METHODIST

Estd : 2008

METHODIST

COLLEGE OF ENGINEERING AND TECHNOLOGY

Approved by AICTE New Delhi | Affiliated to Osmania University, Hyderabad

Abids, Hyderabad, Telangana, 500001

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

MICROPROCESSOR AND MICROCONTROLLER LABORATORY

STUDENT LABORATORY MANUAL

(As per 2018-2019 Academic Regulations)

B.E VI SEMISTER E&CE

SUBJECT CODE: PC 652 EC

Name:____

Roll No.:____

METHODIST

Estd : 2008

METHODIST
COLLEGE OF ENGINEERING AND TECHNOLOGY
Approved by AICTE New Delhi | Affiliated to Osmania University, Hyderabad
Abids, Hyderabad, Telangana, 500001

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

MICROPROCESSOR AND MICROCONTROLLER LABORATORY

STUDENT LABORATORY MANUAL

(As per 2018-2019 Academic Regulations)

B.E VI SEMISTER E&CE

SUBJECT CODE: PC 652 EC

Prepared by

Mr. M. Mahesh Babu, Assistant Professor
Mrs. Zeenath, Assistant Professor

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

Vision of the Institute:

To produce ethical, socially conscious and innovative professionals who would contribute to sustainable technological development of the society.

Mission of the Institute:

To impart quality engineering education with latest technological developments and interdisciplinary skills to make students succeed in professional practice

To encourage research culture among faculty and students by establishing state of art laboratories and exposing them to modern industrial and organizational practices

To inculcate humane qualities like environmental consciousness, leadership, social values, professional ethics and engage in independent and lifelong learning for sustainable contribution to the society

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

Vision of the Department:

To strive to become centre of excellence in Education, Research with moral, ethical values and serve society

Mission of the Department:

M1: To provide Electronics & Communication Engineering knowledge for successful career either in industry or research

M2: To develop Industry-Interaction for innovation, product oriented research and development.

M3: To facilitate value added education combined with hands-on trainings

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

Program Educational Objectives:

PEO 1: Apply the knowledge of Basic sciences and Engineering in designing and implementing the solutions in emerging areas of Electronics and Communication Engineering.

PEO 2: Pursue the research or higher education and practise profession.

PEO 3: Adapt to the technological advancements for providing the sustainable Engineering solutions to meet organisation/society needs

PEO 4: Work as an individual or in a team with professional ethics and values.

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

Program Outcomes:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. Conduct investigations of complex problems: 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.
5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. Communication: 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 presentations, and give and receive clear instructions.
11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. Life-long learning: 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.

Program Specific Outcomes (PSOs):

PSO1: Professional Competence: Apply the knowledge of Electronics & Communication Engineering principles in different domains like VLSI, Signal processing, Communication, Embedded system & Control Engineering.

PSO2: **Technical Skills**: Able to design and implement products using the cutting- edge software and hardware tools and hence provide simple solutions to complex problems.

PSO3: Social consciousness: Graduates will be able to demonstrate the leadership qualities and strive for the betterment of organization, environment and society

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

Laboratory Code of Conduct

1. Students should report to the concerned labs as per the time table schedule.
2. Studentswhoturnuplatetothelabswillinnocasebepermittedtoperformtheexperiment scheduled for the day.
3. Students should bring a note book of about 100 pages and should enter the readings/observations into the note book while performing the experiment.
4. After completion of the experiment, certification of the concerned staff in-charge in the observation book is necessary.
5. Staff member in-charge shall award 25 marks for each experiment based on continuous evaluation and will be entered in the continuous internal evaluation sheet.
6. The record of observations along with the detailed experimental procedure of the experiment performed in the immediate last session should be submitted and certified by the staff member in-charge.
7. Not more than three students in a group are permitted to perform the experiment on a set-up for equipment-based labs. Only one student is permitted per computer system for computer-based labs.
8. The group-wise division made in the beginning should be adhered to, and no student is allowed to mix up with different groups later.
9. The components required pertaining to the experiment should be collected from the stores in-charge, only after duly filling in the requisition form/log register.
10. When the experiment is completed, students should disconnect the setup made by them, and should return all the components/instruments taken for the purpose.
11. Any damage of the equipment or burn-out of components will be viewed seriously by either charging penalty or dismissing the total group of students from the lab for the semester/year.
12. Students should be present in the labs for the total scheduled duration.
13. Students are required to prepare thoroughly to perform the experiment before coming to Laboratory.

14. Procedure sheets/data sheets provided to the students, if any, should be maintained neatly and returned after the completion of the experiment.

METHODIST
COLLEGE OF ENGINEERING AND TECHNOLOGY
Approved by AICTE New Delhi | Affiliated to Osmania University, Hyderabad
Abids, Hyderabad, Telangana, 500001
Estd : 2008

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

Course Code	**Course Title**						**Core/ Elective**
PC652EC	**MICROPROCESSOR AND MICROCONTROLLER LAB**						Core
Prerequisite	Contact Hours per Week				CIE	SEE	Credits
MPMC PC 603EC	L	T	D	P			
	-	-	-	2	25	50	1

Course objectives:

1. Apply Assembly language programs on 8086 trainer kit in standalone/serial
2. Classify interface modules into input /output and Memory interfaces with 8086
3. Develop and execute the embedded C programming concepts of 8051 microcontroller.
4. Design and develop 8051embedded C programs for various interface modules.
5. Develop Interface with serial and I2C bus.

Course Outcomes:

1. Apply different addressing modes & Model programs using 8086 Instruction set
2. Explain the usage of string instructions of 8086 for string manipulation, Comparison
3. Develop interfacing applications using 8086 processor
4. Design different programs using C cross compilers for 8051 controller
5. Develop interfacing applications using 8051 controller

PART- A

1. Use of 8086 trainer kit and execution of programs. (Instruction set for simple Programs using 4 to 5 lines of instruction code under different addressing modes for data transfer, manipulation, Arithmetic operations)
2. Branching operations and logical operations in a given data.
3. Multiplication and division.
4. Single byte, multi byte Binary and BCD addition and subtraction
5. Code conversions.
6. String Searching and Sorting.
7. Interface a stepper motor to 8086 using 8255 PPI

8. Interface a USART 8251 to 8086 for serial data transfer/Receive

PART - B

[Experiments for 8051 using any C- Cross Compiler & appropriate hardware]

1. Familiarity and use of 8051/8031 Microcontroller trainer, and execution of programs.
2. Instruction set for simple Programs (using 4 to 5 lines of instruction code).
3. Timer and counter operations & programming using 8051.
4. Serial communications using UART
5. Programming using interrupts
6. Interfacing 8051 with DAC to generate waveforms.
7. Interfacing traffic signal control using 8051.
8. Program to control stepper motor using 8051.
9. ADC interfacing with 8051
10. Serial RTC interfacing with 8051
11. LCD interfacing with 8051

Note:

1. Preliminary explanation of the features and use of the tools must be made in 2/3 theory periods.
2. A total of not less than 12 experiments must be carried out during the semester with at least 6 from each part.

Suggested Reading:

1. Myke Predko *Programming and Customizing the 8051 Microcontroller,* TMH,2005
2. Mazidi M.A, Mazidi J.G & Rolin D. Mckinlay, “The 8051 Microcontroller & Embedded systems using Assembly and C” 2/e, Pearson Education, 2007.

METHODIST
COLLEGE OF ENGINEERING AND TECHNOLOGY
Approved by AICTE New Delhi | Affiliated to Osmania University, Hyderabad
Abids, Hyderabad, Telangana, 500001
Estd : 2008

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

Methodist College of Engineering & Technology

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

Experiment No.	Title of the Experiment	Date	Page No.	Marks				Remarks/ Signature
				E	O	R	T	
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								
11								
12								
13								

14								

E: Experiment (10 Marks) O: Observation (10 Marks) R: Record (5Marks) T: Total (25 Marks)

INTRODUCTION

INTRODUCTION TO ESA 86/88E KIT

ESA86/88 E, is an economical and powerful general -purpose microcomputer system the can be operated with 8086 or 8088 CPU that may be used as an instructional and learning aid and also as a development tool in R&D labs and industries.

8086 and 8088 are third generation CPUs from INTEL that differ primarily in their external data paths. 8088 users an 8- bit wide data bus while 8086 uses a 16-bit wide data bus. ESA 86/88E can be operated with either CPU and the only possible difference would be in the speed of execution (with 8088 CPU, a small speed degradation occurs, because of the 8-bite wide data bus). In either case, the CPU is operated in maximum mode.

1.1 SPECIFICATIONS

- **Central Processor**
  - 8086 or 8088 CPU operating a 5MHz in maximum mode.
- **Co-Processor**
  - On-board 8087 Numeric Data processor (optional)
- **Memory**
  - ESA 86/88E provides a total of 128 K Bytes of onboard memory.
    - 64 K Bytes of ROM using two 27256 EPROMs
    - 64 K Bytes of ROM using two 62256 Static RAMs.
- **Onboard Peripherals & interfacing Options**
- **8251A** - Universal Synchronous/Asynchronous Receiver/Transmitter supporting standard baud rates from 110 to 19,200. Baud rate is selected through on-board DIP switch setting.
- **8253-5-** Programmable Interval Timer; Timer 0 is used for Baud clock generation Timer 1 and Timer 2 are available to the user.
- **8255A-** 3 Programmable Peripheral Interface Provide up to 72 Programmable I/O line. One 8255 is used for controlling LCD and reading DIP Switch. Two 8255s are for the user, of which one is populated by default and the other is optional.
- **8288-** Bus Controller used for generating control in Maximum Mode Operation.
- **8042/8742 UPI** (Universal Peripheral Interface).

8051 FAMILY OF MICROCONTROLLERS

8051 family of microcontrollers and its derivatives are increasingly becoming popular for instrumentation and control applications due to its speed and powerful instruction set which are essential for real-time applications. This has created the need for a good trainer and development tools. ESA 51E (a low cost version of ESA 51) provides complete solution for this requirement. It can be used as a flexible instructional aid in academic institutions and a powerful development kit in R&D Labs. The system firmware provides stand-alone mode monitor, serial monitor, single line assembler, disassemble and drivers for EPROM programmer and parallel printer interfaces. ESA 51E is supplied with comprehensive and user-friendly documentation.

MAIN FEATURES

- ESA 51E operates on single +5V power supply either in stand-alone mode using PC keyboard and LCD or with host PC through serial (USB/RS-232-C) interface in serial mode implemented using the on chip serial port of microcontroller.
- Stand-alone and serial monitor programs support the entry of user programs, editing and debugging facilities like single stepping and full speed execution of user programs.
- Line assembler & disassembler both in stand - alone and serial modes.
- Total on-board memory is 128K bytes of which 88K bytes RAM has battery backup provision.
- 48 I/O lines and four programmable interval timers.
- 9 Port lines of 8051 brought out to the right angle ribbon cable connector including INT1.
- Buffered bus signals are available through ribbon cable connector for easy system expansion.
- Driver software for file upload/download to/from host PC.

ACCESSORIES (OPTIONAL)

- Power Adapter: +5V @ 3A (SMPS)
- PC keyboard for stand-alone mode of operation.
- EPROM programmer interface (2716 through 27512).
- 8751 Adapter for the above interface.
- Interface Modules for training purpose : Keyboard, Elevator, Display, ADC with DAC, Dual DAC, 8 bit-16 Channel ADC, 12 bit 8 Channel ADC, Logic Controller, Traffic Lights, Tone Generator, Stepper Motor, Opto Isolated Input, Opto Isolated Output, Relay Output etc.

- Power Supply : +5V @ 3A; +12V @ 250mA; -12V @ 100mA and +30V @ 100mA (required for some of the above interfaces).
- 3.6V Ni-Cd battery for power backup to RAM.
- Parallel printer cable.

CENTRAL PROCESSOR

8051 MCU @ 11.0592 MHz.

MEMORY

Two JEDEC sockets provide following ROM : 32K bytes of system firmware using 27256. RAM : 96K bytes of memory, out of which 32K bytes program memory (using Upper half of 628128) and 64K bytes is data memory (using lower half of 628128). Upper most 8K bytes of data memory are reserved for I/O addressing and I/O expansion.

PERIPHERALS

8155: Static HMOS 256 bytes RAM with I/O ports and timer. RAM reserved for monitor, 14-bit timer is available for user and port lines are used for LCD & system configuration.

8255: PPI, Two nos., one supplied, and another for user expansion. One of them can be used for parallel printer interface.

8253: Programmable interval timer. Three16 bit programmable timers available for user

KBD CNTRL: Universal Peripheral Interface used to interface PC keyboard.

STUDY OF 8051 MICROCONTROLLER

SPECIFICATIONS:

1) 128 KB Memory
2) 64 KB EPROM
3) 64 KB RAM of which 4 KB of data memory
4) Clock frequency 11.0592 MHz.
5) 48 I/O lines using two 8255's terminated in two 26 pin each.
6) I/O serial: one RS-232 compatible interface using a 9 pin D type female connector.
7) Keyboard : External PC-AT keyboard
8) Display : Alphanumeric LCD module (4 line x 20 char)

PROCEDURE:

Step 1: press E (enter)

Step 2: EXPAND (display)

Step 3: Enter

Step 4: 8051 Line Assembler / Disassembler (Display)

Step 5: C =

Step 6: Press “A” Key (C = A) (Enter)

Step 7:Enter the Program

Step 8: Press enter Key

Step 9: C = (Display)

Step 10:Press Q (C = Q) (Enter)

Step 11: PRINT OFF COMMAND = (Go to Command Mode)

Execution of the Program

Step 12: Press G

Step 13:Goto? (Display) (Enter)

Step 14: Burst (Display)

Step 15:ADDR (Initial Address of the Program)

Step 16:Enter

Step 17: Wait Done

Checking the Contents of the register:

Press `S`(Enter)

Substitute (Display) (Enter)

Ext. Memo (Display)

Press Any Key

REGISTER (Display) (Enter)

General (Enter)

Name (Enter)

A = (result)

On pressing enter subsequently all register contents will display

Experiment No: 1

Simple Programs under different addressing modes

1.1 Aim: To perform the Assembly Language programs on different addressing modes of 8086 Microprocessor.

1.2 Apparatus:

1. 8086 Trainer kit
2. Power supply
3. Key board

1.3 Programs:

1.3.1 Immediate Addressing Mode

In this type of addressing immediate data is a part of instruction, and appears in the form of successive byte or bytes.

Algorithm:

1. Immediate data is moved in AL register
2. Immediate data is moved in AH register
3. Immediate data is moved in BX register
4. End the program

ADDRESS		**Machine Code**				**SYMBOLIC ASSEMBLER INSTRUCTIONS**			
SEGMENT	OFFSET	1	2	3	4	LABEL	OPCODE	OPERANDS	COMMENTS
0000	2000					START	MOV MOV MOV INT	AL,10 AH,20 BX,0FFFF 03	10H is stored in AL Reg. 20H is stored in AH Reg. 0FFFF H data is stored in BX register. Terminate the program.

INPUTS:

AL =

AH =

OUTPUTS:

AX =

BX =

BX =

1.3.2 Register to Register Addressing Mode

In register addressing mode, the data is stored in a register and it is referred using the particular register. All the registers, except IP (instruction pointer) may be used. EX: Mov BX, AX. In the example, a 16-bit data which is there in AX register is moved into BX register. Both the source and destination are registers only.

Algorithm:

1. Immediate data is moved in AX register
2. Move the data from source register to the BX register
3. End the program

ADDRESS		**Machine Code**				**SYMBOLIC ASSEMBLER INSTRUCTIONS**			
SEGMENT	OFFSET	1	2	3	4	LABEL	MNEMONIC	OPERANDS	COMMENTS
0000	3000					START	MOV	AX,0ABCD	0ABCD H is stored in AX register.
							MOV	BX,AX	0ABCD H data in AX is moved to BX register.
							INT	03	Terminate the program.

INPUTS:

AX =

OUTPUTS:

AX =

BX =

1.3.3 Direct Addressing Mode

In the direct addressing mode, a 16-bit memory address (offset) is directly specified in the instruction as a part of it.

EX: Mov AX, [3000h]

In the above example, the data stored in the memory location 3000h is moved into AX register that is, the contents of memory location 3000h is stored in AL and the contents of memory location 3001h is stored in AH.

Algorithm:

1. Move the data from source pointer to the Accumulator register.
2. Move the data from source pointer to the CL, CH registers.
3. End the program

ADDRESS		Machine Code				SYMBOLIC ASSEMBLER INSTRUCTIONS			
SEGMENT	OFFSET	1	2	3	4	LABEL	MNEMONIC	OPERANDS	COMMENTS
0000	2400					START	MOV	AX,[2000]	
							MOV	CL,[2002]	
							MOV	CH,[2003]	
							INT	03	

INPUTS:

2000 =

2001 =

2002 =

2003

OUTPUTS:

AX =

CX =

1.3.4 Register Indirect Addressing Mode

Sometimes, the address of the memory location which contains data or operand is determined in an indirect way using the offset registers. This mode of addressing is known as register indirect addressing mode. In this addressing mode, the offset address of the data is in either BX or SI or DI registers. The data is supposed to be available at the address pointed to by the content of any of the above registers.

EX: Mov AL, [BX]   Mov AL, [SI]   Mov AL, [DI]

In the above example, the data stored in the memory location pointed by BX register is moved into AX register.

Algorithm:

1. Initialize the pointer value in index register
2. Copy the data from source pointer to the Accumulator register.
3. End the program

ADDRESS		Machine Code				SYMBOLIC ASSEMBLER INSTRUCTIONS			
SEGMENT	OFFSET	1	2	3	4	LABEL	MNEMONIC	OPERANDS	COMMENTS
0000	3000					START	MOV MOV MOV INT	SI,2100 AL,[SI] AX,[SI] 03	

INPUTS:	OUTPUTS:
2100 =	AL =
2101	AX =

1.3.5 Based Indexed Addressing Mode

The effective address of data is formed, in this addressing mode, by adding content of a base register (any one of BX or BP) to the content of an index register (any one of SI or DI). The default segment register may be ES or DS.

Ex: MOV AX, [BX] [SI]

Here, BX is the base register and SI is the index register. The effective address is computed as 10h*DS + [BX] + [SI].

Algorithm:

1. Initialize the pointer value in index register
2. Initialize the pointer value in BX register
3. Copy the data from source pointer to the Accumulator register.
4. Copy the data from Accumulator register to the Base indexed pointer.
5. End the program

ADDRESS		Machine Code				SYMBOLIC ASSEMBLER INSTRUCTIONS			
SEGMENT	OFFSET	1	2	3	4	LABEL	MNEMONIC	OPERANDS	COMMENTS
0000	2500					START	MOV MOV MOV MOV MOV	SI,2200 BX,2300 AL,[BX+SI] AH,[BX+SI+1] [BX+SI+2],AX	

							INT	03	

INPUTS:

4500 =

4501 =

OUTPUTS:

AX =

4502 =

4503 =

1.4 Result: we have studied and verified the outputs of simple programs using 8086 Microprocessor instruction set.

1.5 Exercise Questions:

1. Write an 8086 program to copy a 16-bit value into the register or memory location using different addressing modes.
2. Write an 8086 program to copy 35h into memory locations 4000h to 4004h using register indirect addressing mode using: a) Without a loop and b) With a loop
3. Write a program to move a source data block starting at address location 3000h to a destination block whose address is 4000h. The length of the source block is in CX register.

1.6 Viva Questions:

1) List all the modern microprocessor
2) Name some 16 bit Processor (8086, 80286, 80386L, EX)
3) Name some 32 bit processors (80386DX, 80486, PENTIUM OVERDRIVE)
4) How many bit that 8086 microprocessor supports?
5) What is the size of data bus of 8086?
6) What is the size of address bus of 8086?
7) What is the maximum memory addressing capacity of 8086?
8) Which are the basic parts of 8086?
9) What is an addressing mode?

Experiment No: 2

ARITHMETIC, LOGICAL AND BRANCHING OPERATIONS

2.1 AIM: To perform the Assembly Language programs on Arithmetic, Logical and Branching instructions of 8086 Microprocessor.

2.2 APPARATUS:

1. 8086 Trainer kit
2. Power supply
3. Key board

2.3 PROGRAMS

ARITHMETIC INSTRUCTIONS

Program 2.3.1: Write an ALP to add two 8-bit numbers using registers and place the result in other register.

Algorithm:

1. Initialize two 8-bit data in registers
2. add the data in registers
3. verify the result in register
4. End the program

ADDRESS		**Machine Code**	**SYMBOLIC ASSEMBLER INSTRUCTIONS**			
SEGMENT	OFFSET		LABEL	MNEMONIC	OPERANDS	COMMENTS
0000	2000		START	MOV	AL,05	
				MOV	BL,07	
				ADD	AL,BL	
				INT	03	

INPUTS:

AL =

BL =

OUTPUTS:

AL =

Program 2.3.2: Write an ALP to add two 16-bit numbers using registers and place the result in other register ignoring the possible overflow.

Algorithm:

1. Initialize two 16-bit data in registers
2. add the data in registers
3. verify the result in register
4. End the program

ADDRESS		**Machine Code**	**SYMBOLIC ASSEMBLER INSTRUCTIONS**			
SEGMENT	OFFSET		LABEL	MNEMONIC	OPERANDS	COMMENTS
0000	2000		START	MOV MOV ADD INT	AX,1234 BX,5678 AX,BX 03	

INPUTS: **OUTPUTS:**

AX = AX =

BX =

Program 2.3.3: Write an ALP to subtract two 16-bit numbers using registers and place the result in other register.

Algorithm:

5. Initialize two 16-bit data in registers
6. Subtract the data in registers
7. Store the result in some other register
8. End the program

ADDRESS		**Machine Code**	**SYMBOLIC ASSEMBLER INSTRUCTIONS**			
SEGMENT	OFFSET		LABEL	MNEMONIC	OPERANDS	COMMENTS
0000	2000		START	MOV MOV	AL,05 BL,07	

				SUB	AL,BL	
				MOV	DX,AX	
				INT	03	

INPUTS:	**OUTPUTS:**
AL =	AL =
BL =	DX =

Program 2.3.4: Write an ALP to subtract two 16-bit numbers using registers and place the result in other register.

Algorithm:

1. Initialize two 16-bit data in registers
2. Subtract the data in registers
3. Store the result in some other register
4. End the program

ADDRESS		**Machine Code**	**SYMBOLIC ASSEMBLER INSTRUCTIONS**			
SEGMENT	OFFSET		LABEL	MNEMONIC	OPERANDS	COMMENTS
0000	2000		START	MOV	AX,0056	
				MOV	BX,0044	
				SUB	AX,BX	
				MOV	DX,AX	
				INT	03	

INPUTS:	**OUTPUTS:**
AX =	AX =
BX =	DX =

LOGICAL INSTRUCTIONS

Program 2.3.5: Write an ALP to perform AND logic of two 8-bit numbers using registers.

Algorithm:

1. Initialize two 8-bit data in registers
2. Perform AND logic of data in registers
3. verify the result in register

4. End the program

ADDRESS		Machine Code	SYMBOLIC ASSEMBLER INSTRUCTIONS			
SEGMENT	OFFSET		LABEL	MNEMONIC	OPERANDS	COMMENTS
0000	2000		START	MOV MOV AND INT	AL,12 BL,16 AL,BL 03	

INPUTS: **OUTPUTS:**

AL = AL =

BL =

Program 2.3.6: Write an ALP to perform OR logic of two 8-bit numbers using registers.

Algorithm:

1. Initialize two 8-bit data in registers
2. Perform OR logic of data in registers
3. verify the result in register
4. End the program

ADDRESS		Machine Code	SYMBOLIC ASSEMBLER INSTRUCTIONS			
SEGMENT	OFFSET		LABEL	MNEMONIC	OPERANDS	COMMENTS
0000	3000		START	MOV MOV OR INT	AL,[4050] BL,[4051] AL,BL 03	

INPUTS: **OUTPUTS:**

AL = AL =

BL =

Program 2.3.7: Write an ALP to perform XOR logic of two 8-bit numbers using Register Indirect addressing.

Algorithm:

1. Initialize memory pointer in register
2. Load two 8-bit data from memory into two registers
3. Perform XOR logic of data in registers
4. verify the result in register
5. End the program

ADDRESS		**Machine Code**	**SYMBOLIC ASSEMBLER INSTRUCTIONS**			
SEGMENT	OFFSET		LABEL	MNEMONIC	OPERANDS	COMMENTS
0000	5000		START	MOV	SI,2050	
				MOV	AL,[SI]	
				INC	SI	
				MOV	BL,[SI]	
				XOR	AL,BL	
				INT	03	

INPUTS:	**OUTPUTS:**
2050 =	AL =
2051 =	2052 =

BRANCHING OPERATIONS:-

Program 2.3.8: Write an ALP to perform largest number from a given Two numbers using Register Indirect addressing.

Algorithm:

1. Initialize two numbers into registers
2. Compare two numbers in a registers
3. If condition is true go to step 6
4. If condition is false Load greater number to acc
5. verify the result in register

6. End the program

FINDING LARGEST BYTE OF GIVEN TWO BYTES

ADDRESS		**Machine Code**	**SYMBOLIC ASSEMBLER INSTRUCTIONS**			
SEGMENT	OFFSET		LABEL	MNEMONIC	OPERANDS	COMMENTS
0000	2000		START	MOV	AL,36	
				MOV	BL,78	
				CMP	AL,BL	
				JA	300A	
				MOV	AL,BL	
				INT	03	

INPUTS:	**OUTPUTS:**
AL =	AL =
BL =	

Program 2.3.9: Write 8086 Assembly language program to find the average of 5 numbers stored in a given series starts from memory offset 5001.

Algorithm:

1. Initialize two memory locations(source and destination) in to registers
2. Clear Accumulator, load count value into count register and also copy count into another register
3. Perform addition of first value from memory location to 8bit Acc.
4. Also perform addition with carry of Ah with 00h value.
5. Increment source memory location for accessing next value
6. Decrement count register value by 1
7. Check zero flag status, If this(jump if not zero) condition is true go to step 3, If this(jump if not zero) condition is false go to next step
8. Perform average by dividing Accumulator by stored count value
9. Store the result from Accumulator to memory location
10. End the program

ADDRESS		**Machine Code**	**SYMBOLIC ASSEMBLER INSTRUCTIONS**			
SEGMENT	OFFSET		LABEL	MNEMONIC	OPERANDS	COMMENTS
0000	2000		START	MOV	SI,5001	
				MOV	DI,6001	
				MOV	CL,05	
				MOV	BL,CL	
			NEXT:	**ADD**	**AL,[SI]**	
				ADC	AH,00	
				INC	SI	
				DEC	CL	
				JNZ	**NEXT**	
				DIV	BL	
				MOV	[DI],AX	
				INT	03	

INPUTS:	**OUTPUTS:**
5001 =	AX =
5002 =	6001 =
5003 =	6002 =
5004 =	
5005 =	

2.4 RESULT:

We have verified and noted the outputs of Arithmetic, logical and branching instructions.

2.5 EXERCISE QUESTIONS:

1) Write a program to add two 8-bit decimal numbers using registers.
2) Write a program to subtract two 8-bit decimal numbers using registers.
3) Write an ALP to find smallest no from the given array.
4) Write an 8086 ALP program to find the factorial of a number
5) Write an 8086 ALP program to sort an integer array in descending order.
6) Write an ALP to sort a given set of 16bit unsigned integers into ascending order using bubble sort algorithm.
7) Write an 8086 ALP program to find sum of Even numbers in a given series

2.6 VIVA QUESTIONS:

1) Explain about MOV instruction with examples.
2) List the different addressing modes in 8086.
3) What are the functions of BIU?
4) What are the functions of EU?
5) Which are the registers present in 8086?
6) What do you mean by pipelining in 8086?
7) How many general purpose registers are available in 8086?

Experiment No: 3

Multiplication and division

3.1 AIM: To perform the Assembly Language programs on multiplication and division operations of 8086 Microprocessor.

3.2 APPARATUS:

1. 8086 Trainer kit
2. Power supply
3. Key board

3.3 PROGRAMS:

Program 3.3.1: Write an 8086 ALP Program to multiply two 8-bit unsigned numbers.

Algorithm:

1. Load accumulator with 1st 8-bit number.
2. Load BL register with 2nd 8-bit number.
3. Initialize memory pointer in register
4. Perform multiply operation of data in registers
5. End the program

ADDRESS		**Machine Code**	**SYMBOLIC ASSEMBLER INSTRUCTIONS**			
SEGMENT	OFFSET		LABEL	MNEMONIC	OPERANDS	COMMENTS
0000	2000		START	MOV	AL,05	
				MOV	BL,07	
				MUL	BL	
				INT	03	

INPUTS:

AL =

BL =

OUTPUTS:

AX =

Program 3.3.2: Write an 8086 ALP Program to division two 8-bit unsigned numbers.

Algorithm:

1. Load accumulator with 1st 8-bit number.
2. Load BL register with 2nd 8-bit number.
3. Initialize memory pointer in register
4. Perform division operation of data in registers
5. End the program

ADDRESS		**Machine Code**	**SYMBOLIC ASSEMBLER INSTRUCTIONS**			
SEGMENT	OFFSET		LABEL	MNEMONIC	OPERANDS	COMMENTS
0000	2000		START	MOV	AL,06	
				MOV	BL,03	
				DIV	BL	
				INT	03	

INPUTS: **OUTPUTS:**

AL = AX =

BL =

Program 3.3.3: Write an 8086 ALP Program to multiply two 16-bit unsigned numbers.

Algorithm:

1. Load accumulator with 1st 16-bit number.
2. Load BL register with 2nd 16-bit number.
3. Initialize memory pointer in register
4. Perform multiply operation of data in registers
5. End the program

ADDRESS		**Machine Code**	**SYMBOLIC ASSEMBLER INSTRUCTIONS**			
SEGMENT	OFFSET		LABEL	MNEMONIC	OPERANDS	COMMENTS
0000	2000		START	MOV	AX,0056	
				MOV	BX,0044	
				MUL	BX	

				INT	03	

INPUTS: **OUTPUTS:**

AX = AX =

BX = DX =

Program 3.3.4: Write an 8086 ALP Program to division two 16-bit unsigned numbers.

Algorithm:

1. Load accumulator with 1st 16-bit number.
2. Load BL register with 2nd 16-bit number.
3. Initialize memory pointer in register
4. Perform division operation of data in registers
5. End the program

ADDRESS		**Machine Code**	**SYMBOLIC ASSEMBLER INSTRUCTIONS**			
SEGMENT	OFFSET		LABEL	MNEMONIC	OPERANDS	COMMENTS
0000	2000		START	MOV	AX,0128	
				MOV	BX,0004	
				DIV	BX	
				INT	03	

INPUTS: **OUTPUTS:**

AX = AX =

BX = DX =

3.4 RESULT:

we have verified the outputs of ALP of multiplication and division operation.

3.5 EXERCISE QUESTIONS:

1. Write an 8086 ALP Program to multiply two 8-bit signed numbers
2. Write an 8086 ALP Program to multiply two 16-bit signed numbers
3. Write an 8086 ALP Program to divide 16-bit unsigned number by an 8-bit unsigned number.
4. Write an 8086 ALP Program to divide 16-bit signed number by an 8-bit signed number.

3.6 VIVA QUESTIONS:

1. Explain multiplication and division instruction in 8086 instruction set with examples.
2. What is the difference between instructions MUL & IMUL?
3. How many pin IC 8086 is?
4. What is the difference between instructions DIV & IDIV?
5. During division operation in which registers the result will be stored?

Experiment No: 4

Single byte, Multi byte Binary and BCD addition and subtraction

4.1 AIM: To perform the Assembly Language programs on Single byte, Multi byte Binary and BCD addition and subtraction operations of 8086 Microprocessor.

4.2 APPARATUS:

1. 8086 Trainer kit
2. Power supply
3. Key board

4.3 PROGRAMS:

Program 4.3.1: Write an ALP for adding two multi byte binary numbers. The two strings of binary numbers starts from memory location 3100h & 3110h respectively. The result is stored from memory location 3120h onwards.

Algorithm:

1. Initialize the source pointers, destination pointer & a counter
2. Clear Acc register
3. Move the data stored in first source location to Acc register
4. Add with carry the data in Acc register with the data in second source pointer location
5. Place the sum in destination location
6. Increment the pointers & repeat the process till the count becomes zero
7. End the program

ADDRESS		**Machine Code**	**SYMBOLIC ASSEMBLER INSTRUCTIONS**			
SEGMENT	OFFSET		LABEL	MNEMONIC	OPERANDS	COMMENTS
0000	2000		START	MOV	SI,3100	
				XOR	AX,AX	
				MOV	DI,3120	
				MOV	CX,0005	
			REPT1:	**ADC**	**AL,[SI]**	
				MOV	[DI],AL	
				INC	SI	

				INC LOOP MOV ADC MOV INT	DI REPT1 AL,00 AL,AL [DI+1],AL 03	

INPUTS:		**OUTPUTS:**
3100 =	3110 =	3120 =
3101 =	3111 =	3121 =
3102 =	3112 =	3122 =
3103 =	3113 =	3123 =

Program 4.3.2: Write an ALP for subtracting two multi byte binary numbers. The two strings of binary numbers starts from memory location 3100h & 3110h respectively. The result is stored from memory location 3120h onwards.

Algorithm:

1. Initialize the source pointers & destination pointer & a counter
2. Clear Acc register
3. Move the data stored in first source location to Acc register
4. Subtract with barrow the data in Acc register with the data in second source pointer location
5. Place the difference in destination location
6. Increment the pointers & repeat the process till the count becomes zero
7. End the program

ADDRESS		**Machine Code**	**SYMBOLIC ASSEMBLER INSTRUCTIONS**			
SEGMENT	OFFSET		LABEL	MNEMONIC	OPERANDS	COMMENTS
0000	2000		START **REPT1:**	MOV MOV XOR MOV MOV	SI,3100 DI,3110 AX,AX BX,3120 CX,0005	

				MOV SBB MOV INC INC LOOP INT	**AL,[SI]** AL,[DI] [BX],AL SI DI REPT1 03	

INPUTS: **OUTPUTS:**

3100 =	3110 =	3120 =
3101 =	3111 =	3121 =
3102 =	3112 =	3122 =
3103 =	3113 =	3123 =

Program 4.3.3: Write an ALP for adding two Single byte BCD numbers. Where numbers are stored from starting memory address 2100 and store the result into memory address 2600 and carry at 2601.

Algorithm:

1. Load data from memory offset 2100 to register AL (first number).
2. Load data from memory offset 2101 to register BL (second number).
3. Add these two numbers (contents of register AL and register BL).
4. Apply DAA instruction (decimal adjust).
5. Store the result (content of register AL) to memory offset 2600.
6. Set register AL to 00.
7. Add contents of register AL to itself with carry.
8. Store the result (content of register AL) to memory offset 2601.

ADDRESS		**Machine Code**	**SYMBOLIC ASSEMBLER INSTRUCTIONS**			
SEGMENT	OFFSET		LABEL	MNEMONIC		COMMENTS
				OPCODE	OPERANDS	
0000	2000		START	MOV MOV ADD	AL,[2100] BL,[2101] AL,BL	

				DAA MOV MOV ADC MOV INT	 [2600],AL AL,00 AL,AL [2601],AL 03	

INPUTS:

2100 =

2101 =

OUTPUTS:

2600 =

2601 =

Program 4.3.4: Write an ALP for subtracting single byte BCD numbers. The two bytes of BCD numbers starts from memory location 2100h & 2200h respectively. The result is stored from memory location 2600h onwards.

Algorithm:

1. Initialize the source pointers & destination pointer & a counter
2. Clear Acc register
3. Move the data stored in first source location to Acc register
4. Add with carry the data in Acc register with the data in second source pointer location
5. Convert the binary value into its BCD equivalent
6. Place the sum in destination location
7. Increment the pointers & repeat the process till the count becomes zero
8. End the program

ADDRESS		**MACHINE CODE OF MNEMONIC**	**SYMBOLIC ASSEMBLER INSTRUCTIONS**			
				MNEMONIC		
SEGMENT	**OFFSET**		**LABEL**	**OPCODE**	**OPERANDS**	**COMMENTS**
0000	2000		START	MOV MOV SUB	AL,[2100] BL,[2101] AL,BL	

				DAS MOV MOV ADC MOV INT	 [2600],AL AL,00 AL,AL [2601],AL 03	

INPUTS: **OUTPUTS:**

2100 =	2200 =	2600 =
2101 =	2201 =	2601 =

Program 4.3.5: Write an ALP for adding multi byte BCD numbers. The two strings of BCD numbers starts from memory location 2100h & 2200h respectively. The result is stored from memory location 2600h onwards.

Algorithm:

1. Initialize the source pointers & destination pointer & a counter
2. Clear Acc register
3. Move the data stored in first source location to Acc register
4. Subtract with barrow the data in Acc register with the data in second source pointer location
5. Convert the binary value into its BCD equivalent
6. Place the difference in destination location
7. Increment the pointers & repeat the process till the count becomes zero
8. End the program

ADDRESS		**Machine Code**	**SYMBOLIC ASSEMBLER INSTRUCTIONS**			
SEGMENT	OFFSET		LABEL	MNEMONIC	OPERANDS	COMMENTS
0000	2000		START	MOV MOV XOR MOV MOV	SI,[2100] DI,[2200] AX,AX CX,0005 BX,2600	

			REPT1:	**MOV**	**AL,[SI]**	
				ADD	AL,[DI]	
				DAA		
				MOV	[BX],AL	
				MOV	AL,00	
				ADC	AL,AL	
				MOV	[BX+1],AL	
				INC	SI	
				INC	DI	
				INC	BX	
				LOOP	REPT1	
				INT	03	

INPUTS:		**OUTPUTS:**
2100 =	2200 =	2600 =
2101 =	2201 =	2601 =
2102 =	2202 =	2602 =
2103 =	2203 =	2603 =
2104 =	2204 =	2604 =

Program 4.3.6: Write an ALP for subtracting multi byte BCD numbers. The two strings of BCD numbers starts from memory location 2100h & 2200h respectively. The result is stored from memory location 2600h onwards.

Algorithm:

1. Initialize the source pointers & destination pointer & a counter
2. Clear Acc register
3. Move the data stored in first source location to Acc register
4. Subtract with barrow the data in Acc register with the data in second source pointer location
5. Convert the binary value into its BCD equivalent
6. Place the difference in destination location
7. Increment the pointers & repeat the process till the count becomes zero
8. End the program

ADDRESS		MACHINE CODE	SYMBOLIC ASSEMBLER INSTRUCTIONS			
SEGMENT	**OFFSET**		**LABEL**	**MNEMONIC**	**OPERANDS**	**COMMENTS**
0000	2000		START	MOV	SI,[2100]	
				MOV	DI,[2200]	
				XOR	AX,AX	
				MOV	CX,0005	
				MOV	BX,2600	
			REPT1:	**MOV**	**AL,[SI]**	
				SUB	AL,[DI]	
				DAS		
				MOV	[BX],AL	
				MOV	AL,00	
				SBB	AL,AL	
				MOV	[BX+1],AL	
				INC	SI	
				INC	DI	
				INC	BX	
				LOOP	REPT1	
				INT	03	

INPUTS: **OUTPUTS:**

2100 =	2200 =	2600 =
2101 =	2201 =	2601 =
2102 =	2202 =	2602 =
2103 =	2203 =	2603 =
2104 =	2204 =	2604 =

4.4 Result: we have analysed, verified the outputs of Binary addition, BCD addition and subtraction operations.

4.5 Exercise Questions:

1. Write an 8086 ALP Program to convert ASCII Code into BCD number.
2. Write an 8086 ALP Program to convert Hexadecimal number into BCD number.

4.6 Viva Questions:

1. What is the size of instruction queue in 8086?
2. What is the function of 01h of INT 21h?
3. How many minimum address lines are required to access 512 KB?
4. What is the supply requirement of 8086?
5. What is the relation between 8086 processor frequency & crystal Frequency?
6. Explain Functions of Accumulator or AX register?
7. What is Physical address? And how it is generated?

Experiment No: 5

CODE CONVERSION

5.1 AIM: To perform the following programs on code conversion using instruction set of 8086 Microprocessor.

1. Binary code to Gray code conversion.
2. Hexadecimal to ASCII code conversion.
3. ASCII to Hexadecimal code conversion.

5.2 APPARATUS:

1. 8086 Trainer kit
2. Power supply
3. Key board

5.3 PROGRAM:

Program 5.3.1: Write an 8086 ALP Program to perform Binary number to Gray code conversion.

Algorithm:

1. Load accumulator with 1st 8-bit number.
2. Load BL register with Accumulator value.
3. Logical shift right AL by one time
4. Perform XOR operation of data in registers
5. End the program

ADDRESS	MACHINE CODE	LABEL	MNEMONIC		COMMENTS
SEGMENT:OFFSET			OPCODE	OPERANDS	
0000 : 2000		START	MOV	AL,08	
			MOV	BL,AL	
			SHR	AL,01	
			XOR	AL,BL	
			INT	03	

INPUTS: **OUTPUTS:**

AL = AX =

MASM Program:

```
        ASSUME CS: CODE, DS: DATA
        DATA SEGMENT
          OPR1 DB 12h
        DATA ENDS
        CODE SEGMENT
                START: MOV AX, DATA
                        MOV DS, AX
                        MOV AL, OPR1
                        MOV BL, AL
                        SHR AL, 01
                        XOR AL, BL
                        INT 03
        CODE ENDS
        END START
```

Program 5.3.2: Write an 8086 ALP Program to perform Hexadecimal to ASCII code conversion.

Algorithm:

1. Load accumulator with 1st 8-bit number.
2. Compare Accumulator value with 9.
3. Jump if Acc. Value is greater than 9 to step 6.
4. Perform addition operation of Acc. With 30 value.
5. Jump unconditionally to step 7.
6. Perform addition operation of Acc. With 37 value.
7. End the program.

ADDRESS	**MACHINE**	**LABEL**	**MNEMONIC**		**COMMENTS**
SEGMENT:OFFSET	**CODE**		OPCODE	OPERANDS	
0000 : 2000		START	MOV CMP JA	AL,0B AL,09 **DOWN**	

			ADD	AL,30	
			JMP	**END**	
		DOWN:	ADD	AL,37	
		END:	INT	03	

INPUTS:	**OUTPUTS:**
AL = 0Bh	AX =
AL =	AX =

MASM Program:

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
  OPR1 DB 42h
DATA ENDS
CODE SEGMENT
        START: MOV AX, DATA
                MOV DS, AX
                MOV AL, OPR1
                CMP AL, 09h
                JA   REPT1
                ADD AL, 30h
                JMP REPT2
        REPT1:          ADD AL, 37h
        REPT2:          INT 03
CODE ENDS
END START
```

Program 5.3.3: Write an 8086 ALP Program to perform ASCII code to Hexadecimal conversion.

Algorithm:

1. Load accumulator with 1st 8-bit number.
2. Compare Accumulator value with 39.
3. Jump if Acc. Value is greater than 9 to step 6.
4. Perform subtraction operation of Acc. With 30 value.
5. Jump unconditionally to step 7.

6. Perform subtraction operation of Acc. With 37 value.
7. End the program.

ADDRESS	MACHINE	LABEL	MNEMONIC		COMMENTS
SEGMENT:OFFSET	CODE		OPCODE	OPERANDS	
0000 : 2000		**START**	MOV	AL,42	
			CMP	AL,39	
			JA	**DOWN**	
			SUB	AL,30	
			JMP	**END**	
		DOWN:	SUB	AL,37	
		END:	INT	03	

INPUTS:	**OUTPUTS:**
AL = 42h	AX =
AL =	AX =

MASM Program:

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
  OPR1 DB 42h
DATA ENDS
CODE SEGMENT
        START: MOV AX, DATA
                MOV DS, AX
                MOV AL, OPR1
                CMP AL, 39h
                JA   REPT1
                SUB AL, 30h
                JMP REPT2
        REPT1:          SUB AL, 37h
        REPT2:          INT 03
CODE ENDS
END START
```

5.4 RESULT: Here, we have noted outputs of code conversion programs, and also verified the practical values with Theoretical calculated values.

5.5 Exercise Questions:

1. Write an 8086 ALP Program to convert gray code to binary code.
2. Write an 8086 ALP Program to convert an 8 bit BCD number to hexadecimal number.
3. Write an 8086 ALP Program to convert ASCII code to BCD.
4. Write an 8086 ALP Program to convert an 8 bit BCD number to hexadecimal number.
5. Write an ALP to convert 2 digit packed BCD number into its Binary equivalent number.
6. Write an ALP to convert temperature from degree centigrade into degree Fahrenheit using C=5/9*(F-32).

5.6 Viva Questions:

1. Which are pointers present in this 8086 and what its use?
2. How many segments present in it and what its use?
3. Explain the Functions of BX register?
4. Explain the Functions of CX register?
5. Explain the Functions of DX register?
6. Which is by default pointer for CS/ES?
7. Explain different branching instruction in 8086.

Experiment No: 6

STRING SEARCHING AND SORTING

61 AIM: To perform the following programs on strings using instruction set of 8086 Microprocessor.

1. String Searching
2. String sorting a) Ascending order b) Descending order

6.2 APPARATUS:

1. 8086 Trainer kit
2. Power supply
3. Key board

6.3 PROGRAM:

String Searching:

Program 6.3.1: To search for a character in the given string and store result in to Acc. The data is present in memory location 2500h.

Algorithm:

1. Initialize the source pointer, & counter register.
2. Move the string byte which has to be searched in AL register.
3. Compare string byte from source location with the value in AL register
4. Jump to End (step 7) if two values/strings equal
5. Repeat the process till the value becomes equal or count becomes zero
6. Store the result in destination location
7. End the program

ADDRESS	**MACHINE CODE**	**LABEL**	**MNEMONIC**		**COMMENTS**
SEGMENT:OFFSET			OPCODE	OPERANDS	
0000:2000		START	MOV	SI,2500	
			MOV	CL,05	
			MOV	BL,07	
		UP	MOV	AL,[SI]	
			CMP	AL,BL	
			JE	END	

			INC DEC JNZ MOV	SI CL UP AL,00	
		END	INT	03	

INPUTS: **OUTPUTS:**

2500 = 01 H	2502=	2504=	AL = 07 H
2501 = 02H	2503=		BL=00H

MASM Program:

Write an assembly language program to search a number or character from a string.

PROGRAM:

```
ASSUME CS:CODE, DS:DATA, ES:EXTRA
DATA SEGMENT
STRING1 DB "ENTER A CHARACTER: $"
STRE  DB  " FOUND $"
STRNE  DB  " NOT FOUND $"
DATA  ENDS
EXTRA SEGMENT
STRING2  DB  "METHODIST COLLEGE 12345 $"
STRLEN  DW  ($-STRING2)
EXTRA  ENDS
CODE SEGMENT
START:   MOV AX, DATA
         MOV DS, AX
         MOV AX, EXTRA
         MOV ES, AX
         MOV DX, OFFSET STRING1
         MOV AH, 09H
         INT 21H
         MOV AH, 01H
         INT 21H
```

```
                REPNE SCASB
                JZ LABLE
                MOV DX, OFFSET STRNE
                MOV AH, 09H
                INT 21H
                JMP EXIT
        LABLE:  MOV DX, OFFSET STRE
                MOV AH, 09H
                INT 21H
         EXIT:  INT 03H
          CODE ENDS
          END START
```

Program 6.3.2: write an assembly language program for arranging a given array of number in ascending order using 8086.

Algorithm:

1. Initialize the source pointer, destination pointer & counter register
2. Move the inner count byte(CL) into outer count register(DL)
3. Move source value (which has to be compared) to AL register
4. Compare AL with second source pointer value
5. If AL value smaller than source pointer value, goto step 7
6. Swap values in AL register and source pointer
7. Increment source pointer and Decrement count value
8. Repeat from step 4 process till the count becomes zero
9. Decrement outer count value(DL)
10. Repeat from step 3 process till the count becomes zero
11. End the program

String Sorting:

a) Ascending order:

ADDRESS	MACHINE	LABEL	MNEMONIC		**COMMENTS**
SEGMENT:OFFSET	CODE		OPCODE	OPERANDS	

0000:2000		START	XOR	AX,AX	
			MOV	CL,05	
			MOV	DL,CL	
			DEC	CL	
			MOV	BX,2100	
			MOV	AL,[BX]	
		L2:	**CMP**	**AL,[BX+1]**	
			JC	CONT	
		L1:	**XCHG**	**AL,[BX+1]**	
			MOV	[BX],AL	
		CONT:	**INC**	**BX**	
			LOOP	L1	
			DEC	DL	
			MOV	CL,DL	
			LOOP	L2	
			INT	03	

INPUTS:	**OUTPUTS:**
2100 = 05 H	2100 = 01 H
2101 = 03H	2101 = 02H
2102 = 01 H	2102 = 03 H
2103 = 02H	2103 = 04H
2104 = 04H	2104 = 05H

MASM Program:

```
 ASSUME CS: CODE, DS: DATA
DATA SEGMENT
LIST DB 03H, 18h, 05H, 09H, 43h, 20H
COUNT EQU 06
DATA ENDS
CODE SEGMENT
START:       MOV AX, DATA
             MOV DS, AX
```

```
                        XOR AX, AX
                        MOV DL, COUNT-1
              AGAIN:    MOV CL, DL
                        MOV SI, OFFSET LIST
              REPT1:    MOV AL, [SI]
                        CMP AL, [SI+1]
                        JB NEXT
                        XCHG [SI+1], AL
                        XCHG [SI], AL
              NEXT:         INC SI
                        LOOP  REPT1
                        DEC DL
                        JNZ  AGAIN
                        INT 03H
                CODE ENDS
              END START
```

String Sorting: b) Descending order:

Program 6.3.3: write an assembly language program for arranging a given array of number in descending order using 8086.

Algorithm:

1. Initialize the source pointer, destination pointer & counter register
2. Move the inner count byte(CL) into outer count register(DL)
3. Move source value (which has to be compared) to AL register
4. Compare AL with second source pointer value
5. If AL value greater than source pointer value, goto step 7
6. Swap values in AL register and source pointer
7. Increment source pointer and Decrement count value
8. Repeat from step 4 process till the count becomes zero
9. Decrement outer count value(DL)
10. Repeat from step 3 process till the count becomes zero
11. End the program

ADDRESS	**MACHINE**	**LABEL**	**MNEMONIC**		**COMMENTS**
SEGMENT:OFFSET	**CODE**		OPCODE	OPERANDS	
0000:2000		START	MOV	AX,3000	
			MOV	DS,AX	
			XOR	AX,AX	
			MOV	CL,05	
			MOV	DL,CL	
			DEC	CL	
		L2:	MOV	BX,2100	
		L1:	MOV	AL,[BX]	
			CMP	AL,[BX+1]	
			JNC	CONT	
			XCHG	AL,[BX+1]	
			MOV	[BX],AL	
		CONT:	INC	BX	
			LOOP	L1	
			DEC	DL	
			MOV	CL,DL	
			LOOP	L2	
			INT	03	

INPUTS:	**OUTPUTS:**
2100 = 05 H	2100 =
2101 = 03H	2101 =
2102 = 01 H	2102 =
2103 = 02H	2103 =
2104 = 04H	2104 =

MASM Program:

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
LIST DB 65H, 02H, 46H, 38H, 75H, 01H
COUNT EQU 05
DATA ENDS
CODE SEGMENT
    START:  MOV AX, DATA
            MOV DS, AX
            MOV DL, COUNT
    AGAIN:  MOV CL, DL
            MOV SI, OFFSET LIST
    REPT1:  MOV AL, [SI]
            CMP AL, [SI+1]
            JA NEXT
            XCHG [SI+1], AL
            XCHG [SI], AL
     NEXT:  INC SI
            LOOP REPT1
            DEC DL
            JNZ AGAIN
            INT 03H
       CODE ENDS
END START
```

6.4 RESULT:

We have studied different string operating instructions. We have verified the programs and noted the practical values.

6.5 Exercise Questions:

1. Write an 8086 assembly language program to find the length of the given string.
2. Write an assembly language program to display the given string.(print a given string using DOS commands)
3. Write an assembly language program to reverse the given string.
4. Write an ALP to perform non-overlapped and overlapped block transfer (with and without string specific instructions). Block containing data can be defined in the data segment.
5. Write a program to transfer a block of 4 bytes, starting address is 2500 and transfer the block at address 2600 by using string instructions.

6.6 Viva Questions:

1. Which register is used as COUNT in 8086 mp?
2. What is the function of INT 03 instruction?
3. What is the difference between minimum mode and maximum mode of 8086?
4. What are the process control instructions?
5. What is a difference between HALT and NOP instruction?
6. Which flag bit is effected when JNZ is executed?
7. What is the difference between CMP and SUB instructions
8. What is the difference between MOV and XCHG instruction?
9. Which are strings related instructions?
10. How LOOP instruction can function?

Experiment No: 7

STEPPER MOTOR INTERFACE TO 8086 MP KIT

7.1 AIM: To write an assembly language program for Stepper Motor Interface assumed to be connected over connector J4 of the 8086 trainer kit.

7.2 APPARATUS:

1. 8086 Trainer kit
2. Power supply
3. Key board

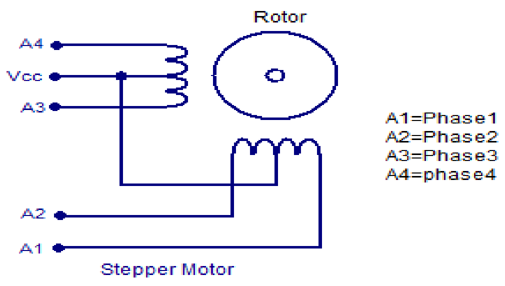
7.3 Theory:

Stepper motor.

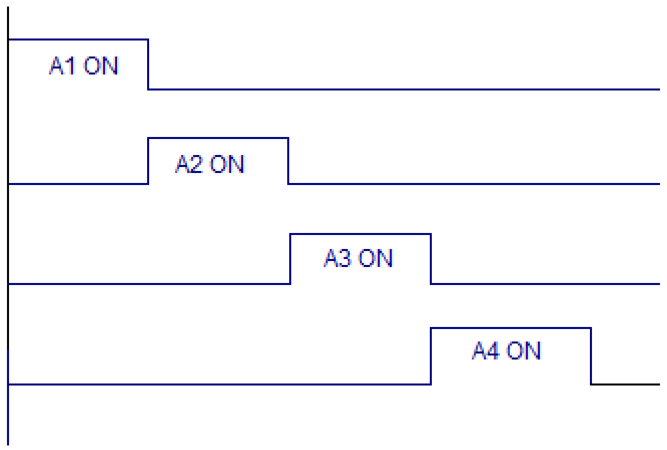
A Stepper Motor is a brushless, synchronous DC Motor. It has many applications in the field of robotics and mechatronics. The total rotation of the motor is divided into steps. The angle of a single step is known as the stepper angle of the motor. There are two types of stepper motors **Unipolar** and **Bipolar**. Due to the ease of operation unipolar stepper motor is commonly used by electronics hobbyists. For more details please read the article Stepper Motor or Step Motor. Stepper Motors can be easily interfaced with a microcontroller using driver ICs such as L293D orULN2003.

Stepper motor is a brush less motor which converts electrical pulses into mechanical rotation. As the name indicates it rotates in steps according to the input pulses. A stepper motor usually have a number of field coils (phases) and a toothed rotor. The step size of the motor is determined by the number of phases and the number of teeth on the rotor. Step size is the angular displacement of the rotor in one step. If a stepper motor has 4 phases and 50 teeth, it takes 50×4=200 steps to make one complete rotation. So step angle will be 360/200=1.8°.

The stepper motor we are using has 4 poles and a 1/64 reduction gear mechanism for increasing torque. The step angle of the motor is 5.64°. But when considering the reduction gear, the step angle of the output shaft is 5.64/64°. The internal schematic of the stepper motor is given below.



The stepper motor is rotated by switching individual phases ON for a given time one by one. The sequence is given in the graph below.



BASICS OF STEPPER MOTOR:

Stepper motor is brushless which take digital signals. It converts digital pulses into mechanical shaft rotation. Rotation is divided into steps and a separate pulse is sent for each step. For each pulse motor rotates a few degrees which are mostly 1.8 degree angle. As we interface it with controller, so when digital pulses increase in frequency, the stepping movement of motor converts to continuous rotation of motor. So we can say that speed of rotation is directly proportional to the frequency of pulses given by controller.

WINDING ARRANGEMENT:

For a two phase stepper motor, there are two winding arrangements of electromagnetic coils.

- Unipolar
- Bipolar

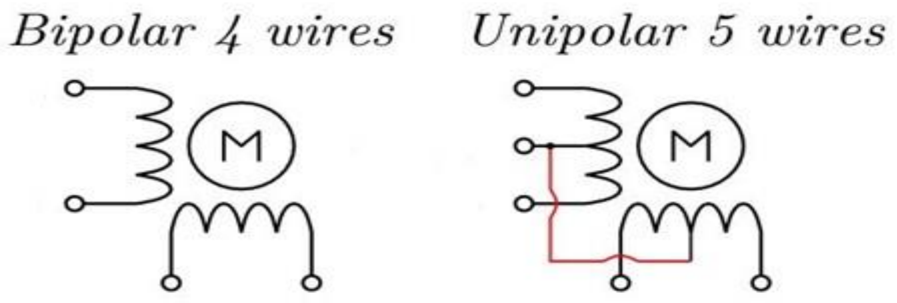
UNIPOLAR MOTOR:

- Only one winding with center tap per phase
- Each winding section is switched on for each direction of magnetic field

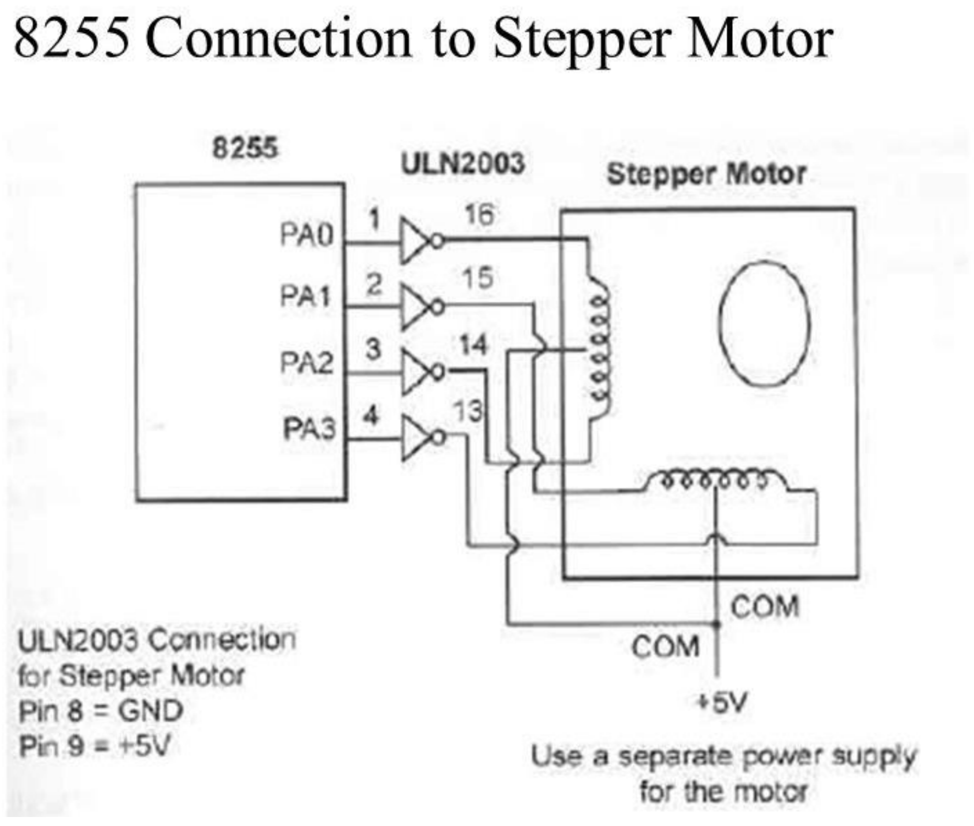
- Center tap of each winding is made common

BIPOLAR MOTOR:

- A single winding per phase
- Current in winding must be reversed to reverse a magnetic pole
- Two leads per phase, none are common



Circuit diagram:



8255 Connection to Stepper Motor
8255
ULN2003
Stepper Motor
PA0
PA1
PA2
PA3
1
2
3
4
16
15
14
13
COM
COM
+5V
ULN2003 Connection for Stepper Motor
Pin 8 = GND
Pin 9 = +5V
Use a separate power supply for the motor

The circuit diagram for interfacing stepper motor to 8255 is shown above. PA.0, PA.1, PA.2 and PA.3 pins are used for controlling the phases A1, A2, A3 and A4 of the stepper motor respectively. ULN2003 is used for driving the individual phases of the stepper motor. ULN2003 is a darlington transistor array used for driving high current loads such as relays and motors. ULN2003 has 8 individual channels each with 1A capacity. The channels can be paralleled to increase the current capacity. Each channel is fitted with individual freewheeling diodes. The ULN2003 is operated in current sinking mode. Each channel is activated by giving a logic LOW at the corresponding input. For example if we make pin 1 of ULN2003 LOW, phase A1 of the stepper motor gets switched ON.

7.4 PROGRAM:

7.4.1 Stepper Motor rotation in Clockwise direction:

ADDRESS		**MACHINE**	**LABEL**	**MNEMONIC**		**COMMENTS**
SEGMENT	**OFFSET**	**CODE**		**OPCODE**	**OPERANDS**	
0000	2000			MOV	DX, 0FFE6	; Initialize all 8255
0000				MOV	AL, 80	; Ports as output
0000				OUT	DX, AL	
0000				MOV	DX, 0FFE2	
0000				MOV	AL, 88	; O/p data to ports
0000			REPEAT:	OUT	DX, AL	
0000				CALL	DELAY	; Introduce delay
0000				ROR	AL, 1	; rotate data byte
0000				JMP	REPEAT	; rotation of motor
0000			DELAY:	MOV	CX, 0800	; Delay subroutine
0000			L1:	LOOP	L1	
0000				RET		

7.4.2 Stepper Motor rotation in Anti-Clockwise direction:

ADDRESS		**MACHINE**	**LABEL**	**MNEMONIC**		**COMMENTS**
SEGMENT	**OFFSET**	**CODE**		**OPCODE**	**OPERANDS**	

0000	2000			MOV	DX, 0FFE6	; Initialize all 8255
0000				MOV	AL, 80	; Ports as output
0000				OUT	DX, AL	
0000				MOV	DX, 0FFE2	
0000				MOV	AL, 88	; O/p data to ports
0000			REPEAT:	OUT	DX, AL	
0000				CALL	DELAY	; Introduce delay
0000				ROL	AL, 1	; rotate data byte
0000				JMP	REPEAT	; rotation of motor
0000			DELAY:	MOV	CX, 0800	; Delay subroutine
0000			L1:	LOOP	L1	
0000				RET		

7.5 RESULT:

We have observed rotation of stepper motor in clockwise and anti clockwise.

7.6 Viva Questions:

1. Define RS-232.
2. What is use of IN and OUT instruction?
3. What is the function of 8284?
4. Briefly describe how direct and indirect Jumps take place in 8086.
5. What is the function of AD0-AD15 pins in 8086?
6. Why do we need maximum mode in 8086 MP?
7. Explain the operating mode of 8255?
8. What is the use of A0 and A1 pins of 8255?
9. Write the control word format in the BSR mode.

10. Give control word to set PC-5 bit for 8255?
11. Give BSR control word to set and reset PC-3 separately for 8255.

Experiment No: 8

USART 8251 Interface to 8086 for serial data transfer/Receive

8.1 AIM: To write an assembly language program to transfer/receive data using interfacing of USART 8251 to the 8086 trainer kit.

8.2 APPARATUS:

1. 8086 Trainer kit
2. Power supply
3. Key board

8.3 Theory:

The 8086 trainer kit is specifically designed to help students to master the required skills in the area of embedded systems. The kit is designed in such way that all the possible features of the microprocessor will be easily used by the students.

8251 (USART)

The RS232C interface of PS-TIMER & USART comprises of the universal synchronous/asynchronous receiver/transmitter 8251 (USART), RS232C driver max 232. The 8251A is used here as a peripheral device for serial communication and is programmed by the CPU to operate using virtually any serial data transmission technique. The USART accepts data characters from the CPU in parallel format and then converts them into a continuous serial data stream for transmission. Simultaneously, it can receive serial data streams and convert them into parallel data characters for the CPU. The CPU can read the status of the USART at any time. These include data transmission errors and control signals.

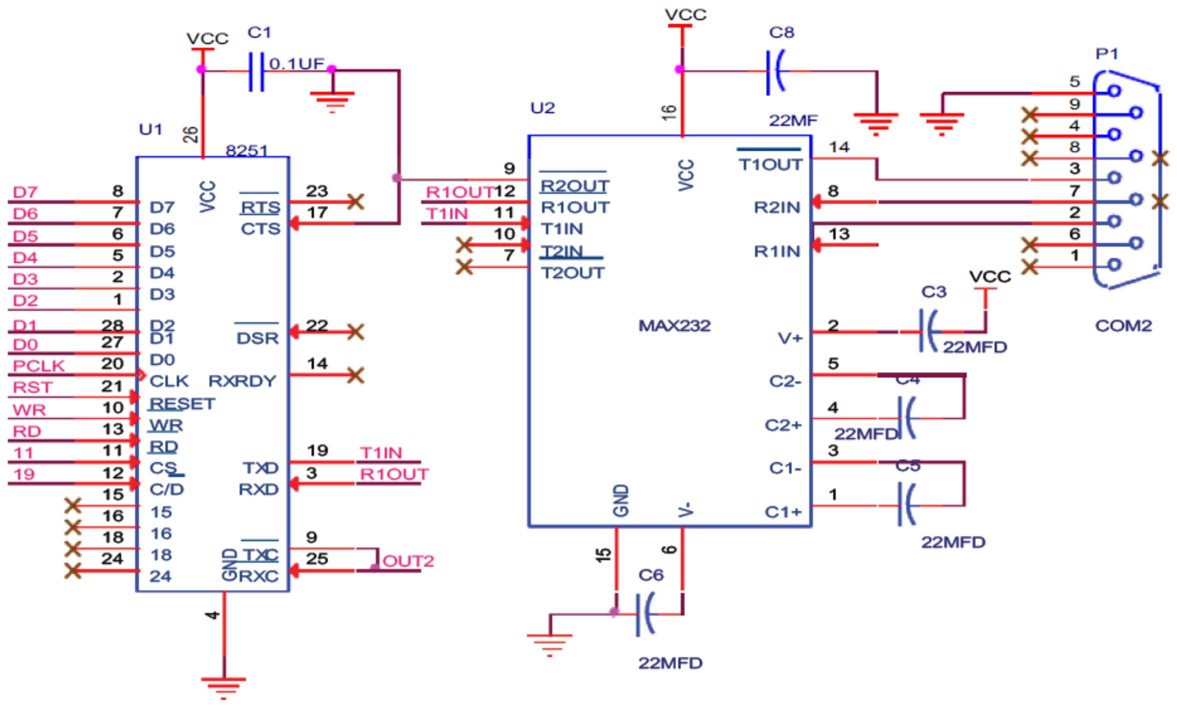
The 8251 is also initialized by specifying both command as well as the mode word. In the Experiment whatever data is transmitted from the CPU (with the help of RS – 232) will be received by the 8251 and then will be transmitted back to the CPU and displayed on the screen.

Interfacing 8251 with 8086

Microprocessor don’t have the direct serial communications, so to communicate the data serially to a device we need a driver to send a character serially, here we use 8251. 8251 is a driver which converts the parallel data to a serial data the pin detail and control pins are given so according to that we can develop the hardware.

Obviously, 8251 is not directly compatible with these signal levels. Standard method to interfaceRS232C and TTL levels is with MC1488 quad TTL-to-RS232C drivers and MC1489 quad RS232C-to-TTL receivers. Of the 25 handshake signals provided by the RS232C standard, we will discuss only four signals which are used in our design. They are the RTS, CTS, RxD and TxD signals.

8.4 Circuit Diagram of Interfacing 8251 with 8086



8.5 Program:

MEMORY	OPCODE		MNEMONICS
1100	Be 00 15		MOV SI,1500H
1103	B0 36		MOV AL,36H
1105	BA 06 FF		MOV DX,FF06
1108	EE		OUT DX,AL
1109	B0 40		MOV AL,40H
110B	BA 04 FF		MOV DX,FF04
110E	EE		OUT DX,AL
110F	B0 01		MOV AL,01H
1111	BA 04 FF		MOV DX,FF04
1114	EE		OUT DX,AL
1115	B1 05	RELOAD:	MOV CL,05H

```
1117        BA 12 FF    CHECK:     MOV DX,FF12
111A        EC                     IN AL,DX
111B        24 02                  AND AL,02H
111D        74 F8                  JZ CHECK
111F        BA 10 FF               MOV DX,FF10
1122        EC                     IN AL,DX
1123        88 04                  MOV [SI],AL
1125        46                     INC SI
1126        3C 3F                  CMP AL,3FH
1128        75 EB                  JNZ RELOAD
112A        FE C9                  DEC CL
112C        75 E9                  JNZ CHECK
112E        CD 02                  INT 02
1130        CD 02                  INT 02
ADDRESS DATA
1500   48H,45H,4CH,4CH
1504   4FH,2DH,38H,30H (HELLO-8086)
1508   38H,35H,0AH,0DH
150C   END
```

8.6 Result: Here, we observed the data transfer operation from kit to pc vice-versa.

8.7 Viva Questions:

1. What is the reset address of 8086?
2. What are the special functions of AX register?
3. What is the importance of BX in 8086?
4. What are Assembler directives? Describe its use.
5. What is stack? Give stack related instructions.
6. What is function of XLAT instruction?
7. Give examples for 8 / 16 / 32 bit Microprocessor?
8. Describe ASSUME and OFFSET directive.

9. What is the use of SEGMENT and ENDS directives?
10. What is the function of 4Ch of INT 21h?

Experiment No.9

Familiarity and use of 8051/8031 Microcontroller trainer, and execution of programs.

9.1 Aim: Learn the use of 8051/8031 Microcontroller trainer, and execution of programs.

9.2 Apparatus:

1. 8051 Microcontroller development kit
2. Power supply
3. Keyboard

9.3 Programs

9.3.1 (a). Write a program in 8051 to add two 16-bit numbers. The numbers are 3CE7H and 3B8DH. Place the sum in registers R7 and R6; R6 has the lower byte.

Algorithm:

1. Initialize the first lower byte in Acc register & add it with second lower byte
2. Store the lower sum in R6 register
3. Take the first upper byte in Acc register & add it with second upper byte along with carry
4. Store the upper byte of the result in R7 register
5. End the program

Source Code:

ADDRESS	OBJECT CODE	MNEMONIC
8000	C3	CLR C
8001	74 E7	MOV A, #0E7H
8003	24 8D	ADD A,#8DH
8005	FE	MOV R6,A
8006	74 3C	MOV A,#3CH
8008	34 3B	ADDC A,#3BH
800A	FF	MOV R7,A
800B	80 FE	HERE:SJMP HERE

(Relative address = Target address – PC contents)

Enter the codes using the format given below.

<EXAMMEM><PRGMEM> 8000 <NXT> DATA <NXT>…..<NXT>EXEC

Note: After executing the program using GO<8000>EXEC press ‘BREAK’ key.

Press EXAMREG key twice to check result in registers R6 and R7 respectively.

Format:

<EXAMREG><EXAMREG><BITMEM>6<NEXT><NEXT>

9.3.1(b). Write an 8051 program to multiply two unsigned 8-bit binary numbers. The numbers are stored in memory locations 8050h and 8051h. Store the result in 8060h and 8061h.

Algorithm:

1. Initialize the pointer DPTR & save the multiplier in B reg
2. Again initialize the pointer DPTR & take the multiplicand in Acc register
3. Multiply two 8-bit data
4. Store the result in the respective memory locations
5. End the program

Sample data: (8050) = 41h $(65)_{10}$

(8051)= 08h

Source Code:

```
            ORG 8000H               ; Starting address
             MOV DPTR,#8051h         ; Initialize DPTR
            MOVX A,@DPTR             ; Multiplier in A reg
            MOV 0F0h,A              ; Save Multiplier in B reg
            MOV DPTR,#8050h         ; Initialize DPTR
            MOVX A,@DPTR             ; Multiplicand in A reg
            MUL AB                   ; Multiply
            MOV DPTR,#8060h          ; Store the result
            MOVX @DPTR,A             ; in respective memory locations
            INC DPTR
            MOV A,0F0h
            MOVX @DPTR,A
        HERE:SJMP HERE               ; End the program
```

Output: (8060)= 08h

(8061)= 02h

i.e., 65X8 = $(520)_{10}$ = 0208h

9.3.1(c). Write an 8051 program to divide the number in 8050h by the number in 8051h.Store the quotient and remainder in 8060h & 8061h of data memory respectively.

Algorithm:

1. Initialize the pointer DPTR & save the divisor in B reg
2. Again initialize the pointer DPTR & take the dividend in Acc register
3. Divide two 8-bit data
4. Store the result in the respective memory locations
5. End the program

```
 Sample data: (8050) =41h     Dividend
                          (8051)=08h      Divisor


          Result        :  (8060) =08h     Quotient
                              (8061)=01h      Reminder
```

Hint: Similar to multiplication program.

Store initial divisor in B reg and dividend in A reg.

Use instruction DIV AB in place of MUL AB.

After division operation, quotient is in A reg & remainder is in B reg.

9.4 Result:

we have studied and verified the outputs of simple programs using 8051 Microcontroller instruction set.

Experiment No: 10

Programs using different addressing modes

10.1Aim: Write an 8051 ALP programs using different addressing modes.

10.2 Apparatus:

1. 8051 Microcontroller development kit
2. Power supply
3. Keyboard

10.3 Programs

10.3.1. Write an 8051 program to copy the value 55H into RAM memory locations 40H to 44H using

A) Direct addressing mode
B) Register addressing mode without using Loop and
C) With Loop

Algorithm:

1. Initialize the data in Acc register & copy it directly in the memory location for direct addressing mode

2. Initialize the data in Acc register which has to be moved

3. Move the data from Acc register to the pointer location

4. Repeat the process by incrementing the pointer for 'n' times (without a loop) & initialize the counter & repeat the loop foe 'n' times (with a loop)

5. End the program

Source Code:

A) Using direct addressing mode

```
               ORG 8000H          ; Starting address
               MOV A,#55h         ; Move the immediate data in A reg
               MOV 40h,A          ; Copy A to RAM Locations
               MOV 41h,A
               MOV 42h,A
               MOV 43h,A
               MOV 44h,A
        HERE:  SJMP HERE          ; End the program
```

B) Using reg-indirect addressing mode without loop

```
            ORG 8000H           ; Starting address
            MOV A,#55h          ; Move the immediate data in A reg
            MOV R0,#40h         ; Initialize the pointer R0
            MOV @R0,A           ; Move data from A reg to R0 location
            INC R0              ; Increment the pointer
            MOV @R0,A
            INC R0
            MOV @R0,A
            INC R0
            MOV @R0,A
            INC R0
            MOV @R0,A
       HERE:SJMP HERE           ; End the program
```

C) With Loop

```
            ORG 8000H
            MOV A,#55h        ; Move the immediate data in A reg
            MOV R0,#40h       ; Initialize the pointer R0
            MOV R2,#05h       ; Initialize the counter
      AGAIN:MOV @R0,A         ; Move data from A reg to R0 location
            INC R0            ; Increment R0
            DJNZ R2,AGAIN ; Decrement & jump if R2!=0 to AGAIN
       HERE:SJMP HERE         ; End the program
```

10.3.2(a). Six bytes of data are stored in memory locations starting at 50H. Add all the bytes. Use register R7 to save any carries generated. Store the sum at memory locations 60H & 61H.

Algorithm:

1. Initialize the source pointer & counter registers

2. Clear Acc & the register to save carry

3. Add the data in Acc register with data in source pointer location

4. Check the carry flag i.e., if cy=1 then increment the register else repeat steps 3 & 4 till count becomes zero

5. Store the result in the desired memory locations

6. End the program

Sample data: (50)=10h, (51)=25h, (52)=2AH, (53)=4Fh, (54)=60h, (55)=3Fh

Source Code:

```
          ORG 8000H          ; Starting address
          MOV R0,#50h        ; Initialize pointer R0
          MOV R2,#06h        ; Initialize the counter R2
          CLR A              ; Initial sum=0
          MOV R7,A           ; Clear R7 to save carry
    AGAIN:ADD A,@R0          ; Add data at R0 with A reg
          JNC NEXT           ; Jump if cy=0 to NEXT
          INC R7             ; Keep track of carries
    NEXT: INC R0             ; Increment pointer
          DJNZ R2,AGAIN      ; Decrement & jump if R2!=0 to AGAIN
          MOV 60h,A          ; Store LSBy of sum
          MOV 61h,R7         ; Store MSBy of sum
    HERE:JMP HERE            ; End the program
```

Output = (60)=4Dh

(61)=01h (MS byte) ; 014Dh

Format: For entering, executing & Checking results.

Enter source code

<EXM MEM> <PRG MEM> 8000 <NXT> DATA <NXT>……. <EXEC>

Feed Sample Data

<EXM MEM> <INTDATA> 50 <NXT> DATA <NXT> DATA…….. <EXEC>

<EXM MEM> <INTDATA> 60 <NXT>00XT>00<EXEC>

Run the Program

<GO><8000> <EXEC>

Reset

Check Results

<EXM MEM> <INT DATA> 60 <NXT>….

10.3.2(b). Write an 8051 program to copy a block of 10 bytes of data from RAM locations starting at 35h to RAM locations starting at 60h.

Algorithm:

1. Initialize the source pointer, destination pointer & a counter register
2. Move the data from source pointer register to the Acc register
3. Move the data from Acc register to the destination pointer location
4. Increment both the pointer registers
5. Repeat the loop for 'n' times
6. End the program

Sample Problem:

Source block

(35)=10h, (36) =20h, (37) = 30h, (38) = 40h, (39) = 50h,

(3A) = 60h, (3B) = 70h, (3C) = 80h, (3D) = 90h, (3E) =A0h

Source Code:

```
      ORG 8000H          ; Starting address
      MOV R0,#35h        ; Source pointer
      MOV R1,#60h        ; Destination pointer
      MOV R3,#0Ah        ; Counter
BACK:MOV A,@R0           ; Move data from R0 to A reg
      MOV @R1,A          ; Move data from A reg to R0 location
      INC R0             ; Increment R0
      INC R1             ; Increment R1
      DJNZ R3,BACK       ; Decrement & jump if R3!=0 to BACK
 HERE:SJMP HERE          ; End the program
```

Output: (60)=10h, (61) =20h, (62) = 30h, (63) = 40h, (64) = 50h,

(65) = 60h, (66) = 70h, (67) = 80h, (68) = 90h, (69) =A0h

10.3.2(c).A byte is stored in register R0. Write an 8051 Program to find the number of 1's in a byte stored in R0 and Store the number of 1's in register R2.

Algorithm:

1. Initialize Acc register with the byte for which number of ones has to be counted & counter register

2. Rotate the byte one bit left with carry & check the carry flag

3. If cy=1 then increment result register

else decrement the count by 1 & repeat the steps 2 & 3 till the

count becomes zero

4. End the program

Source Code:

```
          Let R0 = AAh    i.e.,  10101010
          ORG 8000H            ; Starting address
          MOV R0,#AAh          ; Move data in R0
          MOV A,R0             ; Take it in A reg
          MOV R2,#00h          ; Clear R2 reg
          MOV R1,#08h          ; Initialize the counter R1
     LOOP:RLC A                ; Rotate left with  cy
          JNC CONT             ; Jump if cy=0 to CONT
          INC R2               ; Increment R2
     CONT:DJNZ R1,LOOP         ; Decrement & jump if R1!=0 to LOOP
     HERE:SJMP HERE            ; End the program
```

Output: R2=04h

10.3.2(d). Write an 8051 program to find the number 64h from the set of five readings starting from address location 50H to 54h. If present store 00h in R0, otherwise store FFh in R0.

Algorithm:

1. Initialize the source pointer & counter register

2. Compare the byte in source pointer location with the byte which has to be searched

3. If both the bytes are equal, store 00h in desired register else repeat steps 2 & 3 till the count becomes zero

4. If the byte is not found, store FFh in desired register

5. End the program

Sample Problem (1):

(50) =76h, (51) =45h, (52) =64h, (53) =25h, (54) =22h

Source Code:

```
             ORG 8000H              ; Starting address
             MOV R1,#50h            ; Initialize the source pointer R1
             MOV R2,#05h            ; Initialize the counter R2
       LOOP:CJNE @R1,#64H,CONT      ; Compare & jump not equal to CONT
             MOV R0,#00h            ; Store 00H in R0 reg
      HERE1:SJMP HERE1              ; End the program
       CONT:INC R1                  ; Increment R1 reg
            DJNZ R2,LOOP            ; Decrement & jump if R2!=0 to LOOP
            MOV R0,#FFh             ; Store FFH in R0 reg
     HERE2:SJMP HERE2               ; End the program
```

Output = (R0) = 00h

Sample prob (2)

Replace data in (52) by 94h

Result = (R0) = FFh

10.4 Result:

We have studied and verified the outputs of simple programs on addressing modes using 8051 Microcontroller instruction set.

Experiment No.11

Timer and counter operations & programming using 8051

11.1 AIM: To Perform Timer0 and Timer1 in Counter Mode and Gated Mode Operation.

11.2 APPARATUS:

1. 8051kit with keyboard,
2. Timer module kit,
3. FRC cables
4. Power supply.

11.3 PROCEDURE:

1. Make the power supply connections from 4-way power mate connector on the ALS-NIFC-09 board.
   - +5V blue wire
   - Ground black wire
2. Connect 26-pin flat cable from interface module to P1 of the trainer kit.
3. Enter the program in the RAM location in 9000 and execute the program GO<STARTING ADDRESS><EXEC>

11.4 PROGRAMS

11.4.1 PROGRAM TO VERIFY TIMER '0'- COUNTER MODE:

ADDRESS	OPCODE	LABEL	MNEMONICS
9200			MOV A,TMOD (TMOD=89)
			ORL A,#05H
			MOV TMOD,A
			SETB TRO (TRO=8C)
			LCALL 68EAH
		LOOP:	MOV DPTR,#0194H
			MOV A,TLO (TLO=8A)
			MOVX @DPTR,A
			INC DPTR
			MOV A,THO (THO=8C)

			MOVX @DPTR,A
			LCALL 6748H
			SJMP LOOP

Execution:1) short jp1 of 1&2 pins and press sw1 for manual increment
2) Short jp1 of 2&3 pins for auto increment

11.4.2 PROGRAM TO VERIFY TIMER-1 COUNTER MODE:

ADDRESS	OPCODE	LABEL	MNEMONICS
9100			MOV A, TMOD (TMOD=89)
			ORL A,#50H
			MOV TMOD,A
			SETB TR1 (TR1=8E)
			LCALL 68EAH
		LOOP	MOV DPTR,#0194H
			MOV A,TL1 (TL1=8B)
			MOVX @DPTR,A
			INC DPTR
			MOV A,TH1 (TH1=8D)
			MOVX @DPTR,A
			LCALL 6748H
			SJMP LOOP

Execution: 1) short jp1 of 5&6 pins and press sw2 for manual increment

2) Short jp2 of 4&5 pins for auto increment

11.5 RESULT: Programs for Timer 0 and Timer 1 in Counter Mode and Gated Mode Operations performed**.**

11.6 Viva Questions:

1) What is the reset address of 8086?

2) What is the size of flag register in 8086? Explain all.

3) What is the difference between 08H and 01H functions of INT 21H?

4) Which is faster- Reading word size data whose starting address is at even or at odd address of memory in 8086?

5) Which is the default segment base: offset pairs?

11.7 EXERCISE:

1. write an ALP program to study timer-1 gated mode

Experiment No.12

Communication with a Host Computer System

12.1 Aim: To Perform Communication Operation with a Host Computer System.

12.2 Apparatus:

1. 8051 kit with keyboard,
2. Computer System
3. Power supply.

12.3 Theory:

ESA 31 operating in the serial mode, can be connected to either a CRT terminal or a host computer system. When a computer system is the controlling element, it must be executing driver software to communicate with the ESA 31 target kit.

ESA 31 is supplied with DOS communication driver package XT51 which allow the user to establish serial communication between the trainer and a host PC thro its Asynchronous com ports (COMI and COM2).

Installation:

a) Configure ESA 86/88E for serial mode of operation and set the serial port of ESA 86/88E for 9600 Baud rate and No parity (keep DIP switches 1 and 4 in ON position)

b) Connect the PC to ESA 31 trainer over COM1/COM2 serial port using the RS232C serial interface cable connecter.

The Serial mode of operation:

a) Supports for downloading user programs in to the target ESA 31 kit from a Host computer system in INTEL HEX format.

b) Also supports for uploading user program to host PC and saving them as HEX files to a system.

Using of X8051 Cross Assembler:

A convenient way of creating a file to be downloaded in to ESA31 is to use a cross assembler for 8051 that can generate the object code in extended HEX format. X8051 is such a package.

It is a powerful cross assembler for 8051. It can run on any PC/XT/AT compatible system and supports all the standard mnemonics, pseudo-opcodes (directives) and addressing modes of 8051.

Steps involved in creating a HEX file are as follows:-

Step1:- Select the path folder directory

```
C:\> cd\
C:\> cd51
C:\> edit filename
```

Step2:- Create a source file using the DOS text editor and save it as filename.asm.

12.4 Example Program:

Write a source program for 16 bit addition using DOS text editor. The numbers are 3CE7H and 3B8DH. Place the sum in R7 and R6; R6 should have the lower byte.

```
        ORG 8000H
        CLR C
        MOV A ,#0E7H
        ADD A,#8DH
        MOV R6,A
        MOV A,#3CH
        ADDC A,#3BH
        MOV R7,A
        HERE:SJMP HERE
```

Save and exit from editor.

Let the source file be saved as add 16.asm

Step3:- Assemble the source file add16.asm using X8051 to create an object file add16.obj as follows:

```
C:\51 x8051
Listing destination <N,T,D,E,L,P<CR>=N>: d
Generate cross reference? <Y/N <CR>=N>: n
Input Filename: add16.asm
Output Filename:
8051 CROSS ASSEMBLER-VERSION 4.00f
Input Filename: add16.asm
Output Filename: add16.obj
Lines Assembled:________ Assembly Errors:_______
C:\51
```

Step4:- Link the single file add16.obj

Specify code offset and options H for HEX format.

This process creates a hex file add16.hex that can download into ESA-31 kit.

```
C:\51 link51
Linker copyright<c>1985-version 4.00g
Input Filename: add16.obj
Enter Offset for 'CODE': 0
Input Filename: add16.obj
Output Filename: add16.hex
Options(D,S,A,M,X,H,E,T,1,2,3,(CR)=Default):h
Link Errors: ________ Output Format:_______
C:\51
```

Step 5:- (Optional)

Check the directory to see the files Created for add16

```
C:\51 dir
```

Also check the list file for add16 as

```
C:\51 edit add16.list
```

Step 6:-

Set the system in the serial communication mode using the XT51 command

```
C:\86>xt51
```

Now the following message will appears on the Screen

```
___________________XT51 Version x, y___________________
ELECTRO SYSTEM ASSOCIATES PVT LTD
BANGALORE
Press any Key to Continue
_______________________________________________________
```

XT51 Checks for the presence of communication ports COM1 and COM2.

If Serial communication is established successfully, the command prompt ‘.’ appears on the screen otherwise the communication parameters are set appropriately using ALT+S command and continue.

Subsequently during the POWER ON RESET, the following sign on message appears on the screen followed by command prompt.

ESA 31 MONITOR VERSION x.y

Step 7:-

Download the program hex file from host PC to ESA 51 trainer using the CTRL+D command

CTRL+D

Specify download filename: add16.hex

Specify memory type: P

Specify starting address: 8000

Specify ending address:

Downloading program

Run the program using G command as

G 8000

Press Break Key

Note:

If input data is to be entered use <MD> command to enter the data.

Press ESC key to return to command Prompt. >

Use the M (Modify memory) command to examine the contents of specified memory locations.

Further, if the locations are in RAM, their contents can be altered if desired.

Format: - M {P|D|I|B} addresses 1[address 2] <CR>

Ex 1: Examine a series of RAM locations starting at 8820H and modify the contents of the location 8822H

```
>MD 8820
8820 XX<CR>
8821 XX<CR>
8822 XX 55<CR>
8823 XX<ESC>
```

EX 2: To enter data at internal RAM locations starting at 40H

```
>MI 40<CR>
```

```
40 XX 21<CR>
41 XX 22<CR>
43 XX 55<CR>
44 XX<ESC>
```

> M (Display Memory) Command

This command is used to display the contents of the Program or External or Internal Memory.

Format:

```
M {P|D|I}, address1.address2<CR>
```

EX:

To display the Contents of 5 bytes from location 8020H

```
>MD 8020, 8024<CR>
```

Experiment No.13

Programming using interrupts

13.1 AIM: Write ALP in 8051 to allow the external interrupt 1.

13.2 APPARATUS: 8051 with keyboard interrupt kit module.

13.3 PROCEDURE:

1. Make the power supply connections from 4-way power mate connector on the ALS- NIFC-09 board.
    - +5V blue wire
    - Ground black wire
2. Connect 26-pin flat cable from interface module to P1 of the trainer kit.
3. Enter the program in the RAM location in 9000 and execute the program GO<STARTING ADDRESS><EXEC>

13.4 PROGRAM:

MEMORY LOCATION	OPCODE	LABEL	MEMONIC
8000			ORG 8000
		AGAIN	LJMP AGAIN
			ORG 0013
			SETB P1.3
			MOV R3,#255
		BACK	DJNZ R3,BACK
			CLR P1.3
			ORG 30H
		MAIN	MOV IE,#10000100B
		HERE	SJMP HERE
			LCALL 03

13.5 RESULT: program for interrupt handling in 8051 verified.
13.6 Viva Questions:

1) Can we use SP as offset address holder with CS?

2) Which is the base registers in 8086?

3) Which is the index registers in 8086?

4) What do you mean by segment override prefix?

5) Whether micro reduces memory requirements?

13.7 EXERCISE:

1. Write an alp program to find the length of the given array using masm software.
2. Write an alp program to find the sum of „n" numbers using masm software.

Experiment No.14

Interfacing 8051 with DAC to generate waveforms

14.1 AIM:

To write and execute program in 8051 assembly language for interfacing a DAC interface module with ESA 31 microcontroller trainer kit.

14.2 APPARATUS:

1. ESA 31 Microcontroller trainer kit
2. Dual channel DAC module
3. Power supply units
4. 26 Pin connector cable
5. CRO

14.3 THEORY:

To use DAC, initialize 8255A for mode 0 operation with port A and port B as output. Output data on the appropriate port and observe output wave form at Xout and Yout of the DAC using CRO.

The 16 bit port addresses for 8255A available at J2 connector are:

Port A	Equ	E800H
Port B	Equ	E801H
Port C	Equ	E802H
Port D	Equ	E803H

Note: Port A controls Xout and Port B controls Yout of DAC interface module.

14.4 PROGRAMS:

Write an ALP to generate Saw tooth (Up-going and Down-going)

Write an ALP to generate Triangular waveform

Write an ALP to generate Symmetrical Square wave

Write an ALP to generate

Up-going stair case with 5 steps

Down-going stair case with 5 steps

```
; Assume the DAC interface is connected over J2 of the ESA 31 trainer.
                    ORG                  8000H
                    PORT_A      EQU      E800H
```

```
PORT_B      EQU    E801H
PORT_C      EQU    E802H
CWR         EQU    E803H
```

14.4.1. Program to generate Continuous up going saw tooth.

Algorithm:

1. Initialize 8255 in mode 0 & all ports as output ports
2. Initialize A with 00h
3. Send the data to the ports A & B through A register
4. Increment A
5. Repeat the steps 3 & 4 for the generation of continuous waveform

Source Code:

```
      ORG 8000H               ; Starting address
      MOV DPTR,#0E803H
      MOV A,#80H              ; Initialize 8255A for mode 0
      MOVX @DPTR,A            ; with PA & PB as OUT
      CLR A                   ; Start with value 00H
AGAIN:MOV DPTR, #0E800H       ; Point to Port A
      MOVX @DPTR,A            ; Out to Port A
      INC DPTR                ; Increment DPTR
      MOVX @DPTR,A            ; Out to Port B
      INC A                   ; Increment DAC input
      SJMP   AGAIN            ; Repeat forever
```

Result: The output waveform is observed on CRO & amplitude and time period is measured.

14.4.2. Program to generate continuous down going saw tooth

Algorithm:

1. Initialize 8255 in mode 0 & all ports as output ports
2. Initialize A with FFh
3. Send the data to the ports A & B through A register
4. Decrement A
5. Repeat the steps 3 & 4 for the generation of continuous waveform

Source Code:

```
            ORG 8000H                  ; Starting address
            MOV DPTR,#0E803H
            MOV A,#80H                 ; Initialize 8255A for mode 0
            MOVX @DPTR,A               ; with PA & PB as OUT
            MOV A,#0FFH                ; Start with value FFH
      AGAIN:MOV DPTR,#0E800H           ; Point to Port A
            MOVX @DPTR,A               ; Out to Port A
            INC DPTR                   ; Increment DPTR
            MOVX @DPTR,A               ; Out to Port B
            DEC A                      ; Decrement DAC input
            SJMP AGAIN                 ; Repeat forever
```

Result: The output waveform is observed on CRO & amplitude and time period is measured.

14.4.3. Program to generate continuous triangular waveform

Algorithm:

1. Initialize 8255 in mode 0 & all ports as output ports
2. Initialize A with 00h
3. Send the data to the ports A & B through A register
4. Increment A
5. Repeat the steps 3 & 4 for the generation of up-going saw-tooth waveform
6. Repeat the steps 2-5 with A = FFh & decrementing A for the generation of down-going saw-tooth
7. Repeat the steps 2-6 for the generation of continuous waveform

Source Code:

```
            ORG 8000H              ; Starting address
            MOV DPTR,#0E803H
            MOV A,#80H             ; Initialize 8255A for mode 0
            MOVX @DPTR,A           ; with PA & PB as OUT
            CLR A                  ; Start with value 00H
         UP:MOV DPTR,#0E800H       ; Point to Port A
            MOVX @DPTR,A           ; Out to Port A
```

```
             INC DPTR               ; Increment DPTR
             MOVX @DPTR,A           ; Out to Port B
             INC A                  ; Increment DAC input
             CJNE A,#0FFH,UP        ; Compare & jump if  A!=FFH to UP
       DOWN:MOV DPTR,#0E800H        ; Point to Port A
             MOVX @DPTR,A           ; Out to Port A
             INC DPTR               ; Increment DPTR
             MOVX @DPTR,A           ; Out to Port B
             DEC A                  ; Decrement DAC input
             CJNE A,#00H,DOWN       ; Compare & jump if  A!=00H to DOWN
             SJMP   UP              ; Repeat forever
```

Result: The output waveform is observed on CRO & amplitude and time period is measured.

14.4.4. Program to generate Symmetrical Square Wave

Algorithm:

1. Initialize 8255 in mode 0 & all ports as output ports
2. Initialize A with 00h
3. Send the data to the port A through A register & provide the delay
4. Make A = FFh, send it through port A & provide the delay
5. Repeat the steps 2-4 for the generation of continuous waveform

Source Code:

```
             ORG 8000H              ; Starting address
             MOV DPTR,#0E803H
             MOV A,#80H             ; Initialize 8255A for mode 0
             MOVX @DPTR,A           ; with PA & PB as OUT
        BACK:MOV A,#0FFH            ; Start with value FFH
             MOV DPTR,#0E800H       ; Point to Port A
             MOVX @DPTR,A           ; Out to Port A
             MOV R0,#0FFH           ; Move FFH to R0
        DLY1:DJNZ R0,DLY1           ; for delay
             MOV A,#00H             ; Now start with value FFH
```

```
            MOVX @DPTR,A          ; Out to Port A
            MOV R0,#0FFH          ; Move FFH to R0
       DLY2:DJNZ R0,DLY2          ; for same delay
            SJMP BACK             ; Repeat forever
```

Result: The output waveform is observed on CRO & amplitude and time period is measured.

14.4.5. Program for Stair case (Up-going) with 5 steps

Algorithm:

1. Initialize 8255 in mode 0 & all ports as output ports
2. Initialize A with 00h
3. Add 33h to A & send the data to the port A through A register & provide the delay
4. Compare A with FFh & Repeat the steps 2 & 3 for the generation of continuous waveform

Source Code:

```
            ORG 8000H               ; Starting address
            MOV DPTR,#0E803H
            MOV A,#80H              ; Initialize 8255A for mode 0
            MOVX @DPTR,A            ; with PA & PB as OUT
            MOV A,#00H              ; Start with value 00H
        RPT:ADD A,#33H              ; Add 33H to A reg
            MOV DPTR,#0E800H        ; Point to Port A
            MOVX @DPTR,A            ; Out to Port A
            MOV R0,#0FFH            ; Move FFH to R0
       DLY1:DJNZ R0,DLY1            ; for delay
            CJNE A,#0FFH,RPT        ; Compare & jump if A!=FFH to RPT
            INC A                   ; Increment A reg
            MOVX @DPTR,A            ; Out to Port A
            MOV R0,#0FFH            ; Move FFH to R0
       DLY2:DJNZ R0,DLY2            ; for delay
            SJMP RPT                ; Repeat forever
```

14.5 Result: The output waveform is observed on CRO & amplitude and time period is measured.

Experiment No.15

Interfacing traffic signal control using 8051

15.1 AIM: To write an assembly language program for traffic light signal control interfacing with 8051. Assume that to be connected over connector J7 of the 8051 trainer kit.

15.2 APPARATUS:

1. 8051 Trainer kit
2. Power supply
3. Traffic light kit

15.3 THEORY:

The traffic light interface simulates the control and operation of traffic lights at a junction of four roads. The interface provides a set of 6 LED indicators at each of the four corners. Each of these LED s can be controlled by a port line. Thus the interface allows the user to simulate a variety of traffic simulations using appropriate software routines.

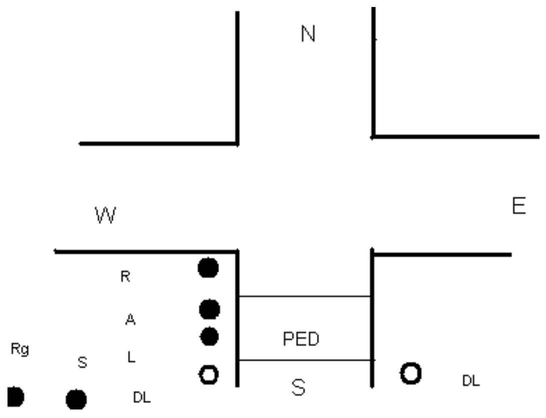
DESCRIPTION OF THE CIRCUIT:

The organization of 6 LED s is identical at each of the four corners. The organization with reference to the LED s at "South-West" corner is shown in the figure below:

R = SOUTH RED

A = SOUTH AMBER L = SOUTH LEFT

S= SOUTH STRAIGHT Rg=SOUTH RIGHT DL=SOUTH PEDESTRIAN



The five LED s (except "Pedestrian") will be ON or OFF depending on the state of corresponding port line LED is ON, if the Port line is Logic „HIGH" and LED is OFF, if it is at logic „LOW‶. The last LED marked

DL is a set of two dual color LED s and they both will be either RED or GREEN depending on the state of the corresponding port line RED if the port line is logic HIGH and GREEN if the port line is logic LOW.

There are four such sets of LED s and these are controlled by 24 port lines of 8255A. Each port line is inverted and buffered using 7406 (open collector inverter buffers) and is used to control an LED. Dual color LEDs are controlled by a port line and its complement.

INSTALLATION:

The interface module has 26-pin connector at one edge of the card. This is used for connecting the interface over J2 of the ESA 31 trainer. The trainer can be in KEYBOARD MODE or SERIAL MODE.

24 LEDS AND CORRESPONDING PORT LINES: PORT A:

D7	D6	D5	D4	D3	D2	D1	D0
ER	EA	ERg	EL	SR	SA	SRg	SL

PORT B:

D7	D6	D5	D4	D3	D2	D1	D0
WR	WA	WRg	WL	NR	NA	NRg	NL

PORT C:

D7	D6	D5	D4	D3	D2	D1	D0
EP	SP	WP	NP	SS	ES	NS	WS

15.4.1.PROBLEM STATEMENT:

Generate the sequence for P_A, P_B, and P_C such that the following traffic situations are simulated.

Vehicles from SOUTH can go NORTH and WEST Vehicles from WEST can go NORTH

Vehicles from NORTH can go SOUTH Pedestrians can cross on EAST

Vehicles from EAST can go WEST and SOUTH Vehicles from WEST can go EAST

Vehicles from SOUTH can go WEST Pedestrians can cross on NORTH

Vehicles from EAST can go SOUTH

Vehicles from NORTH can go SOUTH and EAST Vehicles from SOUTH can go NORTH Pedestrians can cross on WEST

Vehicles from EAST can go WEST

Vehicles from WEST can go EAST and NORTH Vehicles from NORTH can go EAST Pedestrians can cross on SOUTH

No vehicle movement

Pedestrians can cross on all four roads.

The system moves from one state to another state after fixed time delay. The state transition is indicated by turning ON all the AMBER LEDs and all Pedestrians RED LEDs for a fixed duration. The sequence of the above states is repeated again and again.

Algorithm:

1. Initialize 8255 in mode 0 & all ports as output ports
2. Load the look-up table with port values according to the traffic situations
3. Send the port values through the respective port addresses
4. Provide the delay in between the two states
5. Repeat the process to control the traffic continuously

```
; Program memory from 8000H to 804FH
                    ORG                     8000H
                    PORT A    EQU           E800H
                    PORT B    EQU           E801H
                    PORT C    EQEU          E802H
                    CWR       EQU           E803H
; Enter the data mentioned below from 0000H to 001EH in data memory.

                    PORTS:      DB    10H, 81H, 7AH        ;State 1
                    DB                44H, 44H, 0F0H       ;All Ambers ON
                    DB                08H, 11H, 0E5H       ;State 2
                    DB                44H, 44H, 0F0H       ;All Ambers ON
                    DB                81H, 10H, 0DAH       ;State 3
                    DB                44H, 44H, 0F0H
                    DB                11H, 08H, 0B5H       ;State 4
                    DB                44H, 44H, 0F0H
                    DB                88H, 88H, 00H        ;State 5
                    DB                44H, 44H, 0F0
                    DB                00H                  ;Dummy
```

Result: The output is observed on traffic light interface module.

15.4.2. The following sequence of simple traffic conditions are simulated as: Condition 1

Vehicles from SOUTH can go NORTH and WEST

Vehicles from WEST can go NORTH

Vehicles from NORTH can go SOUTH

Pedestrian can cross on EAST

Condition 2

No vehicle movement

Pedestrians can cross on all four roads

Algorithm:

1. Initialize 8255 in mode 0 & all ports as output ports
2. Load the look-up table with port values according to the traffic situations
3. Send the port values through the respective port addresses
4. Provide the delay in between the two states
5. Repeat the process to control the traffic continuously

PORTS: Db A0h, 81h, 7Ah

Db 44h, 44h, 0F0h

Db 88h, 88h, 00h Db 44h, 44h, 0F0h

Db 00h

Result: The output is observed on traffic light interface module.

EXPERIMENT NO.16

Programs to control stepper motor using 8051

16.1 AIM: To write and execute program in 8051 assembly language for interfacing a stepper motor module with ESA 51 microcontroller trainer kit.

16.2 APPARATUS:

1. ESA 8051/31 Microcontroller trainer kit
2. Stepper motor module
3. Power supply units
4. 26 Pin FRC connector cable

16.3 THEORY:

Data acquisition and control represents the most popular applications of microprocessors & microcontrollers. Stepper motor control is a very popular application of microprocessors & microcontrollers in control area as stepper motors are capable of accepting pulses directly from the processors & controllers and move accordingly.

There are two types of Stepper motors:

Permanent Magnet (PM)

Variable Reluctance (VR)

OPERATION OF STEPPER MOTOR:

Stepper motor consists of two important parts, the stator and the rotor. The stator normally has 4 windings on four wheels whereas the rotor is magnetic in nature and has got teeth on it, which is magnetized as North and South poles.

WORKING:

Stepper motor works on the principle of repulsion between magnets. One input to the stepper motor is given in the form of pulses, provided to the windings on the poles as 1000, 0100, 0010, 0001. The windings are provided with input by the 8051 microcontroller through the Port A pins of 8255.

Stator is responsible for creating the magnetic field and rotating the rotor.

SPECIFICATIONS OF THE STEPPER MOTOR USED:

The motor is reversible on the application of a torque of 3Kgcm. The power requirement is +5V DC at 1.2A current per winding at full torque. The step angle is 1.8°, i.e., for every single excitation, the motor shaft rotates by 1.8°.For the motor to rotate one full revolution (360°), number of steps required is

$$360^{o} / 1.8^{o} = 200$$

The stepper motor used has four stator windings which are brought out through colored wires terminated at a 4 pin polarized female connector. The remaining two wires (White & Black wires) are shorted and terminated at 2 pin polarized female connector.

LOOPING:

The number of times the stepper motor should loop is given by:

Count = No. of teeth on rotor X total No. of rotations.

The Port A pins of 8255 (PA0, PA1, PA2, PA3) are used. The values that have to be sent to Port A to drive the stepper motor in clock wise direction are 88h, 44h, 22h, 22h and anti clock wise direction are 11h, 22h, 44h, 88h.

16.4 CIRCUIT DESCRIPTION:

The stepper motor interface uses four transistor pairs (SL100 & 2N3055) in a Darlington pair configuration. Each Darlington pair is used to excite the particular winding of the motor connected to 4 pin connector on the interface. The inputs to these transistors are from the 8255 PPI I/O lines of the microcontroller trainer kit. ‘Port A’ lower nibble PA0, PA1, PA2, PA3 is the four lines brought out to the 26 pin FRC male connector J1 on the interface module. The freewheeling diodes across each winding protect transistors from switching transients.

INSTALLATION:

The interface has two no. of 3 pins and one four pin connectors. Plug in four pin polarized connector of the motor to interface and the 3 pin connector of the motor to the 3 pin connector of the interface marked as “WHT BLK”. Connect the 3 pin female connector of the stepper motor power supply to the connector on the interface marked as “GND +5V/12V”. Connect the 26 core flat ribbon cable to J1 connector on the interface module and the other end of the cable to microcontroller 8051 trainer kit J2.

Switch on power to the trainer kit as well as the stepper motor. Key in the program required for the application and executes the same. When the program is executed, the motor shaft rotates in steps at the speed depending upon the delay between successive steps, which is generated and can be controlled by the program. The direction of rotation can also be controlled through software.

CALCULATIONS:

No. of teeth on rotor = N1 = 50

No. of poles on stator = 8

No. of teeth on stator = 8X5 = N2 = 40

Step angle = $\frac{360^{o}\,(N1 - N2)}{N1 * N2} = 1.8^{O}$

The step angle is 1.8^{O} i.e. for every single excitation; the motor shaft rotates by 1.8^{O}.

```
            PORT A      EQU         E800H
            PORT B      EQU         E801H
            PORT C      EQU         E802H
```

```
        CWR         EQU         E803H
```

16.5 PROGRAM

16.5.1. Write an 8051 program to drive the Stepper motor continuously in clockwise direction.

Algorithm:

1. Initialize 8255 in mode 0 & all ports as output ports
2. Load the input pattern in Acc register & send it through port A address
3. Rotate right the value of Acc register
4. Provide the delay
5. Repeat the process for continuous rotation

Source Code:

```
      ORG 8000H             ; Starting address
      MOV DPTR, #0E803H
      MOV A, #80H           ; Initialize 8255A for mode 0
      MOVX @DPTR, A         ; with PA & PB as OUT
      MOV A, #88H           ; Move the input pattern to A
BACK:MOV DPTR, #0E800H      ; Point to Port A
      MOVX @DPTR, A         ; Out to Port A
      RR A                  ; Rotate right A reg
      MOV R4, #10H          ; Delay routine
LOOP:MOV R3, #0FFH
DLY1:DJNZ R3, DLY1
      DJNZ R4, LOOP
      SJMP BACK             ; Repeat continuously
```

Result: The motor shaft rotates continuously in clockwise direction.

16.5.2. Write an 8051 program to drive the Stepper motor continuously in anti-clockwise direction.

Algorithm:

1. Initialize 8255 in mode 0 & all ports as output ports
2. Load the input pattern in Acc register & send it through port A address
3. Rotate left the value of Acc register
4. Provide the delay
5. Repeat the process for continuous rotation

Source Code:

```
      ORG 8000H             ; Starting address
      MOV DPTR, #0E803H
```

```
            MOV A, #80H             ; Initialize 8255A for mode 0
            MOVX @DPTR, A           ; with PA & PB as OUT
            MOV A, #88H             ; Move the input pattern to A
      BACK:MOV DPTR, #0E800H        ; Point to Port A
            MOVX @DPTR, A           ; Out to Port A
            RL A                    ; Rotate left A reg
            MOV R4, #10H            ; Delay routine
      LOOP:MOV R3, #0FFH
      DLY1:DJNZ R3, DLY1
            DJNZ R4, LOOP
            SJMP BACK               ; Repeat continuously
```

Result: The motor shaft rotates continuously in anti-clockwise direction.

16.5.3. Write an 8051 program to drive the Stepper motor 5 times clockwise & 3 times anti-clockwise direction.

Algorithm:

1. Initialize 8255 in mode 0 & all ports as output ports
2. Initialize the counter for clockwise rotation & register for step size
3. Load the input pattern in Acc register & send it through port A address
3. Rotate right the value of Acc register
4. Provide the delay
5. Repeat the process for clockwise rotation till the count becomes zero
6. Repeat the process for anti-clockwise rotation
7. Repeat the steps 2-6 for continuous rotation

Source Code:

```
            ORG 8000H               ; Starting address
            MOV DPTR,#0E803H
            MOV A,#80H              ; Initialize 8255A for mode 0
            MOVX @DPTR,A            ; with PA & PB as OUT
            MOV R1,#05H             ; Initialize the counter
            MOV R0,#0C8H            ; Initialize the step size
            MOV A,#88H              ; Move the input pattern to A
     BACK1:MOV DPTR,#0E800H         ; Point to Port A
            MOVX @DPTR,A            ; Out to Port A
```

```
              RR A                          ; Rotate right A reg
              MOV R4,#10H                   ; Delay routine
       LOOP1:MOV R3,#0FFH
        DLY1:DJNZ R3,DLY1
             DJNZ R4,LOOP1
             DJNZ R0,BACK1
             DJNZ R1,BACK1
              MOV R1,#03H                 ; Initialize the counter
              MOV R0,#0C8H                ; Initialize the step size
              MOV A,#11H                  ; Move the input pattern to A
       BACK2:MOVX @DPTR,A                 ; Out to Port A
              RL A                      ; Rotate left A reg
              MOV R4,#10H               ; Delay routine
       LOOP2:MOV R3,#0FFH
        DLY2:DJNZ R3,DLY2
              DJNZ R4,LOOP2
              DJNZ R0,BACK2
             DJNZ R1,BACK2
             SJMP BACK1                ; Repeat continuously
```

Result: The motor shaft rotates in clockwise direction 5 times & anti-clockwise direction 3 times.

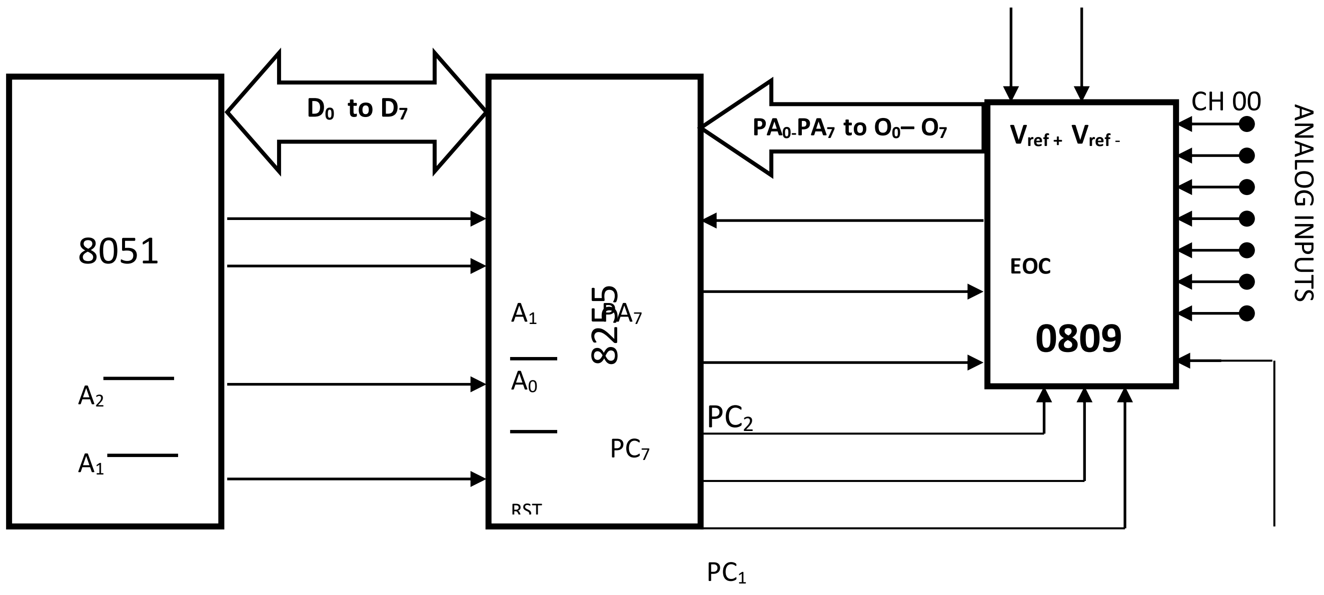
Experiment No.17

ADC interfacing with 8051

17.1 AIM: to interface ADC 0809 with 8051 microcontroller using 8255 ports and measurement of analog voltage.

17.2 APPARATUS REQUIRED: 8051 kit, Power supply unit, ADC module, 26-pin FRC connector.

17.3 INTERFACING DIAGRAM:



8086 Interfacing with ADC0809 through PPI8255

17.4 THEORY:

The analog to digital converter chips 0808 & 0809 are 8-bit CMOS, successive approximation converters. Successive approximation technique is one of the fast techniques for analog to digital conversion. The conversion delay is 100 microseconds at a clock frequency of 640 KHz, which is quite low as compared to other converters. These converters do need any external zero or full scale adjustments as they are already taken care of the internal circuits.

The different pins of 0809 ADC are :

I/P_0-I/P_7	: Analog Inputs
ADD A,B,C	: Address lines for analog inputs
O_7 -O_0	: Digital 8 bit output with O_7 MSB & O_0 LSB
SOC	: Start of Conversion signal pin
EOC	: End of conversion signal pin
OE	: Output latch enable pin if high enables output

CLK	: clock input for ADC
Vcc,GND	: Supply pins +5V and ground

Vref+ & Vref-: Reference voltage positive (+5V max) reference voltage negative (0 Volts min)

17.5 PROCEDURE:

Connect 5V, GND to the interface using the 4 way power-mate

Color codes of the 4 way power connecter (power-mate) on the interface

+5V Blue, GND Black

Connect 26 core flat cable from the 8051 to ADC

Enter the program as per the listing given in the manual.

Execute the program

GO <STARTING ADDRESS><EXEC>

1. POLLED MODE: The message "ENTER CHANNEL NO' is displayed on LCD of the kit in 8051 the 'E' is displayed on the kit. Enter the channel no through keyboard
   (only 0 to 7) and press EXEC. Digital value of the selected channel will be displayed on data field of display.

17.6 PROGRAM:

ADC CONVERSION POLLED MODE

```
2023   CONTROL    EQU    2023H          ;control port address for 8255
2020   PORTA      EQU    2020H          ;porta address for 8255
2021   PORTB      EQU    2021H          ;portb address for 8255
2022   PORTC      EQU    2022H          ;portc address for 8255
677D   UPDD       EQU    677DH          ;display routine addr in data field
679F   GETK       EQU    679FH          ;get & display key in addr field
1020   PUTK       EQU    1020H
2077   RX_BYTE    EQU    2077H
68EA   CLDIS      EQU    68EAH          ;CLRDISPLAY
6946   SET_DD     EQU    6946H          ;SET CURSER POINT
6919   DISPLAY    EQU    6919H          ;DISPLAY

8000          ORG          8000H

8000          12  68  EA   LCALL   CLDIS              ;CLEAR DISPLAY
8003          00           NOP
8004          00           NOP
8005          75  F0  80   MOV     B,#80H
8008          12  69  46   LCALL   SET_DD             ;SET CONTROL WORD
```

```
800B            90  80  7C      MOV       DPTR,#MSG1
800E            12  69  19      LCALL   DISPLAY               ;SEND MESSAGE TO DISPLAY
8011            12  80  6F      LCALL   DELAY

8014            90  20  23      MOV      DPTR,#CONTROL
8017            74  90          MOV      A,#90H               ;PORT a I/P, PORT b,c O/P
8019            F0              MOVX   @DPTR,A
801A            7F  00GET_NEXT:MOV      R7,#00H
801C            00              NOP
801D            12  67  9F      LCALL  GETK                   ;GET KEY FOR CH SELCTN
8020            E5  31          MOV     A,31H
8022            54  0F          ANL      A,#0FH               ;GETTING EXACT CHANNEL NO
8024            FF              MOV     R7,A
8025            EF        ADC: MOV     A,R7
8026            90  20  22      MOV     DPTR,#PORTC           ;OUT PUT CH NO
8029            F0              MOVX  @DPTR,A
802A            74  0D          MOV     A,#0DH                ;SET PC6 (OE)
802C            90  20  23      MOV     DPTR,#CONTROL
802F            F0              MOVX  @DPTR,A
8031            90  20  23      MOV     DPTR,#CONTROL
8034            74  0F          MOV     A,#0FH                ;PC7 SET START\ALE
8036            F0              MOVX  @DPTR,A
8036            7B  30          MOV     R3,#30H               ;30MSEC
8038            7C  FF    LOP:MOV     R4,#FFH
803A            00        LOP1:NOP
803B            00              NOP
803C            DC  FC          DJNZ     R4,LOP1
803E            DB  F8          DJNZ     R3,LOP
8040            74  0E          MOV     A,#0EH                ;PC7 RESET
8042            F0              MOVX  @DPTR,A
8043            00              NOP
8044            00              NOP
8045            90  20  23      MOV     DPTR,#CONTROL
8048            74  0C          MOV     A,#0CH                ;RESET PC6 TO READ EOC
804A            F0              MOVX  @DPTR,A
804B            90  20  20AD1:MOV     DPTR,#PORTA           ;POLL  EOC LINE
804E            E0              MOVX  A, @DPTR
804F            20  E7  F9      JB        ACC.7,AD1           ;HI TO LO
8052            90  20  20AD2:MOV     DPTR,#PORTA           ;POLL  EOC LINE
8055            E0              MOVX  A, @DPTR                ;LO TO HI
8056            54  80          ANL      A,#80H
8058            30  E7  F7      JNB      ACC.7,AD2
805B            74  0D          MOV     A,#0DH                ;SET PC6 (OE)
805D            90  20  23      MOV     DPTR,#CONTROL
8060            F0              MOVX  @DPTR,A
8061            90  20  20      MOV     DPTR,#PORTA           ;READ DIGITAL VALUE
8064            E0              MOVX  A, @DPTR
8065            FE              MOV     R6,A
8066            12  67  7D      LCALL UPDD                    ;DISPLAY IN DATA FIELD
8069            12  80  6F      LCALL DELAY
806C            02  80  1A      LJMP   GET_NEXT               ;GO FOR NEXT CHANNEL
806F
```

```
806F              DELAY:
806F        7A  0F          MOV     R2,#0FH
8071        7C  FF       L1:MOV     R4,#FFH              ;93 MSEC DELAY
8073        7B  FF L_OOP:MOV     R3,#FFH
8075        DB FE LO_OP:DJNZ    R3,LO_OP
8077        DC FA           DJNZ    R4,L_OOP.
8079        DA F6           DJNZ    R2,L1
807B        22              RET
807C

807C        41  44  43  20  50  MSG1  DB  ‘ADC POLLED MODE’,0DH
8081        4F  4C  4C  45  44
8086        20  4D  4F  44  45
808B        0D

808C                        END
```

17.7 RESULT: Thus interfaced ADC with 8051 microcontroller.

Experiment No.18

Serial RTC interfacing with 8051

18.1 Aim: To write an assembly language program for LEDs and switches interfacing with 8051. Assume that to be connected over connector J7 of the 8051 trainer kit.

18.2 Apparatus:

1. 8051 Trainer kit
2. Power supply
3. Serial RTC interfacing kit

18.3 Theory:

A Real Time Clock (RTC) is basically just like a watch – it runs on a battery and keeps time for you even when there is a power outage. Using an RTC, you can keep track of long timelines, even if you reprogram your microcontroller or a power plug.

The real time clock (RTC) is widely used device that provides accurate time and date for many applications.

Many systems such as IBM pc come with RTC chip on mother board.RTC chip uses an internal battery which keeps time and date even when the power is off. In some microcontrollers have inbuilt RTC while others requires interfacing.

Most widely used RTC chip is DS1307 from Dallas Semiconductor.

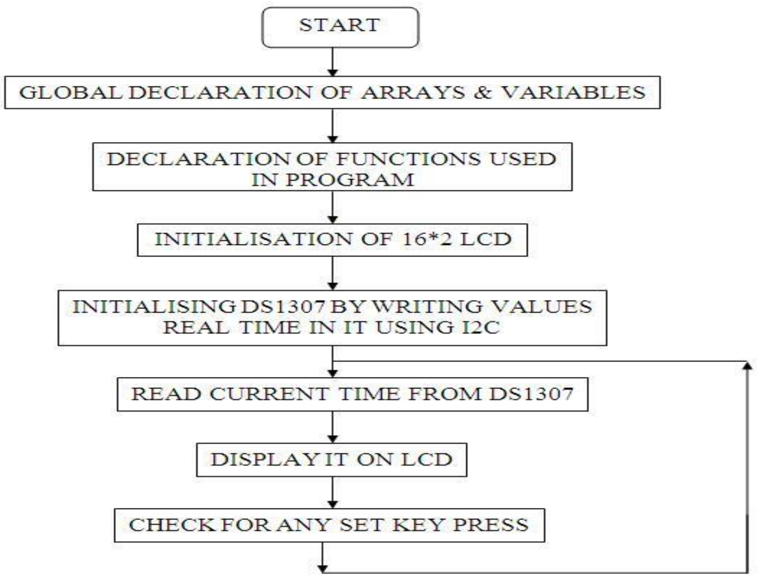
The chip DS1307 with Philips microcontroller P89V51RD2. Microcontroller communicates with DS1307 by using I2C communication protocol. I2C interface can operate with data transfer rate up to 400k bits per second. Microcontroller can operate in transmitter or receiver mode at a time. Received data from RTC chip we displayed by using LCD. All the settings related to timing and date, we did it by using two push buttons.

Connections: Following table shows the use of microcontroller pins in circuit designing

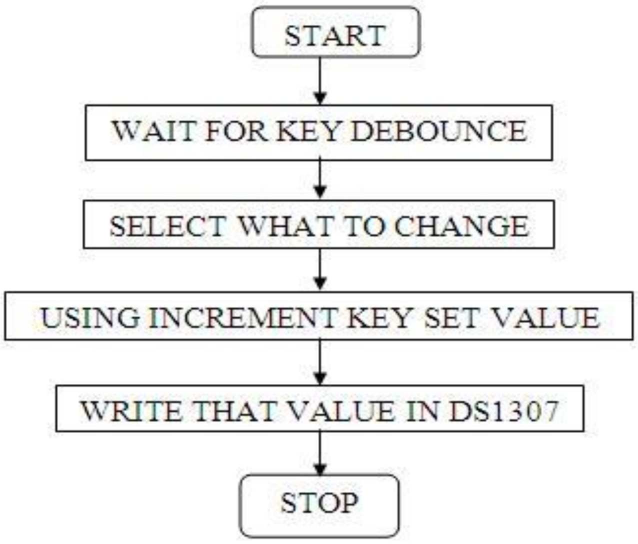
Micro-controller Pin	Connection
P1.0 – P1.7	D0- D7 of LCD
P2.0	SET Button
P2.1	RS pin of LCD

P2.2	INC. Button
P2.3	En pin of LCD
P2.4	SCL of DS1307
P2.5	SCL of DS1307

18.4 Circuit Flowcharts & Proteus Outputs


START
GLOBAL DECLARATION OF ARRAYS & VARIABLES
DECLARATION OF FUNCTIONS USED IN PROGRAM
INITIALISATION OF 16*2 LCD
INITIALISING DS1307 BY WRITING VALUES REAL TIME IN IT USING I2C
READ CURRENT TIME FROM DS1307
DISPLAY IT ON LCD
CHECK FOR ANY SET KEY PRESS

a. **Main program's flowchart:**



SET TIME FLOWCHART:

18.5 Source Code:

```
/ interfacing ds1307 with 80C51

#include<reg51.h>

/* pins used for external h/w */

sbit RS=P2^1;  //connect p2.1 to rs pin of lcd

sbit EN=P2^3;  //connect p2.3 to en pin of lcd

sbit SCL=P2^4;  //i2c clock pin

sbit SDA=P2^5;  //i2c data pin

sbit SET=P2^0; //set button pin

sbit INR=P2^2; //increment button pin

/* some required define(s)*/

#define delay_us _nop_(); //generates 1 microsecond delay

#define LCD P1 //port1 connected to LCD data pins

#define SCLHIGH  SCL=1;

#define SCLLOW   SCL=0;

#define SDAHIGH  SDA=1;

#define SDALOW   SDA=0;

/*various functions used in whole program*/

void _nop_(void);

void init_lcd(void);

void cmd_lcd(unsigned char);

void write_lcd(unsigned char);

void display_lcd(unsigned char *);

void delay_ms(unsigned int);

void init_rtc(void);

void set_rtc(void);

void strt_msg(void);
void start(void);

void stop(void);

void send_byte(unsigned char);

unsigned char receive_byte(unsigned char);

void write_i2c(unsigned char,unsigned char,unsigned char);

void set_value(void);

void stpwtch(void);

unsigned char read_i2c(unsigned char,unsigned char);

//Give time here to set initial values to ds 1307 as specified in timekeeper register
unsigned char clock[]={0x00,0x59,0x23,0x04,0x20,0x10,0x11};
//clock[]={seconds,minutes,hours,day_of_week,date,month,year};

unsigned char stp[]={0x00,0x00,0x00};
//stopwatch initial value

unsigned char *s[]={"SUN","MON","TUE","WED","THU","FRI","SAT"};

unsigned char slave_ack,add=0,c,k,sas;

unsigned int num;

void main(void)
{
        init_lcd();

        strt_msg();

        //COMMENT THIS SECTION WHILE TRANSFRING PROGRAM SECOND TIME IN H/W
        init_rtc();

        //always do this
        while(1)
        {
                if(SET==0)
                set_value();

                c=read_i2c(0xd0,0x02);//read hours register and display on LCD
                /* IMP rtc ds 1307 understands BCD no.sys. our 8051 uC understands HEX no.sys.
                 and LCD requires ASCII data,,,,,,,,,,,,,,,,
                 e.g. lets consider if data read from 1307 is 12 (BCD) i.e. 0001 0010 (BIN)
                 so 8051 consider it as 18 (DECIMAL)
                 x1=(18/16)+48=49(ASCII) i.e. lcd will show 1 and
                 x2=(18%16)+48=50(ASCII) i.e. lcd will show 2
                 i.e. 12 on lcd                              */
                write_lcd((c/16)+48);
                write_lcd((c%16)+48);
                write_lcd(':');
                sas = c & 0x20;


                c=read_i2c(0xd0,0x01);//read minutes register and display on LCD
                write_lcd((c/16)+48);
                write_lcd((c%16)+48);
                write_lcd(':');

                c=read_i2c(0xd0,0x00);//read seconds register and display on LCD
                write_lcd((c/16)+48);
                write_lcd((c%16)+48);
                write_lcd(' ');

                display_lcd(s[read_i2c(0xd0,0x03)]);//read day register and display
                //write_lcd(*s[read_i2c(0xd0,0x03)]);

                cmd_lcd(0xc0);// Go to starting position of 2nd line of LCD

                c=read_i2c(0xd0,0x04);//read date register and display on LCD
                write_lcd((c/16)+48);
                write_lcd((c%16)+48);
                write_lcd('/');

                c=read_i2c(0xd0,0x05);//read month register and display on LCD
                write_lcd((c/16)+48);
                write_lcd((c%16)+48);
                write_lcd('/');

                write_lcd('2'); //write 1st 2 digits of year bcoz only last 2 bits are stored in rtc
                write_lcd('0');
                c=read_i2c(0xd0,0x06);//read year register and display on LCD
                write_lcd((c/16)+48);
                write_lcd((c%16)+48);

                write_lcd(32);    //THIS SECTION SHOWS am/pm
                if(sas == 0x20)
                {
                        display_lcd("AM");
                        //write_lcd(49);
                }
                else
                {
                        //write_lcd(48);
                          display_lcd("PM");
                }

                delay_ms(110);
                cmd_lcd(0x01); // Go to starting position of 1st line of LCD
        }
```

```
}



void start(void) //starts i2c, if both SCK & SDA are idle
{
        if(SCL==0) //check SCK. if SCK busy, return else SCK idle
        return;
        if(SDA==0) //check SDA. if SDA busy, return else SDA idle
        return;

        SDALOW  //High to Low transition on data line SDA makes d start condition
        delay_us

        SCLLOW  //clock low
        delay_us
}

 void stop(void) //stops i2c, releasing the bus
{
        SDALOW //data low
        SCLHIGH //clock high
        delay_us
        SDAHIGH //Low to High transition on data line SDA makes d stop condition
        delay_us
}

void send_byte(unsigned char c) //transmits one byte of data to i2c bus
{
        unsigned mask=0x80;
        do   //transmits 8 bits
        {
        if(c&mask)  //set data line accordingly(0 or 1)
        SDAHIGH //data high
        else
        SDALOW  //data low

        //generate colck
        SCLHIGH   //clock high
        delay_us

        SCLLOW   //clock low
        delay_us

        mask/=2;  //shift mask
        }while(mask>0);

        SDAHIGH  //release data line for acknowledge
        SCLHIGH  //send clock for acknowledge
        delay_us
        slave_ack=SDA; //read data pin for acknowledge
        SCLLOW  //clock low
        delay_us
}

unsigned char receive_byte(unsigned char master_ack) //receives one byte of data from i2c bus
{
        unsigned char c=0,mask=0x80;
        do   //receive 8 bits
        {
        SCLHIGH  //clock high
        delay_us

        if(SDA==1)  //read data
        c|=mask;    //store data
          SCLLOW   //clock low
          delay_us
          mask/=2;  //shift mask
        }while(mask>0);

        if(master_ack==1)
        SDAHIGH  //don't acknowledge
        else
        SDALOW  //acknowledge

           SCLHIGH  //clock high
           delay_us

           SCLLOW  //clock low
           delay_us

        SDAHIGH  //data high

        return c;
}

 void write_i2c(unsigned char device_id,unsigned char location,unsigned char c)
//writes one byte of data(c) to slave device(device_id) at given address(location)
{
do
{
start();       //starts i2c bus
send_byte(device_id);   //select slave device
if(slave_ack==1)   //if acknowledge not received, stop i2c bus
stop();
}while(slave_ack==1);  //loop until acknowledge is received

send_byte(location);  //send address location
send_byte(c);  //send data to i2c bus
stop();  //stop i2c bus
}
```

```
unsigned char read_i2c(unsigned char device_id,unsigned char location)
//reads one byte of data(c) from slave device(device_id) at given address(location)
{
unsigned char c;
do
{
start();   //starts i2c bus
send_byte(device_id); //select slave device
if(slave_ack==1) //if acknowledge not received, stop i2c bus
stop();
}while(slave_ack==1); //loop until acknowledge is received
 send_byte(location);  //send address location
stop(); //stop i2c bus
start(); //starts i2c bus
send_byte(device_id+1); //select slave device in read mode
c=receive_byte(1); //receive data from i2c bus
stop(); //stop i2c bus
return c;
}

void init_lcd(void)
//initialize lcd
{
delay_ms(10); //delay 10 milliseconds

cmd_lcd(0x0e); //lcd on, cursor on
delay_ms(10);

cmd_lcd(0x38); //8 bit initialize, 5x7 character font, 16x2 display
delay_ms(10);

cmd_lcd(0x06); //right shift cursor automatically after each character is displayed
delay_ms(10);

cmd_lcd(0x01); //clear lcd
delay_ms(10);

cmd_lcd (0x80);
}

void cmd_lcd(unsigned char c)
//transmit command or instruction to lcd
{
EN=1;
RS=0; //clear register select pin
LCD=c; //load 8 bit data
EN=0; //clear enable pin
delay_ms(2); //delay 2 milliseconds
}

void write_lcd(unsigned char c)
//transmit a character to be displayed on lcd
```

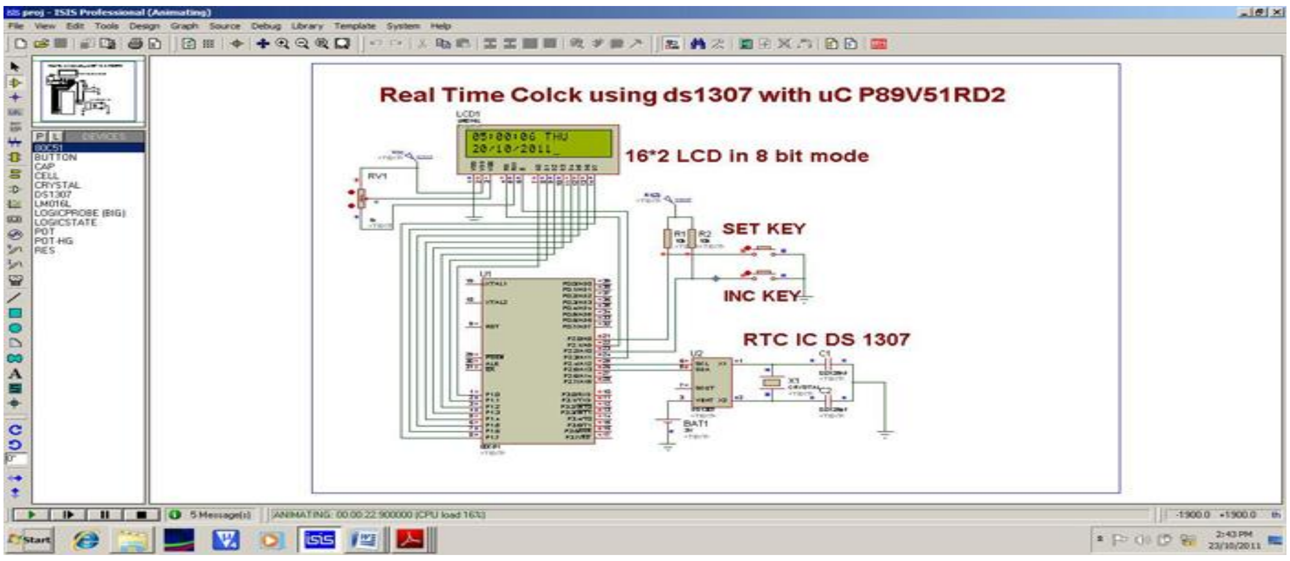
```
{
EN=1; //set enable pin
RS=1; //set register select pin
LCD=c;  //load 8 bit data
EN=0; //clear enable pin
delay_ms(2); //delay 2 milliseconds
}

 void display_lcd(unsigned char *s)
//transmit a string to be displayed on lcd
{
while(*s)
write_lcd(*s++);
}

void delay_ms(unsigned int i)
//generates delay in milli seconds
{
unsigned int j;
while(i-->0)
{
for(j=0;j<500;j++)
{
   ;
}
}
}

void set_value(void)
//this function used for setting time using SET & INC buttons or pins
{
cmd_lcd(0x80);
display_lcd("WELCOME TO TIME");
cmd_lcd(0xC0);
display_lcd(" SET WIZARD !!!");
delay_ms(300);
cmd_lcd(0x01);
cmd_lcd(0x80);
display_lcd(" SET YOUR RTC ?");
cmd_lcd(0xC0);
display_lcd("YES        NEXT");
while(1)
{
if(SET==0)
{
set_rtc();
break;
}
if(INR==0)
{
cmd_lcd(0x01);
stpwtch();
```

```
break;
}
}
}

void init_rtc()
{
while(add<=6)    //update real time clock ic i.e. ds1307 with time
{
write_i2c(0xd0,add,clock[add]);
add++;
}
}

void strt_msg()
{
unsigned int i,j=0;
display_lcd("Welcome to RTC");
cmd_lcd(0xc0);
display_lcd("<<<<<<<<>>>>>>>");
delay_ms(300);    //"...(#@#@#)..."
cmd_lcd(0x01);
display_lcd("SKIP INTRODUCTION");
cmd_lcd(0xc0);
display_lcd("YES           NO");
while(1)
{
if(SET==0)
{
delay_ms(40);
break;
}
if(i==1000)
{
j++;
i=0;
}
if(INR==0|j==100)
{
cmd_lcd(0x01);
display_lcd("THIS PROJECT IS");
cmd_lcd(0xc0);
display_lcd("DONE BY T.E. ELN");
delay_ms(500);
cmd_lcd(0x01);
display_lcd("<ROLL NO> <NAME>");
delay_ms(250);
cmd_lcd(0x01);
display_lcd("38 AVINASH PATIL");
cmd_lcd(0xc0);
display_lcd("44 AMIT SALUNKHE");
delay_ms(550);
```

```
cmd_lcd(0x01);
display_lcd("46 SWAPNIL SANKPAL");
cmd_lcd(0xc0);
display_lcd("48 SUMIT SHEKHAR");
delay_ms(550);
cmd_lcd(0x01);
display_lcd("49 PRANAV SHINDE");
delay_ms(300);
cmd_lcd(0x01);
display_lcd("STARTING RTC....");
cmd_lcd(0xC0);
for(i=0;i<17;i++)
{
display_lcd(".");
delay_ms(15);
}
break;
}
i++;
}
cmd_lcd(0x01);
}

void set_rtc()
{
   unsigned char cnt=0x00;
unsigned char q,p,i=0x00;
while(1)
{
if(SET==0x00)
{
cnt++;
delay_ms(50);
cmd_lcd(0x01); // Go to starting position of 1st line of LCD
cmd_lcd(0xc0);
display_lcd("NEXT     INC");
cmd_lcd(0x80);
switch(cnt)
{
case 1:
  display_lcd("Minutes:");
break;
case 2:
  display_lcd("Hours  :");
break;
case 3:
  display_lcd("Day    :");
break;
case 4:
  display_lcd("Date   :");
break;
case 5:
  display_lcd("Month  :");
break;
   case 6:
  display_lcd("Year   :");
}
}
if(INR==0)
break;
if(cnt>6)
return;
}
cmd_lcd(0xc0);
display_lcd("SAVE     INC");
while(1)
{   if(INR==0)
 {
     delay_ms(10);
  cmd_lcd(0x8A); // Go to starting position of 2nd line of LCD
  p++;
  delay_ms(20);
   switch(cnt)
{
case 1:
if(p>59)
{
  p=0;
}
break;
case 2:
  if(p>23)
{
  p=0;
}
   break;
case 3:
   if(p>7)
{
  p=0;
}
break;
case 4:
   if(p>31)
{
  p=0;
}
break;
case 5:
   if(p>12)
{
  p=0;
}
       break;
   case 6:
   if(p>99)
{
  p=0;
}
}
  q=(p/10)<<4;
  q=q|(p%10);
  write_lcd((q/16)+48);
  write_lcd((q%16)+48);
 }
if(SET==0)
   break;
}
write_i2c(0xD0,cnt,q);
cmd_lcd(0x01);
display_lcd("SAVING CHAGES");
cmd_lcd(0xc0);
display_lcd("PLEASE WAIT");
// delay_ms(100);
for(i=0;i<6;i++)
{
display_lcd(".");
delay_ms(40);
}
cmd_lcd(0x01);
}
```

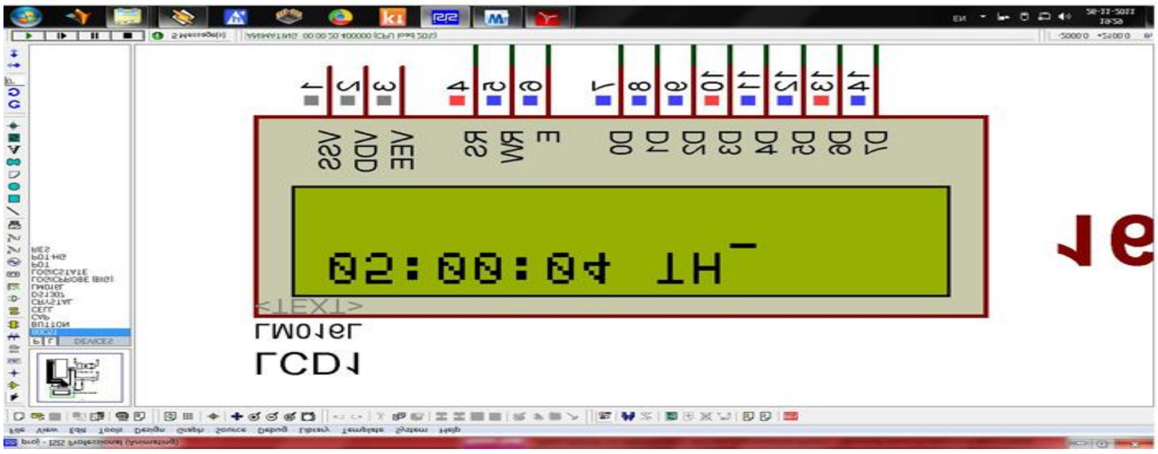
18.6 Result:

Proteus Window showing output



```
05:00:06 THU
20/10/2011_
```

ANIMATING: 00:00:22.900000 (CPU load 16%)


VSS
VDD
VEE
RS
RW
E
D0
D1
D2
D3
D4
D5
D6
D7
LCD1
LM016L
<TEXT>

Experiment No.19
LCD interfacing with 8051

19.1 AIM: Interface an LCD with 8051 microcontroller.

19.2 APPARATUS:

1. 8051 Trainer kit
2. LCD module
3. FRC cables
4. Power Supply.

19.3 PROCEDURE:

1. Make the power supply connections from 4-way power mate connector on the ALS- NIFC-09 board.
   - ▪.............. +5V blue wire
   - ▪.............. Ground black wire
2. Connect 26-pin flat cable from interface module to P1 of the trainer kit.
3. Enter the program in the RAM location in 9000 and execute the program GO<STARTING ADDRESS><EXEC>

19.4 PROGRAM:

```
CNTRL          EQU    2043H            ; 8255 control port address
PORTC          EQU    2042H            ; 8255 port C address
PORTB          EQU    2041H            ; 8255 port B address
PORTA          EQU    2040H            ; 8255 port A address
FUNCTION_SET   EQU    38H              ; display commands
DIS_ON_OFF EQU 0EH
RETURN_HOME    EQU    02H
MODE_SET   EQU 06H
CLEAR_DIS  EQU 01H
DDRAM_ADD      EQU    80H
CNT EQU 40H

          CNT1 EQU
```

```
41H CNT2
EQU 42
```

ADDRESS	OP CODE	LABEL	MNEMONICS
			MOV SP,#50H
			MOV PSW,#00H
			MOV CNT2,#10H
			MOV R0,#14H
			MOV R1,#FFH
			LCALL DELAY
			MOV DPTR,#CNTRL
			MOV A,#80H
			MOVX @DPTR,A
			LCALL SET_CON_LINES
		BACK	MOV R2,#03H
			LCALL SET_WR_CON_LINES
			MOV A,#00H
			MOVX @DPTR,A
			MOV DPTR,#PORTA
			MOV A,#FUNCTION_SET
			MOVX @DPTR,AMOV
			DPTR,#CNTRL
			MOV A,#05H
			MOVX @DPTR,A
			NOP
			NOP
			MOV A,#04H
			MOVX @DPTR,A
			MOV R0,#06H
			MOV R1,#E4H
			LCALL DELAY
			DJNZ R2,BACK

			LCALL CHK_BUSY

			LCALL SET_WR_CON_LINES MOV A,#00H MOVX @DPTR,A MOV DPTR,#PORTA MOV A,#DIS_ON_OFF MOVX @DPTR,A MOV DPTR,#CNTRL MOV A,#05H MOVX @DPTR,A NOP NOP MOV A,#04H MOVX @DPTR,A LCALL CHK_BUSY LCALL SET_WR_CON_LINES MOV A,#00H MOV DPTR,#PORTA MOV A,#RETURN_HOME MOVX @DPTR,A MOV DPTR,#CNTRL MOV A,#05H MOVX @DPTR,A NOP NOP MOV A,#04H MOVX @DPTR,A LCALL CHK_BUSY LCALL SET_WR_CON_LINES MOV A,#00H MOVX @DPTR,A MOV DPTR,#PORTA

			MOV A,#MODE_SET MOVX @DPTR,A MOV DPTR,#CNTRL MOV A,#05H MOVX @DPTR,A NOP NOP MOV A,#04H MOVX @DPTR,A LCALL CHK_BUSY LCALL SET_WR_CON_LINES MOV A,#00H MOVX @DPTR,A MOV DPTR,#PORTA MOV A,#CLEAR_DIS MOVX @DPTR,A MOV DPTR,#CNTRL MOV A,#05H MOVX @DPTR,A NOP NOP MOV A,#04H MOVX @DPTR,A MOV CNT1,#02H MOV CNT,#08H MOV R0,#DDRAM_ADD LCALL CHK_BUSY LCALL SET_WR_CON_LINES MOV A,#00H MOV DPTR,#PORTA MOV A,R0

		BACK3	MOVX @DPTR,A MOV DPTR,#CNTRL MOV A,#05H MOVX @DPTR,A NOP NOP MOV A,#04H MOVX @DPTR,A CLR A MOV DPTR,#MSG MOVX A,@DPTR MOV R1,A INC DPTR PUSH DPH PUSH DPL LCALL CHK_BUSY LCALL SET_WR_CON_LINES MOV A,#01H MOVX @DPTR,A MOV DPTR,#PORTA MOV A,R1 MOVX @DPTR,A MOV DPTR,#CNTRL MOV A,#05H MOVX @DPTR,A NOP NOP MOV A,#04H

			MOVX @DPTR,A POP DPL POP DPH

		F1	CLR A PUSH R0 PUSH R1 MOV R0,#7FH MOV R1,#FFH LCALL DELAY POP R1 POP R0 DJNZ CNT,BACK3 DJNZ CNT1,F1 DJNZ CNT2,FORW1 LJMP FORW MOV CNT,#08H PUSH DPH PUSH DPL LCALL CHK_BUSY LCALL SET_WR_CON_LINES MOV A,#00H MOVX @DPTR,A MOV DPTR,#PORTA MOV A,#C0H MOVX @DPTR,A MOV DPTR,#CNTRL MOV A,#05H MOVX @DPTR,A NOP

			NOP MOV A,#04H MOVX @DPTR,A POP DPL POP DPH

		FORW1	CLR A LJMP BACK3 PUSH DPH PUSH DPL MOV R0,#DDRAM_ADD LCALL CHK_BUSY LCALL SET_WR_CON_LINES MOV A,#00H MOVX @DPTR,A MOV DPTR,#PORTA MOV A,R0 MOVX @DPTR,A MOV DPTR,#CNTRL MOV A,#05H MOVX @DPTR,A NOP NOP MOV A,#04H MOVX @DPTR,A MOV CNT,#08H MOV CNT1,#02H POP DPL POP DPH

	FORW **SET_CON_LINES:**	CLR A LJMP BACK3 :LCALL 0003H MOV DPTR,#CNTRL MOV A,#01H MOVX @DPTR,A MOV A,#03H MOVX @DPTR,A

		CHK_BUSY: **BACK2**	MOV A,#04H MOVX @DPTR,A RET MOV DPTR,#CNTRL MOV A,#90H MOVX @DPTR,A MOV A,#04H MOVX @DPTR,A MOV A,#00H MOVX @DPTR,A MOV A,#03H MOVX @DPTR,A MOV A,#05H MOVX @DPTR,A MOV DPTR,#PORTA MOVX A,@DPTR MOV B,A MOV DPTR,#CNTRL

			MOV A,#04H MOVX @DPTR,A
		F2 **SET_WR_CON_LINES**:	MOV A,B JNB A.7,F2 LJMP BACK2 MOV DPTR,#CNTRL MOV A,#80H MOVX @DPTR,A RET MOV DPTR,#CNTRL MOV A,#04H MOVX @DPTR,A MOV A,#02H
		DELAY: LOOP1: LOOP:	MOVX @DPTR,A RET PUSH R1 NOP DJNZ R1,LOOP POP R1 DJNZ R0,LOOP1 RET

19.5 RESULT: program for interfacing an LCD with 8051 microcontroller performed.

19.6 Viva:

1) What do you mean by emulator?
2) Stack related instruction?
3) What do you mean by 20 dup (0)?
4) Which flags of 8086 are not present in 8085?

19.7 EXERCISE:

1) Write an alp program to perform an operation to find the cubes of a given number using masm software
2) Write an alp program to perform an operation to find the cubes of a given numbers using MP trainer kit

BEYOND THE SYLLABUS

Experiments beyond the Syllabus

1. FACTORIAL OF A GIVEN NUMBER

1.1 AIM: To write an assembly language program to factorial of a given number using MASM Tool.

1.2 APPARATUS:

1. PC with windows OS
2. MASM Tool

1.3 PROGRAM:

```
ASSUME CS: CODE, DS: DATA
    DATA SEGMENT
    C DB  04H
    B DB 01 DUP (0)
    DATA ENDS
    CODE SEGMENT
     START: MOV AX, DATA
              MOV DS, AX
              MOV AX, 0001H
              MOV CX, C
     NEXT: MUL CX
              LOOP NEXT
              MOV  B, AX
              INT 03
              CODE ENDS
              END START
```

1.4 Output:

```
  -g      ax=0018 bx=0000 cx=0000 dx=0000 sp=0000 bp=0000 si=0000 di=0000
ds: 1098 es=1088 ss=1098 cs=1099 ip=0013
-d ds:0   04 18
-u ds:0
```

2. THE MEDIAN FROM THE GIVEN ARRAY OF NUMBERS

2.1 AIM: To write an assembly language program for 8086 to pick the median from the given array of numbers.

2.2 APPARATUS:

1. PC with windows OS
2. MASM Tool

2.3 PROGRAM:

```
ASSUME DS: DATA, CS: CODE
        DATA SEGMENT
                A DB 0AH,03H,07H,05H,04H,08H,0EH,01H
                COUNT EQU 08H
                MEDIAN DB 01H DUP(0)
        DATA ENDS
        CODE SEGMENT
        START:MOV AX, DATA
                MOV DS, AX
                MOV DL, COUNT-1
                MOV CL, 0000H
        L3:     MOV CL, DL
                MOV SI, OFFSET A
        L2:     MOV AL, [SI]
                CMP AL, [SI+1]
                JC L1
                XCHG [SI+1], AL
                XCHG [SI], AL
        L1:     INC SI
                LOOP L2
                DEC DL
                JNZ L3
                LEA SI, A
                MOV AX, 0000H
                MOV AX, COUNT
                MOV BL, 02H
                DIV BL
                CMP AH, 00H
                JZ L4
                ADD SI, AX
                MOV DL, [SI]
                MOV MEDIAN, DL
                JMP EXIT
        L4:     ADD SI, AX
                MOV AX, 0000H
                MOV AL, [SI]
                ADD AL, [SI-1]
                MOV BL, 02H
                DIV BL
                MOV MEDIAN, AL
        EXIT:   INT 03H
        CODE ENDS
        END START
```

3. THE GIVEN STRING IS A PALINDROME OR NOT

3.1 AIM: To write an assembly language program to reverse the given string and verify whether it is a palindrome or not.

3.2 APPARATUS:

1. PC with windows OS
2. MASM Tool

3.3 PROGRAM:

```
ASSUME DS:DATA, CS:CODE, ES:EXTRA
                 DATA SEGMENT
                        STRING1 DB “CMRTC”
                        STRLEN EQU ($-STRING1)
                        MSG1 DB “THE GIVEN STRING IS A PALINDROME$”
                        MSG2 DB “THE GIVEN STRING IS NOT A PALINDROME$”
                DATA ENDS
               EXTRA SEGMENT
                       STRING2 DB STRLEN DUP(0)
                EXTRA ENDS
               CODE SEGMENT
                           START:MOV AX, DATA
                            MOV DS, AX
                            MOV AX, EXTRA
                            MOV ES, AX
                            MOV AX, 0000H
                            MOV SI, OFFSET STRING1
                            MOV DI, OFFSET STRING2
                           ADD DI, STRLEN-1
                           MOV CX, STRLEN
                        L1:MOV AL, DS:[SI]
                           MOV ES:[DI], AL
                           INC SI
                           DEC DI
                           LOOP L1
                          MOV SI,OFFSET STRING1
                          MOV DI,OFFSET STRING2
```

```
                  MOV CX, STRLEN
                 CLD
                  REP CMPSB
                  JE PAL
                 MOV DX, OFFSET MSG2
                MOV AH,09
                INT 21H
                 JMP NEXT
            PAL: MOV DX, OFFSET MSG1
                 MOV AH, 09H
                 INT 21H
          NEXT:INT 03
     CODE ENDS
     END START
```

3.4 Output:

Methodist-The given string is not a palindrome.

RADAR- The given string is a palindrome.

4. POSITIVE & NEGATIVE NUMBERS IN A GIVEN SERIES

4.1 AIM: To write an assembly language program to find the number of positive& negative numbers in a given series.

4.2 APPARATUS:

1. PC with windows OS
2. MASM Tool

4.3 PROGRAM:

```
ASSUME CS: CODE, DS: DATA
  DATA SEGMENT
  LIST DB 12H, -11H, 13H, -20H, 25H,-14H
  COUNT EQU 07H
  DATA ENDS
  CODE SEGMENT
   START: MOV AX, DATA
          MOV DS, AX
          XOR BX, BX
          XOR DX, DX
          MOV CL, COUNT
          MOV SI, OFFSET LIST
 AGAIN:  MOV AX,[SI]
          SHL AX, 01H
          JC NEG1
           INC BX
           JMP NEXT
 NEG1:  INC DX
 NEXT:   ADD SI, 02
          DEC CL
           JNZ AGAIN
           INT 03
           CODE ENDS
           END START
```

4.4 Output:

```
-g          ax=4cdc bx=0004 cx=0003 dx=0003 sp=0000 bp=0000 si=00007 di=0000
ds=0000 es=11b9 cs=11ca ip=001c
```

5. EVEN &ODD IN A GIVEN SERIES

5.1 AIM: To write an assembly language program to find even &odd in a given series.

5.2 APPARATUS:

1. PC with windows OS
2. MASM Tool

5.3 PROGRAM:

```
ASSUME CS: CODE, DS: DATA
 DATA SEGMENT
 LIST DB 12H, 11H, 13H, 20H, 25H, 18H
 COUNT EQU 06H
  DATA ENDS
 CODE SEGMENT
 START:  MOV AX, DATA
           MOV DS, AX
           XOR BX, BX
           XOR DX, DX
           MOV CL, COUNT
           MOV SI, OFFSET LIST
AGAIN:  MOV AX, [SI]
           ROR AX, 01H
           JC NODD
            INC BX
            JMP NEXT
NODD:     INC DX
 NEXT:    ADD SI, 02
           DEC CL
            JNZ AGAIN
            INT 03
            CODE ENDS
            END START
```

5.4 Output:

```
 -g               ax=4c03  bx=0003  cx=0003  dx=0003  sp=0000  bp=0000  si=0006  di=0000
00     ds=11c9 es=11b9 ss=11c9 cs=11ca ip=001c
```

6. PARALLEL COMMUNICATION BETWEEN TWO 8086 MICROPROCESSOR KITS

6.1 AIM: To write an assembly language program to develop a Parallel Communication between two 8086 Microprocessor kits using PPI(8255).

6.2 APPARATUS:

1. PC with windows OS
2. MASM Tool

6.3 PROGRAM:

```
KIT 1: MOV DX,0FFE6
        MOV AL,80H
        OUT DX,AL
       MOV DX,0FFE0
       MOV AL,55H
       OUT DX,AL
        INT 03
KIT 2: MOV DX,0FFE6
       MOV AL,90H
       OUT DX,AL
       MOV DX,0FFE0
       IN  AL, DX
        INT 03
```

6.4 Output:

```
        Kit 1:                    Kit 2:
      G  4000                       G  4000
       AX=0055                       AX=0955
```

7. INTERFACING LED'S AND SWITCHES WITH 8051 KIT

7.1 AIM: To write an assembly language program for LEDs and switches interfacing with 8051. Assume that to be connected over connector J7 of the 8051 trainer kit.

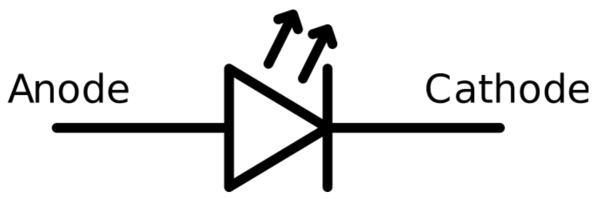
7.2 APPARATUS:

4. 8051 Trainer kit
5. Power supply
6. Key board

7.3 THEORY:

A **light-emitting diode** (**LED**) is a two-lead semiconductor light source. It is a p–n junction diode that emits light when activated.[5] When suitable currents applied to the leads, electrons are able to recombine with electron holes within the device, releasing energy in the form of photons. This effect is called electroluminescence, and the colour of the light (corresponding to the energy of the photon) is determined by the energy band gap of the semiconductor. LEDs are typically small (less than 1 mm^2) and integrated optical components may be used to shape the radiation pattern.

Electronic symbol



Appearing as practical electronic components in 1962, the earliest LEDs emitted low-intensity infrared light. Infrared LEDs are still frequently used as transmitting elements in remote-control circuits, such as those in remote controls for a wide variety of consumer electronics. The first visible-light LEDs were of low intensity and limited to red. Modern LEDs are available across the visible, ultraviolet, and infrared wavelengths, with very high brightness.

LEDs have many advantages over incandescent light sources, including lower energy consumption, longer lifetime, improved physical robustness, smaller size, and faster switching. Light-emitting diodes are used in applications as diverse as aviation lighting, automotive headlamps, advertising, general lighting, traffic signals, camera flashes, lighted wallpaper and medical devices.

Main LED materials: The main semiconductor materials used to manufacture LEDs are:

- **Indium gallium nitride (InGaN):** blue, green and ultraviolet high-brightness LEDs
- **Aluminum gallium indium phosphate (AlGaInP):** yellow, orange and red high brightness LEDs
- **Aluminum gallium arsenide (AlGaAs):** red and infrared LEDs
- **Gallium phosphate (GaP):** yellow and green LEDs

Benefits of LEDs compared with incandescent and fluorescent illuminating devices, include:

- **Low power requirement**: Most types can be operated with battery power supplies.
- **High efficiency:** Most of the power supplied to an LED or IRED is converted into radiation in the desired form, with minimal heat production.
- **Long life:** When properly installed, an LED or IRED can function for decades.

Typical applications include:

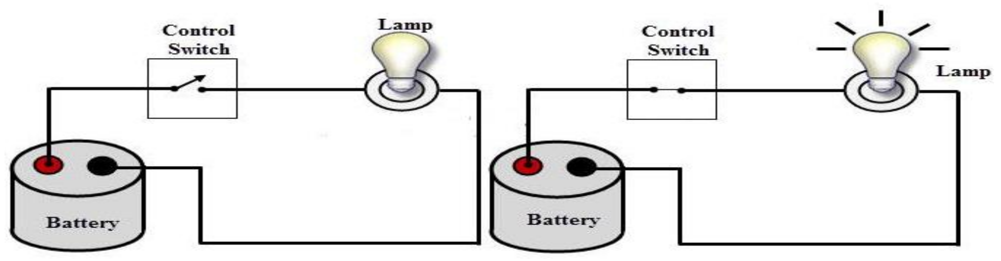
- **Indicator lights:** These can be two-state (i.e., on/off), bar-graph, or alphabetic-numeric readouts.
- **LCD panel backlighting:** Specialized white LEDs are used in flat-panel computer displays.
- **Fiber optic data transmission:** Ease of modulation allows wide communications bandwidth with minimal noise, resulting in high speed and accuracy.
- **Remote control:** Most home-entertainment "remotes" use IREDs to transmit data to the main unit.

SWITCH:

A switch is a device which is designed to interrupt the current flow in a circuit, in other words, it can make or break an electrical circuit. Every electrical and electronics application uses at least one switch to perform ON and OFF operation of the device.

So the switches are the part of a control system and without it, control operation cannot be achieved. A switch can perform two functions, namely fully ON (by closing its contacts) or fully OFF (by opening its contacts).

When the contacts of a switch are closed, the switch creates the closed path for current flow and hence load consumes the power from source. When the contacts of a switch are open, no power will be consumed by the load as shown in below figure.

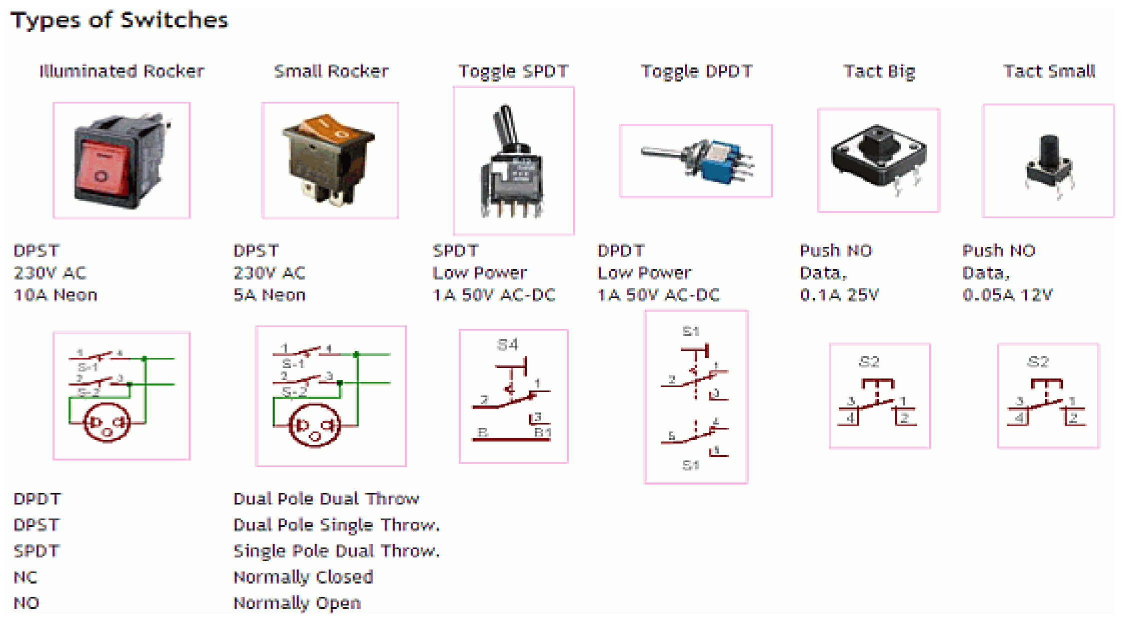


There are numerous switch applications found in wide variety fields such as home, automobiles, industrial, military, aerospace and so on. In some applications multi way switching is employed (like building wiring), in such cases two or more switches are interconnected to control an electrical load from more than one location.

Switches can be of mechanical or electronic type:

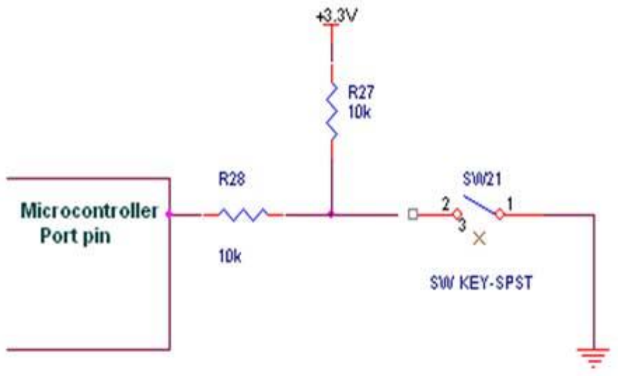
Mechanical switches must be activated physically, by moving, pressing, releasing, or touching its contacts.

Electronic switches do not require any physical contact in order to control a circuit. These are activated by semiconductor action.



Interfacing Switch

Figure shows how to interface the **switch to microcontroller**. A simple switch has an open state and closed state. However, a **microcontroller** needs to see a definite high or low voltage level at a digital input. A **switch** requires a pull-up or pull-down resistor to produce a definite high or low voltage when it is open or closed. A resistor placed between a digital input and the supply voltage is called a "pull-up" resistor because it normally pulls the pin's voltage up to the supply.



We now want to control the **LED** by using switches in **8051 kit**. It works by turning ON a **LED** & then turning it OFF when switch is going to LOW or HIGH.

7.4 PROGRAM:

7.4.1 PROGRAM FOR BLINKING LED'S:

ADDRESS	MACHINE CODE	LABEL	MNEMONIC		COMMENTS
			Opcode	**Operands**	
8000			MOV	0A0,#0E8	; Initialize all 8255
			MOV	R0,#03H	; Ports as output
			MOV	A,#80H	
			MOVX	@R0,A	
			MOV	R0,#00	
			MOV	A, #77	; I/p data from switch
		REPT:	MOVX	@R0, A	
			ACALL	DELAY	; Introduce delay
			RL	A	;
			SJMP	REPT	;
		DELAY:	MOV	R5, #0FFH	; Delay subroutine
		L1:	MOV	R4, #0F0H	
		L2:	DJNZ	R4,L2	
			DJNZ	R5,L1	
			RET		

7.4.2 PROGRAM TO INTERFACE SWITCHES AND LED'S:

ADDRESS	MACHINE CODE	LABEL	MNEMONIC		COMMENTS
			Opcode	**Operands**	
8000			MOV	0A0,#0E8	; Initialize all 8255
			MOV	R0,#03H	; Ports as output
			MOV	A,#82H	
			MOVX	@R0,A	
		REPT:	MOV	R0, #01H	
			ACALL	DELAY	; Introduce delay
			MOVX	A, @R0	; O/p data to ports
			MOV	R0, #00H	
			MOVX	@R0, A	
			ACALL	DELAY	; Introduce delay
			SJMP	REPT	;
		DELAY:	MOV	R5, #0FFH	; Delay subroutine
		L1:	MOV	R4, #99H	
		L2:	DJNZ	R4,L2	
			DJNZ	R5,L1	
			RET		

7.5 RESULT: we have interfaced LEDs and switches with 8051. Verified the outputs LEDs with on –off switches.

7.6 Viva Questions:

1. Explain the register IE format of 8051.
2. Explain the register IP format of 8051.
3. State the use of T0 pin of 8051?
4. Give the functions of each bit in TMOD register.
5. Explain the special function of port – 3 of 8051.
6. What is the function of PCON register?
7. Which register holds the serial data interrupt bits TI and RI.
8. What is the address of the stack when the 8051 is reset?
9. List the all Boolean instructions.
10. List the conditional control transfer instructions.

8. INTERFACING 7-SEGMENT DISPLAY WITH 8051 KIT

8.1 AIM: To write an assembly language program for interfacing 7-segment display with 8051 and display numbers 0 to 9 on 7-segment display. Assume that to be connected over connector J7 of the 8051 trainer kit.

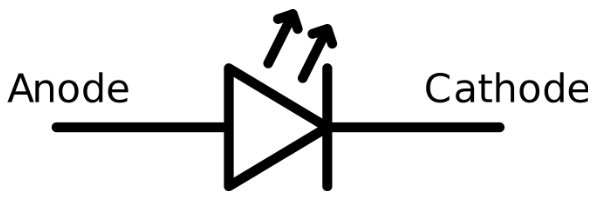
8.2 APPARATUS:

1. 8051 Trainer kit
2. Power supply
3. Key board

8.3 THEORY:

A **light-emitting diode** (**LED**) is a two-lead semiconductor light source. It is a p–n junction diode that emits light when activated. When suitable currents applied to the leads, electrons are able to recombine with electron holes within the device, releasing energy in the form of photons. This effect is called electroluminescence, and the colour of the light (corresponding to the energy of the photon) is determined by the energy band gap of the semiconductor. LEDs are typically small (less than 1 mm^2) and integrated optical components may be used to shape the radiation pattern.

Electronic symbol

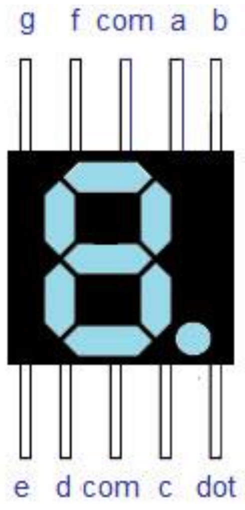


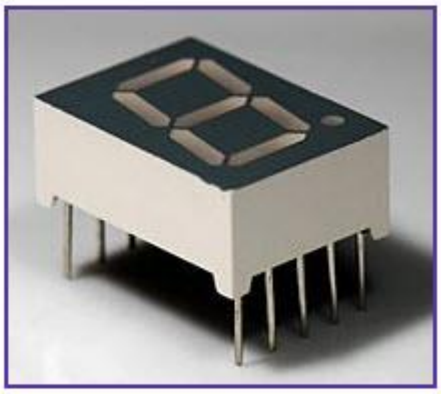
Appearing as practical electronic components in 1962, the earliest LEDs emitted low-intensity infrared light. Infrared LEDs are still frequently used as transmitting elements in remote-control circuits, such as those in remote controls for a wide variety of consumer electronics. The first visible-light LEDs were of low intensity and limited to red. Modern LEDs are available across the visible, ultraviolet, and infrared wavelengths, with very high brightness.

SEVEN SEGMENT DISPLAY

An LED or Light Emitting Diode is a solid state optical pn-junction diode which emits light energy in the form of photons.

The seven segment display is the most common display device used in many gadgets, and electronic appliances like digital meters, digital clocks, microwave oven and electric stove, etc. These displays consist of seven segments of light emitting diodes (LEDs) and that is assembled into a structure like numeral 8. Actually seven segment displays contain about 8-segments wherein an extra 8th segment is used to display dot. This segment is useful while displaying non integer number. Seven segments are indicated as A-G and the eighth segment is indicated as H. These segments are arranged in the form of 8 which is shown in the seven segment display circuit diagram.





7 segment Display Pin Diagram

A seven segment displays are generally available in ten pin package. In that 8 pins relate to the 8 LEDs, the remaining pins at middle are internally shorted. These segments come in two outlines they are common cathode and common anode. In common cathode configuration, the negative terminals are connected to the common pins and the common is connected to the ground. When the corresponding pin is given high, then particular LED glows. In a common anode arrangement, the common pin is given to logic high and the pins of the LED are given low to display a number.

Seven Segment Display Working

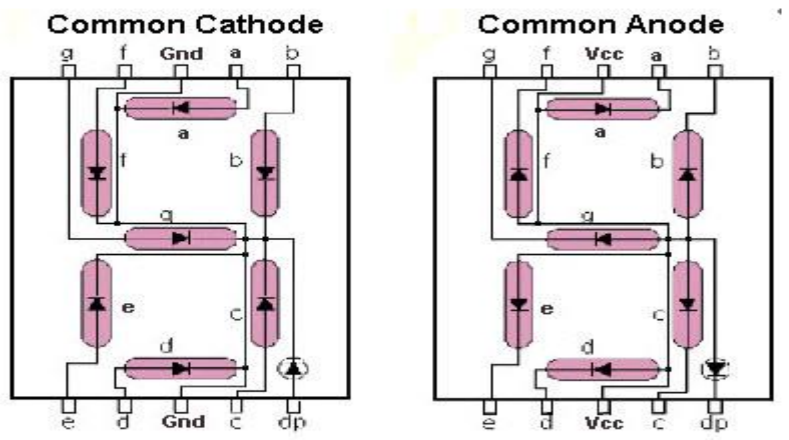
When the power is given to all the segments, then the number 8 will be displayed. If you disconnect the power for segment G (that means 7) then that will result number 0. The circuit of the seven segment display is designed in such a way that the voltage at different pins can be applied at the same time. In the same way, you can form the combinations to display numerals from 0 to 9. Practically, seven segment displays are available with two structures; both the type of displays consists of 10 pins.

The numeric seven segment displays can also display other characters. But generally A-G and L, T, O, S and others are also available. Some problems may occur with the H, X, 2, and Z. Anyways the common seven segment display is numeric only. Alphanumeric displays are also available but cost is little more. These types of displays still have a real purpose due to its high illumination and 7 segment displays are used in dark areas like railway stations. Even 7 segment display based countdown display is used in NASA, which can be read easily even in sunlight.

Types of 7-Segment Displays

There are two types of seven segment displays available in the market. According to the type of application, these displays can be used. The two configurations of seven segment displays are discussed below.

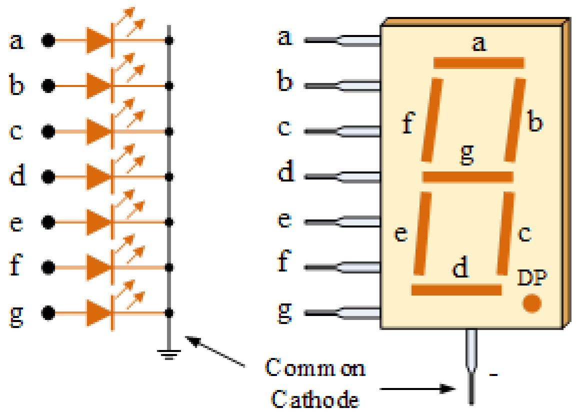
- Common Anode Display
- Common Cathode Display



7- Segment Display Configuration

Common Cathode 7-segment Display

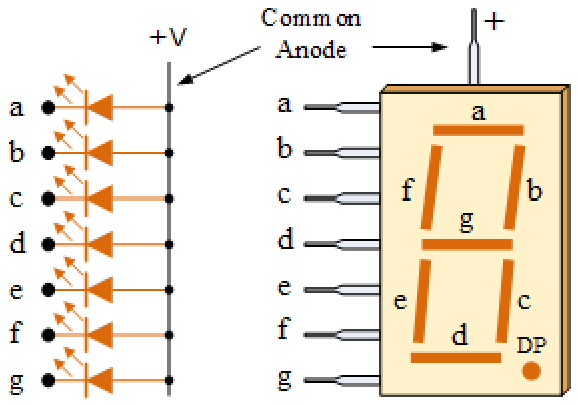
In this type of display, all the cathode connections of the LED segments are connected together to logic 0 or ground. The separate segments are lightened by applying the logic 1 or HIGH signal through a current limiting resistor to forward bias the individual anode terminals a to g.



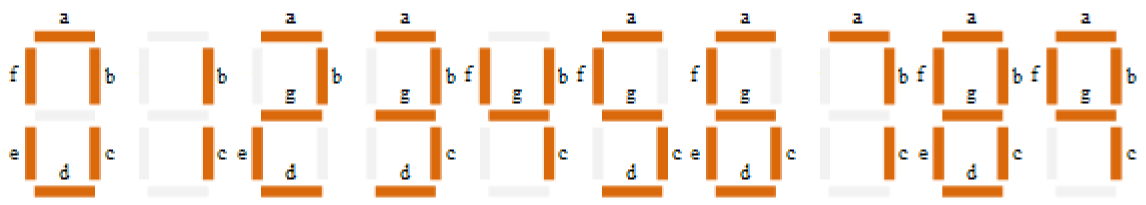
Common Cathode 7-segment Display

Common Anode 7-segment Display

In this type of display, all the anode connections of the LED segments are connected together to logic 1. The separate segments are lightened by applying of the logic 0 or LOW signal through a current limiting resistor to the cathode of the particular segment a to g.



7-Segment Display Segments for all Numbers.



Then for a 7-segment display, we can produce a truth table giving the individual segments that need to be illuminated in order to produce the required decimal digit from 0 through 9 as shown below.

Driving a 7-segment Display

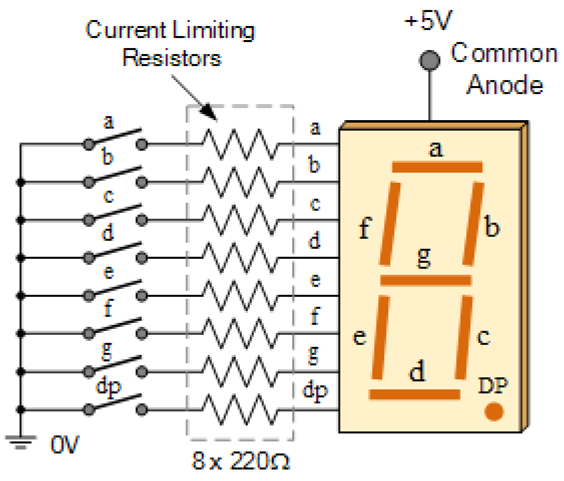
Although a 7-segment display can be thought of as a single display, it is still seven individual LEDs within a single package and as such these LEDs need protection from over current.

LEDs produce light only when it is forward biased with the amount of light emitted being proportional to the forward current. This means that a LEDs light intensity increases in an approximately linear manner with an increasing current. So this forward current must be controlled and limited to a safe value by an external resistor to prevent damage to the LED segments.

The forward voltage drop across a red LED segment is very low at about 2-to-2.2 volts, (blue and white LEDs can be as high as 3.6 volts) so to illuminate correctly, the LED segments should be connected to a voltage source in excess of this forward voltage value with a series resistance used to limit the forward current to a desirable value.

Typically for a standard red coloured 7-segment display, each LED segment can draw about 15 mA to illuminated correctly, so on a 5 volt digital logic circuit, the value of the current limiting resistor would be about 200Ω (5v – 2v)/15mA, or 220Ω to the nearest higher preferred value.

So to understand how the segments of the display are connected to a 220Ω current limiting resistors consider the circuit below.



8.4 PROGRAM:

8.4.1 PROGRAM TO DISPLAY NUMBERS FROM 0 –to- 9:

ADDRESS	MACHINE CODE	LABEL	MNEMONIC		COMMENTS
			Opcode	**Operands**	
8000			MOV	0A0,#0E8	; Initialize all 8255
8003			MOV	R0,#03H	; Ports as output
8005			MOV	A,#80H	

8007			MOVX	@R0,A	
			MOV	R0, #00H	
			MOV	DPTR, #9000	;Starting location of data
			MOV	R1, #0AH	;Count value in R1
		REPT:	MOVX	A,@DPTR	;Get data from memory
			MOVX	@R0, A	; O/p data to ports
			INC	DPTR	;Next data location
			ACALL	DELAY	; Introduce delay
			DJNZ	R1, REPT	
			SJMP	REPT	;
		DELAY:	MOV	R5, #0FFH	; delay subroutine
		L1:	MOV	R4, #0F0H	
		L2:	DJNZ	R4,L2	
			DJNZ	R5,L1	
			RET		

INPUTS:

9000=0C0

9001=0F9

9002=0A4

9003=0B0

9004=99

9005=92

9006=82

9007=0F8

9008=80

9009=90

900A=88

8.5 RESULT: we have interfaced 7-segment display with 8051 and displayed the numbers 0 to 9 on 7-segment display.

8.6 Viva Questions:

1. Define timer operation.

2. Define counter operation.
3. Mention the operating modes of timer/counter in 8051?
4. What is RS 232C?
5. Why are drivers bused in between RS232 and microcontroller?
6. Explain the function of each bit of SCON register.
7. Explain the function of each bit of PCON register.
8. How will you double the baud rate in 8051?
9. What is mode-0 operation in serial communication ports
10. What is mode 3 operation in serial communication ports?