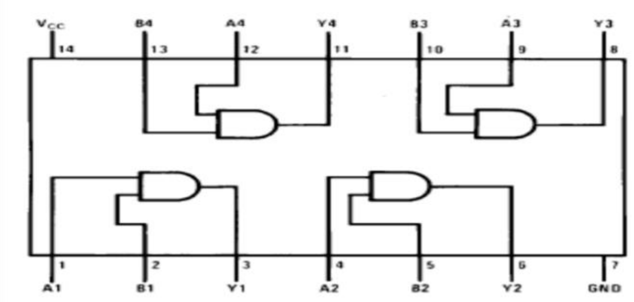
PRACTICAL NO 4

Aim: To study verify the pin configuration of IC 7404, IC 7432, IC 7408, IC 7486, IC 7400, IC 7402, IC 7427

Components: IC 7400, 7402, 7404,7408,7432,7486 Apparatus: Prototyping board (breadboard), DC Power Supply, Connecting Wires Theory:

AND, OR and NOT gates are basic gates. XOR and XNOR are universal gates. Basically, logic gates are electronic circuits because they are made up of number of electronic devices and components. Inputs and outputs of logic gates can occur only in two levels. These two levels are term HIGH and LOW, or TRUE and FALSE, or ON AND off, OR SIMPLY 1 AND 0. A table which lists all possible combinations of input variables and the corresponding outputs is called a „truth table". It shows how the logic circuits output responds to various combinations of logic levels at the inputs

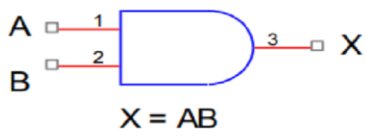
Pin diagram of IC**74LS08**:



AND GATE:-

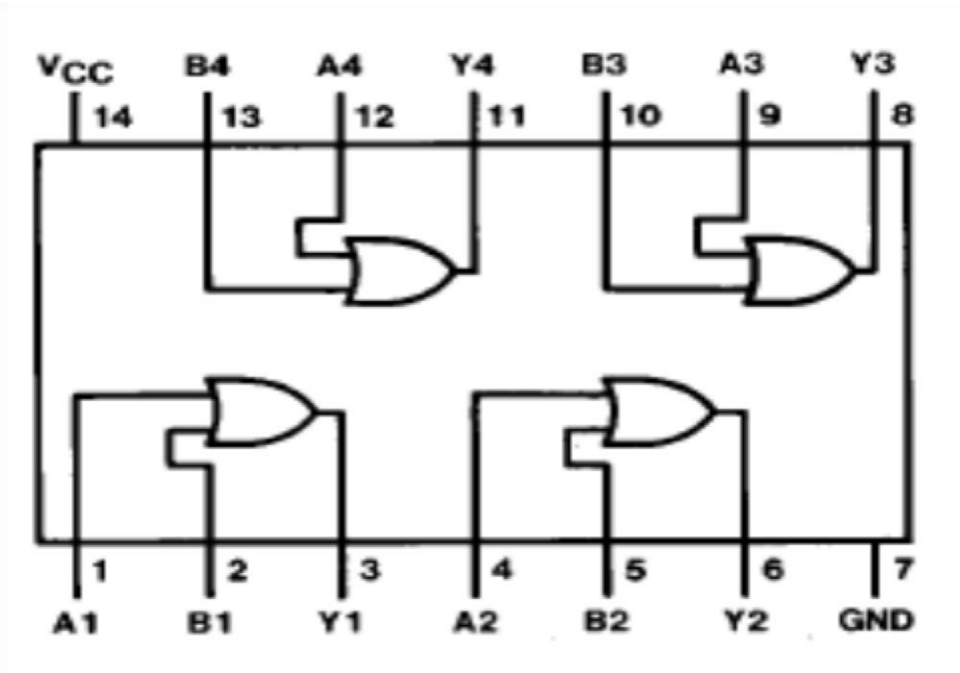
$$\mathbf{X = A.B}$$

Logic symbol:



Truth table:

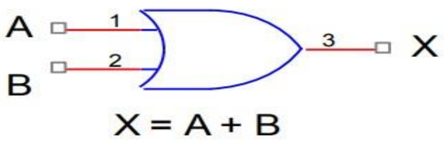
Input		Output
A	**B**	**X**
0	0	0
0	1	0
1	0	0
1	1	1



OR GATE

X = A + B

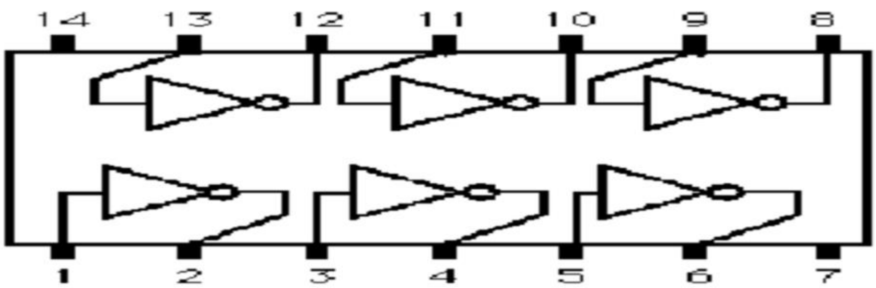
Logic symbol:



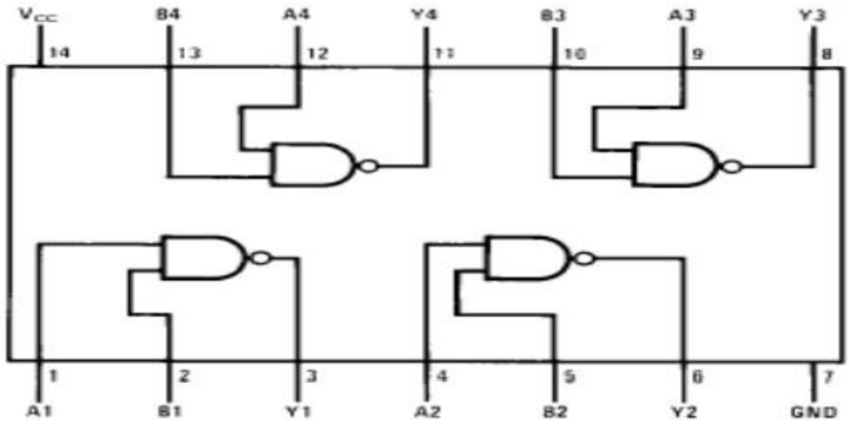
Truth table:

Input		Output
A	B	X
0	0	0
0	1	1
1	0	1

Pin diagram for IC**74LS04**:



Pin diagram for IC**74LS00**:



NAND GATE

RESULT: Thus we have studied the pin configuration of IC 7404, IC 7432, IC 7408, IC 7486, IC 7400, IC 7402, IC 7427

CONCLUSION:

CO	PO Attained
CO1	**PO(1,2,3,4,5,6,7,8,9,10,11,12)**

Viva Questions:

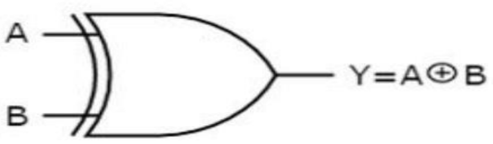
What are logic ICs, and why are they used in digital circuits?
What is the significance of the Vcc and GND pins in an IC? What does DIP (Dual In-Line Package) mean?
How do you verify the working of an IC using a truth table?
What precautions should be taken while handling logic gate ICs?

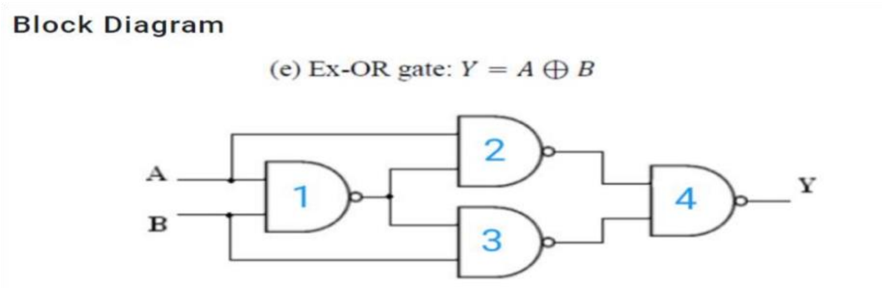
PRACTICAL NO 5

Aim: Implementation of AND, OR, NOT, EX-OR and X-NOR by using NOR Gate and NAND Gate only

Theory: Ex-OR GATE - The full form of Ex-OR gate is Exclusive-OR gate. Its function is the same as that of OR gate except for some cases, when the inputs having an even number of ones.
In other words, the output of an Exclusive-OR gate ONLY goes "HIGH" when its two input terminals are at "DIFFERENT" logic levels with respect to each other. The Exclusive-OR Gate function, or Ex-OR for short, is achieved by combining standard logic gates together to form more complex gate functions that are used extensively in building arithmetic logic circuits, computational logic comparators and error detection circuits.The two-input "Exclusive-OR" gate is basically a modulo two adder, since it gives the sum of two binary numbers.

Boolean Expression **Y = A.B' + 'A.B = (A ⊕ B)**





A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Program: (Students are required to develop Logic for the given problem)

RESULT: Thus we have studied EX-OR and X-NOR by using Gate and NAND Gate only

CONCLUSION:

CO	PO Attained
CO1	PO(1,2,3,4,5,6,7,8,9,10,11,12)

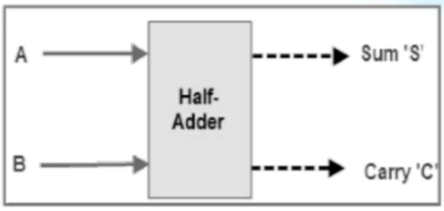
Viva Queations:

What are universal gates? Why are NAND and NOR called universal gates?
Can you implement all logic gates using only NAND or NOR gates?
How do you implement a NOT gate using only NAND and NOR gates?
How can we convert NAND and NOR gates into basic gates?
Which gate is more efficient for implementation: NAND or NOR? Why?

PRACTICAL NO 6

Aim: Construct Half adder Full adder by using AND, OR, NOT, EX-OR, Gate and verify truth Table. **Theory: 1)Half Adder**

Half adder is a combinational circuit that performs simple addition of two binary numbers. If we assume A and B as the two bits whose addition is to be performed, the block diagram and a truth table for half adder with A, B as inputs and Sum, Carry as outputs can be tabulated as follows.



Truth Table			
Input		Output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

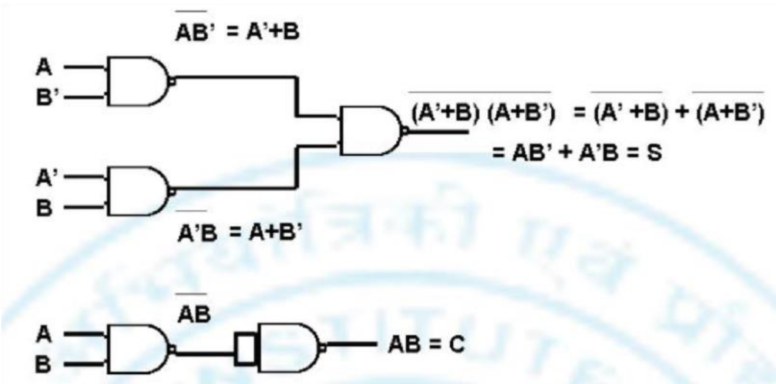
Block diagram and truth table of half adder

RESULT: Thus we have studied and performed C program to perform 3*3 matrix multiplication.

Half Adder using NAND gates:

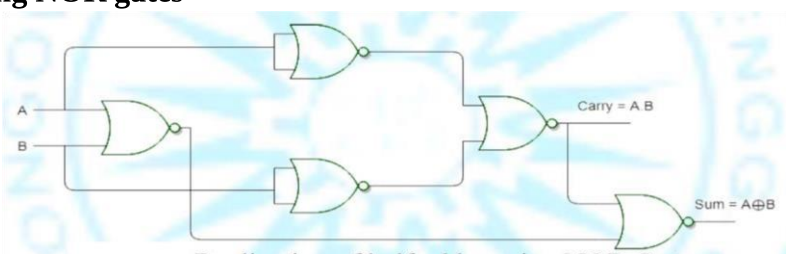
Five NAND gates are required in order to design a half adder. The circuit to realize half adder using NAND gates is shown below.

Half Adder using NOR gates



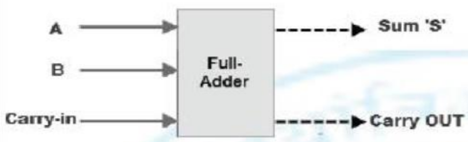
Realization of half adder using NAND gates

Half Adder using NOR gates



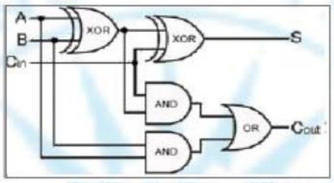
Realization of half adder using NOR Gates

Full Adder



Input			Output	
A	B	Cin	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 8. Full Adder Block Diagram and Truth Table



Full Adder Logic Diagram

Full Adder using NAND gates:
As mentioned earlier, a NAND gate is one of the universal gates and can be used to implement any logic design. The circuit of full adder using only NAND gates is shown below.

RESULT: Thus we have Half adder Full adder by using AND, OR, NOT, EX-OR, Gate and verified truth Table

CONCLUSION:

COURSE OUTCOME	PROGRAM OUTCOME Attained
CO2	PO(1,2,3,4,,8,9,10,12)

Viva Questions:

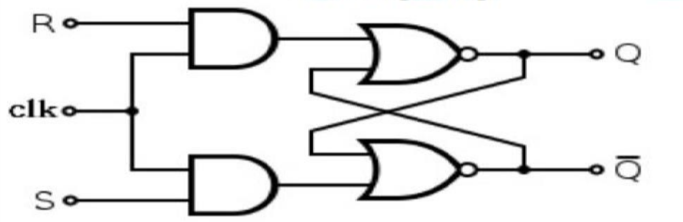
What is the difference between a Half Adder and a Full Adder?
Why does a Half Adder not account for a carry-in (Cin)?
What is the significance of XOR gates in an adder circuit?
How does a Full Adder handle carry propagation?
What is the Boolean expression for the Sum and Carry of a Half Adder?

PRACTICAL NO 7

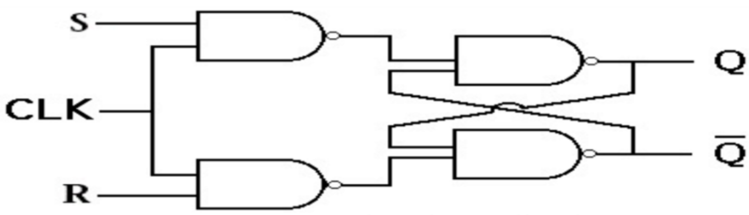
Aim: To Study SR and Clocked SR Flip Flop and T Flip Flop and verify its Truth Table

Theory: The simplest kind of sequential circuit which is capable of storing one bit of information is called latch. The operation of basic latch can be modified, by providing an additional control input that determines when the state of the circuit is to be changed. The latch with additional control input is called the Flip-Flop. The additional control input is either the clock or enable input.

S-R Flip-Flop



Clocked NOR-based S-R Flip-Flop



NAND-based S-R Flip-Flop

S	R	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Not Used

S-R Flip-Flop characteristic Table

RESULT: Thus we have studied SR and Clocked SR Flip Flop

CONCLUSION:

COURSE OUTCOME	PROGRAM OUTCOME Attained
CO2	PO(1,2,3,4,,8,9,10,12)

Viva Questions:

What are the real-life applications of flip-flops?
How are flip-flops used in memory circuits and registers?
What is the role of a T flip-flop in frequency division circuits?
How do we avoid invalid states in an SR flip-flop?
What is edge triggering, and why is it important in flip-flop

PRACTICAL NO 8

Aim: To Study the JK Flip Flop and D Flop and verify its truth table

Theory: Sequential Circuits: The logic circuits whose outputs at any instant of time depend not only on the present input but also on the past outputs are called sequential circuits. The simplest kind of sequential circuit which is capable of storing one bit of information is called latch. The operation of basic latch can be modified, by providing an additional control input that determines when the state of the circuit is to be changed. The latch with additional control input is called the Flip-Flop.

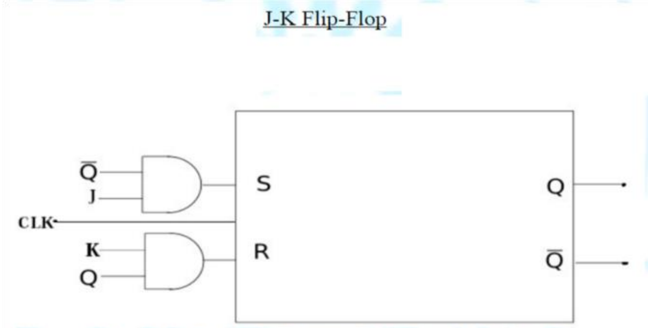
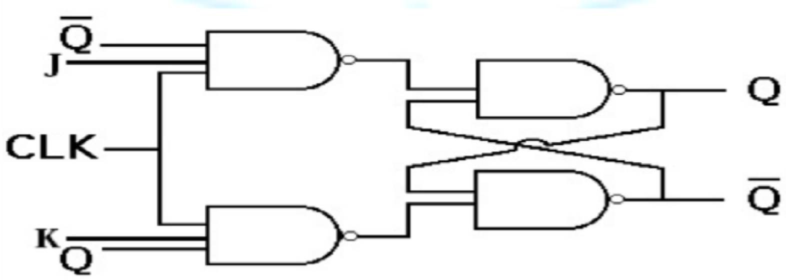


Figure 1: J-K Flip-Flop using S-R Flip-Flop

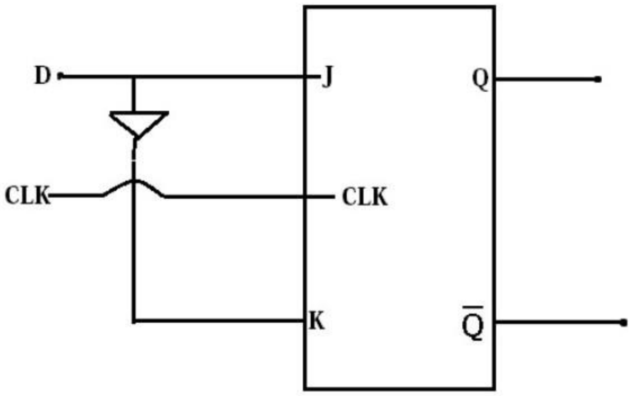


NAND based J-K Flip-Flop

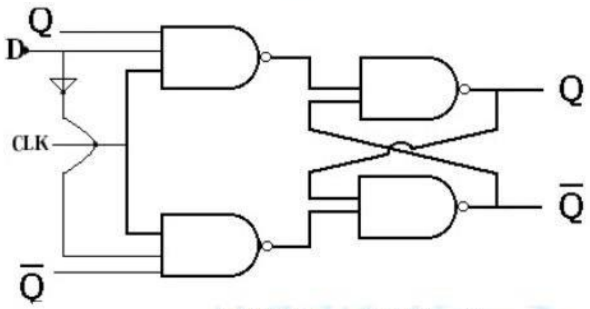
J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	$\overline{Q}$ (t)

: J-K Flip-Flop characteristic Table

D Flip-Flop



: D Flip-Flop



NAND-based D Flip-Flop

D	Q(t+1)
0	1
1	1

: D Flip-Flop characteristic Table

RESULT: Thus we Flip Flop and D Flop and have studied and JK and verified its truth table

CONCLUSION:

COURSE OUTCOME	PROGRAM OUTCOME Attained
CO3	PO(1,2,3,4,5,8,9,10,12)

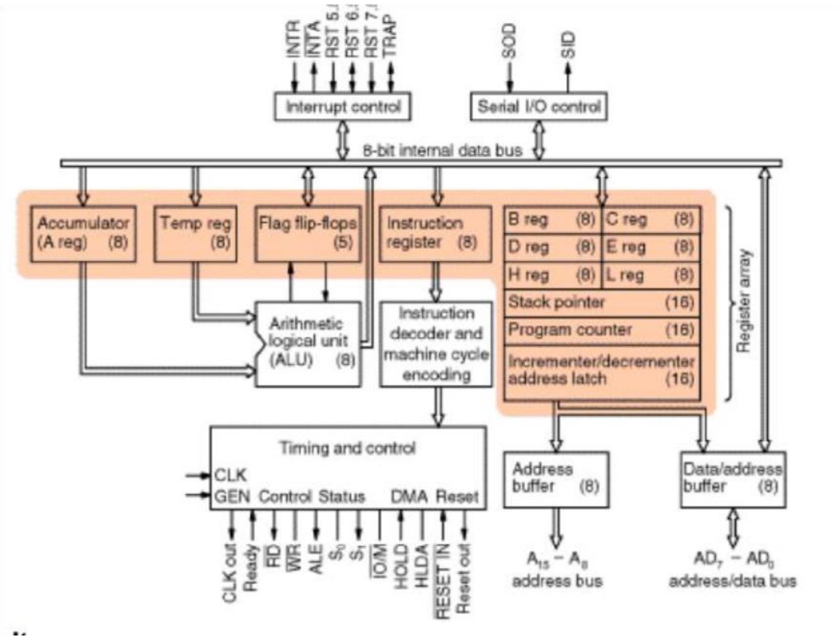
Viva Questions:

What are the real-world applications of JK and D Flip-Flops? How is a D Flip-Flop used in registers and memory circuits?
What is edge triggering, and why is it important in flip-flops? How can a JK Flip-Flop be used as a frequency divider?
What is the role of flip-flops in digital counters and shift registers?

PRACTICAL NO 9

Aim: To study Architecture of 8085 Pin configuration

Theory:



Control Unit Generates signals within uP to carry out the instruction, which has been decoded. In reality causes certain connections between blocks of the uP to be opened or closed, so that data goes where it is required, and so that ALU operations occur. Arithmetic Logic Unit The ALU performs the actual numerical and logic operation such as ‘add’, ‘subtract’, ‘AND’, ‘OR’, etc. Uses data from memory and from Accumulator to perform arithmetic. Always stores result of operation in Accumulator

Registers The 8085/8080A-programming model includes six registers, one accumulator, and one flag register, as shown in Figure. In addition, it has two 16-bit registers: the stack pointer and the program counter. They are described briefly as follows. The 8085/8080A has six general-purpose registers to store 8-bit data; these are identified as B,C,D,E,H, and L as shown in the figure. They can be combined as register pairs - BC, DE, and HL - to perform some 16bit operations. The programmer can use these registers to store or copy data into the registers by using data copy instructions. Accumulator

The accumulator is an 8-bit register that is a part of arithmetic/logic unit (ALU). This register is used to store 8-bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator. The accumulator is also identified as register A. Flags The ALU includes five flip-flops, which are set or reset after an operation according to data conditions of the result in the accumulator and other registers. They are called Zero(Z), Carry (CY), Sign (S), Parity (P), and Auxiliary Carry

(AC) flags; they are listed in the Table and their bit positions in the flag register are shown in the Figure below. The most commonly used flags are Zero, Carry, and Sign.

The microprocessor uses these flags to test data conditions. For example, after an addition of two numbers, if the sum in the accumulator id larger than eight bits, the flip-flop uses to indicate a carry - - called the Carry flag (CY) – is set to one. When an arithmetic operation results in zero, the flip-flop called the Zero(Z) flag is set to one. The first Figure shows an 8-bit register, called the flag register, adjacent to the accumulator. However, it is not used as a register; five bit positions out of eight are used to store the outputs of the five flip-flops.

The flags are stored in the 8-bit register so that the programmer can examine these flags (data conditions) by accessing the register through an instruction These flags have critical importance in the decision-making process of the microprocessor. The conditions (set or reset) of the flags are tested through the software instructions. For example, the instruction JC (Jump on Carry) is implemented to change the sequence of a program when CY flag is set. The thorough understanding of flag is essential in writing assembly language programs. Program Counter (PC) This 16-bit register deals with sequencing the execution of instructions. This register is a memory pointer.

Memory locations have 16-bit addresses, and that is why this is a 16-bit register. The microprocessor uses this register to sequence the execution of the instructions. The function of the program counter is to point to the memory address from which the next byte is to be fetched. When a byte (machine code) is being fetched, the program counter is incremented by one to point to the next memory location Stack Pointer (SP) The stack pointer is also a 16-bit register used as a memory pointer. It points to a memory location in R/W memory, called the stack. The beginning of the stack is defined by loading 16-bit address in the stack pointer. The stack concept is explained in the chapter "Stack and Subroutines." Instruction Register/Decoder Temporary store for the current instruction of a program. Latest instruction sent here Microprocessor 8085 from memory prior to execution. Decoder then takes instruction and 'decodes' or interprets the instruction. Decoded instruction then passed to next stage. Memory Address Register Holds address, received from PC, of next program instruction. Feeds the address bus with addresses of location of the program under execution.

Control Generator Generates signals within uP to carry out the instruction which has been decoded. In reality causes certain connections between blocks of the uP to be opened or closed, so that data goes where it is required, and so that ALU operations occur.

Result: Thus the study of Architecture of 8085 microprocessor is completed.

CONCLUSION:

COURSE OUTCOME	PROGRAM OUTCOME Attained
CO4	PO(1,2,3,4,5,8,9,10,12)

Viva Questions:

What is the role of the Program Counter (PC)?

What is the function of the Stack Pointer (SP)?

What are the different types of interrupts in 8085?

What is the difference between TRAP and RST 7.5 interrupts?

What is the clock frequency of 8085?

What is the function of X1 and X2 pins?

PRACTICAL NO 10

Aim: To Study The Instruction Set Of 8085.Write A Program To Add Two Hexadecimal & Decimal Numbers.

Theory:

Instruction and Data Formats

The various techniques to specify data for instructions are:

1. 8-bit or 16-bit data may be directly given in the instruction itself.
2. The address of the memory location, I/O port or I/O device, where data resides, may be given in the instruction itself.
3. In some instructions, only one register is specified. The content of the specified register is one of the operands.
4. Some instructions specify two registers. The contents of the registers are the required data.
5. In some instructions, data is implied. The most instructions of this type operate on the content of the accumulator.

Due to different ways of specifying data for instructions, the machine codes of all instructions are not of the same length. It may 1-byte, 2-byte or 3-byte instruction.

Addressing Modes

Each instruction requires some data on which it has to operate. There are different techniques to specify data for instructions. These techniques are called **addressing modes**. Intel 8085 uses the following addressing modes:

- **Direct Addressing** ○ **Register Addressing** ○ **Register Indirect Addressing**

In Register Indirect mode of addressing, the address of the operand is specified by a register pair.

Example

- **LXI H, 2500 H** - Load H-L pair with 2500H.
- **MOV A, M** - Move the content of the memory location, whose address is in H-L pair (i.e. 2500 H) to the accumulator.
- **HLT** - Halt.

APPARATUS REQUIRED: -

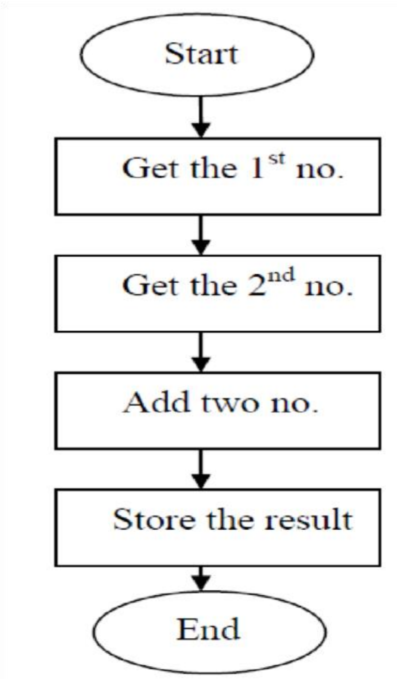
Sr. no.	Name of equipments/components/software	Specification/range/rating/versi on	Quantity
1	8085 Microprocessor programming kit, instruction coding sheet.	SCIENTECH-8085	1
2.	Power supply	A.C (230V Mains)	

Hexadecimal Addition: The program takes the content of 2009, adds it to 200B & stores the result back at 200C.

Steps: 1. Initialize HL Reg. pair with address where the first number is lying.
2. Store the number in accumulator.
3. Get the second number.
4. Add the two numbers and store the result in 200B.
5. Go back to Monitor

Let: (2009 H) = 80 H
(200B H) = 15 H
Result = 80 H + 15 H = 95 H

(2009 H) → A
A → B
(200B H) → A
A + B → A
A → (200C H)



FLOWCHART : -

PROGRAM:-

```
LXI H, 2009 ;        Point 1st no.
MOV A, M    ;        Load the acc.
INX H       ;        Adv Pointer
ADD M       ;        ADD 2nd NO.
INX H       ;        Adv Pointer
MOV M, A    ;        Store Result
RST 5
```

Decimal Addition:

Steps: 1. Initialize HL Reg. pair with address where the first number is lying.

2. Store the number in accumulator.
3. Get the second number.
4. Add the two numbers and store the result in 200B.
5. Go back to Monitor

REULTS:-

Thus the numbers at 2009H and at memory are added.

CONCLUSION:-

Thus the program to add two 8-bit numbers was execuated

Viva Questions:

What is the purpose of an instruction set in 8085?
What are the different types of instructions in 8085?
How many bits does the 8085 microprocessor process at a time?
What is the difference between MOV, MVI, and LXI instructions?
What is the difference between ADD and ADC instructions?

PRACTICAL NO 11

Aim: Write an assembly language program to multiply two 8 bit data.

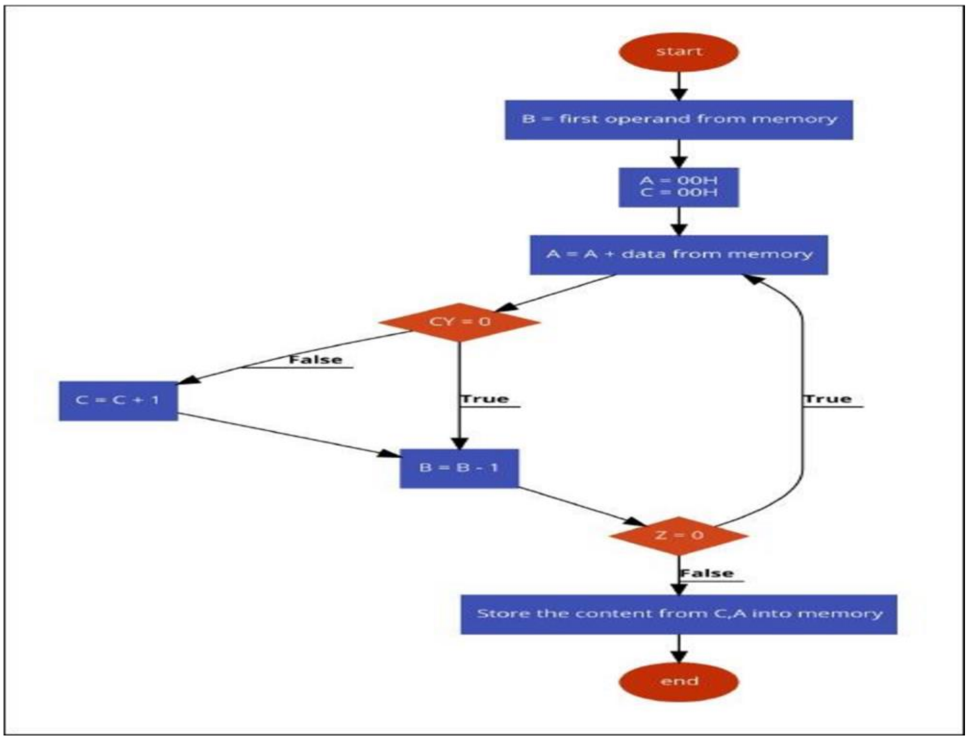
Theory: The 8085 has no multiplication operation. To get the result of multiplication, we should use the repetitive addition method.After multiplying two 8-bit numbers it may generate 1-byte or 2-byte numbers, so we are using two registers to hold the result.

We are saving the data at location 8000H and 8001H. The result is storing at location 8050H and 8051H.

Input

Address	Data
.	.
.	.
.	.
8000	DC
8001	AC
.	.
.	.
.	.

Flow Diagram



Program

Address	HEX Codes	Labels	Mnemonics	Comments
F000	21, 00, 80		LXI H,8000H	Load first operand address
F003	46		MOV B, M	Store first operand to B
F004	23		INX H	Increase HL pair
F005	AF		XRA A	Clear accumulator
F006	4F		MOV C, A	Store 00H at register C
F007	86	LOOP	ADD M	Add memory element with Acc
F008	D2, 0C, F0		JNC SKIP	When Carry flag is 0, skip next task
F00B	0C		INR C	Increase C when carry is 1
F00C	05	SKIP	DCR B	Decrease B register
F00D	C2, 07, F0		JNZ LOOP	Jump to loop when Z flag is not 1
F010	21, 50, 80		LXI H,8050H	Load Destination address
F013	71		MOV M, C	Store C register content into memory
F014	23		INX H	Increase HL Pair
F015	77		MOV M, A	Store Acc content to memory
F016	76		HLT	Terminate the

program

Output

Address	Data
. .	
. .	
. .	
8050	**93**
8051	**D0**
. .	
. .	
. .	

Viva Questions:

Does 8085 have a direct multiplication instruction?
How is multiplication performed using 8085?
What is the role of the accumulator in this program?
Why do we use a loop in this program?
What is the difference between DCR and DCX instructions?

PRACTICAL NO 12

Aim: To verify the principle of mod-n counter.

Theory: To verify the principle of a Mod-n counter, you can build or simulate a counter circuit and observe its behavior. A Mod-n counter cycles through 'n' distinct states before resetting to its initial state, and you can verify this by observing the output sequence after each clock pulse.

Understanding Mod-n Counters

- **Definition:** A Mod-n counter is a sequential logic circuit that counts through a predefined sequence of 'n' states and then resets to its starting state.
- **Example:** A Mod-5 counter would count from 0 to 4 and then reset to 0.
- **Modulus:** The number of states a counter cycles through before resetting is called its modulus.
- **Implementation:** Mod-n counters can be built using flip-flops and logic gates.

Verification Steps

1. **Choose a Modulus (n):** Select a value for 'n' (e.g., Mod-3, Mod-8, Mod-10) to test.
2. **Design or Simulate a Circuit:**
   - **Hardware:** You can build a Mod-n counter using flip-flops (like JK or D flipflops) and logic gates (AND, OR, NOT) to implement the counting and reset logic.
   - **Simulation:** Use a digital logic simulator (like Proteus, Logisim, or Multisim) to model the counter circuit.
3. **Apply Clock Pulses:** Feed a clock signal to the counter circuit.
4. **Observe the Output:** Monitor the output states of the counter (the flip-flops' outputs) after each clock pulse.
5. **Verify the Sequence:**
   - **Counting:** Check if the counter increments through the expected sequence of states (0, 1, 2, ... n-1).
   - **Reset:** Observe if the counter resets to its initial state after reaching the 'n-1' state.
   - **Modulus:** Count the number of states the counter cycles through before resetting. This should equal 'n'.
6. **Repeat for Different Values of 'n':** Test with various values of 'n' to ensure the principle works correctly for different moduli.

Example: Verifying a Mod-3 Counter

1. **Modulus:** n = 3 (Mod-3 counter)
2. **Circuit:** Use 2 flip-flops (they can store 2^2=4 states, but we'll use only 3) and logic gates to implement the Mod-3 counter logic.
3. **Simulation/Hardware:** Simulate or build the circuit and apply clock pulses.
4. **Expected Sequence:** The counter should cycle through the states: 00, 01, 10, and then reset back to 00.
5. **Verification:** Observe the output sequence and verify that the counter resets to 00 after reaching state 10, confirming the Mod-3 behavior.

Design Mod – N synchronous Counter

The value of N can be different from power of 2. Also, the counting sequence may be random for example some cyclic code (8421, 2423 etc). The following method is applied for designing for mod N and any counting sequence. **Design for Mod-N counter** :

The steps for the design are –

Step 1 : Decision for number of flip-flops –

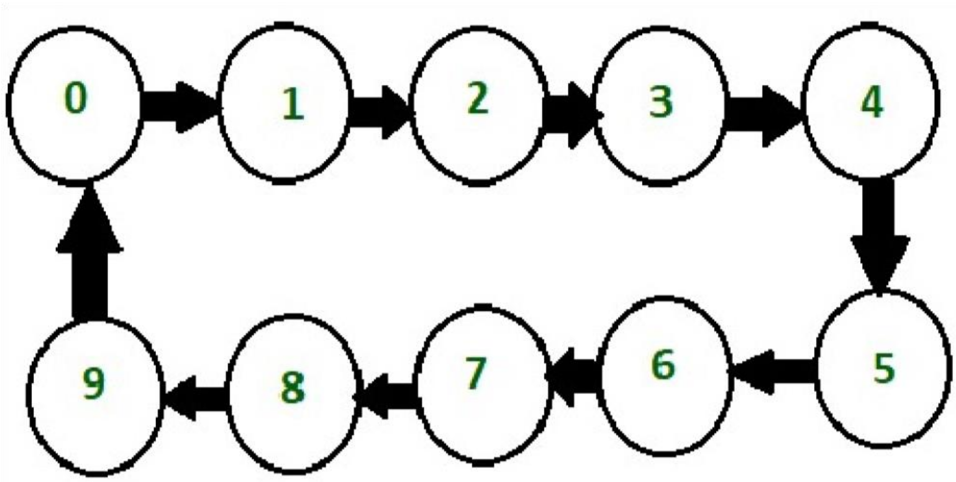
Example : If we are designing mod N counter and n number of flip-flops are required then n can be found out by this equation. $\mathbf{N \le 2^n}$

Here we are designing Mod-10 counter Therefore, N= 10 and number of Flip flops(n) required is **For n =3,** 10<=8, which is false.
For n= 4,10<=16, which is true.
Therefore number of FF required is 4 for Mod-10 counter.

Step 2 : Write excitation table of Flip flops –
Here T FF is used

Previous state(Q_n)	Next state(Q_{n+1})	T
0	0	0
0	1	1
1	0	1
1	1	0

Excitation table of T FF.
Step 3 : Draw state diagram and circuit excitation table –



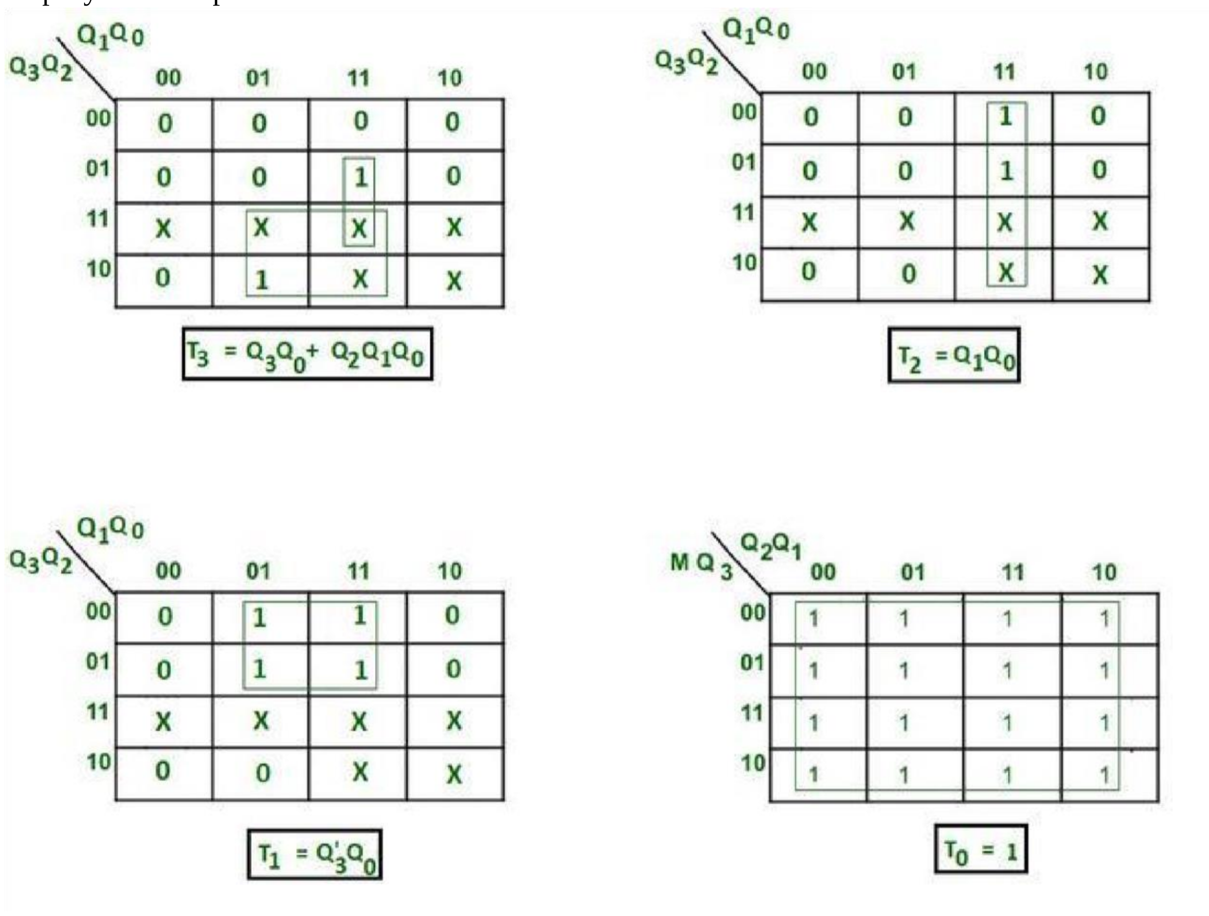
Counting Sequence of Decade counter
A decade counter is called as mod -10 or divide by 10 counter. It counts from 0 to 9 and again reset to 0. It counts in natural binary sequence. Here 4 T Flip flops are used. It resets after Q_3 Q_2 Q_1 Q_0 = 1001.
Circuit excitation table –
Here Q_3 Q_2 Q_1 Q_0 are present states of four flip-flops and Q^*_3 Q^*_2 Q^*_1 Q^*_0 are next counting state of 4 Flip flops. If there is a transition in current state i.e if Q3 value changes from 0 to 1 or 1 to 0 then there's corresponding T(toggle) bit is written as 1 otherwise 0.

Q_3	Q_2	Q_1	Q_0	Q^*_3	Q^*_2	Q^*_1	Q^*_0	T_3	T_2	T_1	T_0
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	1
0	0	1	1	0	1	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	1
0	1	0	1	0	1	1	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	1
0	1	1	1	1	0	0	0	1	1	1	1
1	0	0	0	1	0	0	1	0	0	0	1
1	0	0	1	0	0	0	0	1	0	0	1

Circuit excitation table

Step 4 : Create Karnaugh map for each FF input in terms of flip-flop outputs as the input variable –

Simplify the K map –

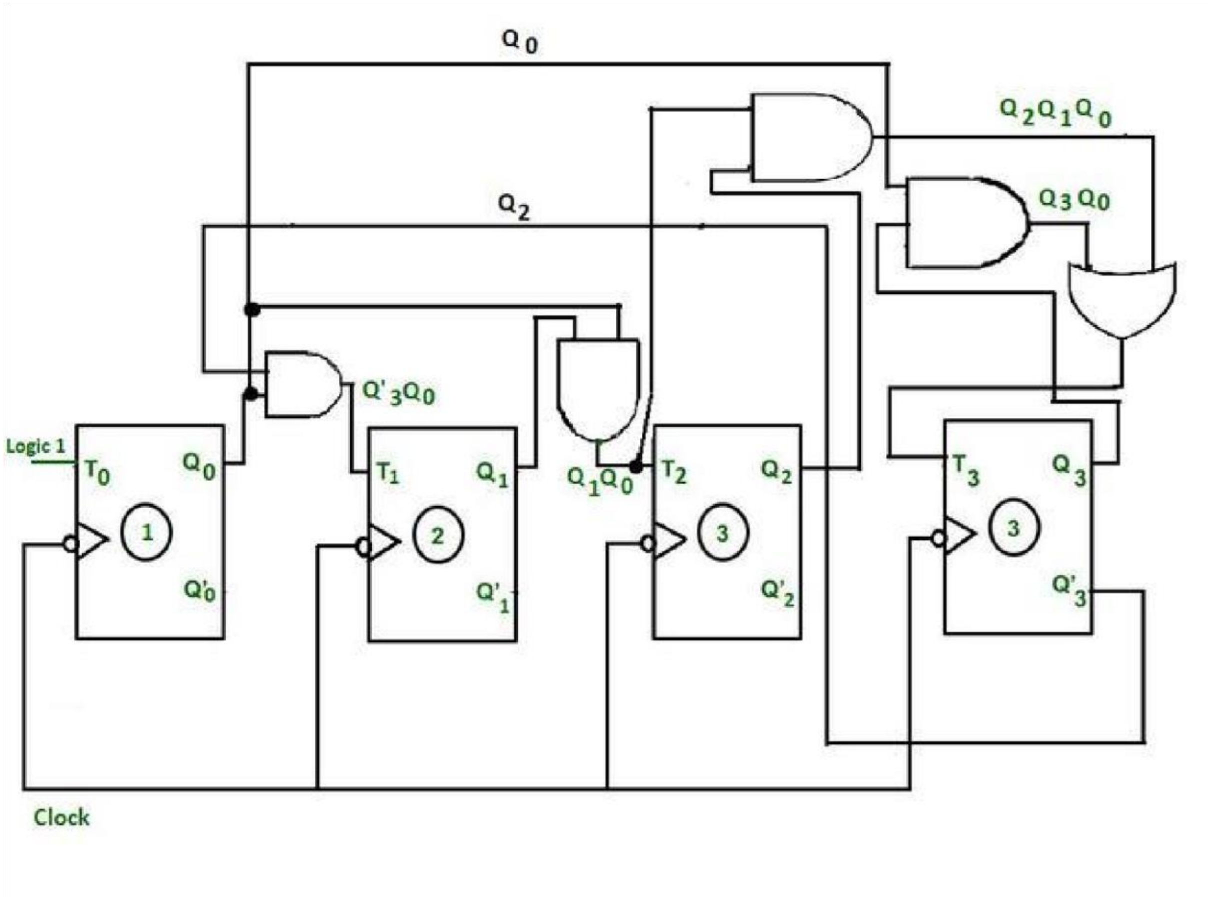


K map for finding minimal expressions.

Step 5 : Create circuit diagram –

Here negative edge triggered clock is used for toggling purpose.

- The clock is provided to every Flip flop at same instant of time.
- The toggle(T) input is provided to every Flip flop according to the simplified equation of K map.



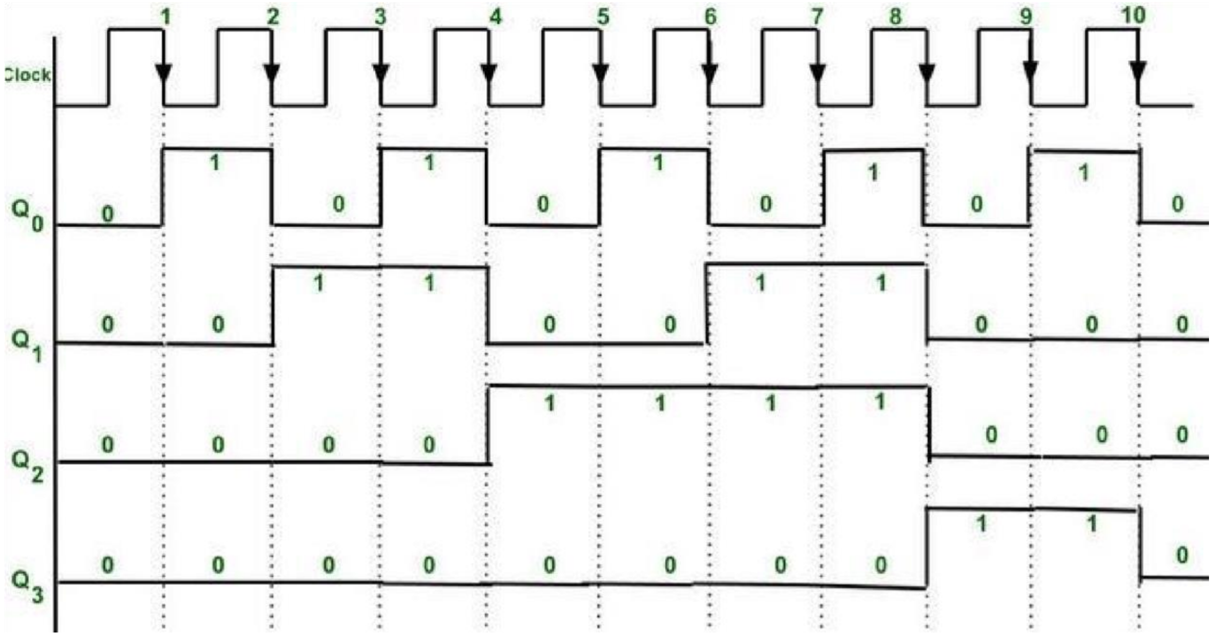
Circuit diagram

Timing diagram : Here toggling is used.

Previous state(Q_n)	T	Next state(Q_{n+1})
0	0	0
0	1	1
1	0	1
1	1	0

Characteristic table of T FF.

The state of a FF will change only when toggle input(T) of a FF is 1.



Timing diagram of synchronous Decade counter

OUTPUT: We get Output as Q_3(MSB) Q_2 Q_1 Q_0(LSB).

After 10th falling edge the output state of all the FFs again becomes 0 0 0 0.

Viva Questions:

What is a counter in digital electronics?
What does Mod-N Counter mean?
How many states does a Mod-6 Counter have?
What is the general formula to determine the number of flip-flops required for a Mod-N Counter?
What is the difference between a synchronous and asynchronous counter?