# DIGITAL CIRCUITS AND EMBEDDED SYSTEMS LAB MANUAL

(As per KTU Syllabus 2015)

Version 2

July 2019

## DEPT. OF ELECTRICAL ENGINEERING

COLLEGE OF ENGINEERING TRIVANDRUM

## PREFACE

This manual is prepared as per the BTech Degree syllabus for Digital Circuits and Embedded Systems Lab in Electrical and Electronics Engineering of A P J Abdul Kalam Technical University. This manual consists of a set of experiments designed to allow students to build, and verify digital circuits, microprocessor and embedded systems. This set of experiments/programs cover relevant topics prescribed in the syllabus and are designed to reinforce the theoretical concepts taught in the classroom with practical experience in the lab.

We take this opportunity to express thanks to Dr. P Sreejaya, Professor and Head of the dept. of Electrical Engineering for her continued interest and encouragement for this work. We are also thankful to all faculty members of Electrical Engineering department for their cooperation in the preparation of this reference record.

This is the revised version of the manual that was prepared in 2017. Every effort has been taken for the correctness of the subject dealt with, suggestions and remarks are welcome.

| | | |
|---|---|---|
| Chief Co-ordinator | : | Dr. Sreejaya P<br>Head of the Department |
| Prepared and compiled by | : | Dr. Lal Priya P S/ Prof. Vivek R S<br>Lab – in – charge |

## DEPARTMENT VISION AND MISSION

### VISION

- Be a centre of excellence and higher learning in Electrical Engineering and allied areas.

### MISSION

- To impart quality education in Electrical Engineering and bring-up professionally competent engineers.
- To mould ethically sound and socially responsible Electrical Engineers with leadership qualities.
- To inculcate research attitude among students and encourage them to pursue higher studies

## Syllabus

***3rd Semester B.Tech (Electrical Engineering)***

Year of Introduction 2016

**EE331 DIGITAL CIRCUITS AND EMBEDDED SYSTEMS LAB 0-0-3**

**DIGITAL CIRCUITS EXPERIMENTS** :

(at least 7 experiments are mandatory)

1. Realisation of SOP & POS functions after K map reduction
2. Half adder & Full adder realization using NAND gates
3. 4-bit adder/subtractor & BCD adder using IC 7483
4. BCD to decimal decoder and BCD to 7-segment decoder & display
5. Study of multiplexer IC and Realization of combinational circuits using multiplexers.
6. Study of counter ICs (7490, 7493)
7. Design of synchronous up, down & modulo N counters
8. Study of shift register IC 7495, ring counter and Johnsons counter
9. VHDL implementation of full adder, 4 bit magnitude comparator

**EMBEDDED SYSTEM EXPERIMENTS**:

(**Out of first six, any two experiments using 8085 and any two using 8086. Out of the last 3 experiments, any two experiments using 8051 or any other open source hardware platforms like PIC, Arduino, MSP430, ARM etc)** ( at least 5 experiment are mandatory)

1. Data transfer instructions using different addressing modes and block transfer.
2. Arithmetic operations in binary and BCD-addition, subtraction, multiplication and division
3. Logical instructions- sorting of arrays in ascending and descending order
4. Binary to BCD conversion and vice versa.
5. Interfacing D/A converter- generation of simple waveforms-triangular wave, ramp etc
6. Interfacing A/D converter
7. Square wave generation.
8. LED and LCD display interfacing
9. Motor control

## Course Outcomes

| | |
|---|---|
| CO1 | Describe basic concepts of digital system components such as basic Gates,Adder, Flip Flops, Counters, Multiplexers, Registers, Decoders |
| CO2 | Verify experimentally SOP & POS solutions, functions 4-bit Adder/Subtractor, 4 bit Counters, 4bit Registers, 8-1 Multiplexers, 4-10 Decoders. |
| CO3 | Develop and execute programmes to perform data transfer, arithmetic / logical operations and code conversions using 8085 |
| CO4 | Interface A/D & D/A CONVERTERS and LED/LCD display to control a motor. |
| CO5 | Design an embedded system for a particular application |

## **Program Educational Objectives (PEOs)**

Graduates will

1. Excel as technically competent Electrical Engineers.

2. Excel in higher studies and build on fundamental knowledge to develop technical skills within and across disciplines.

3. Have an ability to function effectively as members or leaders in technical teams.

4. Adapt to changes in global technological area and social needs through lifelong learning.

## **Program outcomes**

PO1 Apply the knowledge of mathematics, science and engineering fundamentals and an engineering specialization to the solution of complex engineering problems.

PO2 Identify, formulate, review research literature and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences

PO3 Design solutions for complex engineering problems and design system components or processes that meet the specific needs with appropriate consideration for the public health and safety, and the cultural, societal and environmental considerations

PO4 Use research based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5 Create, select and apply appropriate techniques, resources and modern engineering and IT tools including predictions and modelling to complex engineering activities with an understanding of the limitations.

PO6 Apply reasoning informed by the contextual knowledge to assess social, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7 Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of and need for sustainable development.

PO8 Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9 Function effectively as an individual and as a member or leader in diverse teams and in multidisciplinary settings.

PO10 Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentation and give and receive clear instructions.

PO11 Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s work, as a member and leader in a team to manage projects and multidisciplinary environments.

PO12 Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

# CONTENTS

# I. DIGITAL CIRCUITS LAB EXPERIMENTS

# 1. FAMILIARISATION OF LOGIC GATES

## Aim

To verify the truth tables of TTL AND, OR, NOT, NAND, NOR and XOR gates.

## Basic TTL gates

Commonly used basic TTL gates are:

| | |
|---|---|
| 7408 Quad two input AND gates | 7432 Quad two input OR gates |
| 7404 Hex inverters | 7400 Quad two input NAND gates |
| 7402 Quad two input NOR gates | 7486 Quad two input XOR gates |

## Truth Tables

The logic symbols for AND, OR, NOT, NAND, NOR and XOR gates are given in figure 1.1. The truth tables of AND, OR, NOT, NAND, NOR and XOR gates are given below.

AND gate

| Input | | Output |
|---|---|---|
| A | B | Y=A.B |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

OR gate

| Input | | Output |
|---|---|---|
| A | B | Y=A+B |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

NOT gate

| Input | Output |
|---|---|
| A | $Y = \bar{A}$ |
| 0 | 1 |
| 1 | 0 |

NAND gate

| Input | | Output |
|---|---|---|
| A | B | Y=$\overline{A.B}$ |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

NOR gate

| Input | | Output |
|---|---|---|
| A | B | Y=$\overline{A + B}$ |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

XOR gate

| Input | | Output |
|---|---|---|
| A | B | Y=$\overline{A \oplus B}$ |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

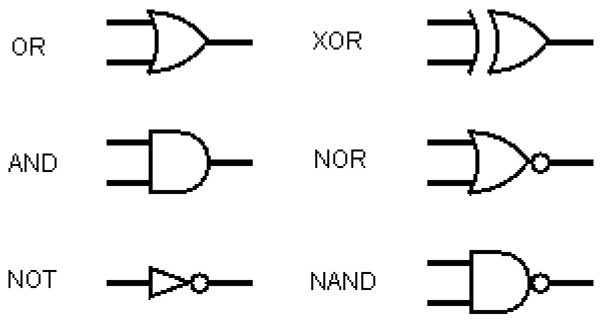


Figure 1.1 Logic symbols of gates

The pin details of ICs are given in figure 1.2.

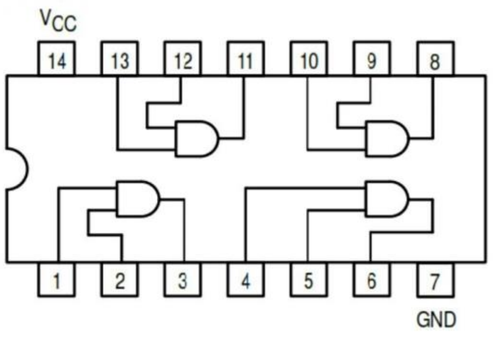


Figure 1.2.a IC 7408

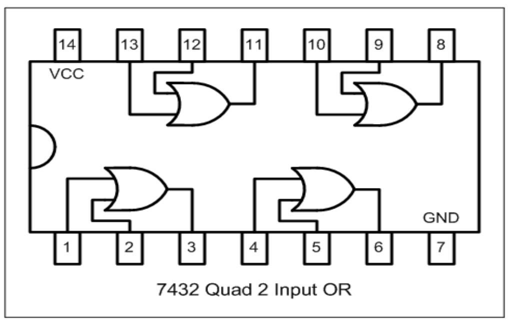


Figure 1.2.b IC 7432

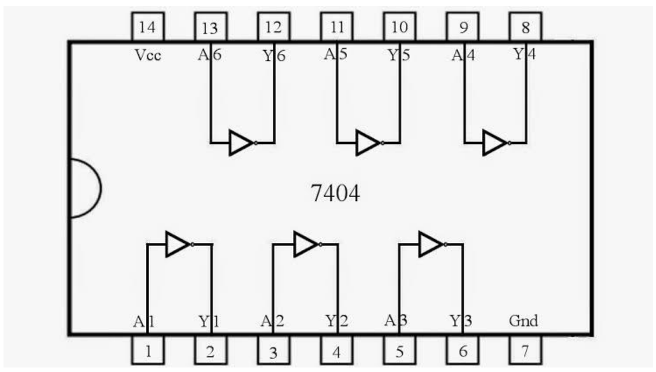


Figure 1.2.c IC 7404

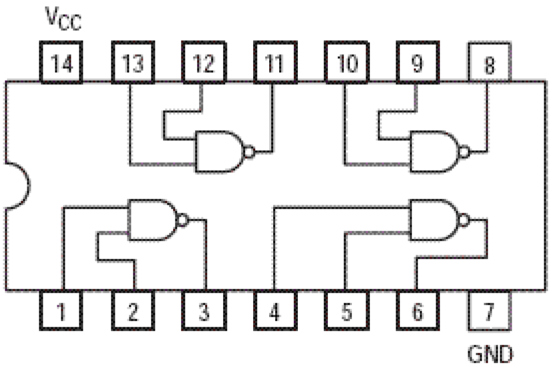


Figure 1.2.d IC 7400

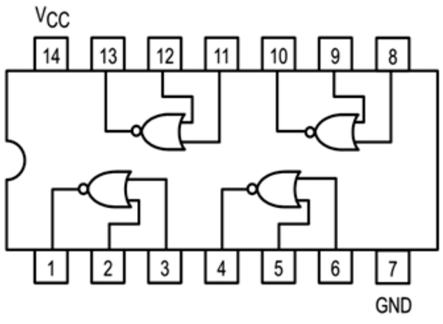


Figure 1.2.e IC7402

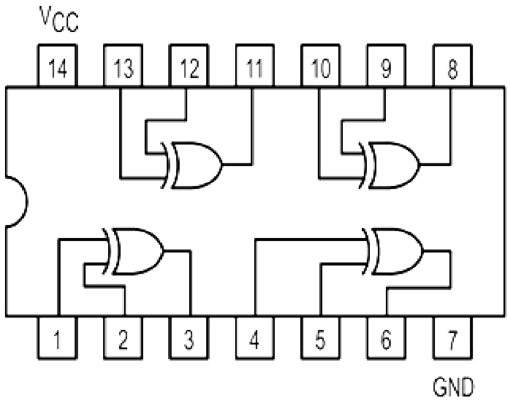


Figure 1.2.f IC 7486

**Procedure**

**1.1. AND gate**

Identify the terminals of IC 7408 and connect one gate from the quad AND gates of the IC as in figure 1.2.a. Connect 5 V dc between $V_{CC}$ and GND terminals. The positive terminal of the supply must be connected to the $V_{CC}$ terminal. The LED connected between $V_{CC}$ and the output terminal is used to indicate the logic state of the gate. Using 0 V for the logic 0 input and + 5V for logic 1 input, determine the logic states of the output for various combination of the input by noting whether the LED is glowing or not. When the LED is glowing, the logic output is zero and when not glowing, it is 1.

### 1.2. OR GATE

Connect one out of the four OR gates of the IC 7432 as in figure 1.2.b.Verify the truth table of the OR gate as in the case of AND gate.

### 1.3. NOT (Inverter) gate

Verify the truth table of the NOT gate by connecting one of the six inverter gates of IC 7404 as in figure 1.2.c

### 1.4. NAND gate

Connect one gate of the four two input NAND gates of the IC 7400 as in figure 1.2.d. Verify the truth table of the NAND gate.

### 1.5. NOR gate

Wire up the circuit of figure 1.2.e. using one of the four two input NOR gates of IC 7402. Verify the truth table by applying various input combination and observing the output.

### 1.6. XOR gate

Connect one XOR gate of the four gates of IC 7486 as in figure 1.2.f. Verify the truth table of the XOR gate given above.

### Questions

1. Make a NOT gate using (a) a 2 input NAND gate (b) a 3 input NAND gate.

2. How will you use a 3-input (a) NAND gate as a 2-input NAND gate (b) OR gate as a 2input OR gate

3. What are the values of the voltages measured at the output of a TTL gate corresponding to 0 and 1 levels respectively?

4. Comment on the magnitude and direction of the input/output current of a gate corresponding to 0 level and 1 level.

# 2. Realisation of SOP and POS Functions

**Aim**

1. To verify De-Morgan’s theorem.
2. To realize SOP and POS functions after K map reduction.

**De morgan’s Theorem**

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

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

**Procedure**

Verification of Demorgan’s theorem can be done using the circuits of fig 2.1 and 2.2. The circuits of figure 2.1a and 2.1b produce the same output for the same sets of inputs A and B. Verify DeMorgan’s first law. Similarly the circuits of fig.2.2a and 2.2b can be used to verify the second law.

Realize the following functions through NAND gates.

$$f = \sum(0,1,2,5,7,9,10)+\mathrm{d}(3,8,15)$$

$$f = \prod(0,1,2,3,8,9,10,13,15)+\mathrm{d}(4,5,11,14)$$

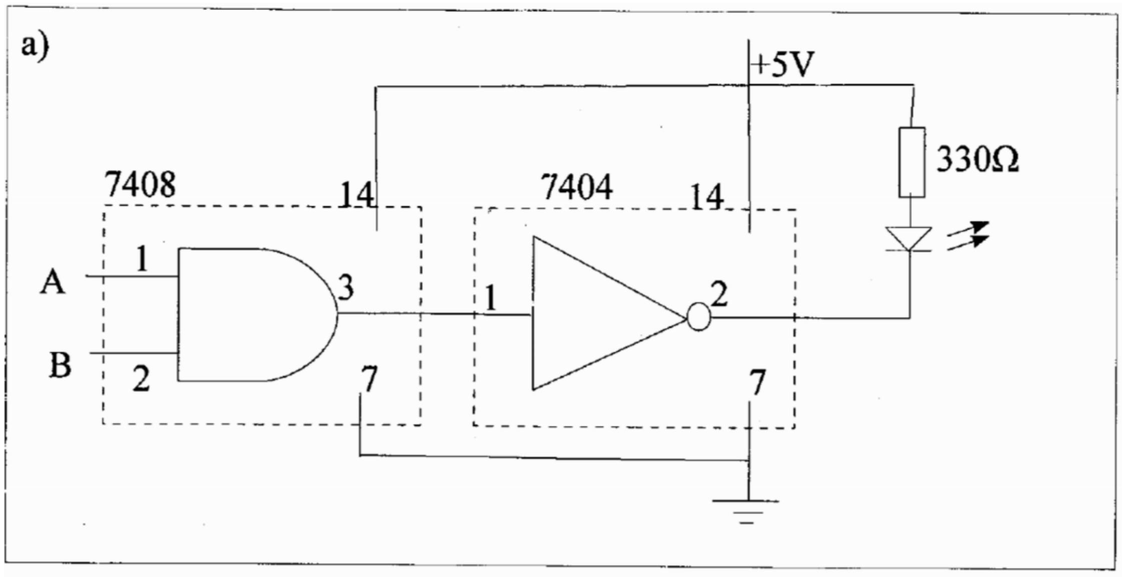


Fig. 2.1 (a) $\overline{A.B}$

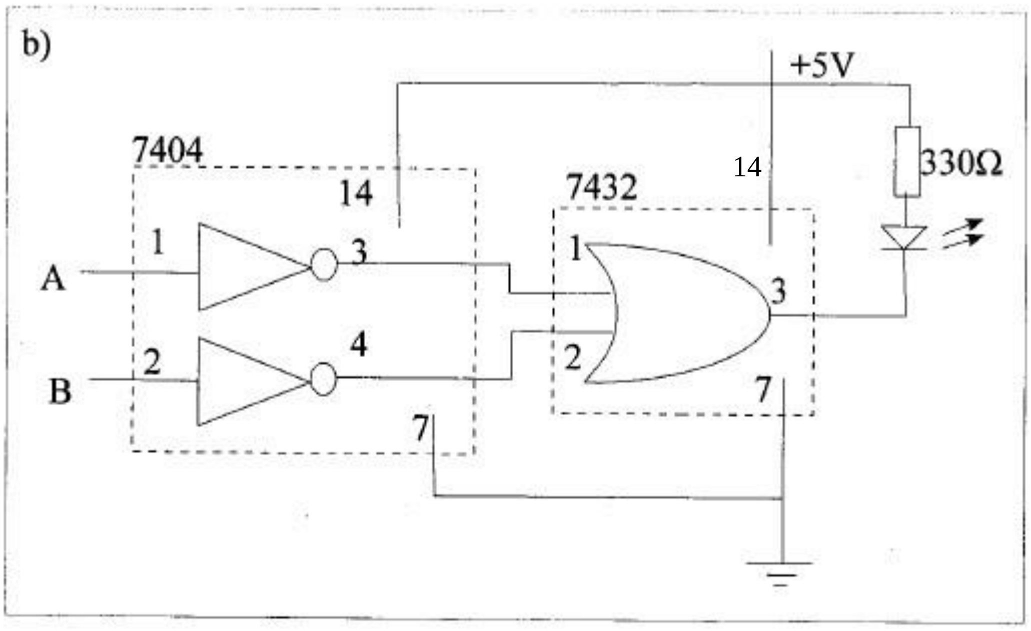


Fig. 2.1 (b) $\bar{A}+\bar{B}$

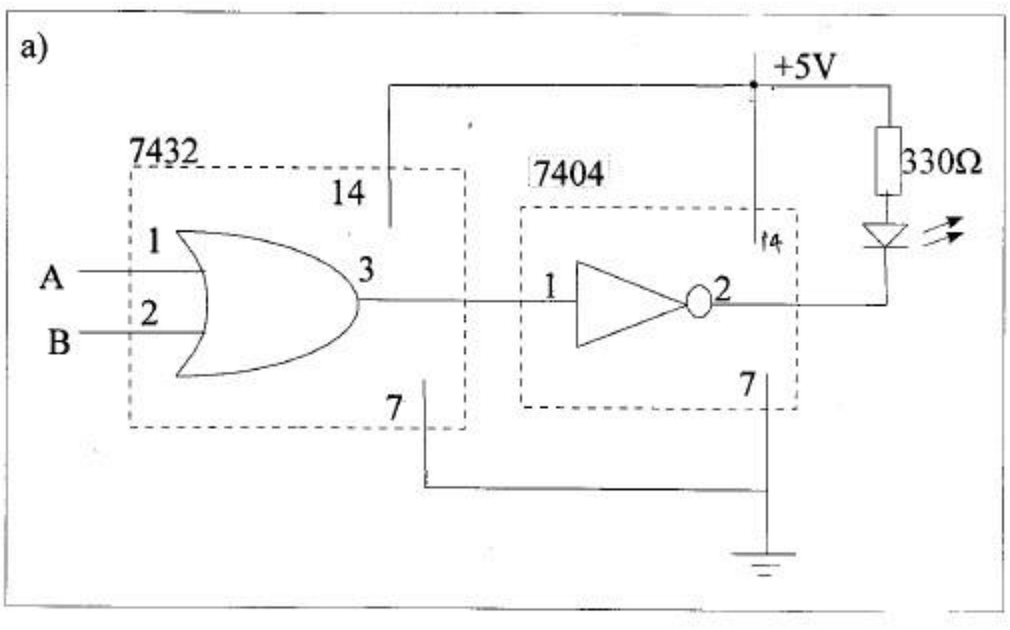


Fig. 2.2 (a) $\overline{A+B}$

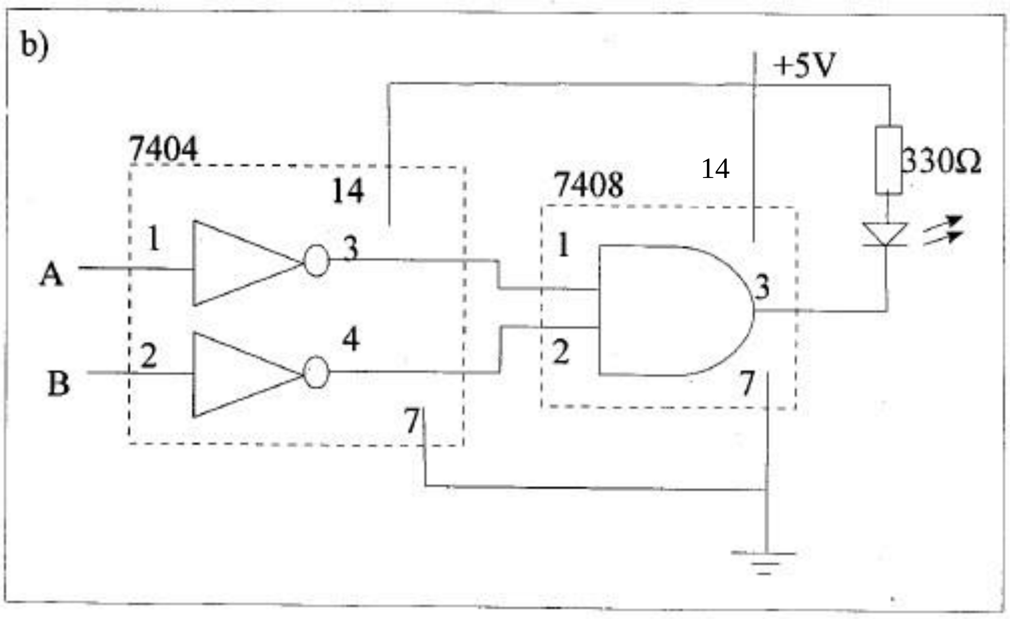


Fig. 2.2 (b) $\bar{A}+\bar{B}$

**SOP**

$f = \sum(0,1,2,5,7,9,10) + d(3,8,15)$

K map

| | $\overline{C}\overline{D}$ | $\overline{C}D$ | $CD$ | $C\overline{D}$ |
|---|---|---|---|---|
| $\overline{A}\overline{B}$ | 1 | 1 | X | 1 |
| $\overline{A}B$ | 0 | 1 | 1 | 0 |
| $AB$ | 0 | 0 | X | 0 |
| $A\overline{B}$ | X | 1 | 0 | 1 |

$$Y = \overline{\overline{\overline{B}\overline{D}}.\overline{\overline{A}D}\,\overline{\overline{B}\overline{C}}}$$

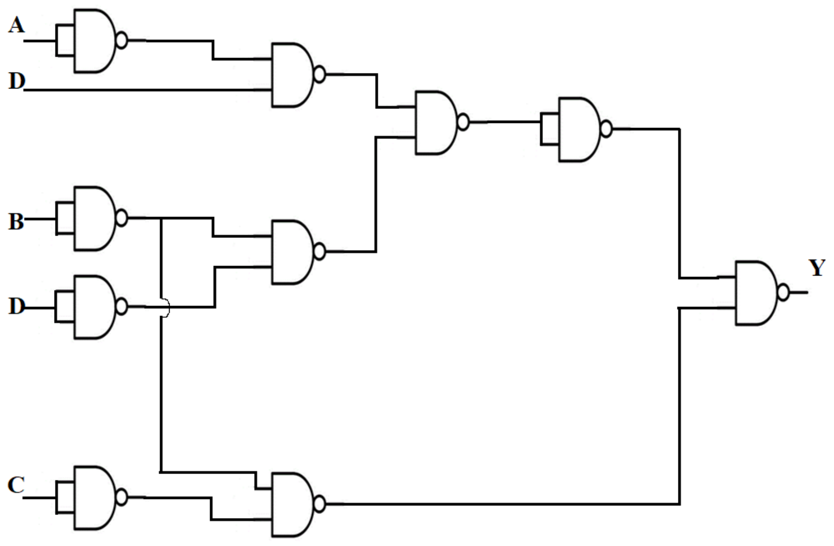


Fig. 2.3 Implementation of SOP

**POS**

$f = \prod(0,1,2,3,8,9,10,13,15)+\mathrm{d}(4,5,11,14)$

K map

| | $\overline{C}\overline{D}$ | $\overline{C}D$ | $CD$ | $C\overline{D}$ |
|---|---|---|---|---|
| $\overline{A}\overline{B}$ | **0** | **0** | **0** | **0** |
| $\overline{A}B$ | **X** | **X** | **1** | **1** |
| $AB$ | **1** | **0** | **0** | **X** |
| $A\overline{B}$ | **0** | **0** | **X** | **0** |

$Y = B\overline{AD}$

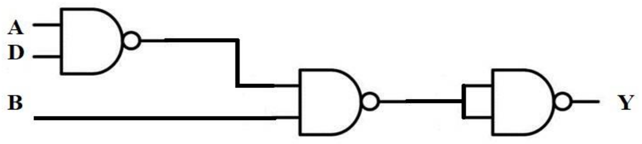


Fig.2.4 Implementation of POS

# 3. HALF ADDER AND FULL ADDER

**Aim**

1. To set up half adder circuit using NAND gates only and to verify the truth table.
2. To set up full adder circuit using XOR, AND and OR gates and to verify the truth table.
3. To set up full adder circuit using NAND gates only and to verify the truth table.

**Half Adder**

A logic circuit used for the addition of two one bit signals is known as a half adder. Its logic diagram and implementation using NAND gates is shown in Fig. 3.1.

**Full Adder**

Addition of two multi bit numbers is performed serially one bit (column wise) at a time from right to left. When two bits in column are added it is necessary that carry from the addition in previous column is also added. The combinational logic circuit which achieves this is called full adder. Fig. 3.2 shows the logic diagram of the full adder.

**Procedure**

Wire up the circuit of the half adder using IC7400. Apply proper inputs and verify the truth table. Assemble the full adder circuit using AND, OR and XOR gates and verify the truth table. Assemble the full adder circuit NAND gates and verify the truth table.

**Questions**

1. Design a full adder circuit using only NAND gates.

2. How will you use

a. A full adder as a half adder?

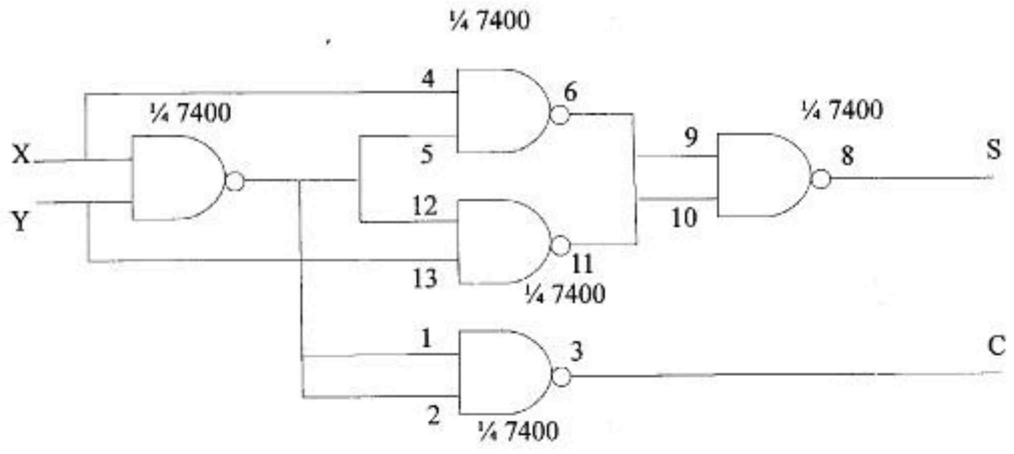
b. A full subtractor as a half subtractor?

Table 3.1: Truth table of Half Adder

| **X** | **Y** | **S** | **C** |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

Table 3.2: Truth table of Full Adder

| **X** | **Y** | **$C_{in}$** | **$S_0$** | **$C_c$** |
|---|---|---|---|---|
| 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 |



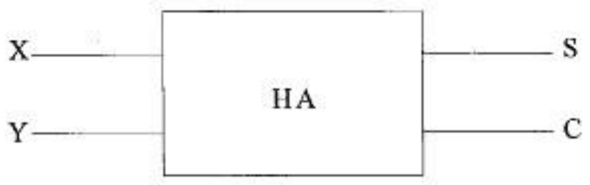
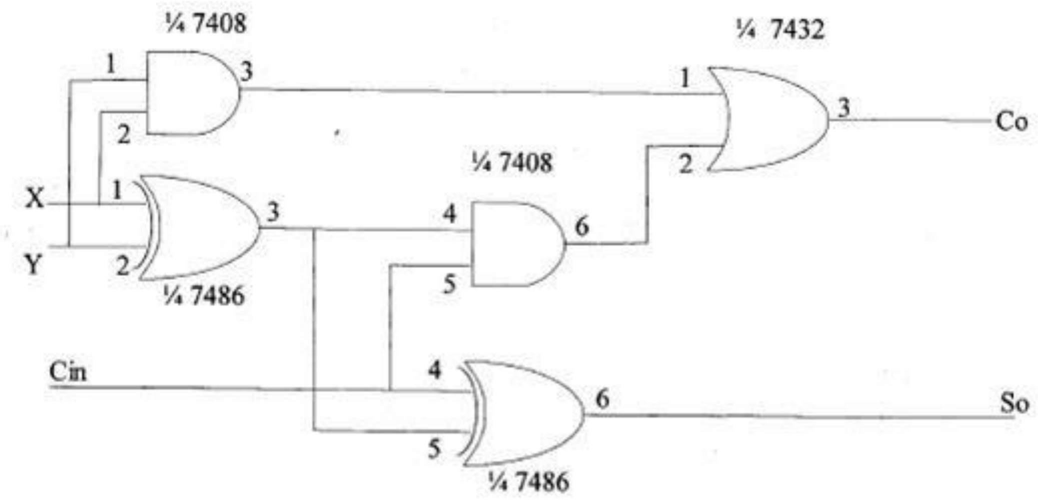


Fig.3.1 Half adder circuit using NAND gates



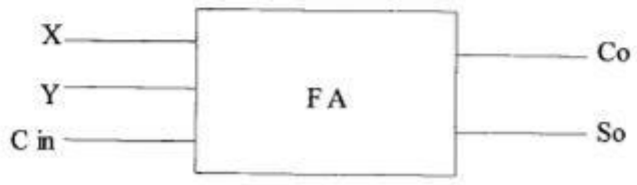


Fig. 3.2 Full adder circuit using XOR, AND and OR gates

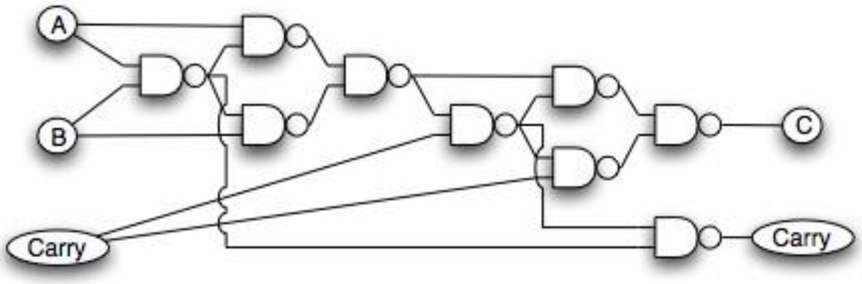


Fig. 3.3 Full adder circuit using only NAND gates

# 4. 4 Bit Adder/Subtractor and BCD Adder

**Aim**

1. To set up and verify the operation of a 4 bit adder using IC 7483.
2. To set up and verify the operation of a 4bit subtractor using IC 7483.
3. To set up and verify the operation of a BCD adder using IC 7483.

**IC7483**

IC7483 performs addition of two 4 bit binary numbers. These are full adders with their sums brought out as $S_0, S_1, S_2, S_3$. $S_0$ being the sum of the LSB column. The carryout from the MSB column is available at pin 14; pin 13 is the carry input. The pin configuration of the IC is given in Fig. 4.1.

**Procedure**

**4bit adder**

Connect the circuit as shown in Fig. 4.2. The $C_{in}$ input is grounded and the A and B inputs are applied at the appropriate input terminals. Verify the addition operation for various values of the two numbers A and B.

**4 bit subtractor**

The circuit is shown in figure 4.3. It performs the subtraction of 4 bit number B from another A by the 2's complement addition method. Wire up the circuit as shown with the input $C_{in}$ high. Apply various values for numbers A and B and verify the subtraction operation.

**4bit adder / subtractor**

The circuit is shown in figure 4.4. It performs addition / subtraction of 4 bit numbers A and B. Wire up the circuit as shown with the input $C_{in}$ low / high. Apply various values for numbers A and B and verify the addition and subtraction operation.

**BCD adder**

The circuit is shown in figure 4.5. It performs BCD addition of 4 bit numbers A and B. Wire up the circuit as shown with the input $C_{in}$ low. Apply various values for numbers A and B and verify the BCD addition operation.

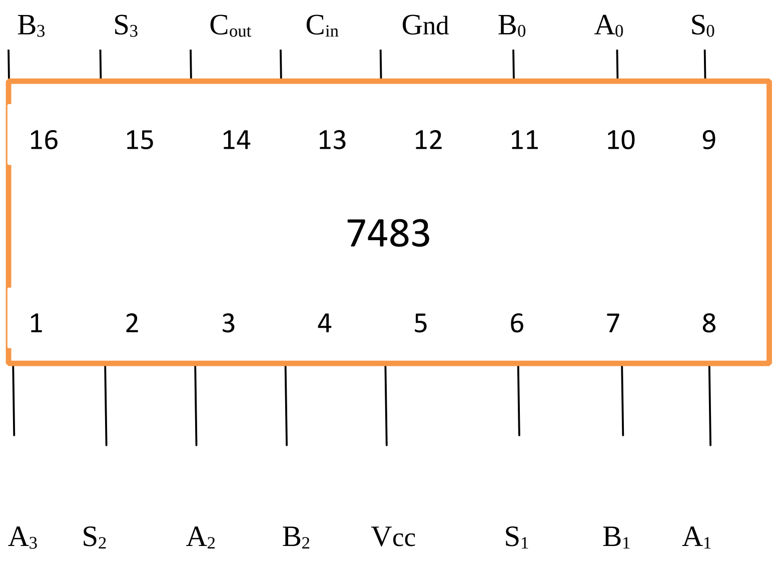


Fig. 4.1: Pin Details

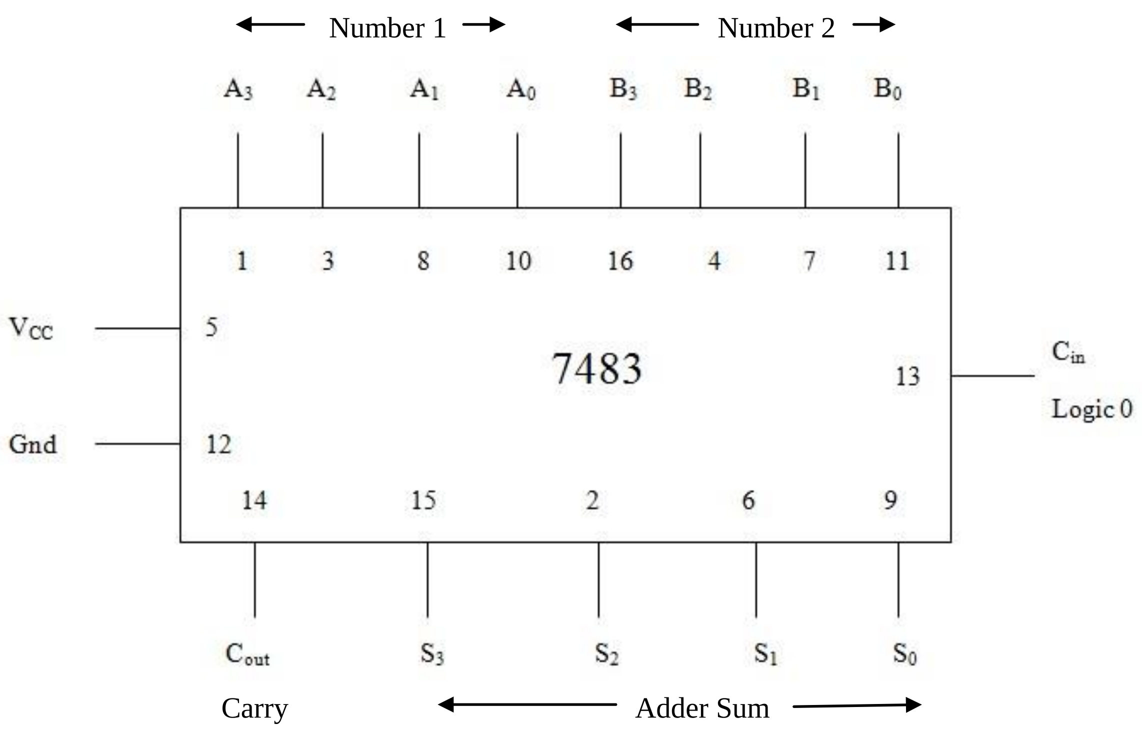


Fig. 4.2: 4 Bit Adder

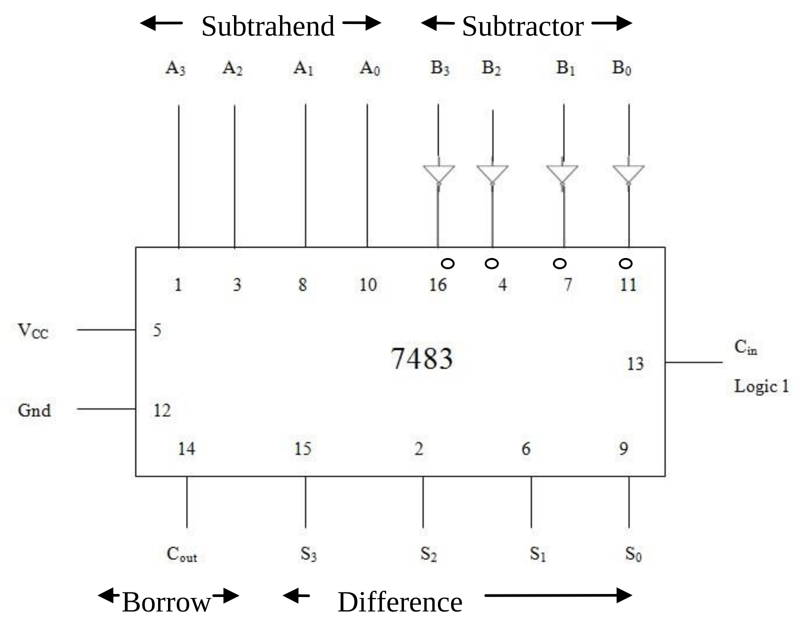


Fig. 4.3: 4 Bit Subtractor

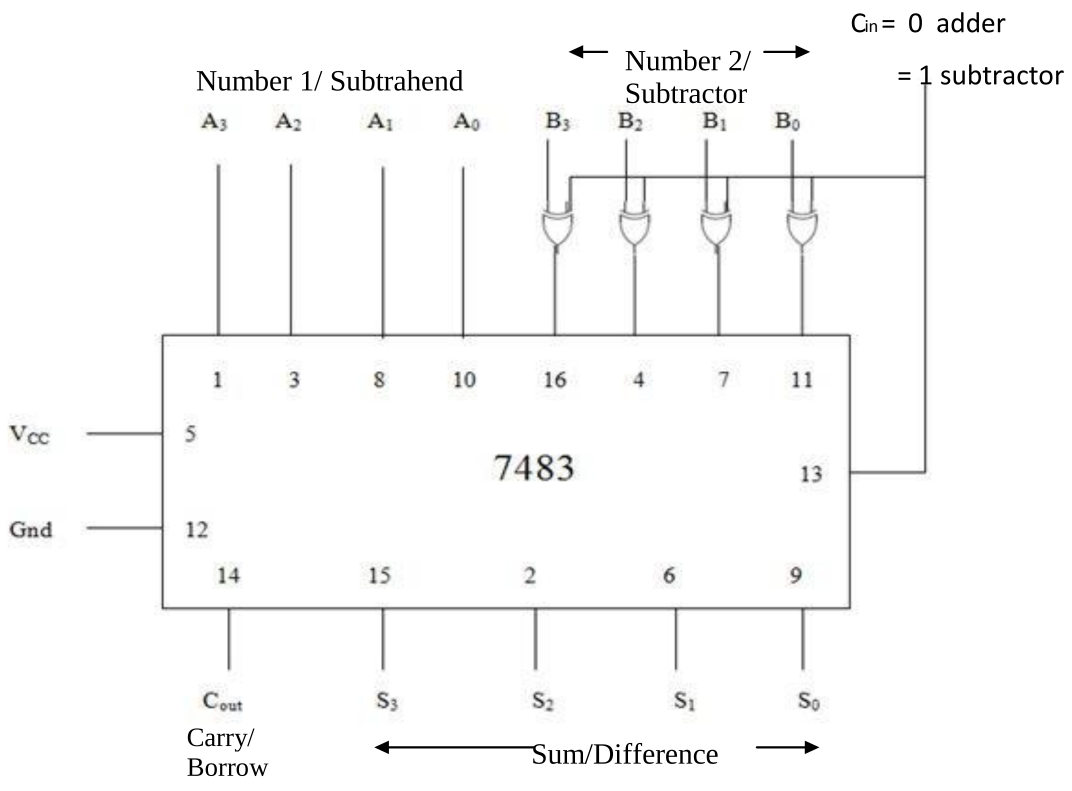


Fig. 4.4: 4 Bit Adder/Subtractor

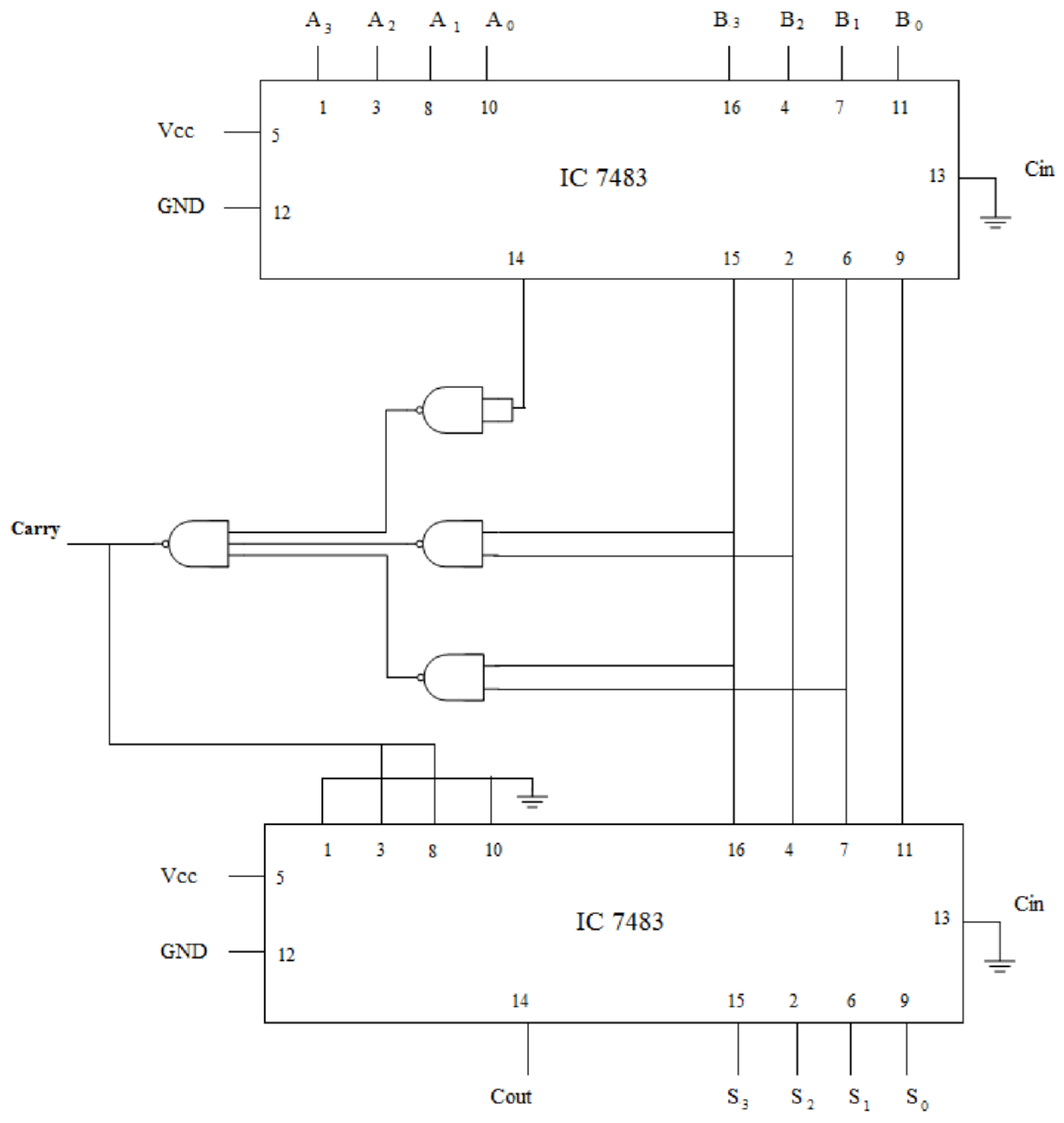


Fig. 4.5: BCD adder

# 5.a. BCD TO DECIMAL DECODER

## Aim

1. To study the operation of BCD to decimal decoder IC 7442.

## BCD to Decimal Decoder

Using 10 nos. of 4-input NAND gates a BCD to decimal decoder can be assembled as shown in Fig 5.a.1. ABCD are the BCD inputs and 0-9 are the outputs. The outputs are active low.

## IC7442

7442 is a TTL BCD to decimal decoder with active high inputs and active low outputs and is capable of driving LEDs. It uses inverters to obtain the complements of A, B, C, D inputs. It is a 16 pin IC with pin numbers 1 to 7 as outputs 0 to 6, pin number 8 ground, pin numbers 9 to 11 as outputs 7 to 9, pin numbers 12 to 15 as inputs D, C, B, A and pin number 16 is $V_{cc}$.

## Procedure

Wire up the circuits of Fig. 5.a.2. Apply BCD inputs 0000 through 1001 and observe the decimal outputs on the corresponding LEDs. Verify the truth table given below.

Table 5.a.1: Truth table of BCD to decimal decoder

| *No* | *D* | *C* | *B* | *A* | *Y0* | *Y1* | *Y2* | *Y3* | *Y4* | *Y5* | *Y6* | *Y7* | *Y8* | *Y9* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | L | L | L | L | L | H | H | H | H | H | H | H | H | H |
| 1 | L | L | L | H | H | L | H | H | H | H | H | H | H | H |
| 2 | L | L | H | L | H | H | L | H | H | H | H | H | H | H |
| 3 | L | L | H | H | H | H | H | L | H | H | H | H | H | H |
| 4 | L | H | L | L | H | H | H | H | L | H | H | H | H | H |
| 5 | L | H | L | H | H | H | H | H | H | L | H | H | H | H |
| 6 | L | H | H | L | H | H | H | H | H | H | L | H | H | H |
| 7 | L | H | H | H | H | H | H | H | H | H | H | L | H | H |
| 8 | H | L | L | L | H | H | H | H | H | H | H | H | L | H |
| 9 | H | L | L | H | H | H | H | H | H | H | H | H | H | L |
| 10 | H | L | H | L | H | H | H | H | H | H | H | H | H | H |
| 11 | H | L | H | H | H | H | H | H | H | H | H | H | H | H |
| 12 | H | H | L | L | H | H | H | H | H | H | H | H | H | H |
| 13 | H | H | L | H | H | H | H | H | H | H | H | H | H | H |
| 14 | H | H | H | L | H | H | H | H | H | H | H | H | H | H |
| 15 | H | H | H | H | H | H | H | H | H | H | H | H | H | H |

**Questions**

1. What is meant by priority reader?

2. Design a decimal to excess-3 encoder using NAND gates.

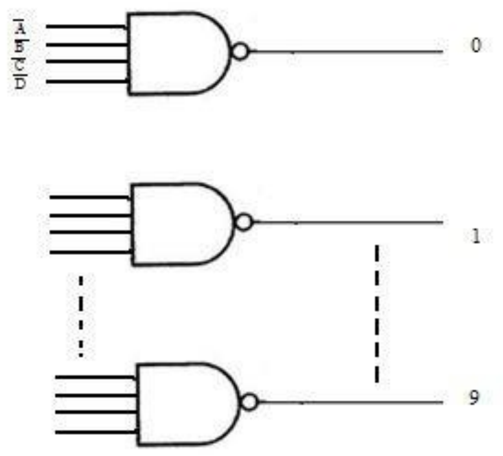


Fig. 5.a.1: BCD to decimal decoder

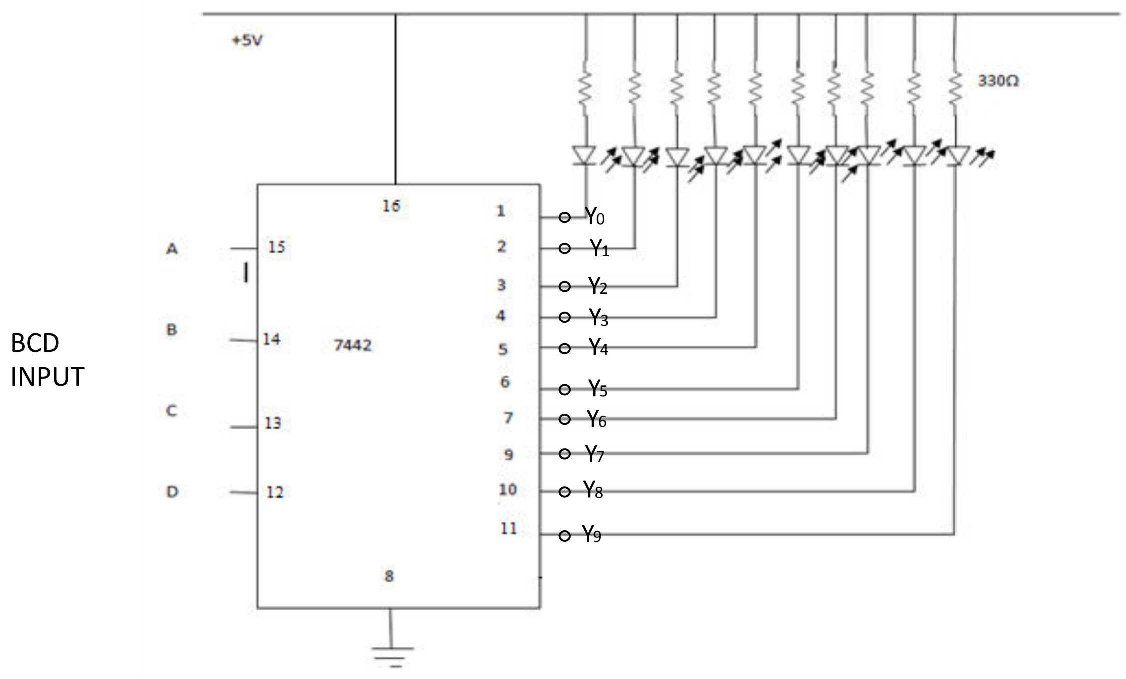


Fig. 5.a.2: BCD to decimal decoder wiring circuit

# 5.b.SEVEN SEGMENT DISPLAY AND DECODER/DRIVER

**Aim**

To verify the truth table of the seven segments LED display using BCD to 7 segment decoder/driver.

**7segment LED display**

A seven segment LED display consists of seven individual LEDs positioned as shown in Fig. 5.b.2, each diode forming a segment. These diodes are mounted on a common substrate and the whole unit comes in a single package. For some seven segment LEDs, the anodes of the segment diodes are made to terminate at a common point. To display a numeric digit in the common anode seven segment LED, the anode segment is connected to $+V$. The cathode pins of the corresponding segment LEDs are grounded for displaying the required digit. The seven segments of the display are termed a, b, c, d, e, f and g. There is one dot LED in the display device. This is for displaying the decimal point wherever necessary.

The cathodes of seven segment LEDs must be grounded through series resistances typically 330Ω each. The potential difference across the LED when forward biased is typically 1.6V-2.5V and the current required is in the range of 5-20mA.

**BCD to seven segment decoder**

IC7447 is a BCD to 7segment decoder/driver which offers active low, high sink current outputs incorporates automatic leading and or leading edge zero blanking control (RBI and RBO). Lamp tests may be performed at any time when the BI/RBO node is HIGH. An overriding blanking input BI can be used to control the lamp intensity or to inhibit the outputs. The pin details and the logic diagram of the device are shown in Fig.5.b.1. The pin details of the seven segment display FND507 are given in Fig .5.b.2.

**Procedure**

Connect the seven segment display and BCD to 7 segment decoder/driver. Verify the truth table by applying proper inputs.

Table 5.b.1: Truth table for BCD to seven segment decoder

| | | INPUTS | | | | | | OUTPUTS | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Fn | LT | RBI | D | C | B | A | BI/RBO | a | b | c | d | e | f | g | note |
| 0 | H | H | L | L | L | L | H | L | L | L | L | L | L | H | A |
| 1 | H | X | L | L | L | H | H | H | L | L | H | H | H | H | A |
| 2 | H | X | L | L | H | L | H | L | L | H | L | L | H | L | |
| 3 | H | X | L | L | H | H | H | L | L | L | L | H | H | L | |
| 4 | H | X | L | H | L | L | H | H | L | L | H | H | L | L | |
| 5 | H | X | L | H | L | H | H | L | H | L | L | H | L | L | |
| 6 | H | X | L | H | H | L | H | H | H | L | L | L | L | L | |
| 7 | H | X | L | H | H | H | H | L | L | L | H | H | H | H | |
| 8 | H | X | H | L | L | L | H | L | L | L | L | L | L | L | |
| 9 | H | X | H | L | L | H | H | L | L | L | H | H | L | L | |
| BI | X | X | X | X | X | X | L | H | H | H | H | H | H | H | B |
| RBI | H | L | L | L | L | L | L | H | H | H | H | H | H | H | C |
| LT | L | X | X | X | X | X | H | L | L | L | L | L | L | L | D |

**Notes:**

A. BI/RBO serves as blanking input/ripple blanking output. BI and RBI must be open or held high. Input may be high or low.
B. When a low level is applied to the blanking input all segment outputs got a high level regardless of other inputs.
C. When ripple blanking input and other inputs A, B, C and D are at low level with the lamp test input at high level all segment outputs go to a high level and RBO goes to low level (response condition).
D. When BI/RBO is open or held at high and a low is applied to LT all segment outputs go to low.

**Questions**

1. Explain the difference between encoder and decoder.

2. Explain the following terms:
    a. Zero blanking
    b. Leading zero blanking

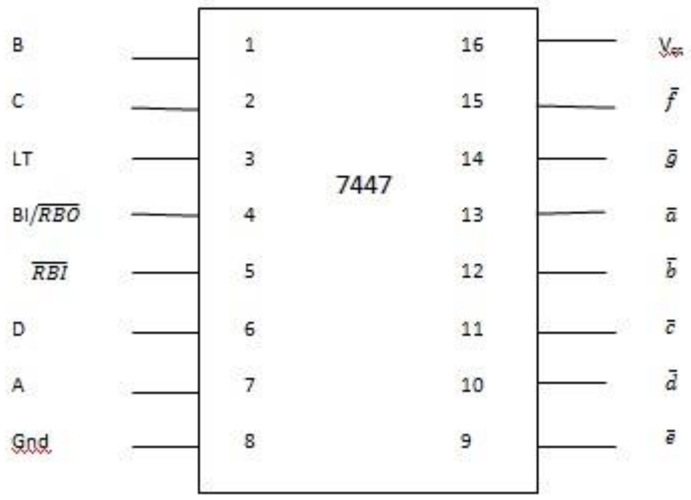


Fig. 5.b.1: Pin diagram of IC7447

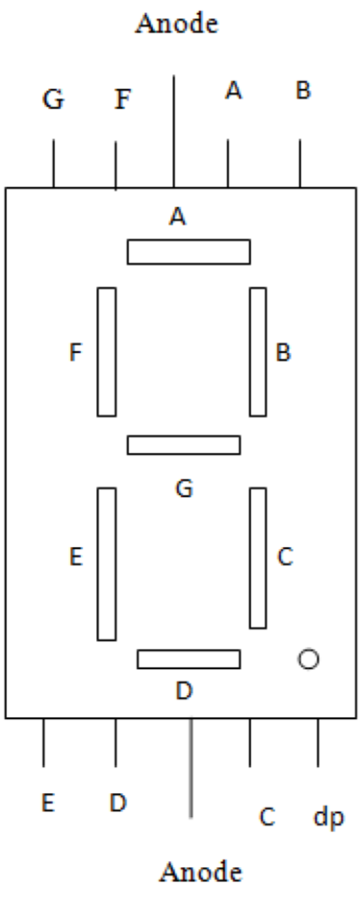


Fig. 5.b.2: 7 segments LED display using FND 507

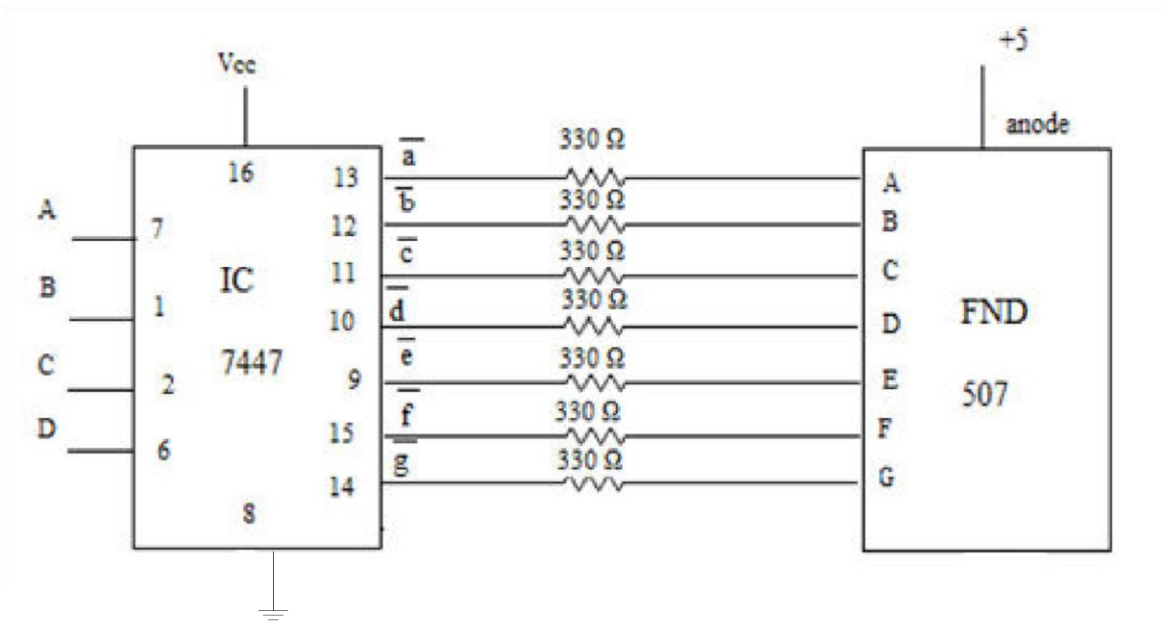


Fig .5.b.3: BCD to seven segment display wiring circuit

# 6 (a) MUX using gates and to study a MUX ICs

**Aim**

To study a 4 : 1 multiplexer using gates and to study a MUX ICs

**Multiplexer**

Multiplexer (Mux) is a combinatorial circuit which selects one of the inputs and route it to the output. A multiplexer has data input lines, data select lines and output.

The logic symbol of a 4: 1 multiplexer is shown in Fig. 6.a.1. According to the two bit binary code on the data select inputs, corresponding data input line will be selected and routed to the output.

From the truth table 6.a.1, it can be seen that output

$$Y = D_0S_1S_2 + D_1\overline{S_1}S_2 + D_2S_1\overline{S_2} + D_3\overline{S_1}\overline{S_2}$$

This Boolean expression can be realized using gates.

**IC 74151**

It is an 8 : 1 multiplexer with 16 pin IC package. It has three data select inputs $S_0$, $S_1$ and $S_2$ and an active low strobe input. The data inputs are $D_0$ through $D_7$. Three bit binary number at the data select inputs decides the data input line that is to be directed to the output Y. A logic low at the strobe input activates the chip. The pin diagram of IC 74151 is shown in figure 6.a.2.

**IC 74153**

It is a dual 4 : 1 multiplexer IC. It has four inputs in each section and $Y_0$ and $Y_1$ are the corresponding outputs. $G_0$ and $G_1$ are the corresponding active low strobe inputs to these sections. Select lines $S_1$ and $S_0$ are common for both sections. The pin diagram of IC 74153 is shown in figure 6.a.3.

**Procedure**

**Multiplexer using gates**

Set up the circuit as in figure 6.a.4. Input all four combinations at $S_1$ $S_0$ one by one, observe corresponding output.

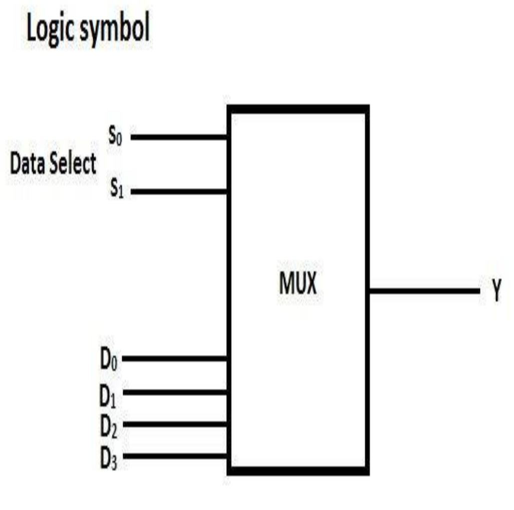
**IC 74151**

Apply logic 0 to the strobe input. Apply logic 0 or 1 randomly to data inputs $D_0$ through $D_7$. Apply binary numbers from 000 to 111 at select lines and observe the output Y from pin no. 5 corresponding to the select line inputs.

**IC 74153**

Apply logic 0 to the active low strobe input $G_0$ and $G_1$. Apply logic 0 or 1 randomly to data inputs $A_0$, $B_0$, $C_0$ and $D_0$. Apply binary numbers from 00 to 11 at select lines and observe the output $Y_0$ from pin no. 7 corresponding to the select line inputs.

Apply logic 0 or 1 randomly to data inputs $A_1$, $B_1$, $C_1$ and $D_1$. Apply binary numbers from 00 to 11 at select lines and observe the output $Y_1$ from pin no. 9 corresponding to the select line inputs.



.Fig. 6 .a.1 4:1 MUX

| $S_1$ | $S_2$ | Y |
|---|---|---|
| 0 | 0 | $D_1$ |
| 0 | 1 | $D_2$ |
| 1 | 0 | $D_3$ |
| 1 | 1 | $D_4$ |

Table. 6.a.1 Truth Table of 4:1 MUX

Pin out diagrams of 74151

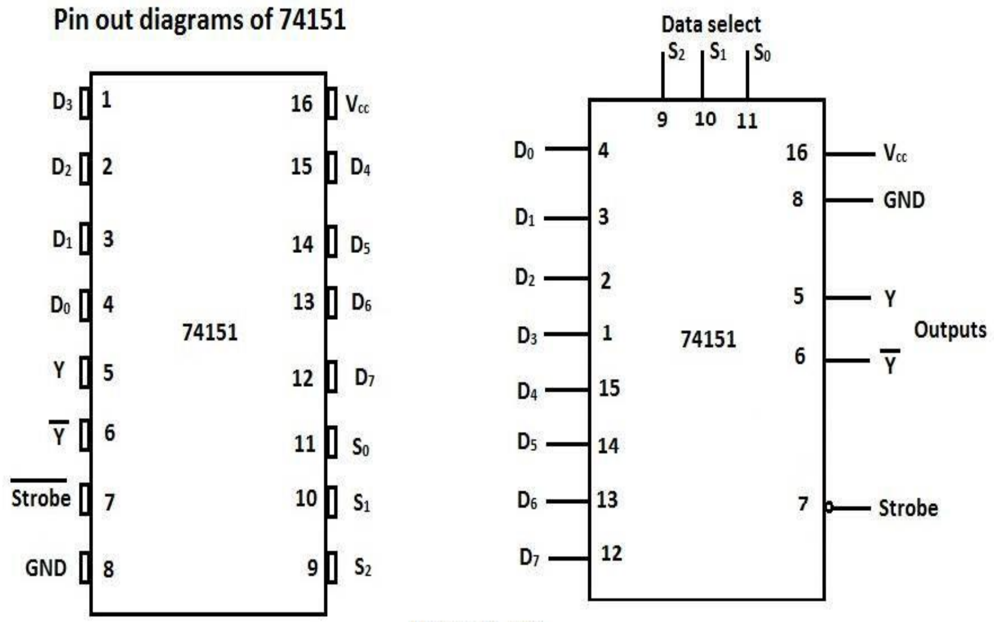


Fig. 6.a.2

Pin out diagrams of 74153

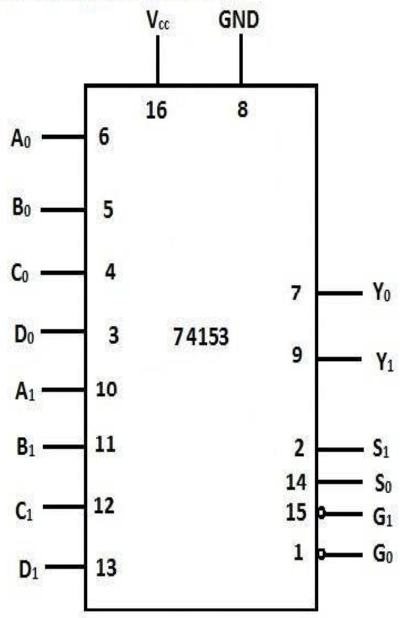
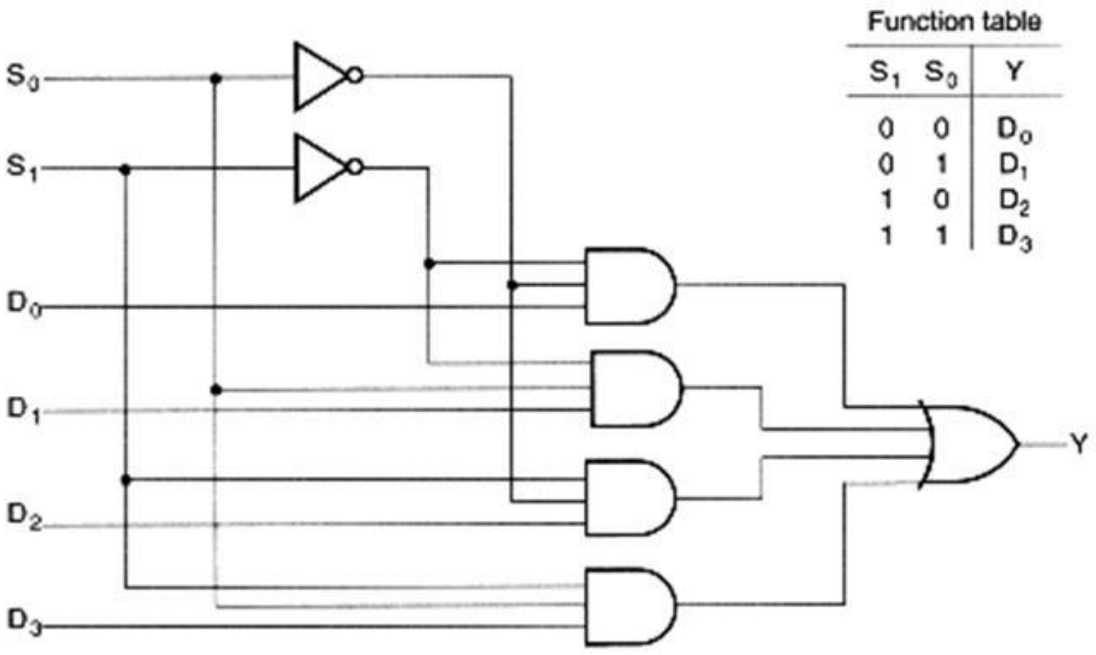


Fig. 6.a.3

| Inputs | | | Output |
|---|---|---|---|
| Strobe $G_0$ | $S_1$ | $S_0$ | $Y_0$ |
| 0 | 0 | 0 | $A_0$ |
| 0 | 0 | 1 | $B_0$ |
| 0 | 1 | 0 | $C_0$ |
| 0 | 1 | 1 | $D_0$ |
| 1 | X | X | 0 |

Table. 6.a.2



Function table

| $S_1$ | $S_0$ | Y |
|---|---|---|
| 0 | 0 | $D_0$ |
| 0 | 1 | $D_1$ |
| 1 | 0 | $D_2$ |
| 1 | 1 | $D_3$ |

Fig. 6.a.4 Implementation of 4:1 MUX using gates.

## 6(b) Combinatorial circuits using MUX

**Aim**

To realize the following function using multiplexer IC

$f = \sum m(0,1,3,6,8,9,10,12,13,14)$

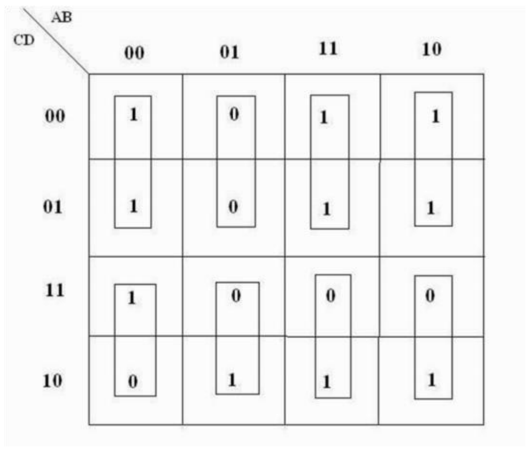
**Theory**

Multiplexers can be used to realize logic circuits. A multiplexer with n number of select lines can be used to realize an n variable Boolean expression. With additional logic gates or circuits an n+1 variable Boolean expression can be realized with the same multiplexer.

The implementation table is shown in table. 6.b.1.

Each grouped pairs correspond to eight data inputs to the MUX. Top left pair indicates that when ABC = 000, data input $D_0$ should be 1. Bottom left pair indicates that when ABC = 001, data input $D_1$ should be same as D. Bottom right pair indicates that when ABC = 101, data input $D_5$ should be $\overline{D}$ and so on.

Table. 6.b.1 K Map



| CD \ AB | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | 0 | 1 | 1 |
| 01 | 1 | 0 | 1 | 1 |
| 11 | 1 | 0 | 0 | 0 |
| 10 | 0 | 1 | 1 | 1 |

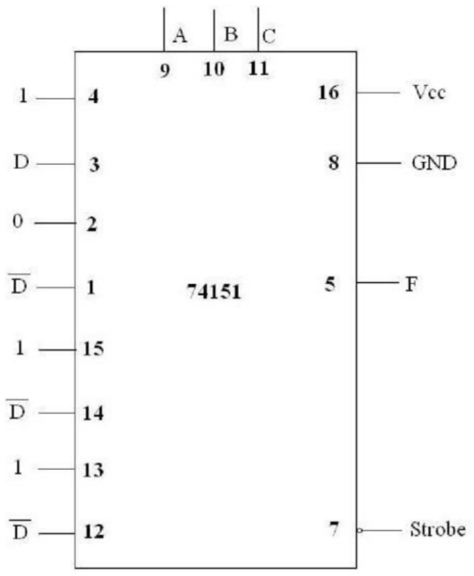


Fig. 6.b.1: Pin diagram of IC74151

# 7. Study of IC7490 and IC7493

## IC 7493

This IC is a 4 bit binary counter which can be used in either mod 8 or mod 16 configurations. The logic figure of the IC is given in fig. 7.3. The reset inputs R1 and R2 are active high and a high level at both inputs are necessary to reset all flip flops simultaneously. All the 4 flip flops have their J and K inputs connected to $V_{cc}$ . If clock is applied to input B, the outputs will appear at $Q_B$, $Q_C$, $Q_D$ and this is a mod 8 ripple counter. On the other hand, if the clock is applied at the input A and $Q_A$ is connected to input B, it is mod-16, 4-bit ripple counter. The outputs are $Q_A$, $Q_B$, $Q_C$ and $Q_D$.

## IC 7490

Fig 7.4 shows the basic internal structure of 7490. $FF_A$ is mod 2 counter and $FF_B$, $FF_C$, $FF_D$ constitute a mod 5 counter. The mod 5 and mod 2 counter can be used independently or in combination. If $Q_A$ is connected to input B and the pulse to be counted are applied at input A, the circuit is a BCD counter.

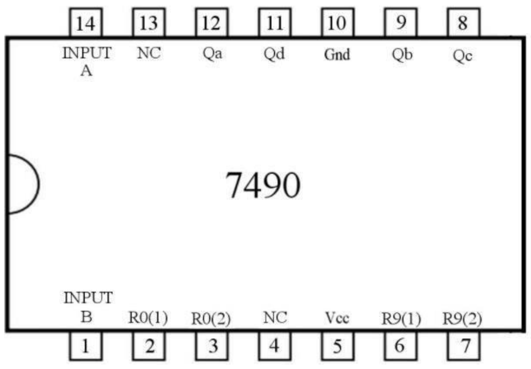


Fig. 7.1: Pin diagram of IC7490

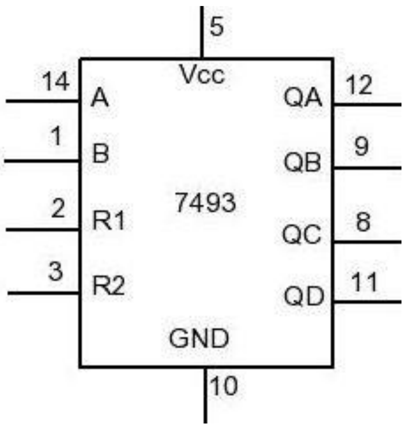


Fig. 7.2: Pin diagram of IC7493

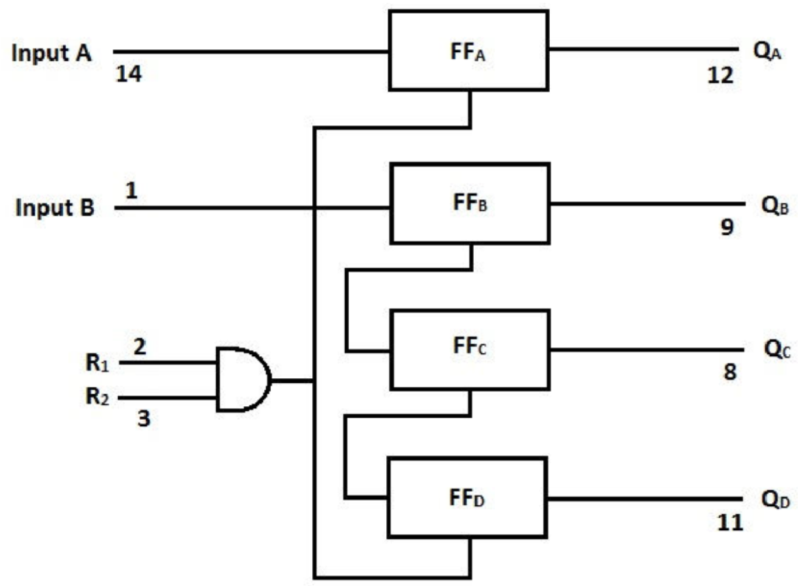


Fig. 7.3: Connections of IC 7493

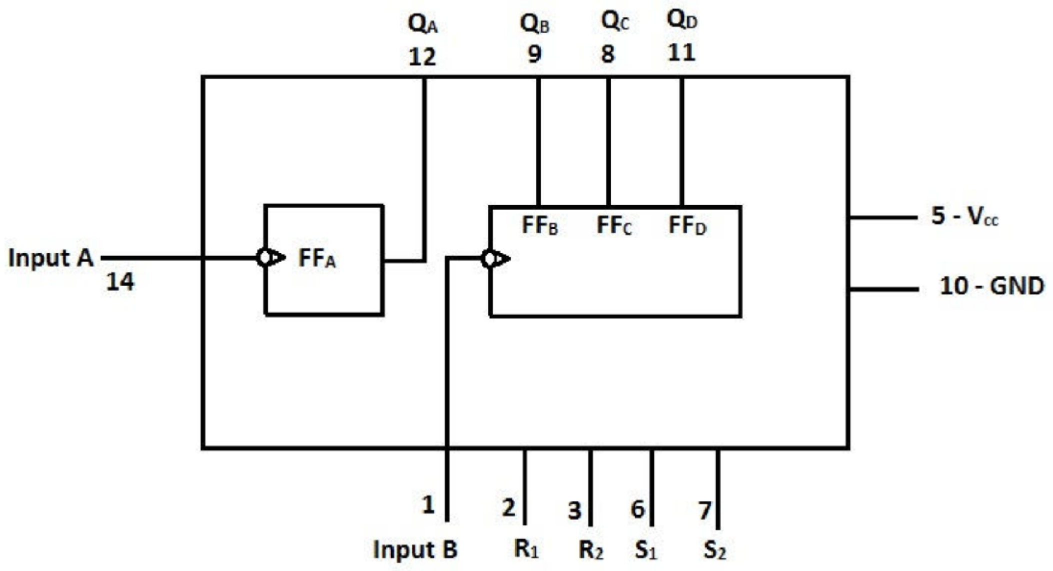


Fig. 7.4: Internal connections of IC 7490

# 8. Design of synchronous up, down & modulo N counters

**Aim**

1. To study the design and implementation of 3 bit synchronous up/down counter.
2. To study the working of the 4-bit binary counter IC 7493 and the decade counter IC 7490
3. To set up a counter of modulus N using IC 7493

**Synchronous up, down counter**

A counter is a register capable of counting number of clock pulse arriving at its clock input. Counter represents the number of clock pulses arrived. An up/down counter is one that is capable of progressing in increasing order or decreasing order through a certain sequence. An up/down counter is also called bidirectional counter. Usually up/down operation of the counter is controlled by up/down signal. When this signal is high counter goes through up sequence and when up/down signal is low counter follows reverse sequence.

**State Diagram**

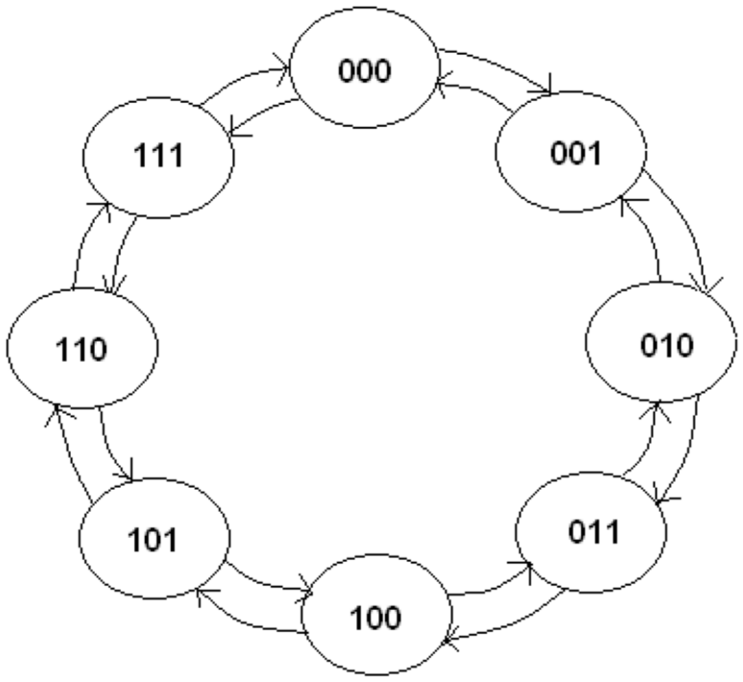


Fig. 8.1: State Diagram

Table 8.1: Characteristics Table

| Q | $Q_{t+1}$ | J | K |
|---|---|---|---|
| 0 | 0 | 0 | X |
| 0 | 1 | 1 | X |
| 1 | 0 | X | 1 |
| 1 | 1 | X | 0 |

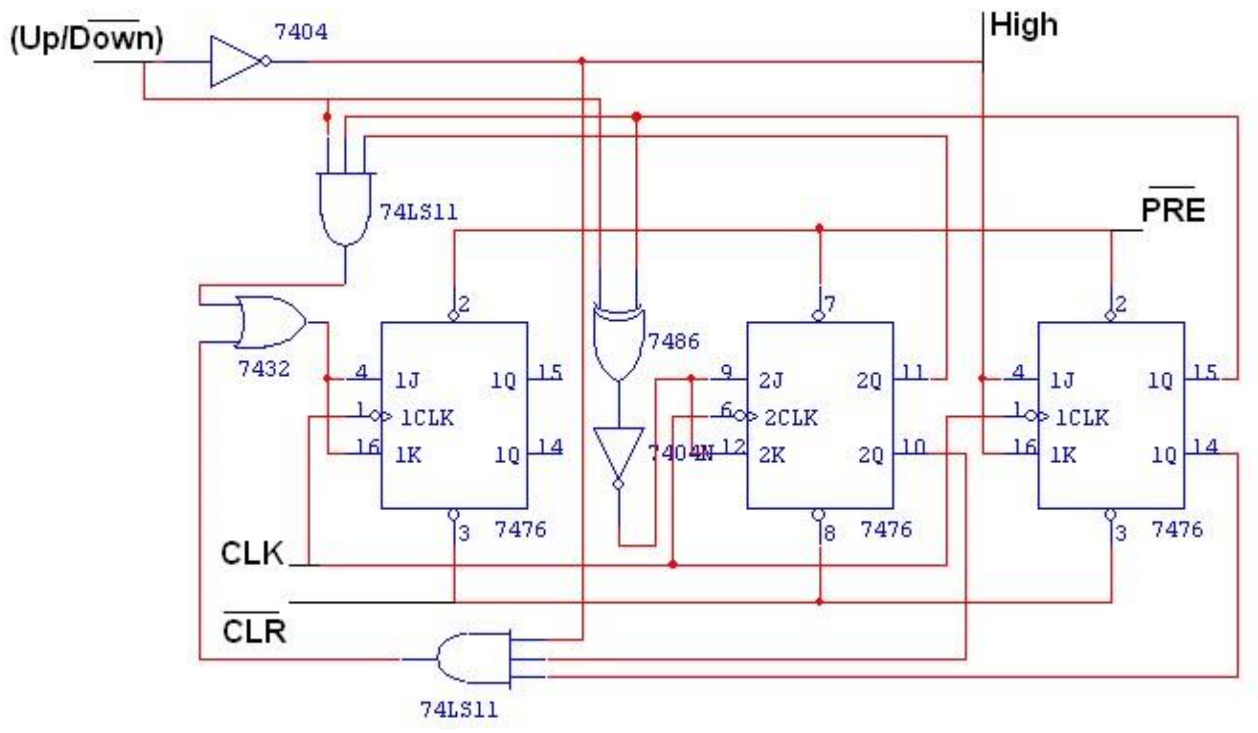


Fig. 8.2: Logic Diagram

Table 8.2: Truth Table

| Input | Present State | | | Next State | | | A | | B | | C | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Up/Down | $Q_A$ | $Q_B$ | $Q_C$ | $Q_{A+1}$ | $Q_{B+1}$ | $Q_{C+1}$ | $J_A$ | $K_A$ | $J_B$ | $K_B$ | $J_C$ | $K_C$ |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | X | 1 | X | 1 | X |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | X | 0 | X | 0 | X | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | X | 0 | X | 1 | 1 | X |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | X | 0 | 0 | X | X | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | X | 1 | 1 | X | 1 | X |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | X | X | 0 | X | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | X | X | 1 | 1 | X |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | X | 0 | X | X | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | X | 0 | X | 1 | X |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | X | 1 | X | X | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | X | X | 0 | 1 | X |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | X | X | 1 | X | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 | X | 0 | 0 | X | 1 | X |
| 1 | 1 | 0 | 1 | 1 | 1 | 0 | X | 0 | 1 | X | X | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | X | 0 | X | 0 | 1 | X |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | X | 1 | X | 1 | X | 1 |

**Procedure**

1. Connections are given as per circuit diagram.
2. Logical inputs are given as per circuit diagram.
3. Observe the output and verify the truth table.

**Binary counter**

A binary counter can be constructed using clocked J-K flip flops. All the J and K inputs are tied to $+V_{cc}$ . The system clock drives the second and so on. Each flip flop toggles with a negative transition at its clock input. A divide by N (modulo N) counter capable of counting up to (N-1) requires n flip flops where $2^n=N$. For example, Mod 8 counter which can count up to 7 requires 3 flip flops. It is called Modulo N counter because it has N different output states.

**IC 7493**

This IC is a 4 bit binary counter which can be used in either mod 8 or mod 16 configurations. The logic figure of the IC is given in fig. 7.3. The reset inputs R1 and R2 are active high and a high level at both inputs are necessary to reset all flip flops simultaneously. All the 4 flip flops have their J and K inputs connected to $V_{cc}$ . If clock is applied to input B, the outputs will appear at $Q_B$, $Q_C$, $Q_D$ and this is a mod 8 ripple counter. On the other hand, if the clock is applied at the input A and $Q_A$ is connected to input B, it is mod-16, 4-bit ripple counter. The outputs are $Q_A$, $Q_B$, $Q_C$ and $Q_D$.

**IC 7490**

Fig 7.4 shows the basic internal structure of 7490. $FF_A$ is mod 2 counter and $FF_B$, $FF_C$, $FF_D$ constitute a mod 5 counter. The mod 5 and mod 2 counter can be used independently or in combination. If $Q_A$ is connected to input B and the pulse to be counted are applied at input A, the circuit is a BCD counter.

**Modulo N counter**

The modulus of a counter is the total number of states through which the counter can progress. To construct a modulo N counter it is necessary to have a ripple chain of n flip flops such that n is the smallest number for which $2^n > N$. IC 7493 whose pin details are given in fig 7.2 has 4 flip flops and so using this IC a counter of any modulus up to 16 can be set up. All flip flops of 7493 have direct reset inputs, which are active low. The output of a NAND gate is connected to all the direct reset inputs of the flip flops. Using a proper feedback connection, it is possible to reset all the flip flops at count N. This can be achieved by connecting each input ($R_1$, $R_2$) of the NAND gate to the Q output of a flip flop which becomes 1 at count N. If more than one flip flop outputs become 1 at count N, AND gates can be used to connect these outputs to the reset inputs of the counter. For example, a modulo 13 counter can be set up connecting $Q_D$ and $Q_C$ to $R_1$ and through an AND gate and $Q_A$ directly to $R_2$

**Procedure**

Wire up the circuit for mod 13 counter by connecting $Q_A$ to B input and to $R_1$ and $Q_D$ and $Q_C$ to $R_2$ through an AND gate. Connect LED indicators to $Q_A$, $Q_B$, $Q_C$ and $Q_D$. Apply clock manually to A input and verify that the counter is a mod 13 counter. Repeat for mod 9 and mod 11. In both these cases, $Q_A$ is connected to the B input. The feedback to inputs are obtained from $Q_A$, $Q_B$ through an AND gate and $Q_D$ direct for mod 11 and $Q_A$ to $R_1$ and $Q_D$ to $R_2$ for mod 9.

**Questions**

1. How will you use the 7490 IC to design a symmetrical divide by 10 frequency divider?

2. Set up a circuit using 7490 IC to count up to 999.

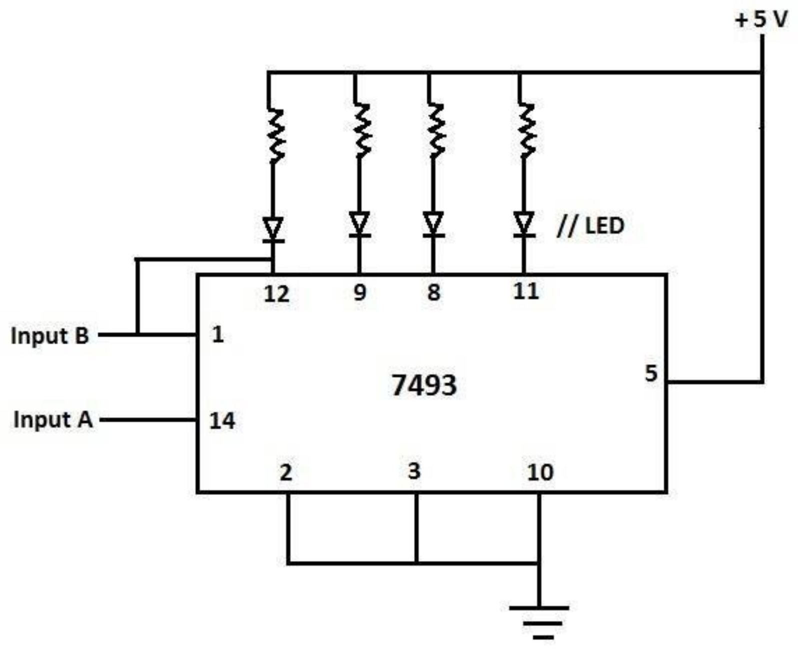


Fig. 8.3: Binary Counter

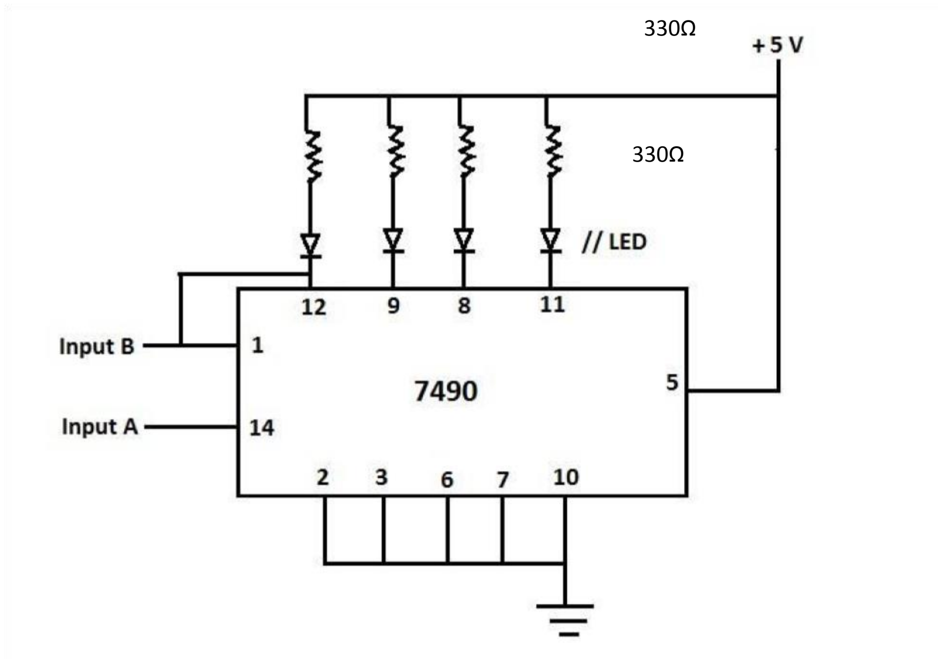


Fig. 8.4: Decade counter

# 9. Study of shift register IC 7495, ring counter and Johnsons counter

**Aim**

1) To verify the parallel operation, shift left, rotate left, shift right and rotate right operations and serial input operations on IC 7495.

2) To verify the operation of a ring counter and Johnsons counter.

**Shift register**

A flip flop can be used to store one bit. A series of flip flops connected in cascade is used to store a word. Such a cascade of flip flops is called a register. A shift register is one, in which the information stored can be shifted one position at a time when one clock pulse is applied. The data can be shifted in either direction (left or right). The shift register can be used in four different configurations depending upon the way in which the data is entered into and taken out of it. These are:

1) Serial input, serial output.
2) Parallel input, serial output.
3) Serial input, parallel output.
4) Parallel input, parallel output.

**IC 7495**

IC 7495 is a 4-bit shift register. The data can be entered both in serial as well as parallel form. The data can be shifted in the right or left direction. The pin details of the IC are given in fig 9.1.

**Procedure**

**(1) Parallel load operation.**

The truth table of IC 7495 is given as follows.

| INPUTS | | | | | | | | OUTPUTS | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Mode cntrl | clocks | | Serial | Parallel | | | | | | | | |
| M | Ck 1 Right | Ck 2 Left | | A | B | C | D | $Q_A$ | $Q_B$ | $Q_C$ | $Q_D$ | Operation |
| 1 | X | 1 | X | X | X | X | X | $Q_{AO}$ | $Q_{BO}$ | $Q_{CO}$ | $Q_{DO}$ | |
| 1 | X | ↓ | X | a | b | c | d | a | b | c | d | Parallel-in |
| 1 | X | ↓ | X | $Q_B$ | $Q_C$ | $Q_D$ | d | $Q_B$ | $Q_C$ | $Q_D$ | d | Shift-left |
| 0 | 1 | 0 | X | X | X | X | X | $Q_{AO}$ | $Q_{BO}$ | $Q_{CO}$ | $Q_{DO}$ | |
| 0 | ↓ | X | E | X | X | X | X | E | $Q_{An}$ | $Q_{Bn}$ | $Q_{Cn}$ | Serial-in Shift-right |

*Shift left operation requires external connection of $Q_B$ to A, $Q_C$ to C. Serial data is entered at input D.

**Notes:**

1. a, b, c, d, e are the levels of steady state inputs at A, B, C, D and serial inputs respectively.
2. $Q_{AO}$, $Q_{BO}$, $Q_{CO}$, $Q_{DO}$ are levels of $Q_A$, $Q_B$, $Q_C$, $Q_D$ respectively before the indicated steady state input conditions have been established.
3. $Q_{An}$, $Q_{Bn}$, $Q_{Cn}$, $Q_{Dn}$ are levels of $Q_A$, $Q_B$, $Q_C$, $Q_D$ respectively before the most recent high to low transition ( ↓ ) of the clock has been applied.
4. X means don't care (any input including transition).

The mode control and shift left pins are held high and the 4 bit data is fed to the A B C D input pins. The right shift and serial input pins can be high or low (don't care). Data gets loaded into the Register when the shift left pin is made low by applying a pulse. (i.e., during a high to low transition of the signal).

**(2) Shift right operation**

Connect mode control to logic 0 and apply serial data at the serial input terminal starting from LSB as in fig 9. 2. Apply clock pulse at right shift terminal pin 9 and observe the outputs $Q_A$, $Q_B$, $Q_C$ and $Q_D$. The output can be taken in parallel form or in serial form. For taking output in the serial form apply clock pulse to right shift terminal and take the output at $Q_D$.

**(3) Shift left operation**

Connect mode control to logic 1. Connect $Q_B$ to A, $Q_C$ to B and $Q_D$ to C as in fig 9. 3. Apply serial data at the D input starting from the MSB. Apply the clock pulses at left shift (pin.no.8) and observe the outputs at $Q_A$, $Q_B$, $Q_C$, $Q_D$. Verify the left shift operation. Parallel output can be obtained at $Q_A$, $Q_B$, $Q_C$, $Q_D$ and serial output at $Q_A$ with the clock pulse applied at left shift (pin.no.8).

**(4) Rotate operation (Right / Left)**

During the right shift operation, if $Q_D$ is connected to SERIAL-IN pin, the 4 bit sequence will rotate clockwise (right) i.e., $Q_A$ assumes $Q_D$ state, $Q_B$ that of $Q_A$, $Q_C$ that of $Q_B$, $Q_D$ that of $Q_C$.

For rotating the bits anticlockwise, the connections are the same as for the left shift operation except that $Q_A$ is now connected to the D input pin. For successive left shift operations the bit rotates anticlockwise.

**Ring Counter**

In the shift register (see fig.9.4), if the Q output of the last stage flip flop $Q_D$ (pin 10) is connected to the serial input (pin 1) a ring counter is obtained.

**Johnsons Counter**

In Johnsons counter, the $\bar{Q}$ output of the last stage flip flop $Q_D$ (pin 10) is connected to the serial input (pin 1). (See fig. 9.5)

**Questions**

(1) How will you cascade the IC 7495 to obtain an 8-bit shift register?

(2) How will you use a shift register to multiply or divide a number by 2?

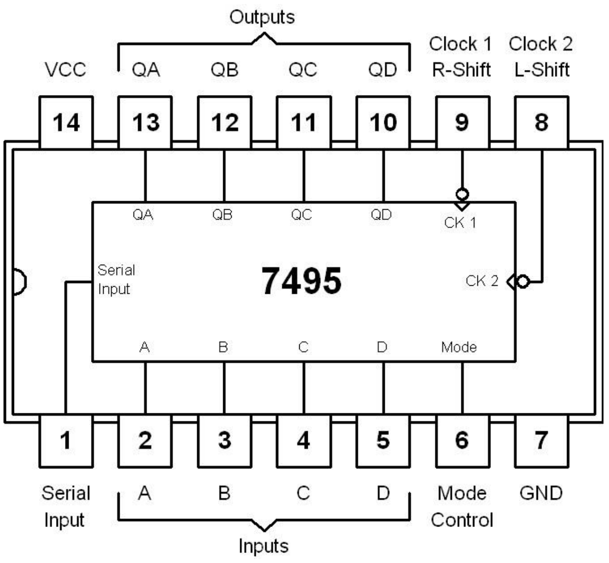


Fig. 9.1: Pin details of IC 7495

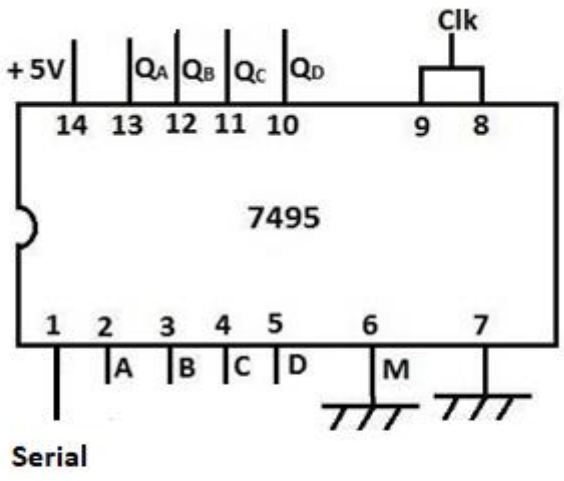


Fig. 9.2 : Right shift register using IC 7495

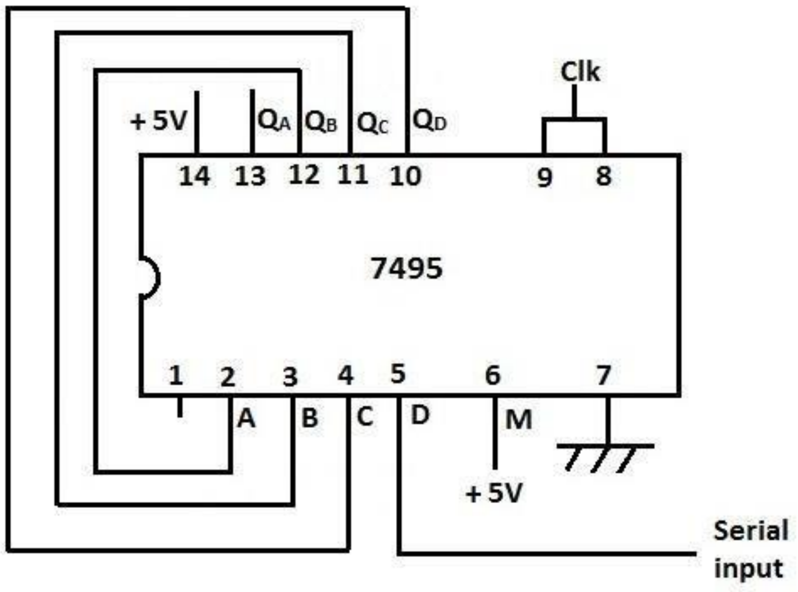


Fig. 9.3: Left shift register using 7495

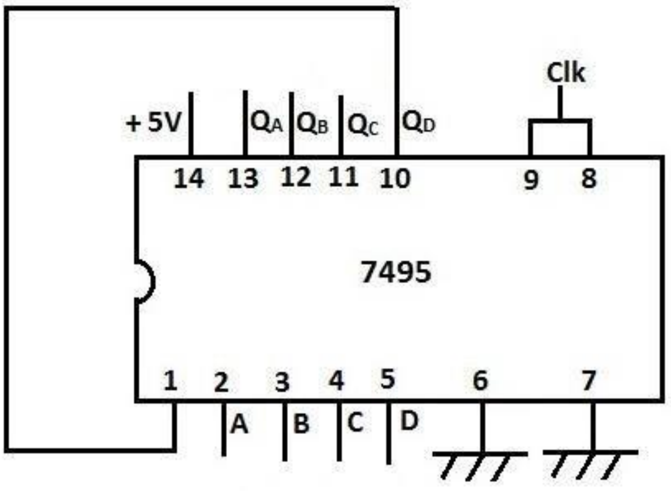


Fig. 9.4: Ring Counter using 7495

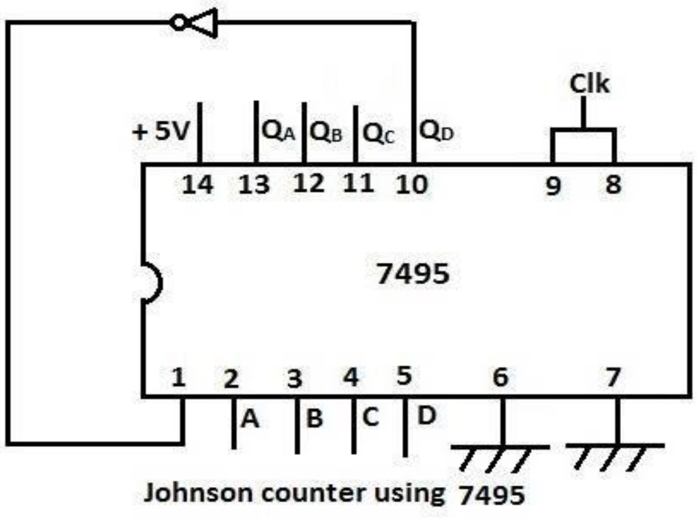


Fig. 9.5 : Johnson Counter using 7495

# 10. VHDL implementation of full adder, 4 bit magnitude comparator

**Aim**

1. To set up magnitude comparator using IC 7485 and verify its function table
2. To study VHDL implementation of Full Adder.
3. To study VHDL implementation of 4 bit magnitude comparator.

**Software used:**

XILINX 8.1 Software installed in a PC.

**Theory**

VHDL is an acronym for Very high speed integrated circuit (VHSIC) Hardware Description Language which is a programming language that describes a logic circuit by function, data flow behavior, and/or structure. This hardware description is used to configure a programmable logic device (PLD), such as a field programmable gate array (FPGA), with a custom logic design.

The general format of a VHDL program is built around the concept of BLOCKS which are the basic building units of a VHDL design. Within these design blocks a logic circuit of function can be easily described.

A VHDL design begins with an ENTITY block that describes the interface for the design. The interface defines the input and output 1ogic signals of the circuit being designed. The ARCHITECTURE block describes the internal operation of the design. Within these blocks are numerous other functional blocks used to build the design elements of the logic circuit being created.

After the design is created, it can be simulated and synthesized to check its logical operation. SIMULATION is a bare bones type of test to see if the basic logic works according to design and concept. SYNTHESIS allows timing factors and other influences of actual field programmable gate array (FPGA) devices to effect the simulation thereby doing a more thorough type of check before the design is committed to the FPGA or similar device.

## VHDL Program Structure

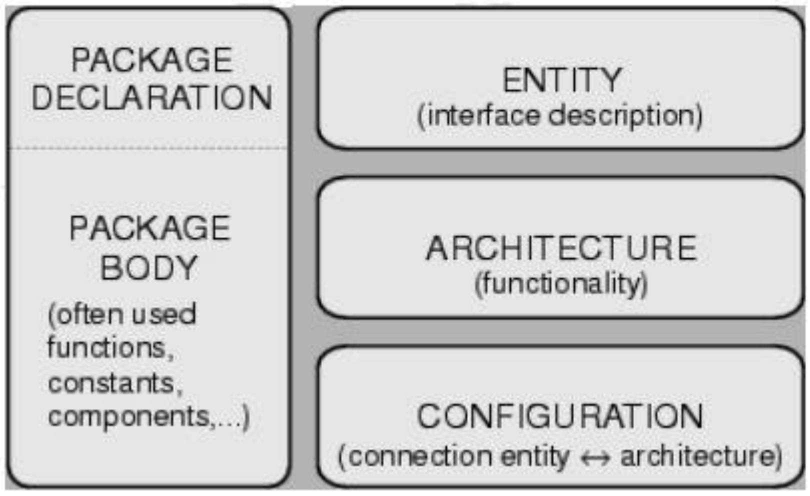


Fig. 10.1 VHDL Program Structure

**entity** entity-name **is**

[**port**(interface-signal-declaration);] **end [entity]**

[entity-name]; **architecture** architecture-name **of**

entity-name **is**

[declarations]

**Begin** architecture body **end**

**[architecture]** [architecture-name];

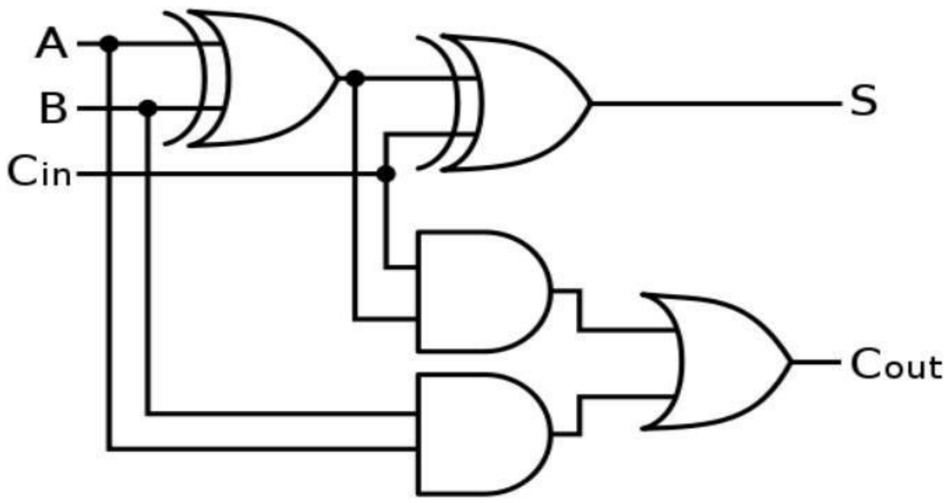


Fig. 10.1: Full Adder

Table 10.1: Truth table of full adder

| Inputs | | Outputs | | |
|---|---|---|---|---|
| A | B | $C_{in}$ | $C_{out}$ | S |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

**Program**

```
Library IEEE;
Use ieee.std_logic_1164_all;
Use ieee.std_logic_arith_all;
Entity FA_2 is
```

```
Port (a, b, cin: in bit; s, c: out bit);
End FA_2;
Architecture FA_2_beh of FA_2 is
Begin
Process (a, b,cin)
Begin
S<=a XOR B XOR Cin;
C<= (a and b) OR (a and cin) OR (b and cin);
End process;
End FA_2_beh;
```

**Questions**

1. What is full adder?

2. Using which gates we design the full adder?

## a) 4-Bit Magnitude Comparator using IC 7485

### IC7485

IC7485 is a 4 bit magnitude comparator. Two 4 bit numbers A = $A_3$, $A_2$, $A_1$, $A_0$ and B = $B_3$, $B_2$, $B_1$, $B_0$ can be compared to give output at one of the pins $>$ out, A = B out, A < B out, when the specified condition is satisfied. Words of greater lengths can be compared by connecting comparators in cascade. The $> , < ad =$ outputs of a stage handling less significant bits are connected to the corresponding $> , < ad =$ inputs of the next stage handling more significant bits. The stage handling the least significant bit must hav1e a high level applied to A = B input. The pin configuration and the function table are given below.

Table 10.2: Function table

| COMPARING INPUTS | | | | CASCADING INPUTS | | | OUTPUTS | | |
|---|---|---|---|---|---|---|---|---|---|
| A3,B3 | A2,B2 | A1,B1 | A0,B0 | A>B | A<B | A=B | A>B | A<B | A=B |
| A3>B3 | X | X | X | X | X | X | H | L | L |
| A3<B3 | X | X | X | X | X | X | L | H | L |
| A3=B3 | A2>B2 | X | X | X | X | X | H | L | L |
| A3=B3 | A2<B2 | A1>B1 | X | X | X | X | L | H | L |
| A3=B3 | A2=B2 | A1<B1 | X | X | X | X | H | L | L |
| A3=B3 | A2=B2 | A1=B1 | X | X | X | X | L | H | L |
| A3=B3 | A2=B2 | A1=B1 | A0>B0 | X | X | X | H | L | L |
| A3=B3 | A2=B2 | A1=B1 | A0<B0 | X | X | X | L | H | L |
| A3=B3 | A2=B2 | A1=B1 | A0=B0 | X | X | H | L | L | H |

**Procedure**

Keeping the A = B input high apply appropriate input and verify the truth table. For $A_n > B_n$ apply 1 to $A_n$ and 0 to $B_n$, similarly for $A_n < B_n$ apply 0 to $A_n$ and 1 to $B_n$. For $A_n = B_n$ try both the cases of $A_n = B_n = 0$ and $A_n = B_n = 1$.

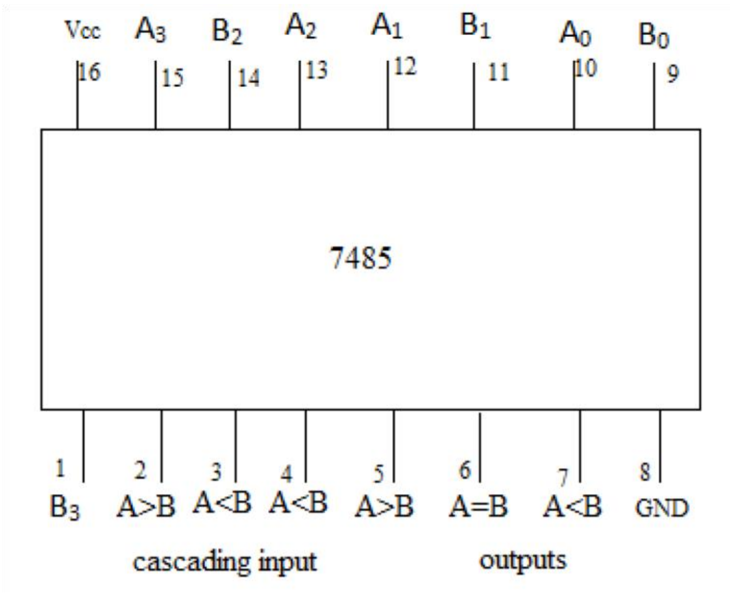


Fig. 10.2: Pin diagram IC7485

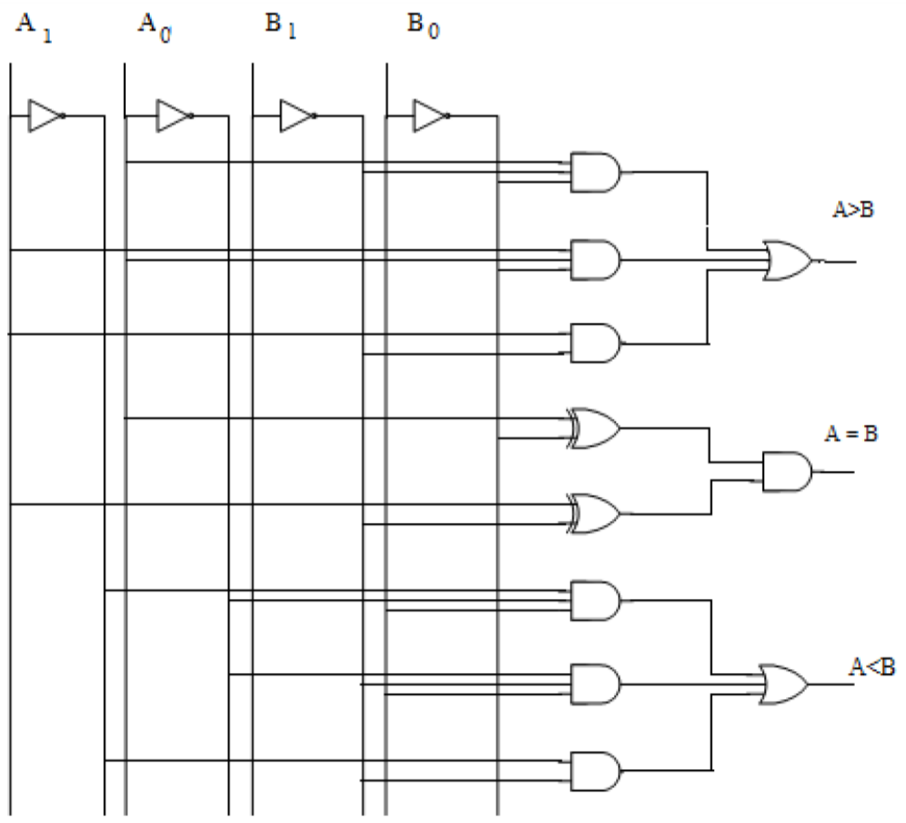


Fig. 10.3: Two bit magnitude comparator

**b) VHDL Implementation of 4-Bit Magnitude Comparator**

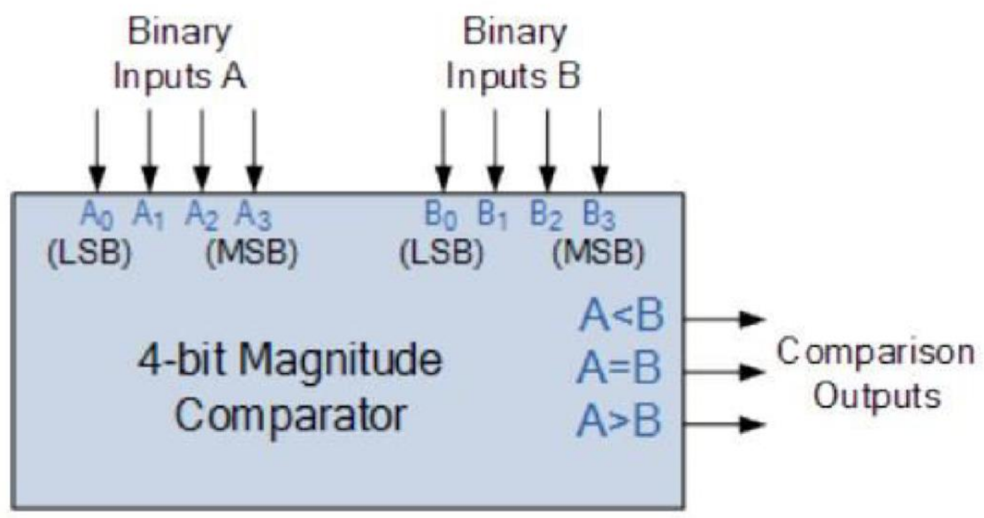


Fig. 10.4: 4-bit Magnitude Comparator

**Program:**

```
Library IEEE;
Use IEEE.std_logic_1164_all;
Use IEEE.std_logic_arith_all;
Entity COM_2 is
Port (a, b: in bit_vector (3 down to 0); z: out bit_vector (2 down to 0));
End COM_2;
Architecture COM_2_beh of COM_2 is
Begin
Process (a, b)
Begin
If (a=b) then
Z<='100';
Elsif (a<b) then
Z<='010';
Elsif (a>b) then
 Z<='001';
End if;
End process;
End COM_2_beh;
```

**Precaution**

Make sure that there is no syntax and semantic error.

**Questions**

1. What is comparator?

2. What are uses of comparator?

3. What is the voltage comparator?

4. What is the no. of outputs in 4- bit comparator?

# II. MICROPROCESSOR AND EMBEDDED SYSTEMS EXPERIMENTS

# 1. Data Transfer Instructions using Different Addressing Modes and Block Transfer

1. Write an ALP for loading registers A, B, C, D, E, H and L with single byte data addressing using immediate addressing

| MEMORY ADDRESS | MACHINE CODES | LABEL | OPCODE | OPERAND | COMMENTS |
|---|---|---|---|---|---|
| 2000 | 3E 01 | START | MVI | A,01 | Load A with 01 |
| 2002 | 06 02 | | MVI | B,02 | Load B with 02 |
| 2004 | 0E 03 | | MVI | C,03 | Load C with 03 |
| 2006 | 16 04 | | MVI | D,04 | Load D with 04 |
| 2008 | 1E 05 | | MVI | E,05 | Load E with 05 |
| 200A | 26 06 | | MVI | H,06 | Load H with 06 |
| 200C | 2E 07 | | MVI | L,07 | Load L with 07 |
| 200E | EF | END | RST | 05 | Return to monitor program |

2. Write an ALP for loading registers B, C, D, E, H and L with same data using register addressing

| MEMORY ADDRESS | MACHINE CODES | LABEL | OPCODE | OPERAND | COMMENTS |
|---|---|---|---|---|---|
| 2020 | 3A 50 20 | START | LDA | 2050 | Load accumulator with 2050 |
| 2023 | 47 | | MOV | B, A | Move the content of A to B |
| 2024 | 4F | | MOV | C, A | Move the content of A to C |
| 2025 | 57 | | MOV | D, A | Move the content of A to D |
| 2026 | 5F | | MOV | E, A | Move the content of A to E |
| 2027 | 67 | | MOV | H, A | Move the content of A to H |
| 2028 | 6F | | MOV | L, A | Move the content of A to L |
| 2029 | EF | END | RST | 05 | Return to monitor program |

3. Write an ALP for loading register pairs BC, DE and HL with 16-bit data using immediate addressing

| MEMORY ADDRESS | MACHINE CODE | LABEL | OPCODE | OPERAND | COMMENTS |
|---|---|---|---|---|---|
| 2050 | 01 50 21 | START | LXI | B, 2150 | Load BC register pair with data 2150 |
| 2053 | 11 51 21 | | LXI | D, 2151 | Load DE register pair with data 2151 |
| 2056 | 21 52 21 | | LXI | H, 2152 | Load HL register pair with data 2152 |
| 2059 | EF | END | RST | 05 | Return to monitor program |

4. Write an ALP to copy a block of data from 4 memory locations to another 4 memory locations using 8-bit data transfer addressing mode direct addressing.

| MEMORY ADDRESS | MACHINE CODES | LABEL | OPCODE | OPERAND | COMMENTS |
|---|---|---|---|---|---|
| 2060 | 3A 50 22 | START | LDA | 2250 | Load accumulator with 2250 |
| 2063 | 32 54 22 | | STA | 2254 | Accumulator content stored in 2254 |
| 2066 | 3A 51 22 | | LDA | 2251 | Load data in 2251 to accumulator |
| 2069 | 32 55 22 | | STA | 2255 | Accumulator data stored in 2255 |
| 206C | 3A 52 22 | | LDA | 2252 | Load data in 2252 to accumulator |
| 206F | 32 56 22 | | STA | 2256 | Accumulator data stored in 2256 |
| 2072 | 3A 53 22 | | LDA | 2253 | Load data in 2253 to accumulator |

| | | | | | |
|---|---|---|---|---|---|
| 2075 | 32 57 22 | | STA | 2257 | Accumulator data stored in 2257 |
| 2078 | EF | END | RST | 05 | Return to monitor program |

5. Repeat 4th ALP using 16-bit data transfer addressing mode direct addressing

| MEMORY ADDRESS | MACHINE CODE | LABEL | OPCODE | OPERAND | COMMENTS |
|---|---|---|---|---|---|
| 2080 | 2A 50 20 | START | LHLD | 2050 | Data in 2050 to L register and data in 2051 to H register |
| 2083 | 22 54 20 | | SHLD | 2054 | L register content to 2054 and H register content to 2055 |
| 2086 | 2A 52 20 | | LHLD | 2052 | Data in 2052 to L register and data in 2053 to H register |
| 2089 | 22 56 20 | | SHLD | 2056 | L register content to 2056 and H register content to 2057 |
| 208C | EF | END | RST | 05 | Return to monitor program |

6. Repeat 4th ALP using 16-bit data transfer addressing mode indirect addressing

| MEMORY ADDRESS | MACHINE CODE | LABEL | OPCODE | OPERAND | COMMENTS |
|---|---|---|---|---|---|
| 2093 | 21 50 20 | START | LXI | H, 2050 | Initialize HL pair |
| 2096 | 01 51 20 | | LXI | B, 2051 | Initialize BC pair |
| 2099 | 11 55 20 | | LXI | D, 2055 | Initialize DE pair |
| 209C | 36 04 | | MVI | M, 04 | Set counter as 4 |

| | | | | | |
|---|---|---|---|---|---|
| 209E | 0A | LOOP | LDAX | B | Load content of memory location whose address is in BC pair to accumulator |
| 209F | 12 | | STAX | D | Store content of accumulator into memory location whose address is in DE pair |
| 20A0 | 03 | | INX | B | Increment BC pair |
| 20A1 | 13 | | INX | D | Increment DE pair |
| 20A2 | 35 | | DCR | M | Decrement count by 1 |
| 20A3 | C2 9E 20 | | JNZ | LOOP | Jump if non zero |
| 20A6 | EF | END | RST | 05 | Return to monitor program |

# 2. Arithmetic Operations in Binary and BCD- Addition, Subtraction, Multiplication and Division

## 1. Addition and subtraction of 8 bit numbers

**Aim**

To evaluate the expression X + Y – Z, where X, Y and Z are 8 bit numbers stored in memory.

**Theory**

Read the numbers X, Y and X from memory to register. Evaluate the expression. Store the result in the memory.

**Program**

| MEMORY ADDRESS | MACHINE CODE | LABEL | OPCODE | OPERAND | COMMENTS |
|---|---|---|---|---|---|
| 2000 | 21 00 21 | START | LXI | H,2100 | Initialize memory pointer |
| 2003 | 7E | | MOV | A, M | Load X in register A |
| 2004 | 23 | | INX | H | Pointer to Y |
| 2005 | 46 | | MOV | B, M | Load Y in register B |
| 2006 | 23 | | INX | H | Pointer to Z |
| 2007 | 4E | | MOV | C, M | Load Z in register C |
| 2008 | 88 | | ADD | B | Sum X+Y in register A |
| 2009 | 91 | | SUB | C | X+Y-Z in register A |
| 200A | 23 | | INX | H | Pointer to store the result |
| 200B | 77 | | MOV | M, A | Store result to memory |
| 200C | EF | END | RST | 05 | Return to monitor program |

**Observations**

| | Memory location | Data |
|---|---|---|
| Input | | |
| | | |
| | | |
| Output | | |

## 2. BCD Addition

**Aim**

To find the sum of two 8 bit (2 digit) BCD numbers.

**Theory**

Add the two BCD numbers using ADD and adjust using DAA.

**Program**

| MEMORY ADDRESS | MACHINE CODE | LABEL | OPCODE | OPERAND | COMMENTS |
|---|---|---|---|---|---|
| 2060 | 21 00 21 | START | LXI | H,2100 | Initialize memory pointer |
| 2063 | 7E | | MOV | A, M | Load first data |
| 2064 | 23 | | INX | H | Pointer to second data |
| 2065 | 86 | | ADD | M | Add both number |
| 2066 | 27 | | DAA | | Convert sum to BCD |
| 2067 | 23 | | INX | H | Pointer to save sum |
| 2068 | 77 | | MOV | M, A | Store sum in memory |
| 2069 | EF | END | RST | 05 | Return to monitor program |

**OBSERVATIONS**

| | Memory location | Data |
|---|---|---|
| Input | | |
| | | |
| Output | | |

## 3. Multi Precision Subtraction

**Aim**

To find the difference of two 16 bit numbers stored in memory.

**Theory**

The two 16 bit numbers are stored in consecutive memory locations. Lower byte of first number is stored in the first memory location, then higher byte of the first number, then lower and higher byte of second number, Lower byte of second number is first subtracted from the lower byte of first number. Then higher byte of second number is subtracted with borrow from higher byte of first number.

**Program**

| MEMORY ADDRESS | MACHINE CODE | LABEL | OPCODE | OPERAND | COMMENTS |
|---|---|---|---|---|---|
| 2000 | 21 00 21 | START | LXI | H,2100 | Initialize memory pointer |
| 2003 | 7E | | MOV | A, M | Load X in register A |
| 2004 | 23 | | INX | H | Pointer to higher digits if first number |
| 2005 | 46 | | MOV | B, M | Higher digits in B |
| 2006 | 23 | | INX | H | Pointer to lower digits of second number |
| 2007 | 4E | | MOV | C, M | Lower digits in C |
| 2008 | 23 | | INX | H | Pointer to higher digits of second number |

| | | | | | |
|---|---|---|---|---|---|
| 2009 | 56 | | MOV | D, M | Higher digits in D |
| 200A | 91 | | SUB | C | Subtract lower digits |
| 200B | 23 | | INX | H | Pointer to save result |
| 200C | 77 | | MOV | M, A | Store lower result |
| 200D | 78 | | MOV | A, B | Higher digits in A |
| 200E | 9A | | SBB | D | Subtract higher digit with borrow |
| 200F | 23 | | INX | H | Pointer to save result |
| 2010 | 77 | | MOV | M, A | Store higher result |
| 2011 | EF | END | RST | 05 | Return to monitor program |

**Observations**

| | Memory location | Data |
|---|---|---|
| Input | | |
| | | |
| | | |
| | | |
| Output | | |
| | | |

## 4. Binary Multiplication

**Aim**

To perform the multiplication of the two binary numbers by repeated addition and shift & add method

### Repeated addition method

**Algorithm**

i. Start
ii. Get the multiplicand

iii. Get the multiplier
iv. Initialize product <- 0
v. Product <- product + multiplicand and multiplier <- multiplier – 1
vi. If multiplier = 0, go to step 7 else go to step 5
vii. Store the result
viii. Stop

**Program**

| MEMORY ADDRESS | MACHINE CODE | LABEL | OPCODE | OPERAND | COMMENTS |
|---|---|---|---|---|---|
| 2000 | 21 02 21 | START | LXI | H,2102 | Initialize memory pointer |
| 2003 | 46 | | MOV | B, M | Load multiplier in B register |
| 2004 | 11 00 00 | | LXI | D, 0000 | Initialize DE pair as 0000 |
| 2007 | 2A 00 21 | | LHLD | 2100 | Load data in HL pair |
| 200A | EB | | XCHG | | Exchange DE with HL pair |
| 200B | 19 | LP: | DAD | D | Add |
| 200C | 05 | | DCR | B | Decrement register B |
| 200D | C2 0B 20 | | JNZ | LP | If not 0 go to LP |
| 2010 | 22 03 21 | | SHLD | 2103 | |
| 2013 | EF | END | RST | 5 | Software interrupt |

**Observations**

| | Memory location | Data |
|---|---|---|
| Input | | |
| | | |
| | | |
| Output | | |
| | | |

**<u>Shift and add method</u>**

**Algorithm**

1. Start

2. Get the multiplier and multiplicand. Set bit counter equal to number of bits
3. Shift multiplier left by one bit
4. If MSB of the multiplier = 1, go to step 5 else go to step 6
5. Add multiplicand to partial product
6. Decrement bit counter. Shift partial sum left.
7. If bit counter = 0, store the result else go to step 3
8. Stop

**Program description**

Consider the example of multiplying two nibbles

```
Multiplicand 1010 x
Multiplier   0101
             ………
             0000
              1010
               0000
                1010
             …………….
             0110010
```

This explains that if the MSB of the multiplier is 1, partial product is shifted to left and the multiplicand is added to the partial product. If the bit of the multiplier is zero, only shifting is done.

**Program**

| MEMORY ADDRESS | MACHINE CODE | LABEL | OPCODE | OPERAND | COMMENTS |
|---|---|---|---|---|---|
| 2000 | 2A 50 21 | START | LHLD | 2150 | Place content of 2150 in L (multiplicand) and 2151 in H(Multiplier) |
| 2003 | EB | | XCHG | | HL to DE |
| 2004 | 7A | | MOV | A, D | Multiplier D to A and multiplicand in E |
| 2005 | 21 00 00 | | LXI | H, 0000 | Clear HL |
| 2008 | 06 08 | | MVI | B, 08 | Register B to rotation count (8) |
| 200A | 16 00 | | MVI | D, 00 | Initialize D |

| | | | | | |
|---|---|---|---|---|---|
| 200C | 1F | NXT BIT | RAR | | Check if multiplicand bit is 1 |
| 200D | D2 11 20 | | JNC | NO ADD | If not stop adding |
| 2010 | 19 | | DAD | D | Add multiplicand to HL |
| 2011 | EB | NO ADD | XCHG | | Place multiplicand in HL |
| 2012 | 29 | | DAD | H | And shift left |
| 2013 | EB | | XCHG | | Retrieve shifted multiplicand |
| 2014 | 05 | | DCR | B | Decrement counter |
| 2015 | C2 0C 20 | | JNZ | NXT BIT | |
| 2018 | 22 52 21 | | SHLD | 2152 | Store result in location 2152 & 2153 |
| 201B | CF | END | | | |

**Observations**

| | Memory location | Data |
|---|---|---|
| Input | | |
| | | |
| Output | | |
| | | |

## 5. Binary Division

### Aim

To write an assembly language program for binary division. The 16 bit dividend is in memory location 2100H and 2101H and the divisor in 2102H. The quotient and the remainder should be stored in 2103H and 2104H respectively.

### Algorithm

1. Start

2. Get the dividend and divisor, initialize counter for 8 bits
3. Shift left the dividend and quotient
4. If divisor > dividend, go to step 6 else go to step 5
5. Dividend = dividend – divisor
6. Decrement counter
7. If count = 0, store the result, else go to step 3
8. Stop

**Program**

| MEMORY ADDRESS | MACHINE CODE | LABEL | OPCODE | OPERAND | COMMENTS |
|---|---|---|---|---|---|
| 2000 | 21 00 21 | START | LXI | H, 2100 | Initialize HL pair as memory pointer |
| 2003 | 46 | | MOV | B, M | Load divisor in B |
| 2004 | 23 | | INX | H | Increment HL |
| 2005 | 7E | | MOV | A, M | Load dividend to accumulator |
| 2006 | 23 | | INX | H | Increment HL |
| 2007 | 0E 00 | | MVI | C, 00 | Initialize quotient as 00 |
| 2009 | B8 | | CMP | B | |
| 200A | DA 13 20 | | JC | LP | |
| 200D | 90 | LP1 | SUB | B | Subtract dividend and divisor |
| 200E | 0C | | INR | C | Increment contents of C |
| 200F | B8 | | CMP | B | Is dividend less than divisor |
| 2010 | D2 0D 20 | | JNC | LP1 | If not jump to LP1 |
| 2013 | 77 | LP | MOV | M, A | Store remainder at 2102 |
| 2014 | 23 | | INX | H | Increment HL |
| 2015 | 71 | | MOV | M, C | Store quotient at 2103 |
| 2016 | EF | END | RST | 05 | Software interrupt |

## Observations

|  | Memory location | Data |
|---|---|---|
| Input |  |  |
|  |  |  |
| Output |  |  |
|  |  |  |

# 3. Logical Instructions – Sorting of Arrays

## 1. Sorting of arrays in ascending order

### Aim

To sort ten bytes of data initially stored in memory location starting from XX00H onwards, in ascending order.

### Algorithm

1. Start
2. Initialize cycle counter, compare counter and address pointer
3. Bring first data into accumulator
4. If accumulator < next data, go to step 6 else go to step 5
5. Exchange data
6. Decrement comparison counter
7. If comparison counter = 0, go to step 8 else go to step 4
8. Decrement cycle counter
9. If cycle counter = 0, stop else go to step 3

### Program description

Program uses the bubble sort technique. In this type of sorting, first and second data will be compared and the bigger will be kept in the third address and so on. After on cycle (ie., N-1 comparisons), largest number will be kept in the last address. In the second cycle of bubble sort (ie., N-2 comparisons) second largest number will be stored in the last but one address.

### Program

| MEMORY ADDRESS | MACHINE CODE | LABEL | OPCODE | OPERAND | COMMENTS |
|---|---|---|---|---|---|
| 2000 | 21 00 21 | LP2 | LXI | H, 2100 | Initialize HL pair as memory pointer |
| 2003 | 0E 07 | | MVI | C, 07 | Set counter as 8 |
| 2005 | 06 00 | | MVI | B, 00 | Set B = 0 |
| 2007 | 7E | LP1 | MOV | A, M | Get first data in accumulator |
| 2008 | 23 | | INX | H | Increment HL |
| 2009 | BE | | CMP | M | Compare first and second |
| 200A | DA 14 20 | | JC | LP | |

| | | | | | |
|---|---|---|---|---|---|
| 200D | 56 | | MOV | D, M | |
| 200E | 77 | | MOV | M, A | |
| 200F | 2B | | DCX | H | |
| 2010 | 72 | | MOV | M, D | |
| 2011 | 23 | | INX | H | Increment HL |
| 2012 | 06 01 | | MVI | B, 01 | |
| 2014 | 0D | LP | DCR | C | |
| 2015 | C2 07 20 | | JNZ | LP1 | |
| 2018 | 05 | | DCR | B | |
| 2019 | CA 00 20 | | JZ | LP2 | |
| 201C | EF | END | RST | 05 | Software interrupts |

**Observations**

| | Memory location | Data |
|---|---|---|
| Input | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| Output | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

## 2. Sorting of array in descending order

**Aim**

Modify the above algorithm to sort in descending order by changing the step 4 (If accumulator > next data, go to step 6 else go to step 5)

**Program**

| MEMORY ADDRESS | MACHINE CODE | LABEL | OPCODE | OPERAND | COMMENTS |
|---|---|---|---|---|---|
| 2000 | 21 00 21 | LP2 | LXI | H, 2100 | Initialize HL pair as memory pointer |
| 2003 | 0E 07 | | MVI | C, 07 | Set counter as 8 |
| 2005 | 06 00 | | MVI | B, 00 | Set B = 0 |
| 2007 | 7E | LP1 | MOV | A, M | Get first data in accumulator |
| 2008 | 23 | | INX | H | Increment HL |
| 2009 | BE | | CMP | M | Compare first and second |
| 200A | D2 14 20 | | JNC | LP | |
| 200D | 56 | | MOV | D, M | |
| 200E | 77 | | MOV | M, A | |
| 200F | 2B | | DCX | H | |
| 2010 | 72 | | MOV | M, D | |
| 2011 | 23 | | INX | H | Increment HL |
| 2012 | 06 01 | | MVI | B, 01 | |
| 2014 | 0D | LP | DCR | C | |
| 2015 | C2 07 20 | | JNZ | LP1 | |
| 2018 | 05 | | DCR | B | |
| 2019 | CA 00 20 | | JZ | LP2 | |
| 201C | EF | END | RST | 05 | Software interrupts |

**Observations**

| | Memory location | Data |
|---|---|---|
| Input | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| Output | | |
| | | |
| | | |
| | | |
| | | |

# 4. Binary to BCD conversion and vice versa

## 1. Binary to BCD conversion

### Aim

Write an assembly language program to convert 8-bit binary to BCD.

### Algorithm

1. Start
2. Set pointer of datum and initialize counter
3. Move datum to accumulator
4. Subtract 64H from accumulator till result becomes negative
5. Cancel the last subtraction and store the quotient and the remainder
6. Divide the remainder further by 10, by repeated subtraction by 0A
7. Store the two quotients and the final remainder
8. Stop

### Program description

0 to 255 is the range of binary numbers allowed in this program. The binary number will be broken down into hundred, tens and units.

### Program

| MEMORY ADDRESS | MACHINE CODES | LABEL | OPCODE | OPERAND | COMMENTS |
|---|---|---|---|---|---|
| 2000 | 3A 00 F1 | START | LDA | F100H | |
| 2003 | 47 | | MOV | B, A | Move the content of A to B |
| 2004 | 16 64 | | MVI | D, 64H | |
| 2006 | CD 1A 20 | | CALL | BCD | |
| 2009 | 61 | | MOV | H, C | Move the content of C to H |
| 200A | 16 0A | | MVI | D, 0AH | |
| 200C | CD 1A | | CALL | BCD | |
| 200F | 79 | | MOV | A, C | |
| 2010 | 07 | | RLC | | |
| 2011 | 07 | | RLC | | |
| 2012 | 07 | | RLC | | |
| 2013 | 07 | | RLC | | |
| 2014 | B0 | | ORA | B | |
| 2015 | 6F | | MOV | L, A | |
| 2016 | 22 01 F1 | | SHLD | F101H | |

| | | | | | |
|---|---|---|---|---|---|
| 2019 | 76 | | HLT | | |
| 201A | 0E 00 | BCD | MVI | C, 00H | |
| 201C | 78 | | MOV | A, B | |
| 201D | 92 | RPTS | SUB | D | |
| 201E | DA 25 20 | | JC | NC | |
| 2021 | 0C | | INR | C | |
| 2022 | C3 1D 20 | | JMP | RPTS | |
| 2025 | 82 | NC | ADD | D | |
| 2026 | 47 | | MOV | B, A | |
| 2027 | C9 | | RET | | |

**Observations**

| | Memory location | Data |
|---|---|---|
| Input | | |
| Output | | |
| | | |

**2. BCD to binary conversion**

**Aim**

Write an assembly language program to convert BCD data to binary data using 8085 microprocessor kit.

**Algorithm**

1. Start the microprocessor
2. Get the BCD data in accumulator and save it in register ‘E’
3. Mark the lower nibble of BCD data in accumulator
4. Rotate upper nibble to lower nibble and save it in register ‘B’
5. Clear the accumulator
6. Move 0AH to ‘C’ register
7. Add ‘A’ and ‘B’ register
8. Decrement ‘C’ register. If zf = 0, go to step 7
9. Save the product in ‘B’
10. Get the BCD data in accumulator from ‘E’ register and mark the upper nibble
11. Add the units (A-ug) to product (B-ug)
12. Store the binary value in memory
13. End the program

**Program**

| MEMORY ADDRESS | MACHINE CODES | LABEL | OPCODE | OPERAND | COMMENTS |
|---|---|---|---|---|---|
| 4100 | 3A 00 42 | START | LDA | 4200 | Get the data in ‘A’ |
| 4103 | 5E | | MOV | E, A | Save in E register |
| 4104 | E6 F0 | | ANI | F0 | Mark the lower nibble |
| 4106 | 07 | | RLC | | Rotate the upper |
| 4107 | 07 | | RLC | | To lower nibble |
| 4108 | 07 | | RLC | | And save in |
| 4109 | 07 | | RLC | | Register B |
| 410A | 47 | | MOV | B, A | Move it from A to B |
| 410B | AF | | XRA | A | Clear the accumulator |
| 410C | 0E 0A | | MVI | C, 0A | Initialize C as ‘0A’ |
| 410E | 08 | | REP | | |
| 410F | 0D | | DCR | C | Decrement C register |
| 4110 | C2 0E 41 | | JNZ | | Jump till value C is 0 |
| 4113 | 47 | | MOV | B, A | Move the value in A to B |
| 4114 | 7B | | MOV | A, E | Get the BCD in A |
| 4115 | E6 0F | | ANI | 0F | Mark the upper nibble |
| 4117 | 80 | | ADD | B | Add A and B |
| 4118 | 32 01 42 | | STA | 4201 | Save the binary data |
| 411B | 76 | | HLT | | Stop |

**Sample input**

| Input address | Value |
|---|---|
| 4200 | 68 |

**Sample output**

| Output address | Value |
|---|---|
| 4201 | 44 |

# 5. Interfacing D/A Converter - Generation of simple waveforms

## 1. Generate a triangular wave

### Aim

Generate a triangular wave of suitable frequency using DAC interface card.

### Theory

A digital number can be converted into an analog number by selectively adding voltage which is proportional to the weight of each binary digit. Different waveforms can be generated using this DAC-0800 module.

### Circuit description

Port A and Port B are connected to channel 1 and channel 2. A reference voltage of 8V is generated using 723 and is given to verify points of DAC 0800. The standard output voltage will be 7.98V when FE is outputted and will be OV when 00 is outputted. The output of DAC-0800 is fed to the operational amplifier to get the final output as X OUT and Y OUT

The DAC interface can be used to generate various waveforms using a microprocessor. In most of the DAC cards the digital outputs from the port A and port B of 8255 are separately converted to analog signals by DAC.

The reference voltage needed for the DACs can be obtained from an on-board voltage regulator. The output from the DACs vary between 0-7.98V corresponding to values between 00 to FF.

To use DAC initialize 8255 in mode 0 with port A and port B as output ports. Output the data

on the appropriate port, and observe the output waveform on an oscilloscope.

### Algorithm

1. Initialize digital data 00.
2. Increment data by 1.
3. If data is not equal to FF, go to step 2, 4. Decrement data by 1.
5. If data is not equal to 00, go to step4,
6. Loop to step 2.

**Program**

| MEMORY ADDRESS | MACHINE CODE | LABEL | OPCODE | OPERAND | COMMENTS |
|---|---|---|---|---|---|
| 2000 | 3E 80 | START | MVI | A, 80H | Initialize 8255 mode 0 |
| 2002 | D3 03 | | OUT | 03H | Port A and Port B are outputs |
| 2004 | AF | | XRA | A | Start value 00H |
| 2005 | D3 00 | LOOP1 | OUT | 00H | Out to DAC |
| 2007 | 3C | | INR | A | Increment DAC input |
| 2008 | FE FF | | CPI | FFH | Check for peak value |
| 200A | C2 05 20 | | JNZ | LOOP1 | No loop back |
| 200D | D3 00 | LOOP2 | OUT | 00H | Out to DAC |
| 200F | 3D | | DCR | A | Decrement the DAC input |
| 2010 | C2 0D 20 | | JNZ | LOOP2 | Minimum value not reached loop back |
| 2013 | C3 05 20 | END | JMP | LOOP1 | Repeat |

**Procedure**

1. Connect the interface card to the microprocessor kit and CRO
2. Connect the output of DAC to channel 1 of CRO
3. Enter and execute the program. Observe the triangular output waveform on the CRO.

## 2. Generate a ramp wave

**Aim**

Generate a ramp wave using a microprocessor kit and a DAC interface card.

**Algorithm**

1. Initialize digital data 00
2. Increment data by 1

3. Loop 2 **Program**

| MEMORY ADDRESS | MACHINE CODES | LABEL | OPCODE | OPERAND | COMMENTS |
|---|---|---|---|---|---|
| 2000 | 3E 80 | START | MVI | A, 80H | Initialize 8255 |
| 2002 | D3 03 | | OUT | 03H | |
| 2004 | 3E 00 | | MVI | A, 00H | |
| 2006 | D3 00 | BACK | OUT | 00H | |
| 2008 | 3C | | INR | A | |
| 2009 | C3 06 20 | END | JMP | BACK | |

# 6. INTERFACING A/D CONVERTER

**Aim**

To convert an analog 0-5V signal to 8bit digital value (00 to FF) and display using a sevensegment display.

**Theory**

This circuit requires an ADC IC (0809), Programmable peripheral interface(PPI)8255. The ADC 0809 is an 8-bit digital to analog converter with 8 channel inbuilt multiplexes and it converts analog voltage input Vi to an 8-bit digital output(D7-Do), It uses the principle of successive approximation technique for conversion process. A pulse applied to the ADC's Start of conversion terminal initiates the conversion process. The time taken for completion of conversion is called conversion time. During $t_c$ the conversion process is taking place, ADC's end of conversion (EOC) output go low. The EOC output returns high only when the conversion is complete.

Features of 0809

- Resolution - 8 bits
- Conversion Time - 100ps at 640KHz
- Single supply voltage - +5V dc
- An8-channel multiplexer with latched control logic
- 0 to 5V analog input voltage
- Clock frequency 10KHz to 1280 KHz
- Conversion delay time (8 CLK period +2ms)

The heart of the single chip data conversion system is its 8-bit analog to digital converter it is having 3 major sections 256 R ladder network, successive approximation register and comparator.

Control word for 8255

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |

= 98H

- Mode 0
- Port A Input port
- Port B Output
- Port Cu Input port
- Port $C_L$ Output port

**Algorithm**

1. Initialize 8255 Port A as input, Port B as output, Port Cu as Input port and Port CL as Output port.

2. Select Channel 0 by sending address 00 through Port B.
3. Make ALE & SOC low, then high and again low to give a pulse at ALE and SOC.
4. Check EOC signal.
5. Is EOC high? If zero, loop to 4.
6. Read input data from Port A.
7. Store and display.
8. Loop to step 2.

**Program**

| MEMORY ADDRESS | MACHINE CODES | LABEL | OPCODE | OPERAND |
|---|---|---|---|---|
| 2000 | 31 FF 20 | START: | LXI | SP,20FFH |
| 2003 | 3E 98 | | MVI | A,98H |
| 2005 | D3 03 | | OUT | 03H |
| 2007 | 3E 00 | LOOP1: | MV1 | A,00H |
| 2009 | D3 01 | | OUT | 01H |
| 200B | 3E 00 | | MVI | A,00H |
| 200D | D3 02 | | OUT | 02H |
| 200F | 3E 03 | | MVI | A,03H |
| 2011 | D3 02 | | OUT | 02H |
| 2013 | 3E 00 | | MVI | A,00H |
| 2015 | D3 02 | | OUT | 02H |
| 2017 | DB 02 | LOOP2: | IN | 02H |
| 2019 | E6 10 | | ANI | 10H |
| 201B | CA 17 20 | | JZ | LOOP2 |
| 201E | 3E 04 | | MVI | A,04H |
| 2020 | D3 02 | | OUT | 02H |
| 2022 | DB 00 | | IN | 00H |
| 2024 | 32 F6 27 | | STA | 27F6H |
| 2027 | CD 47 03 | | CALL | 0347H |
| 202A | 11 00 00 | | LXI | D,0000H |
| 202D | CD BC 03 | | CALL | 03BCH |
| 2030 | CD FA 06 | | CALL | 06FAH |
| 2033 | 11 00 00 | | LXI | D,0000H |
| 2036 | CD BC 03 | | CALL | 03BCH |
| 2039 | C3 07 20 | END: | JMP | LOOP1 |

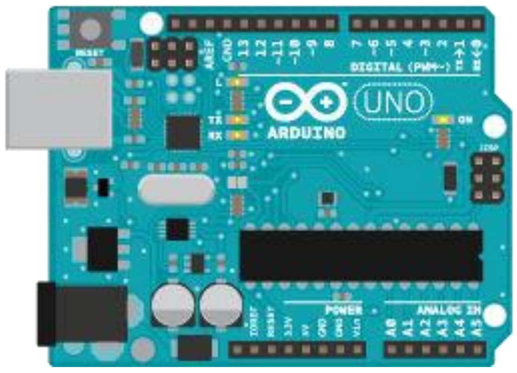
## Observations

| Analog Input | Digital Output |
|---|---|
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

## About arduino

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

Over the years Arduino has been the brain of thousands of projects, from everyday objects to complex scientific instruments. A worldwide community of makers - students, hobbyists, artists, programmers, and professionals - has gathered around this open-source platform, their contributions have added up to an incredible amount of accessible knowledge that can be of great help to novices and experts alike.



Arduino was born at the Ivrea Interaction Design Institute as an easy tool for fast prototyping, aimed at students without a background in electronics and programming. As soon as it reached a wider community, the Arduino board started changing to adapt to new needs and challenges, differentiating its offer from simple 8-bit boards to products for IoT applications, wearable, 3D printing, and embedded environments. All Arduino boards are completely open-source, empowering users to build them independently and eventually adapt them to their particular needs. The software, too, is open-source, and it is growing through the contributions of users worldwide.

# 7. Blinking LED

**Aim**

Write a program to blink LED using Arduino.

**Theory**

Arduino programs are written in the Arduino Integrated Environment(IDE). Arduino IDE is a special software running on your system that allows you to write sketches(synonym for program in Arduino language) for different Arduino boards. Arduino UNO is an entry level Arduino board with enough memory and processing capabilities. Pins 0 to 13 are digital pins, and any one of these can be connected to LED( say pin no. 8). A 680Ω resistor is connected in series to limit the current through the LED.

**Circuit description**

Connect the Arduino hardware to the computer via the USB cable provided. The USB port on the Arduino provides a serial communication over the USB and appears as a virtual COM port to the software on the computer. Open the Arduino IDE program, select the correct board from the tools menu and write the program. Compile and upload the program to the Arduino board. The program makes PIN 8 as output pin. Connect the LED and resistor as shown in the fig 7.1.

**Program**

```
int LED=8; //The digital pin to which LED is connected

void setup( )

{pinMode(LED, OUTPUT); //Declaring pin 8 as output pin

}

void loop( ) //the loop runs again and again

{

digitalWrite(LED, HIGH); //turn ON the LED

delay(1000); //wait for 1 sec

digitalWrite(LED, LOW); //turn OFF the LED

delay(1000); //wait for 1 sec

}
```

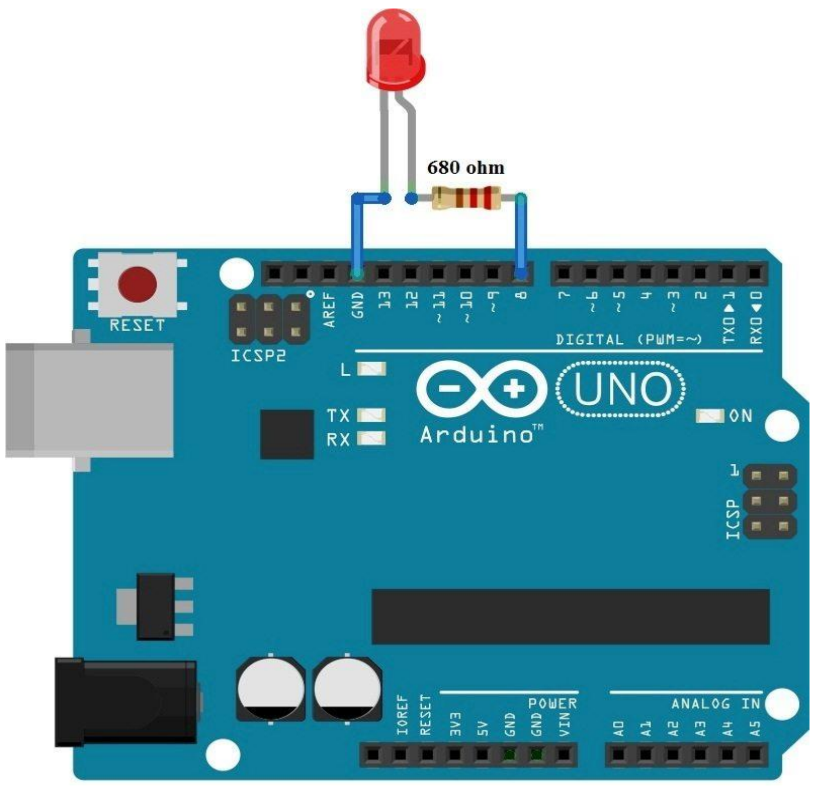


Fig. 7.1 Wired Circuit

## 8. Square Wave Generation

**Aim**

Write a program to generate square wave in Arduino.

**Theory**

Arduino UNO is an entry level Arduino board with enough memory and processing capabilities for small projects like square wave generation (see appendix for technical specifications of Arduino UNO). Pins 0 to 13 are digital pins, and can be used to view the generated square wave. There are numerous techniques for square wave generation, and one has to select the suitable technique based on the project. One way is to continuously make the output signal jump between HIGH and LOW.

**Circuit description**

Connect the Arduino hardware to the computer via the USB cable provided. The USB port on the Arduino provides a serial communication over the USB and appears as a virtual COM port to the software on the computer. Open the Arduino IDE program, select the correct board from the tools menu and write the program. Compile and upload the program to the Arduino board. The program makes PIN 8 as output pin. So, view the square wave output in PIN 8 of the Arduino using a CRO/DSO.

**Program**

```
int PIN = 8; void
setup()

{
        pinMode(PIN, OUTPUT);
}         void
loop()

{
        int state = 0;
        while(1)
        {
                If(state == 0)
                {

                        digitalWrite(PIN, LOW);
                        state = 1;

                }

                else

                {

   digitalWrite(PIN, HIGH);

                        state = 0;

                }

  }
```

# 9. LED and LCD Display Interfacing

## 1. Seven segment led display interfacing

### Aim

Write a program to display numbers in a seven segment LED display.

### Theory

Arduino can be used to interface a seven segment LED display. Seven segment displays are of two types: common anode and common cathode. The difference is the polarity of the LEDs and common terminal. In a common cathode seven-segment display, all seven LEDs plus a dot LED have the cathodes connected to pins 3 and pin 8. To use this display, we need to connect GROUND to pin 3 and pin 8 and, and connect +5V to the other pins to make the individual segments light up. The following diagram shows the internal structure of common-cathode seven-segment display:

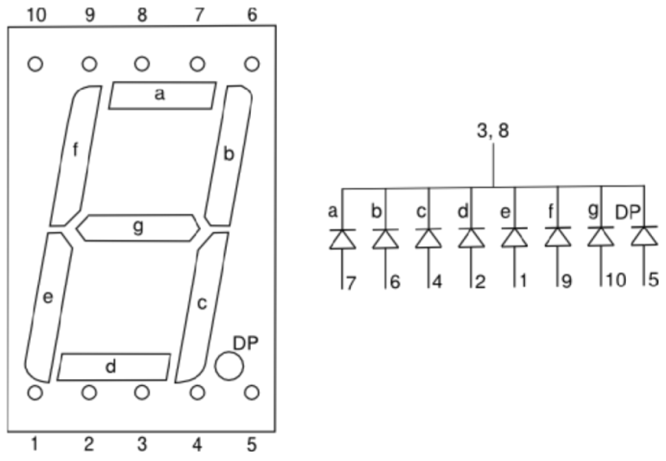


Fig. 9.1 common cathode 7 segment led display

The seven segments are labelled a-g, with the dot being ‘dp’. The common anode display is the exact opposite. In a common-anode display, the positive terminal of all the eight LEDs are connected together and then connected to pin 3 and pin 8. To turn on an individual segment, you ground one of the pins.

Turn on the required segments by outputting a HIGH signal to get the required digit displayed.

**Circuit description**

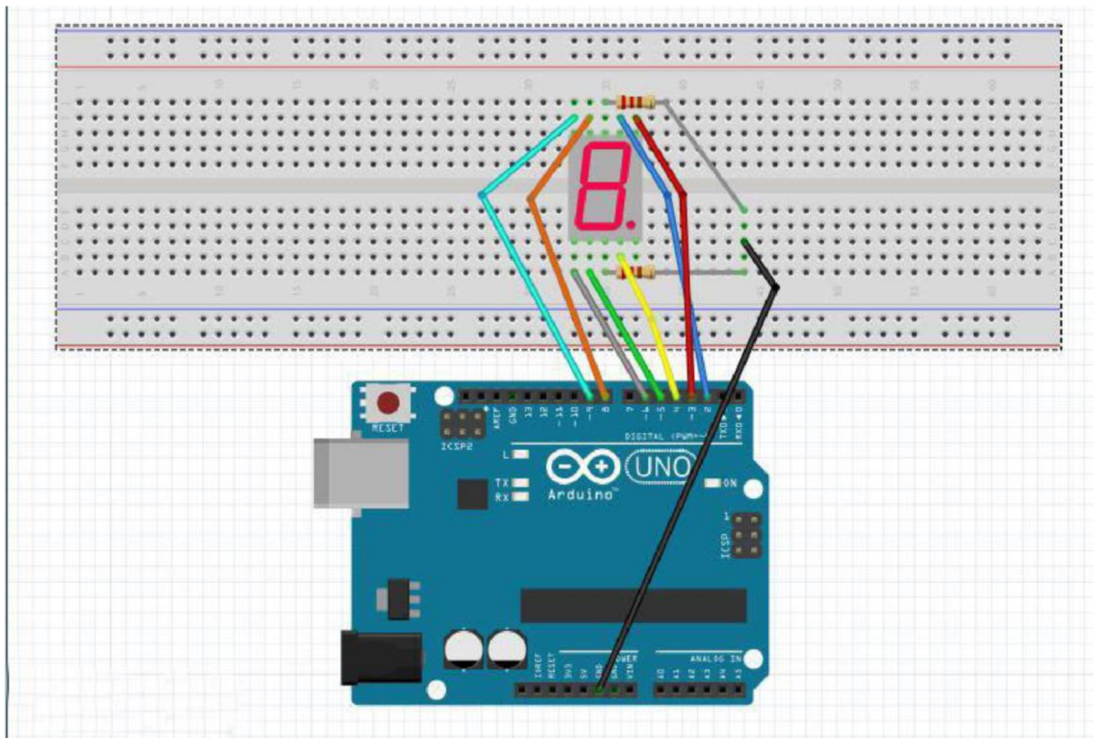


Fig. 9.2 Interfacing arduino and led display

Connect the pins described below:

2. Arduino Pin 2 to Pin 9 of LED display.
3. Arduino Pin 3 to Pin 10 of LED display.
4. Arduino Pin 4 to Pin 4 of LED display. 5. Arduino Pin 5 to Pin 2 of LED display. 6. Arduino Pin 6 to Pin 1 of LED display. 7. Arduino Pin 8 to Pin 7 of LED display.
8. Arduino Pin 9 to Pin 6 of LED display.
9. Arduino GND to Pin 3 and Pin 8 of LED display, each connected with 220 ohm resistors.

**Program**

int a = 2; //For displaying segment "a" int

b = 3; //For displaying segment "b" int c

= 4; //For displaying segment "c" int d =

5; //For displaying segment "d" int e = 6;

//For displaying segment "e" int f = 8;

//For displaying segment "f" int g = 9;

//For displaying segment "g"

```
void setup() {
pinMode(a, OUTPUT);  //A
pinMode(b, OUTPUT);  //B
pinMode(c, OUTPUT);  //C
pinMode(d, OUTPUT);  //D
pinMode(e, OUTPUT);  //E pinMode(f,
OUTPUT);  //F pinMode(g,
OUTPUT);  //G

}        voiddisplayDigit(int
digit)

{
 //Conditions for displaying segment a
if(digit!=1 && digit != 4) digitalWrite(a,HIGH);


 //Conditions for displaying segment b if(digit
!= 5 && digit != 6) digitalWrite(b,HIGH);


 //Conditions for displaying segment c
if(digit !=2) digitalWrite(c,HIGH);


 //Conditions for displaying segment d if(digit != 1 && digit !=4 && digit !=7) digitalWrite(d,HIGH);


 //Conditions for displaying segment e  if(digit
== 2 || digit ==6 || digit == 8 || digit==0)
digitalWrite(e,HIGH);
```

```
 //Conditions for displaying segment f if(digit != 1
&& digit !=2 && digit!=3 && digit !=7)
digitalWrite(f,HIGH); if (digit!=0 && digit!=1 &&
digit !=7) digitalWrite(g,HIGH);

} voidturnOff() {
digitalWrite(a,LOW);
digitalWrite(b,LOW);
digitalWrite(c,LOW);
digitalWrite(d,LOW);
digitalWrite(e,LOW);
digitalWrite(f,LOW);
digitalWrite(g,LOW);

} void loop() {
for(inti=0;i<10;i++)

 { displayDigit(i);
delay(1000);
turnOff();

 }
}
```

**2. Interfacing LCD Display**

**Aim**

Write a program to display a text on the LCD display.

**Hardware required**

i. Arduino UNO

ii. LCD Screen (Hitachi HD44780 driver compatible)
iii. 10k ohm potentiometer
iv. 220 ohm resistor
v. Hook-up wires
vi. Breadboard

**Theory**

The LiquidCrystal library allows you to control LCD displays that are compatible with the Hitachi HD44780 driver. This example sketch prints "Hello World!" to the LCD and shows the time in seconds since the Arduino was reset.

The LCDs have a parallel interface, meaning that the microcontroller has to manipulate several interface pins at once to control the display. The interface consists of the following pins:

A register select (RS) pin that controls where in the LCD's memory you're writing data to. You can select either the data register, which holds what goes on the screen, or an instruction register, which is where the LCD's controller looks for instructions on what to do next.

A Read/Write (R/W) pin that selects reading mode or writing mode

An Enable pin that enables writing to the registers
8 data pins (D0 -D7). The states of these pins (high or low) are the bits that you're writing to a register when you write, or the values you're reading when you read.

There's also a display contrast pin (Vo), power supply pins (+5V and Gnd) and LED Backlight (Bklt+ and BKlt-) pins that you can use to power the LCD, control the display contrast, and turn on and off the LED backlight, respectively.

The process of controlling the display involves putting the data that form the image of what you want to display into the data registers, then putting instructions in the instruction register. The LiquidCrystal Library simplifies this for you so you don't need to know the low-level instructions.

The Hitachi-compatible LCDs can be controlled in two modes: 4-bit or 8-bit. The 4-bit mode requires seven I/O pins from the Arduino, while the 8-bit mode requires 11 pins. For displaying text on the screen, you can do most everything in 4-bit mode, so example shows how to control a 2x16 LCD in 4-bit mode.

**Circuit description**

To wire your LCD screen to your board, connect the following pins:

i. LCD RS pin to digital pin 12
ii. LCD Enable pin to digital pin 11
iii. LCD D4 pin to digital pin 5

iv. LCD D5 pin to digital pin 4

v. LCD D6 pin to digital pin 3

vi. LCD D7 pin to digital pin 2

Additionally, wire a 10k pot to +5V and GND, with its wiper (output) to LCD screens VO pin (pin3). A 220 ohm resistor is used to power the backlight of the display, usually on pin 15 and 16 of the LCD connector.

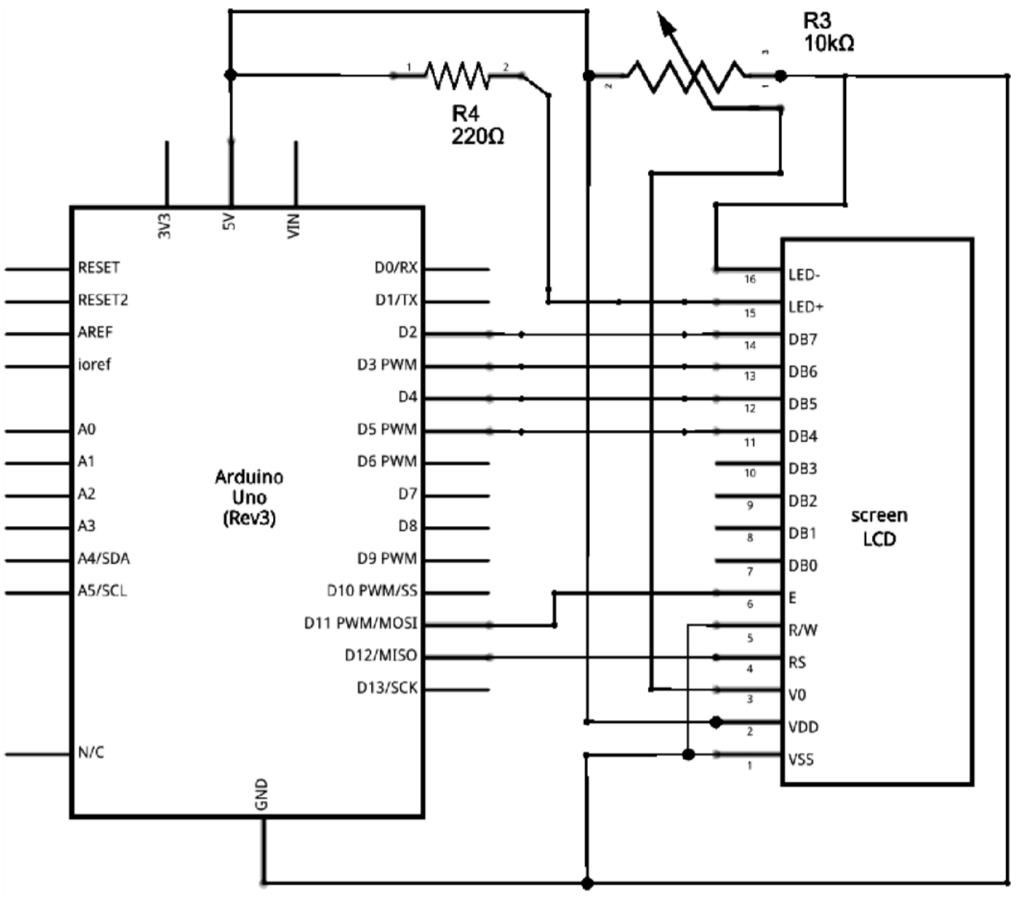


Fig. 9.3: Interfacing Arduino with LCD display

**Program**

```
// include the library code:

#include <LiquidCrystal.h>



// initialize the library with the numbers of the interface pins

LiquidCrystallcd(12, 11, 5, 4, 3, 2);
```

```
void setup() {

  // set up the LCD's number of columns and rows: lcd.begin(16,

2);


  // Print a message to the LCD.

lcd.print("hello, world!");
}



void loop() {

  // set the cursor to column 0, line 1

  // (note: line 1 is the second row, since counting begins with 0):

lcd.setCursor(0, 1);

  // print the number of seconds since reset: lcd.print(millis()

/ 1000);


}
```

# 10. Motor Control

## Aim

Write a program in Arduino IDE to control the speed and direction of a DC motor.

## Hardware required

1. 1 x L298 bridge IC
2. 1 x DC motor
3. 1 x Arduino UNO
4. 1 x breadboard
5. 10 x jumper wires

## Theory

A direct current, or DC, motor is the most common type of motor. DC motors normally have just two leads, one positive and one negative. If you connect these two leads directly to a battery, the motor will rotate. If you switch the leads, the motor will rotate in the opposite direction.

To control the direction of the spin of DC motor, without changing the way that the leads are connected, you can use a circuit called an H-Bridge. An H bridge is an electronic circuit that can drive the motor in both directions. H-bridges are used in many different applications, one of the most common being to control motors in robots. It is called an H-bridge because it uses four transistors connected in such a way that the schematic diagram looks like an "H."

You can use discrete transistors to make this circuit, but for this tutorial, we will be using the L298 H-Bridge IC. The L298 can control the speed and direction of DC motors and stepper motors and can control two motors simultaneously. Its current rating is 2A for each motor. At these currents, however, you will need to use heat sinks.

The pinouts for the L298 are shown below.

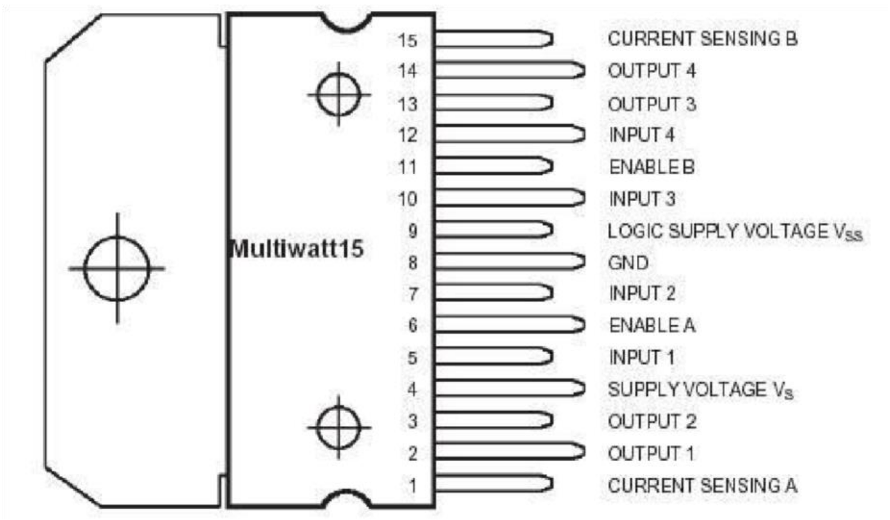


Fig. 9.1: Pinouts of L298

The schematic above shows how to connect the L298 IC to control two motors. There are three input pins for each motor, including Input1 (IN1), Input2 (IN2), and Enable1 (EN1) for Motor1 and Input3, Input4, and Enable2 for Motor2.

Since we will be controlling only one motor, we will connect the Arduino to IN1 (pin 5), IN2 (pin 7), and Enable1 (pin 6) of the L298 IC. Pins 5 and 7 are digital, i.e. ON or OFF inputs, while pin 6 needs a pulse-width modulated (PWM) signal to control the motor speed.

The following table shows which direction the motor will turn based on the digital values of IN1 and IN2.

| IN1 | IN2 | MOTOR |
|---|---|---|
| | | BRAKE |
| 1 | | FORWARD |
| | 1 | BACKWARD |
| 1 | 1 | BRAKE |

IN1 pin of the L298 IC is connected to pin 8 of the Arduino while IN2 is connected to pin 9. These two digital pins of Arduino control the direction of the motor. The EN A pin of IC is connected to the PWM pin 2 of Arduino. This will control the speed of the motor.

To set the values of Arduino pins 8 and 9, we will use the digitalWrite() function, and to set the value of pin 3, we will use the using analogWrite() function.

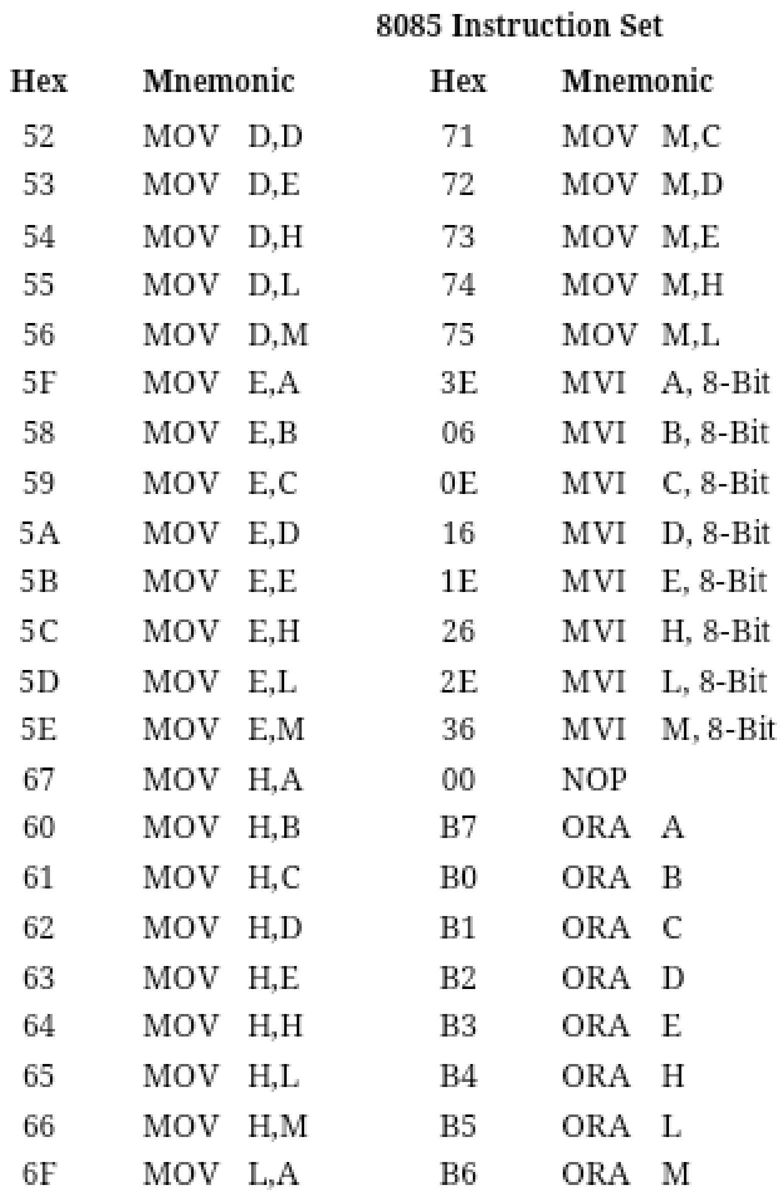
**Circuit description**

1. Connect 5V and ground of the IC to 5V and ground of Arduino.
2. Connect the motor to pins 2 and 3 of the IC.
3. Connect IN1 of the IC to pin 8 of Arduino.
4. Connect IN2 of the IC to pin 9 of Arduino.
5. Connect EN1 of IC to pin 2 of Arduino.
6. Connect SENS A pin of IC to the ground.
7. Connect the Arduino using Arduino USB cable and upload the program to the Arduino using Arduino IDE software.
8. Provide power to the Arduino board using power supply, battery or USB cable. **Program**

   constintpwm = 3;     //initializing pin 3 as pwm constint in_1 = 8; constint in_2 = 9;

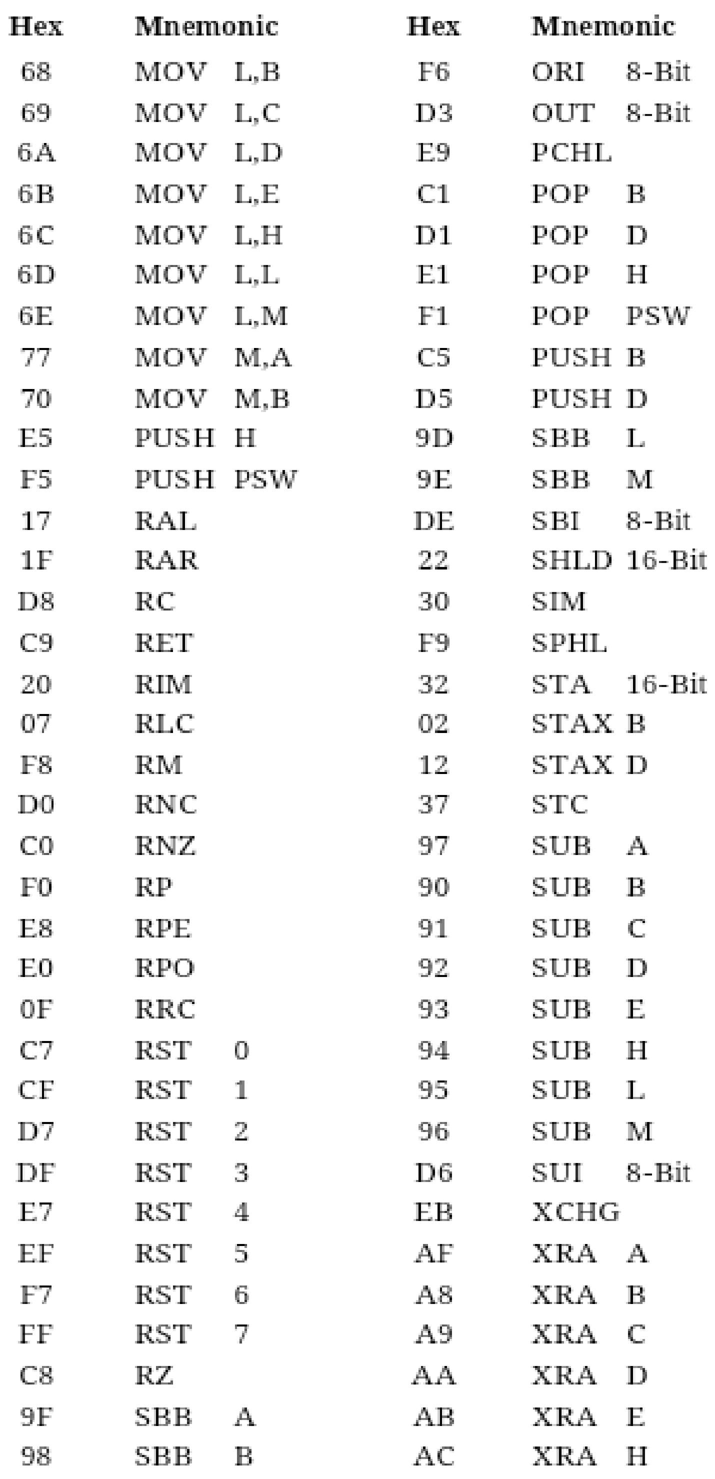
```
//For providing logic to L298 IC to choose the direction of the DC motor  void

setup()


{
pinMode(pwm,OUTPUT) ;   //we have to set PWM pin as output

pinMode(in_1, OUTPUT) ;  //Logic pins are also set as output pinMode(in_2,

OUTPUT) ;


}         void

loop()


{

//For Clock wise motion, in_1 = High, in_2 = Low



digitalWrite(in_1, HIGH) ; digitalWrite(in_2,

LOW) ; analogWrite(pwm,255);




/*setting pwm of the motor to 255 we can change the speed of rotation by changing pwm input
but we are only using arduino so we are using highest value to driver the motor  */



//Clockwise for 3 secs

delay(3000) ;



//For brake digitalWrite(in_1,

HIGH) ; digitalWrite(in_2,

HIGH) ; delay(1000) ;
//For Anti Clock-wise motion - IN_1 = LOW, IN_2 = HIGH
digitalWrite(in_1, LOW) ; digitalWrite(in_2, HIGH) ;
delay(3000) ;


//For brake digitalWrite(in_1,
HIGH) ; digitalWrite(in_2,
HIGH) ; delay(1000) ;

 }
```

# Appendix



**8085 Instruction Set**

| Hex | Mnemonic | Hex | Mnemonic |
|---|---|---|---|
| 52 | MOV D,D | 71 | MOV M,C |
| 53 | MOV D,E | 72 | MOV M,D |
| 54 | MOV D,H | 73 | MOV M,E |
| 55 | MOV D,L | 74 | MOV M,H |
| 56 | MOV D,M | 75 | MOV M,L |
| 5F | MOV E,A | 3E | MVI A, 8-Bit |
| 58 | MOV E,B | 06 | MVI B, 8-Bit |
| 59 | MOV E,C | 0E | MVI C, 8-Bit |
| 5A | MOV E,D | 16 | MVI D, 8-Bit |
| 5B | MOV E,E | 1E | MVI E, 8-Bit |
| 5C | MOV E,H | 26 | MVI H, 8-Bit |
| 5D | MOV E,L | 2E | MVI L, 8-Bit |
| 5E | MOV E,M | 36 | MVI M, 8-Bit |
| 67 | MOV H,A | 00 | NOP |
| 60 | MOV H,B | B7 | ORA A |
| 61 | MOV H,C | B0 | ORA B |
| 62 | MOV H,D | B1 | ORA C |
| 63 | MOV H,E | B2 | ORA D |
| 64 | MOV H,H | B3 | ORA E |
| 65 | MOV H,L | B4 | ORA H |
| 66 | MOV H,M | B5 | ORA L |
| 6F | MOV L,A | B6 | ORA M |



| Hex | Mnemonic | Hex | Mnemonic |
|---|---|---|---|
| 68 | MOV L,B | F6 | ORI 8-Bit |
| 69 | MOV L,C | D3 | OUT 8-Bit |
| 6A | MOV L,D | E9 | PCHL |
| 6B | MOV L,E | C1 | POP B |
| 6C | MOV L,H | D1 | POP D |
| 6D | MOV L,L | E1 | POP H |
| 6E | MOV L,M | F1 | POP PSW |
| 77 | MOV M,A | C5 | PUSH B |
| 70 | MOV M,B | D5 | PUSH D |
| E5 | PUSH H | 9D | SBB L |
| F5 | PUSH PSW | 9E | SBB M |
| 17 | RAL | DE | SBI 8-Bit |
| 1F | RAR | 22 | SHLD 16-Bit |
| D8 | RC | 30 | SIM |
| C9 | RET | F9 | SPHL |
| 20 | RIM | 32 | STA 16-Bit |
| 07 | RLC | 02 | STAX B |
| F8 | RM | 12 | STAX D |
| D0 | RNC | 37 | STC |
| C0 | RNZ | 97 | SUB A |
| F0 | RP | 90 | SUB B |
| E8 | RPE | 91 | SUB C |
| E0 | RPO | 92 | SUB D |
| 0F | RRC | 93 | SUB E |
| C7 | RST 0 | 94 | SUB H |
| CF | RST 1 | 95 | SUB L |
| D7 | RST 2 | 96 | SUB M |
| DF | RST 3 | D6 | SUI 8-Bit |
| E7 | RST 4 | EB | XCHG |
| EF | RST 5 | AF | XRA A |
| F7 | RST 6 | A8 | XRA B |
| FF | RST 7 | A9 | XRA C |
| C8 | RZ | AA | XRA D |
| 9F | SBB A | AB | XRA E |
| 98 | SBB B | AC | XRA H |

| Hex | Mnemonic | | Hex | Mnemonic | |
|---|---|---|---|---|---|
| 99 | SBB | C | AD | XRA | L |
| 9A | SBB | D | AE | XRA | M |
| 9B | SBB | E | EE | XRI | 8-Bit |
| 9C | SBB | H | E3 | XTHL | |
| CE | ACI | 8-Bit | 3F | CMC | |
| 8F | ADC | A | BF | CMP | A |
| 88 | ADC | B | B8 | CMP | B |
| 89 | ADC | C | B9 | CMP | C |
| 8A | ADC | D | BA | CMP | D |
| 8B | ADC | E | BB | CMP | E |
| 8C | ADC | H | BC | CMP | H |
| 8D | ADC | L | BD | CMP | L |
| 8E | ADC | M | BE | CMP | M |
| 87 | ADD | A | D4 | CNC | 16-Bit |
| 80 | ADD | B | C4 | CNZ | 16-Bit |
| 81 | ADD | C | F4 | CP | 16-Bit |
| 82 | ADD | D | EC | CPE | 16-Bit |
| 83 | ADD | E | FE | CPI | 8-Bit |
| 84 | ADD | H | E4 | CPO | 16-Bit |
| 85 | ADD | L | CC | CZ | 16-Bit |
| 86 | ADD | M | 27 | DAA | |
| C6 | ADI | 8-Bit | 09 | DAD | B |
| A7 | ANA | A | 19 | DAD | D |
| A0 | ANA | B | 29 | DAD | H |
| A1 | ANA | C | 39 | DAD | SP |
| A2 | ANA | D | 3D | DCR | A |
| A3 | ANA | E | 05 | DCR | B |
| A4 | ANA | H | 0D | DCR | C |
| A5 | ANA | L | 15 | DCR | D |
| A6 | ANA | M | 1D | DCR | E |
| E6 | ANI | 8-Bit | 25 | DCR | H |
| CD | CALL | 16-Bit | 2D | DCR | L |
| DC | CC | 16-Bit | 35 | DCR | M |
| FC | CM | 16-Bit | 0B | DCX | B |
| 2F | CMA | | 1B | DCX | D |

| Hex | Mnemonic | | Hex | Mnemonic | |
|---|---|---|---|---|---|
| 2B | DCX | H | 01 | LXI | B, 16-Bit |
| 3B | DCX | SP | 11 | LXI | D, 16-Bit |
| F3 | DI | | 21 | LXI | H, 16-Bit |
| FB | EI | | 31 | LXI | SP, 16-Bit |
| 76 | HLT | | 7F | MOV | A,A |
| DB | IN | 8-Bit | 78 | MOV | A,B |
| 3C | INR | A | 79 | MOV | A,C |
| 04 | INR | B | 7A | MOV | A,D |
| 0C | INR | C | 7B | MOV | A,E |
| 14 | INR | D | 7C | MOV | A,H |
| 1C | INR | E | 7D | MOV | A,L |
| 24 | INR | H | 7E | MOV | A,M |
| 2C | INR | L | 47 | MOV | B,A |
| 34 | INR | M | 40 | MOV | B,B |
| 03 | INX | B | 41 | MOV | B,C |
| 13 | INX | D | 42 | MOV | B,D |
| 23 | INX | H | 43 | MOV | B,E |
| 33 | INX | SP | 44 | MOV | B,H |
| DA | JC | 16-Bit | 45 | MOV | B,L |
| FA | JM | 16-Bit | 46 | MOV | B,M |
| C3 | JMP | 16-Bit | 4F | MOV | C,A |
| D2 | JNC | 16-Bit | 48 | MOV | C,B |
| C2 | JNZ | 16-Bit | 49 | MOV | C,C |
| F2 | JP | 16-Bit | 4A | MOV | C,D |
| EA | JPE | 16-Bit | 4B | MOV | C,E |
| E2 | JPO | 16-Bit | 4C | MOV | C,H |
| CA | JZ | 16-Bit | 4D | MOV | C,L |
| 3A | LDA | 16-Bit | 4E | MOV | C,M |
| 0A | LDAX | B | 57 | MOV | D,A |
| 1A | LDAX | D | 50 | MOV | D,B |
| 2A | LHLD | 16-Bit | 51 | MOV | D,C |

## Standard References

1. Thomas L. Floyd, Digital Fundamentals, Pearson Publications, 11th edition
2. M. Morris Mano, Digital Design, Pearson Publications
3. S. Salivahan, Digital Circuits and Designs, Oxford University Press
4. Ramesh Gaonkar, Microprocessor Architecture, Programming, and applications with the 8085, Penram International Publishing
5. John Uffenbeck, Microcomputers and Microprocessors : The 8080, 8085 and Z-80 programming, interfacing and troubleshooting, Prentice Hall, College Div
6. Muhammed Ali Mazidi, The 8051 Microcontroller and Embedded systems using Assembly and C, Pearson Publications

## RUBRICS

**Rubrics used for continuous evaluation in every lab session:**
**All faculty dealing with lab courses shall share these rubrics with students, alongwith Course Outcomes.**

| No. | Parameters | Marks | Unsatisfactory 0-1 | Developing 2-3 | Satisfactory 4 | Exemplary 5 |
|---|---|---|---|---|---|---|
| 1 | Preparation | 5 | The student did not prepare for the experiment with necessary circuits/designs/observation tables/sample waveforms/program algorithms. Does not indicate the date / experiment no. in the observation book. | The student has prepared the circuit diagrams /designs /programs with some of the details omitted. Requires major corrections. | The student has prepared the circuit diagrams /design /programs well with most of the details attended to. The observation book is presented with some details of the experiment plans and expected results. Requires some corrections / guidance. | The student has prepared the circuit diagrams /design /programs well with all of the details attended to. The observation book shows evidence of keen interest by way of presentation, accuracy and experiment plan. Shows independence. No corrections are required from the faculty. |
| 2 | Viva | 5 | The student does not answer any viva questions. Has no idea about the principles behind the experiment or about the objectives. | The student has no clear idea about the objectives of the experiments. Answers a few viva questions. Does not understand the theoretical principles well. Needs improvement. | The student understands some of the objectives. Answers most of the questions. With some hints, the student could understand the principles behind the experiments. | The student understands all of the objectives. Answers all of the questions. The principles behind the experiment are unambiguously understood. |

| | | | | | | |
|---|---|---|---|---|---|---|
| 3 | Performance | 5 | Circuit construction /Program does not match diagram of circuit / algorithm and does not work. Student does not know how to troubleshoot and solve the problem. | Circuit construction /Program has several problems. Student finds it difficult to troubleshoot. Requires assistance. A few partial results were obtained. | Circuit construction / Program execution is good with only minor omissions. Student able to troubleshoot, with some guidance. Circuits /Programs function mostly as planned. Most of the expected results obtained. | Circuit construction /Programs matches diagram in model / algorithm. Appropriate components are used to represent symbols. Construction is excellent and carefully planned. All precautions adopted. Circuits function according to plan. (In the case of programming, all test conditions were satisfied). All expected results were obtained. The student even tries to extend the experiment beyond the stated basic objectives. Student shows independence. |
| 4 | Lab Report /Documenting | 5 | Did not complete the report in the same lab session or next lab session. | Lab report submitted in time, but not complete in all respect. The observations are neither dated nor signed by the student. The inferences are not drawn, and conclusions not presented. The result graphs / waveforms are clumsy and does not provide scale / labels. | Lab report submitted in time, and most of the details are presented. Some of the inferences are presented but not correct / need improvement. The result graphs / waveforms shown are neat in an overall sense, but lacks details. Observations were dated and signed by the student. | Completed lab report and submitted in time. Report is neat and excellently organised, with date and signature. Inferences and conclusions presented show excellent grasp of the student in the concepts. The result graphs / waveforms presented are having all the required details. Student shows excellent abilities to carry out |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | experiment independently. |
| | | | | | | |