Lab Code:20ECL403

Microprocessor and Microcontroller Lab Manual

Department of Electronics & Communication Engineering

Bapatla Engineering College :: Bapatla

(Autonomous)

G.B.C. Road, Mahatmajipuram, Bapatla-522102, Guntur (Dist.)

Andhra Pradesh, India.

E-Mail:bec.principal@becbapatla.ac.in

Web:www.becbapatla.ac.in

Contents

Bapatla Engineering College :: Bapatla
(Autonomous)

Vision

- To build centers of excellence, impart high quality education and instill high standards of ethics and professionalism through strategic efforts of our dedicated staff, which allows the college to effectively adapt to the ever changing aspects of education.

- To empower the faculty and students with the knowledge, skills and innovative thinking to facilitate discovery in numerous existing and yet to be discovered fields of engineering, technology and interdisciplinary endeavors.

Mission

- Our Mission is to impart the quality education at par with global standards to the students from all over India and in particular those from the local and rural areas.

- We continuously try to maintain high standards so as to make them technologically competent and ethically strong individuals who shall be able to improve the quality of life and economy of our country.

Bapatla Engineering College :: Bapatla

(Autonomous)

Department of Electronics and Communication Engineering

Vision

To produce globally competitive and socially responsible Electronics and Communication Engineering graduates to cater the ever changing needs of the society.

Mission

- To provide quality education in the domain of Electronics and Communication Engineering with advanced pedagogical methods.

- To provide self learning capabilities to enhance employability and entrepreneurial skills and to inculcate human values and ethics to make learners sensitive towards societal issues.

- To excel in the research and development activities related to Electronics and Communication Engineering.

Bapatla Engineering College :: Bapatla

(Autonomous)

Department of Electronics and Communication Engineering

Program Educational Objectives (PEO's)

PEO-I: Equip Graduates with a robust foundation in mathematics, science and Engineering Principles, enabling them to excel in research and higher education in Electronics and Communication Engineering and related fields.

PEO-II: Impart analytic and thinking skills in students to develop initiatives and innovative ideas for Start-ups, Industry and societal requirements.

PEO-III: Instill interpersonal skills, teamwork ability, communication skills, leadership, and a sense of social, ethical, and legal duties in order to promote lifelong learning and Professional growth of the students.

Program Outcomes (PO's)

Engineering Graduates will be able to:

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

PO2. 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.

PO3. 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.

PO4. 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.

PO5. 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.

PO6. 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.

PO7.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.

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

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

PO10. 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.

PO11. 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.

PO12. 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.

Bapatla Engineering College :: Bapatla

(Autonomous)

Department of Electronics and Communication Engineering

Program Specific Outcomes (PSO's)

PSO1: Develop and implement modern Electronic Technologies using analytical methods to meet current as well as future industrial and societal needs.

PSO2: Analyze and develop VLSI, IoT and Embedded Systems for desired specifications to solve real world complex problems.

PSO3: Apply machine learning and deep learning techniques in communication and signal processing.

BAPATLA ENGINEERING COLLEGE:: BAPATLA

(Autonomous)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

MICROPROCESSOR AND MICROCONTROLLER LAB

II B.Tech. IV Semester (Code: 20ECL403)

Lectures	:	0 Hours/Week	Tutorial	:	0 Hours/Week	Practical	:	3 Hours/Week
CIE Marks	:	30	SEE Marks	:	70	Credits	:	1.5

Pre-Requisite: None.

Course Objectives: Students will be able to	
➢	Understand the basic programming of 8086 Microprocessor.
➢	Interface the 8086 microprocessor with various peripherals for different applications.
➢	Understand the basic programming of 8051 microcontroller.
➢	Interface the 8051 microcontroller with various peripherals for different applications

Course Outcomes: At the end of the course, student will be able to

CO1	Demonstrate skill on usage of modern tools such as TASM for 8086 microprocessor and KEIL for 8051 microcontroller.
CO2	Develop assembly language programs for various applications using 8086 Microprocessor.
CO3	Develop assembly language programs for various applications using 8051 microconstroller
CO4	Analyze the interfacing of Programmable peripheral devices with 8051 Micro controller.

Mapping of Course Outcomes with Program Outcomes & Program Specific Outcomes

	PO's												**PSO's**		
CO	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**1**	**2**	**3**
CO1	2	3		2	3				3					3	
CO2	2	3		2	3				3					3	
CO3	2	3		2	3				3					3	
CO4	2	3		2	3				3				2	3	2
AVG	2	3		2	3				3				2	3	2

LIST OF EXPERIMENTS

36 Hours

Experiments Based on ALP (8086)

1. Programs on Data Transfer Instructions.
2. Programs on Arithmetic and Logical Instructions.
3. Programs on Branch Instructions.
4. Programs on Subroutines.
5. Sorting of an Array.
6. Programs on Interrupts (Software and Hardware).

7. 8086 Programs using DOS and BIOS Interrupts.
8. Programs on 80386, 80486
9. ARM processor

Experiments Based on Interfacing & Microcontroller (8051)

10. DAC Interface-Waveform generations.
11. Stepper Motor Control.
12. Keyboard Interface / LCD Interface.
13. Data Transfer between two PCs using RS.232 C Serial Port
14. Programs on Data Transfer Instructions using 8051 Microcontroller.
15. Programs on Arithmetic and Logical Instructions using 8051 Microcontroller.
16. Applications with Microcontroller 8051

NOTE: *A minimum of 10(Ten) experiments, choosing 5 (Five) from each part, have to be Performed and recorded by the candidate to attain eligibility for Semester End Examination.*

1. INTRODUCTION TO MDS IDE (MASM/TASM)

AIM: To Familiarize Using Microprocessor Development System Integrated Development Environment – TASM for microprocessor/microcontroller based system design.

Apparatus: 1. Host PC Installed with TASM with DOSBOX supporting MDS IDE Tool Chain

Theory: MDS IDE is a package installed on Host computer to simulate and run 8086 assembly level programs.

This Tool chain comprises of the following tools:

1. Editor
2. Assembler
3. Linker/ Locator
4. Loader (Up & down)

Editor allows to type assembly source file and allows to do modifications and save them for future reference.

Assembler translates source code (.ASM file) to object code (.exe)

Linker allows to link several source files and does address resolution to generate executable file for downloading on to the target.

Loader allows downloading (.exe) file onto the target using RS232C cable from host computer and uploads results on to host for verification.

Procedure:

Using Intel 8086:

Intel 8086 can be used in standalone mode using resident key board for typing in manually assembled Op-codes.
Key words used while working with X8086 based IDE:
1. Edit: opens default editor to type code and file can be saved with .ASM extension
2. TASM FILE.ASM allows assembling .ASM file
3. TLINK FILE.OBJ links several files together and carries out address resolution task.
4. TD FILE.EXE : makes the ground ready to download code for target & to download .EXE file.

RESULT: Explored MDS IDE features to develop Assembly Language Program for Intel 8086/8051.

CYCLE – I

8086 PROGRAMMING

2. ADDITION OF TWO UNSIGNED BYTES

AIM: To write an ALP to add two unsigned words residing in memory locations starting from 6000h and store back the result in next memory locations

PROGRAM :

```
        ASSUME CS:CODE
CODE    SEGMENT
        ORG 3000H
        MOV SI, 6000H
        MOV AX, [SI]
        ADD SI, 02H
        MOV BX, [SI]
        MOV CX, 00H          ; clear CX register for holding carry
        ADD AX, BX
        JNC DOWN
        INC CX
DOWN    MOV 02H[SI], AX
        MOV 04H[SI], CX
        INT 03H
CODE    ENDS
        END
```

OBSERVATIONS AFTER EXECUTION OF CODE:

DATA1:	DATA2:
6000: 0005H	6000: 80F1H
6002: 0008H	6002: FFF0H
RESULT:	RESULT:
6004: 0000H	6004: 80E1H
6006: 0000H	6006: 0001H

3. SUBTRACTION OF TWO UNSIGNED BYTES

AIM: To write an ALP to perform unsigned subtraction on two 16 bit numbers stored in successive memory locations starting from 5000h and store back the difference and borrow in next memory locations

PROGRAM :

```
                OUTPUT 2500AD
                SYMBOLS
                ASSUME CS:CODE
CODE            SEGMENT
                ORG 3000H
                MOV SI, 5000H
                MOV AX, [SI]
                ADD SI, 02H
                MOV BX, [SI]
                MOV CX, 00H
                SUB AX, BX
                JNC DOWN
                DEC CX
DOWN            ADD SI, 02H
                MOV [SI], AX
                MOV 02H[SI], CX
                INT 03H
CODE            ENDS
                END
```

OBSERVATIONS AFTER EXECUTION OF CODE:

DATA1:	DATA2:
5000: 9999H	5000: 2043H
5002: 5875H	5002: 2049H
RESULT:	RESULT:
5004: 4124H	5004: FFFAH
5006: 0000H	5006: FFFFH

4. MULITIPLICATION OF TWO UNSIGNED BYTES

AIM: To write an ALP to multiply two 16 bit unsigned numbers stored in successive memory locations and store back the product in next memory locations.

PROGRAM:

```
            OUTPUT 2500AD
            SYMBOLS
            ASSUME CS:CODE
CODE        SEGMENT
            ORG 5000H
            MOV SI, 7000H
            MOV AX, [SI]
            ADD SI, 02H
            MOV BX, [SI]
            MUL BX
            ADD SI, 02H
            MOV [SI], AX
            MOV 02H[SI], DX
            INT 03H
CODE        ENDS
            END
```

OBSERVATIONS AFTER EXECUTION OF CODE:

DATA1:
7000: 0003H
7002: 0002H
RESULT:
7004: 0006H
7006: 0000H

DATA2:
7000: 9999H
7002: 5875H
RESULT:
7004: CAEDH
7006: 3512H

5. DIVISION (32 Bit by 16 bit unsigned)

AIM: To write an ALP to divide 32 bit number by 16 bit number in the memory locations and store back the quotient and remainder back in the next memory locations.

PROGRAM:

```
                OUTPUT 2500AD
                SYMBOLS
                ASSUME CS:CODE
CODE            SEGMENT
                ORG 5000H
                MOV SI, 7000H
                MOV AX, [SI]
                ADD SI, 02H
                MOV DX, [SI]
                ADD SI, 02H
                MOV BX, [SI]
                DIV BX
                ADD SI, 02H
                MOV [SI], AX
                MOV 02H[SI], DX
                INT 03H
CODE            ENDS
                END
```

OBSERVATIONS AFTER EXECUTION OF CODE:

DATA1:	DATA2:
7000: 4361H	7000: 4360H
7002: 0010H	7002: 0000H
7004:0016H	7004:0016H
RESULT:	RESULT:
7006: BD3EH	7004: 0310H
7008: 000DH	7006: 0000H

6. SIGNED ADDITION (16 BIT)

AIM: To write an ALP to add two signed 16 bit numbers stored in memory locations and store back the result in next memory location.

PROGRAM:

```
            OUTPUT 2500AD
            SYMBOLS
            ASSUME CS:CODE
CODE        SEGMENT
            ORG 3000H
            MOV SI, 6000H
            MOV AX,-0003H
            CWD
            MOV BX,0002H
            ADD AX, BX
            MOV [SI], AX
            MOV 02H[SI], CX
            INT 03H
CODE        ENDS
            END
```

OBSERVATIONS AFTER EXECUTION OF CODE:

DATA1:	DATA2:
6000: -0003H	6000: -0AC3H
6002: 0002H	6002: 0D02H
RESULT:	RESULT:
6004: FFFFH	6004: 023FH
6006: FFFFH	6006: FFFFH

7. SIGNED SUBTRACTION (16 BIT)

AIM: To write an ALP to subtract two signed 16 bit numbers stored in memory locations and store back the result in next memory location.

PROGRAM:

```
            OUTPUT 2500AD
            SYMBOLS
            ASSUME CS:CODE
CODE        SEGMENT
            ORG 3000H
            MOV SI, 6000H
            MOV AX,-0009H
            CWD
            MOV BX,0001H
            SUB AX, BX
            MOV [SI], AX
            MOV 02H[SI], CX
            INT 03H
CODE        ENDS
            END
```

OBSERVATIONS AFTER EXECUTION OF CODE:

DATA1:	DATA2:
6000: -0009H	6000: -0AC3H
6002: 0001H	6002: 0D02H
RESULT:	RESULT:
6004: FFF6H	6004: E83BH
6006: FFFFH	6006: FFFFH

8. SIGNED MULTIPLICATION

AIM: To write an ALP to multiply two signed 16 bit numbers stored in memory locations and store back the result in next memory location.

PROGRAM:

```
        OUTPUT 2500AD
        SYMBOLS
        ASSUME CS:CODE
CODE    SEGMENT
        ORG 3000H
        MOV SI, 6000H
        MOV AL,-0003H
        CBW
        MOV BX,0004H
        IMUL BX
        MOV [SI], AX
        MOV 02H[SI], DX
        INT 03H
CODE    ENDS
        END
```

OBSERVATIONS AFTER EXECUTION OF CODE:

DATA1:	DATA2:
6000: -0003H	6000: -0003H
6002: 0002H	6002: 0D01H
RESULT:	RESULT:
6004: FFF4H	6004: FDBDH
6006: FFFFH	6006: FFFFH

9. SIGNED DIVISION

AIM: To write an ALP to divide two signed 16 bit number by 8 bit number stored in memory locations and store back the result in next memory location.

PROGRAM:

```
                OUTPUT 2500AD
                SYMBOLS
                ASSUME CS:CODE
CODE            SEGMENT
                ORG 3000H
                MOV SI, 6000H
                MOV AX,-0016H
                CWD
                MOV BX,0004H
                IDIV BX
                MOV [SI], AX
                MOV 02H[SI], DX
                INT 03H
CODE            ENDS
                END
```

OBSERVATIONS AFTER EXECUTION OF CODE:

DATA1:
6000: -0016H
6002: 04H
RESULT:
6004: FFFBH
6006: FFFFH

DATA2:
6000: -0FE2H
6002: 04H
RESULT:
6004: FC08H
6006: FFFFH

10. SUM OF 'N' 16-BIT NUMBERS

AIM: To write an ALP to find sum of 'N' 16 bit numbers stored in memory locations and store back the result in successive memory locations.

PROGRAM:

```
        OUTPUT 2500AD
        SYMBOLS
        ASSUME CS:CODE
CODE    SEGMENT
        ORG 3000H
        MOV SI, 4000H
        MOV CX,[SI]
        MOV DX,0000H
        ADD SI, 02H
        MOV AX,[SI]
        DEC CX
UP      ADD [SI],02H
        MOV BX,[SI]
        ADD AX, BX
        JNC DOWN
        INC DX
DOWN    DEC CX
        JNZ UP
        ADD SI,02H
        MOV [SI], AX
        MOV 02H[SI], DX
        INT 03H
CODE    ENDS
        END
```

OBSERVATIONS AFTER EXECUTION OF CODE:

DATA1:	DATA2:
4000: 0003H	4000: 0004H
4002: 0123H	4002: 0123H
4004: 6363H	4004: 6363H
4006: 7525H	4006: 7525H
	4008: A575H

RESULT:

4008: D9ABH

400A: 0000H

RESULT:

400A: 7F20H

400B: 0001H

11. AVERAGE OF 'N' 16-BIT NUMBERS

AIM: To write an ALP to find average of 'N' 16 bit numbers stored in memory locations and store back the result in successive memory locations.

PROGRAM :

```
                OUTPUT 2500AD
                SYMBOLS
                ASSUME CS:CODE
CODE            SEGMENT
                ORG 3000H
                MOV SI, 4000H
                MOV CX,[SI]
                MOV DI,CX
                MOV DX,0000H
                ADD SI, 02H
                MOV AX,[SI]
                DEC CX
UP              ADD [SI],02H
                MOV BX,[SI]
                ADD AX, BX
                JNC DOWN
                INC DX
DOWN            LOOP UP
                MOV CX, DI
                DIV AX,CX
                MOV 02H[SI], AX
                MOV 04H[SI], DX
                INT 03H
CODE            ENDS
                END
```

OBSERVATIONS AFTER EXECUTION OF CODE:

DATA1:	DATA2:
4000: 0004H	4000: 0003H
4002: FFFFH	4002: 0123H
4004: FFFFH	4004: 4568H
4006: FFFFH	4006: ABCDH
4008: FFFFH	
RESULT:	RESULT:
400A: FFFFH	4008: 5678H
400B: 0000H	400A: 0001H

12. ADDITION OF TWO 32-BIT NUMBERS

AIM: To write an ALP to find addition of two 32 bit numbers stored in memory locations and store back the result in successive memory locations.

PROGRAM:

```
                OUTPUT 2500AD
                SYMBOLS
                ASSUME CS:CODE
CODE            SEGMENT
                ORG 5000H
                MOV DI, 00H
                MOV SI, 3000H
                MOV AX, [SI]
                ADD SI, 02H
                MOV BX, [SI]
                ADD SI, 02H
                MOV CX, [SI]
                ADD SI, 02H
                MOV DX, [SI]
                ADD AX, CX
                ADC BX, DX
                JNC DOWN
                INC DI
DOWN            ADD SI, 02H
                MOV [SI], AX
                MOV 02H[SI], BX
                MOV 04H[SI], DI
                INT 03H
CODE            ENDS
                END
```

OBSERVATIONS AFTER EXECUTION OF CODE:

DATA1:	DATA2:
3000: CDABH	3000: 5678H
3002: 4910H	3002: A234H
3004: CDFFH	3004: 3210H
3006: 3122H	3006: A754H
RESULT:	RESULT:
3008: 9BAAH	3008: 8888H
3006: 7A33H	300A: 4988H
300C: 0000H	300C: 0001H

13. SMALLEST AND LARGEST OF GIVEN SET OF NUMBERS

AIM: To write an ALP to find smallest / largest of a given set of 16 bit numbers stored in memory locations and store back the result in successive memory locations.

PROGRAM:

```
                OUTPUT 2500AD
                SYMBOLS
                ASSUME CS:CODE
CODE            SEGMENT
                ORG 5000H
                MOV SI, 3000H
                MOV DI, SI
                MOV CX, [SI]
                ADD SI, 02H
                MOV AX, [SI]
                DEC CX
UP              ADD SI, 02H
                MOV BX, [SI]
                CMP AX, BX
                JAE DOWN
                MOV AX, BX
DOWN            LOOP UP
                MOV DX, AX
                MOV 02H[SI], DX
                MOV CX, [DI]
                ADD AX, [DI]
UP1             ADD DI, 02H
                MOV BX, [DI]
                CMP AX, BX
                JC DOWN1
DOWN1           LOOP UP1
                MOV 02H[SI], AX
                INT 03H
CODE            ENDS
                END
```

OBSERVATIONS AFTER EXECUTION OF CODE:

DATA1:	DATA2:
3000: 0003H	3000: 0005H
3002: 0015H	3002: 000AH
3004: 010AH	3004: 111BH
3006: 0001H	3006: 1000H
	3008: 0005H
	300A: 0AAAH
RESULT:	RESULT:
3008: 010AH	3008: 111BH
3006: 0001H	300A: 0005H

14. BUBBLE SORT

AIM: To write an ALP to arrange 16 bit numbers stored in memory locations in ascending and descending order using Bubble sort.

PROGRAM:

```
        OUTPUT 2500AD
        SYMBOLS
        ASSUME CS:CODE
CODE    SEGMENT
        ORG 5000H
        MOV SI, 3000H
        MOV CX, [SI]
        DEC CX
UP      MOV SI, 3002H
        MOV DX, CX
UP1     MOV AX, [SI]
        ADD SI, 02H
        CMP AX, [SI]
        JC DOWN          /* JNC DOWN – FOR DESCENDING ORDER */
        XCHG AX, [SI]
        SUB SI, 02H
        MOV [SI], AX
        ADD SI, 02H
DOWN    DEC DX
        JNZ UP1
        LOOP UP
        INT 03H
CODE    ENDS
        END
```

OBSERVATIONS AFTER EXECUTION OF CODE:

DATA1:	DATA2:
3000: 0004H	3000: 0004H
3002: 000AH	3002: 000AH
3004: 001BH	3004: 001BH
3006: 0000H	3006: 0000H
3008: 0008H	3008: 0008H
RESULT:	RESULT:
3002: 0000H	3002: 001BH
3004: 0008H	3004: 000AH
3006: 000AH	3006: 0008H
3008: 001BH	3008: 0000H

15. PACKED BCD TO ASCII CONVERSION

AIM: To write an ALP to convert Packed BCD to ASCII equivalent

PROGRAM:

```
                OUTPUT 2500AD
                SYMBOLS
                ASSUME CS:CODE
CODE            SEGMENT
                ORG 4000H
                MOV SI, 3000H
                MOV AL, [SI]
                MOV BL, AL
                MOV CL, 04H
                AND AL, 0FH
                OR AL, 30H
                ADD SI, 01H
                MOV [SI], AL
                AND BL, F0H
                ROL BL, CL
                OR BL, 30H
                ADD SI, 01H
                MOV [SI], BL
                INT 03H
CODE            ENDS
                END
```

OBSERVATIONS AFTER EXECUTION OF CODE:

DATA1:	DATA2:
3000: 58H	3000: 12H
RESULT:	RESULT:
3001: 38H	3001: 32H
3002: 35H	3002: 31H

16. ASCII TO PACKED BCD CONVERSION

AIM: To write an ALP to convert ASCII coded byte to packed BCD.

PROGRAM:

```
            OUTPUT 2500AD
            SYMBOLS
            ASSUME CS:CODE
CODE        SEGMENT
            ORG 4000H
            MOV SI, 3000H
            MOV CL, 04H
            MOV AL, [SI]
            INC SI
            MOV BL, [SI]
            SUB AL, 30H
            SUB BL, 30H
            ROL BL, CL
            ADD AL, BL
            INC SI
            MOV [SI], AL
            INT 03H
CODE        ENDS
            END
```

OBSERVATIONS AFTER EXECUTION OF CODE:

DATA1:
3000: 31H
3001: 32H

RESULT:
3002: 21H

DATA2:
3000: 38H
3001: 39H

RESULT:
3002: 98H

17. MOVING BLOCK OF DATA

AIM: To write an ALP to move a block of data from one location to another

PROGRAM:

```
                OUTPUT 2500AD
                SYMBOLS
                ASSUME CS:CODE
CODE            SEGMENT
                ORG 3000H
                MOV SI, 4000H
                MOV DI, 5000H
                MOV CX, [SI]
                ADD SI, 02H
                REP MOVSW
                INT 03H
CODE            ENDS
                END
```

OBSERVATIONS AFTER EXECUTION OF CODE:

DATA1:	DATA2:
4000: 0005H	4000: 0004H
4002: 0001H	4002: FEDCH
4004: 0004H	4004: E304H
4006: 0009H	4006: 5678H
4008: 000AH	4008: 987AH
400A: 0010H	
RESULT:	RESULT:
5000: 0001H	4000: FEDCH
5002: 0004H	4002: E304H
5004: 0009H	4004: 5678H
5006: 000AH	4006: 987AH
5008: 0010H	

18. COUNTING NUMBER OF 1'S IN A GIVEN WORD

AIM: To write an ALP to count number of 1's in a word stored in memory location

PROGRAM:

```
            OUTPUT 2500AD
            SYMBOLS
            ASSUME CS:CODE
CODE        SEGMENT
            ORG 3000H
            MOV SI, 8000H
            MOV AX, [SI]
            MOV BX,00H
            MOV CL, 10H
UP          CLC
            RLC AX, 01H
            JC DOWN1
            LOOP UP
            JMP DOWN2
DOWN1       INC BX
            LOOP UP
DOWN2       ADD SI, 02H
            MOV [SI], BX
            INT 03H
CODE        ENDS
            END
```

OBSERVATIONS AFTER EXECUTION OF CODE:

DATA1:	DATA2:
8000: 1234H	8000: 5123H
RESULT:	RESULT:
8002: 0005H	8002: 0006H

19. PALINDROME (STRING REVERSAL)

AIM: To write an ALP for a string reversal

PROGRAM:

```
                OUTPUT 2500AD
                SYMBOLS
DATA            SEGMENT
STR1            DB ‘LIRIL’, 00H
LEN             EQU ($-STR1)
STR2            DB ‘       ’, 00H
M1              DB ‘PALINROME’, 00H
M2              DB ‘NOT A PALINDROME’, 00H
PRINT           EQU FE00:0013H
DATA            ENDS
CODE            SEGMENT
                ASSUME CS:CODE
                ORG 3000H
                LEA SI, STR1
                LEA DI, STR2
                MOV CX, LEN
                DEC CX
                MOV DX, CX
                ADD DI, CX
                DEC DI
UP              MOV AL, [SI]
                MOV [DI], AL
                INC SI
                DEC DI
                LOOP UP
                MOV CX, DX
                LEA SI, STR1
                LEA DI, STR2
                CLD
                REPE CMPSB
                JNZ DOWN2
                LEA AX, M1
                CALL FAR PRINT
                INT 03H
```

```
DOWN2       LEA AX, M2
            CALL FAR PRINT
            INT 03H
CODE        ENDS
            END
```

OBSERVATIONS AFTER EXECUTION OF CODE:

DATA1:	DATA2:
STR1: ‘LIRIL’	STR1: ‘KITE’
RESULT:	RESULT:
STR2: ‘LIRIL’	STR2: ‘ETIK’
PALINDROME	NOT A PALINDROME

CYCLE – II

8086 INTERFACING

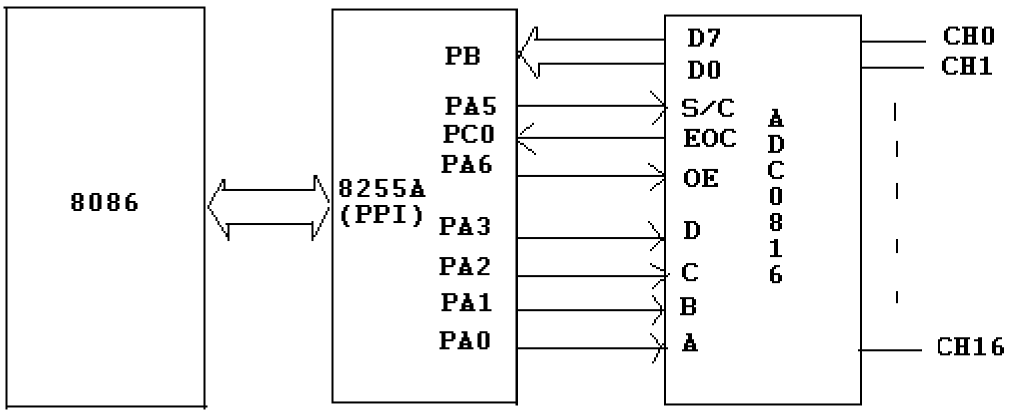
20. ADC (16-Channel) INTERFACE

Introduction:

In many microprocessor based systems analog input Signals have to be converted into digital values. A variety of Analog – to – Digital converters (ADCs) are available for this Purpose. This interface uses ADC 0816 to do analog-to-digital Conversion.

Description of the Circuit:

ADC 0816 is a 16-channel 8-bit A-D converter. One of the 16 channels can be selected by setting the channel select lines to appropriate values. These channel select lines are connected to four lines of port A of 8255A viz. PA3,PA2,PA1,PA0. A 2 MHz crystal-controlled oscillator generates a clock signal which is divided by four (using 74LS74) and is then fed to the clock input of ADC 0816. The data from ADC can be read through port B. Start command can be issued to ADC by asserting PA5. The EOC Signal is read through PC0. The OE signal is sent through PA6.



INTERFACING ADC WITH 8086

```
        OUTPUT 2500AD
        SYMBOLS
CLEAR   EQU FE00:1B0H
PRINT   EQU FE00:1B64H
CODE    SEGMENT
        ASSUME CS:CODE
        ORG 1000H
START   MOV AL,8BH
        MOV DX,FFE7H
        OUT DX,AL
        MOV CL,03H
        MOV CH,01H
        CALL CONVERT
        CALL FAR PRINT
        CALL DELAY
        PUSH AX
        MOV AL,0DH
        CALL FAR CLEAR
        POP AX
        JMP START
CONVERT MOV AL,00H
        OR AL,CL
        MOV DX,FFE1H
        OUT DX,AL
        MOV AL,20H
        OR AL,CL
        OUT DX,AL
        NOP
        NOP
        MOV AL,00H
        OR AL,CL
        OUT DX,AL
        MOV DX,FFE5H
WAIT1   IN AL,DX
        AND AL,01H
        JNZ WAIT1
WAIT2   IN AL,DX
        AND AL,01H
        JZ WAIT2
        MOV AL,40H
        OR AL,CL
        MOV DX,FFE1H
        OUT DX,AL
        NOP
        MOV DX,FFE3H
        IN AL,DX
        PUSH AX
```

```
        MOV AL,00H

        OR AL,CL
        MOV DX,FFE1H
        OUT DX,AL
        POP AX
        RET
DELAY   PUSH CX
        PUSH DX
        MOV DX,0400H
UP2     MOV CX,00FFH
UP      LOOP UP
        DEC DX
        JNZ UP2
        POP DX
        POP CX
        RET
```

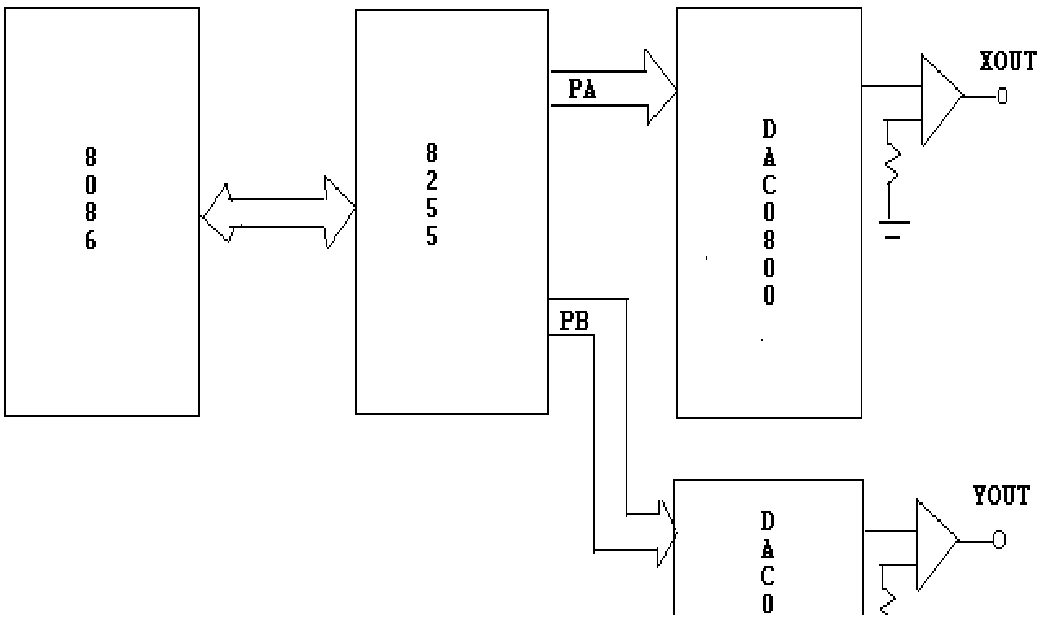
21. DAC INTERFACE

Introduction:

The DAC interface can be used to generate different interesting waveforms using microprocessors. Two eight-bit DACs (DAC 0800) are provided. The digital inputs to these DACs are provided through port A and port B of 8255 used as output ports. The analog outputs of the DACs Can be given to the operational amplifiers which act as current to voltage converters. The outputs from the DACs vary between 0 and 5 V corresponding to values between 00 and FFH. Different waveforms can be observed at the op-amp outputs depending upon the digital input patterns.

Description of the Circuit:

Port A and port B of 8255 PPI are used as output ports. The digital inputs to the DACs are provided through port A and port B of 8255. The analog outputs of the DACs are connected to the inverting inputs of op-amp which acts as current to voltage converter. The output can be taken at points marked Xout and Yout.



INTERFACING DAC WITH 8086

SQUARE WAVE GENERATION

```
        OUTPUT 2500AD
        SYMBOLS
CODE    SEGMENT
        ASSUME CS:CODE
        ORG 1000H
        MOV AL,80H
        MOV DX,FFE7H
        OUT DX,AL
        MOV DX,FFE1H
UP      MOV AL,00H
        OUT DX,AL
        CALL DELAY
        MOV AL,FFH
        OUT DX,AL
        CALL DELAY
        JMP UP
DELAY   MOV CX,00FFH
UP1     LOOP UP1
        RET
CODE    ENDS
        END
```

TRIANGULAR WAVE GENERATION

```
        OUTPUT 2500AD
        SYMBOLS
CODE    SEGMENT
        ASSUME CS:CODE
        ORG 1000H
        MOV AL,80H
        MOV DX,FFE7H
        OUT DX,AL
        MOV DX,FFE1H
UP2     MOV CX,FFH
        MOV AL,00H
UP      OUT DX,AL
        INC AL
        LOOP UP
        MOV CX,FFH
UP1     OUT DX,AL
        DEC AL
        LOOP UP1
        JMP UP2
CODE    ENDS
        END
```

22. KEYBOARD INTERFACE

AIM: To write an ALP to interface a matrix keyboard, read the key pressed and get the display of the key pressed on the monitor.

Introduction:

In many microprocessor based systems, calculator keypad is used as an input device. A calculator keypad can be interfaced to a microprocessor using a dedicated peripheral controller like 8279 keyboard/display controller. In this case, the controller can handle the interface problems like key de bounce, 2 key lock-out and N-key roll-over etc. Further such controllers can directly encode the position of the depressed key. In an alternative approach, the calculator keypad interface is passive and software is used for encoding the key positions and for handling problems like key de bounce, roll-over etc.

The present interface module provides the calculator style calculator keypad consisting of the keys 0 to 9, +, -, x, =, %, C, CE and 2 spare keys. These twenty keys are arranged in a 3X8 matrix. The row lines can be driven through port C (PC2,PC1 & PC0) and status of the column lines can be read through port A.

Circuit Description:

When no key is pressed, all the return lines are low. The row lines are driven high one after another in sequence. When a row is driven high pressing a key in that row causes a corresponding return line to be read as high. Then it can scan for the column for which the key is depressed. The row and column positions can then be used to encode the key. As the scanning of the rows occurs at very high speeds, compared to human reaction times, there is no danger of missing a key depression. The key de bounce can be handled through appropriate software routines.

KEYBOARD INTERFACE WITH 8086

PROGRAM:

```
                OUTPUT 2500AD
PRINT           EQU FE00: 1B64H
CLEAR           EQU FE00: 1B50H
CWR             EQU FEE7H
PTA             EQU FFE1H
PTC             EQU FFE5H
CODE            SEGMENT
                ASSUME CS:CODE
                ORG 1000H
START           MOV DX,CWR
                MOV AL,90H
                OUT DX,AL
L1              MOV DX,PTC
                MOV AL,01H
                OUT DX,AL
                MOV DX,PTA
                MOV CL,FFH
                CALL CHK
                MOV DX,PTC
                MOV AL,02H
                OUT DX,AL
                MOV CL,07H
                CALL CHK
                MOV DX,PTC
                MOV AL,04H
                OUT DX,AL
                MOV DX,PTA
                MOV CL,0FH
                CALL CHK
                JMP L1
                INT 3
CHK             MOV CH,08H
                MOV DX,PTA
                IN AL,DX
L2              INC CL
                SHR AL,01H
                JC PRT
                DEC CH
                JZ RT
                JMP L2
```

```
PRT      DEC CH
         MOV AL,CL
         CALL FAR PRINT
         CALL DELAY
         MOV AL,0DH
         CALL FAR CLEAR
RT       RET
DELAY    PUSH CX
         MOV CX,0FFH
LP       NOP
         NOP
         LOOP LP
         POP CX
         RET
CODE     ENDS
         END
```

23. INTERFACING SEVEN SEGMENT DISPLAY WITH 8086

AIM: To write an ALP to display message "**HELP"** on a 4 digit seven segment LED display.

PROGRAM:

```
                OUTPUT 2500AD
                SYMBOLS
CODE            SEGMENT
PORTC           EQU FFE5H
PORTB           EQU FFE3H
CTLW            EQU FFE7H
VALUE1          BYTE 31H,E3H,60H,90H
VALUE2          BYTE FFH,FFH,FFH,FFH
                ORG 1000H
                MOV DX,CTLW
                MOV AL,80H
                OUT DX,AL
UP6             LEA SI,VALUE1
                LEA DI,VALUE2
                MOV BL,04H
LOOP0           MOV AL,[SI]
                MOV CL,08H
LOOP1           MOV DX,PORTB
                OUT DX,AL
                SHR AL,01H
                CALL CLO
                DEC CL
                JNZ LOOP1
                INC SI
                DEC BL
                JNZ LOOP0
                CALL DELAY
                MOV BL,04H
LOOP3           MOV AL,[DI]
                MOV CL,08H
LOOP2           MOV DX,PORTB
                OUT DX,AL
                SHR AL,01H
                CALL CLO
                DEC CL
                JNZ LOOP2
                INC DI
                DEC BL
                JNZ LOOP3
                CALL DELAY
                JMP UP6
```

```
CLO     PUSH AX
        MOV AL,00H
        MOV DX,FFE5H
        OUT DX,AL
        MOV AL,FFH
        OUT DX,AL
        POP AX
        RET
DELAY   PUSH CX
        PUSH DX
        MOV DX,03H
UP9     MOV CX,1FFFH
UP0     LOOP UP0
        DEC DX

        JNZ UP9
        POP DX
        POP CX
        RET
CODE    ENDS
        END
```

24. INTERFACING STEPPER MOTOR WITH 8086

AIM: To write an ALP to run the given stepper motor for 360 degrees of revolution in

1) Clockwise and
2) Anti clockwise directions

PROGRAM:

Clock wise direction:

```
        OUTPUT 2500AD
        SYMBOLS
CODE    SEGMENT
        ASSUME CS:CODE
        ORG 1000H
        MOV AL,80H
        MOV DX,FFE7H
        OUT DX,AL
        MOV CL,32H
UP2     MOV BL,04
        MOV AL,01
UP1     MOV DX,FFE1H
        OUT DX,AL
        CALL DELAY
        ROL AL,01
        DEC BL
        JNZ UP1
        DEC CL
        JNZ UP2
        INT 3
DELAY   PUSH CX
        MOV CX,0F0FH
UP      NOP
        NOP
        NOP
        LOOP UP
        POP CX
        RET
CODE    ENDS
        END
```

PROGRAM:

Clockwise direction:

```
            OUTPUT 2500AD
            SYMBOLS
CODE        SEGMENT
            ASSUME CS:CODE
            ORG 1000H
            MOV AL,80H
            MOV DX,FFE7H
            OUT DX,AL
            MOV CL,32H
UP2         MOV BL,04
            MOV AL,08
UP1         MOV DX,FFE1H
            OUT DX,AL
            CALL DELAY
            ROR AL,01
            DEC BL
            JNZ UP1
            DEC CL
            JNZ UP2
            INT 3
DELAY       PUSH CX
            MOV CX,0F0FH
UP          NOP
            NOP
            NOP
            LOOP UP
            POP CX
            RET
CODE        ENDS
            END
```

25. LOGIC CONTROLLER

Introduction:

To realize a specific logic programming 8086 interfaced to 8255A. Port B is connected to Inputs and Port A is connected to outputs.

Circuit Description:

The Experimental Set-up contains 8 no. of push button keys interfaced to port A and 8 no. of LEDS connected to port B in Common Cathode configuration. The buttons can be operated to feed inputs for the logic controller and LEDS can be operated to reflect the result of the logic realized. Logic is realized by writing the code accordingly.

Program to realize Logic -OR : Output of an or gate is logic high only when all inputs are at logic high else output is held at logic low.

```
          OUTPUT 2500AD
          SYMBOLS
C:        SEGMENT
          ASSUME CS:C
          ORG 4000H
          MOV AL,8BH          ; COMMAND FOR 8255A MAKING PA: OUTPUT
                              ;PB:INPUT
          MOV DX, FFE7H       ; ADDRESS OF CONTROL REG
          OUT DX,AL
UP:       MOV DX,FFE3H        ; PORT B ADD
          IN AL,DX            ; SCAN KEYS
          CMP AL,00H          ;ESTABLISH BOUNDARY IN LOGIC
          JNE ONLED
          MOV AL,00H
          MOV DX,FFE1H
          OUT DX,AL
          JMP UP
ONLED: MOV DX,FFE1H
          MOV AL, FFH
          OUT DX,AL
          JMP UP
```

8086 INTERFACING
(Beyond syllabi)

26. ELEVATOR INTERFACE

Introduction:

This interface simulates the control and operation of an elevator. Four floors are assumed and for each floor a key and corresponding LED Indicators are provided to serve as request buttons and request status indicators. The elevator itself is represented by a column of ten LED's.

The motion of elevator can be simulated by turning on successive LED's one at a time. The delay between turning off one LED and turning on the next LED can simulate the speed of the elevator. It is possible to read the request status information through one port, reset the request indicators through the another port and control the elevator through another port.

Circuit Description:

This interface has 4 keys marked 0,1,2 & 3 representing the request buttons at the 4 floors. Pressing of key causes the corresponding flip-flop to be set. The outputs of the 4 flip-flops can be read through Port B (PB0, PB1,PB2 & PB3). Also the status of these signals is reflected by a set of 4 LEDs. The flip-flop can be reset through port A (PA4, PA5, PA6 & PA7). A column of 10 LEDs representing the elevator can be controlled through port A (PA0, PA1, PA2 & PA3).

These port lines are fed to the inputs of the decoder 7442, whose outputs are used to control the ON / OFF states of the LEDs, Which simulate the motion of the elevator.

The elevator is operated as follows:

1. Initially, the elevator is at the ground floor.
2. When the elevator reaches any floor, it stays at that floor until the request from another floor is made. When such a request is detected, it moves to that floor.
3. The floor requests are scanned in a fixed order (0, 1, 2 & 3 floors).

ELEVATOR INTERFACE WITH 8086

AIM: To write an ALP to interface Elevator with 8086

PROGRAM:

```
                OUTPUT 2500AD
                SYMBOLS
DATA            EGMENT
FCODE           DB 00H,03H,06H,09H
FCR             DB E0H,D3H,B6H,79H
DATA            NDS
CODE            EGMENT
                ASSUME CS:CODE, DS:DATA
                ORG 1000H
                MOV DX,FFE7H
                MOV AL,81H
                OUT DX,AL
                XOR AX,AX
LOOP1           MOV AL,AH
                OR AL,F0H
                MOV DX,FFE1H
                OUT DX,AL
                MOV DX,FFE3H
LOOP2           IN AL,DX
                AND AL,0FH
                CMP AL,0FH
                JZ LOOP2
                MOV SI,00
FINDF           ROR AL,1
                JNC  FOUND
                INC SI
                JMP FINDF
FOUND           MOV AL,FCODE[SI]
                CMP AL,AH
                JA  LOOP
                JB DOWN
CLEAR           MOV AL,FCLR[SI]
                MOV DX,FFE1H
                OUT DX,AL
                JMP LOOP1
LOOP            CALL DELAY
                INC AH
                XCHG AL,AH
                OR AL,F0H
                MOV DX,FFE1H
                OUT DX,AL
                AND AL,0FH
                XCHG AH,AL
```

```
                CMP AL,AH
                JNZ  LOOP
                JMP    CLEAR
DOWN            CALL DELAY
                DEC AH
                XCHG AH,AL
                OR AL,F0H
                MOV DX,FFE1H
                OUT DX,AL
                AND AL,0FH
                XCHG AL,AH
                CMP AL,AH
                JNZ  DOWN
                JMP  CLEAR
DELAY           MOV CX,800H
UP1             LOOP  UP1
                MOV CX,FFFFH
UP2             LOOP  UP2
                RET
CODE            ENDS
                END
```

27. MICROPROCESSOR BASED AUTOMATIC GATE OPENING / CLOSING

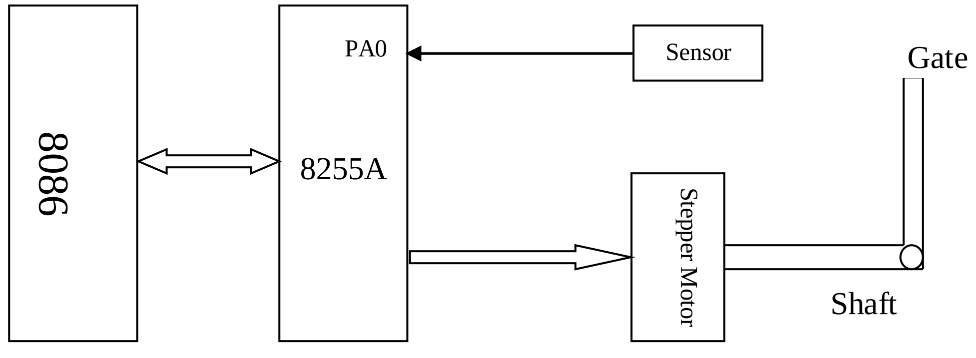
Aim:

To operate the gate (open and close) using 8086 Microprocessor.

Description:

The opening and closing of a gate is controlled by 8086 Microprocessor by sensing the vehicle coming in and going out using a sensor. Here LDR is used as sensor. When the vehicle passes over the sensor, light falling on it gets obstructed and a signal is generated by a signal conditioning circuit. This signal is given to PA0 pin of 8255A. When 8086 detects logic 1 on PA0 of 8255A, it activates the stepper motor to rotate in anticlockwise direction to open the gate. After the vehicle has passed through the gate the 8086 activates the stepper motor again to rotate in clockwise direction to close the gate.

BLOCK DIAGRAM:



PROGRAM:

```
            OUTPUT 2500AD
            SYMBOLS
CODE        SEGMENT
            ASSUME CS: CODE
            ORG 2000H
            MOV AL, 90H
            MOV DX, FFE6H
            OUT DX, AL
            MOV DX, FFE0H
BACK        IN AL, DX
            AND AL, 01H
            JZ BACK
            MOV AL, 80H
            MOV DX, FFE7H
            OUT DX, AL
            MOV CL, 0CH
UP2         MOV BL, 04H
            MOV AL, 08H
UP1         MOV DX, FFE1H
            OUT DX, AL
            CALL DELAY
            ROR AL, 01H
            DEC BL
            JNZ UP1
            DEC CL
            JNZ UP2
            CALL DELAY1
            MOV CL, 0CH
UP5         MOV BL, O4H
            MOV AL, 01H
UP4         MOV DX, FFE1H
            OUT DX, AL
            CALL DELAY
```

```
            ROL AL, 01H
            DEC BL
            JNZ UP4
            DEC CL
            JNZ UP5
            INT 3
DELAY       PUSH CX
            MOV CX, 0F0FH
UP          NOP
            NOP
            NOP
            LOOP UP
            POP CX
            RET
DELAY1      PUSH CX
            PUSH BX
            MOV BX, 0FH
UP7         MOV CX, FFFFH
UP6         NOP
            NOP
            NOP
            LOOP UP6
            DEC BX
            JNZ UP7
            POP BX
            POP CX
            RET
CODE        ENDS
            END
```

28. MICROPROCESSOR BASED TEMPERATURE CONTROLLER

Aim:

To control the temperature of a device under consideration by interfacing RTD (Resistance Temperature Detector) interfaced with 8086 Microprocessor.

Introduction:

An electrical transducer is a sensing device using which a physical, mechanical or optical quantity to be measured is transformed into an electrical signal (voltage or current) proportional to the input. This output can be amplified or modified to suit the requirements of the indicating or controlling equipment. The same output can be converted to a digital format for display, storage, or on-line computation.

Most of the metals exhibit an increase in their resistivity with temperature. Copper, nickel and platinum are metals that exhibit good sensitivity and reproducibility for temperature measurement purpose. The platinum resistance element is the best choice for many applications, because of its inherent reproducibility and accuracy.

The temperature transducer PT 100 converts the change in temperature to change in resistance. This change in resistance can be easily represented as a change in voltage using a resistance bridge. The temperature of the heating element can then be controlled by interfacing the transducer with a microprocessor. For this, the analog output of the transducer is converted to digital output, which is read by the microprocessor, which in-turn controls the power to the heating element. This interface uses a cost effective method of A/D conversion with the help of a Digital-to-Analog converter (DAC). Either successive approximation or counter algorithm is implemented in software, to realize analog-to-digital conversion.

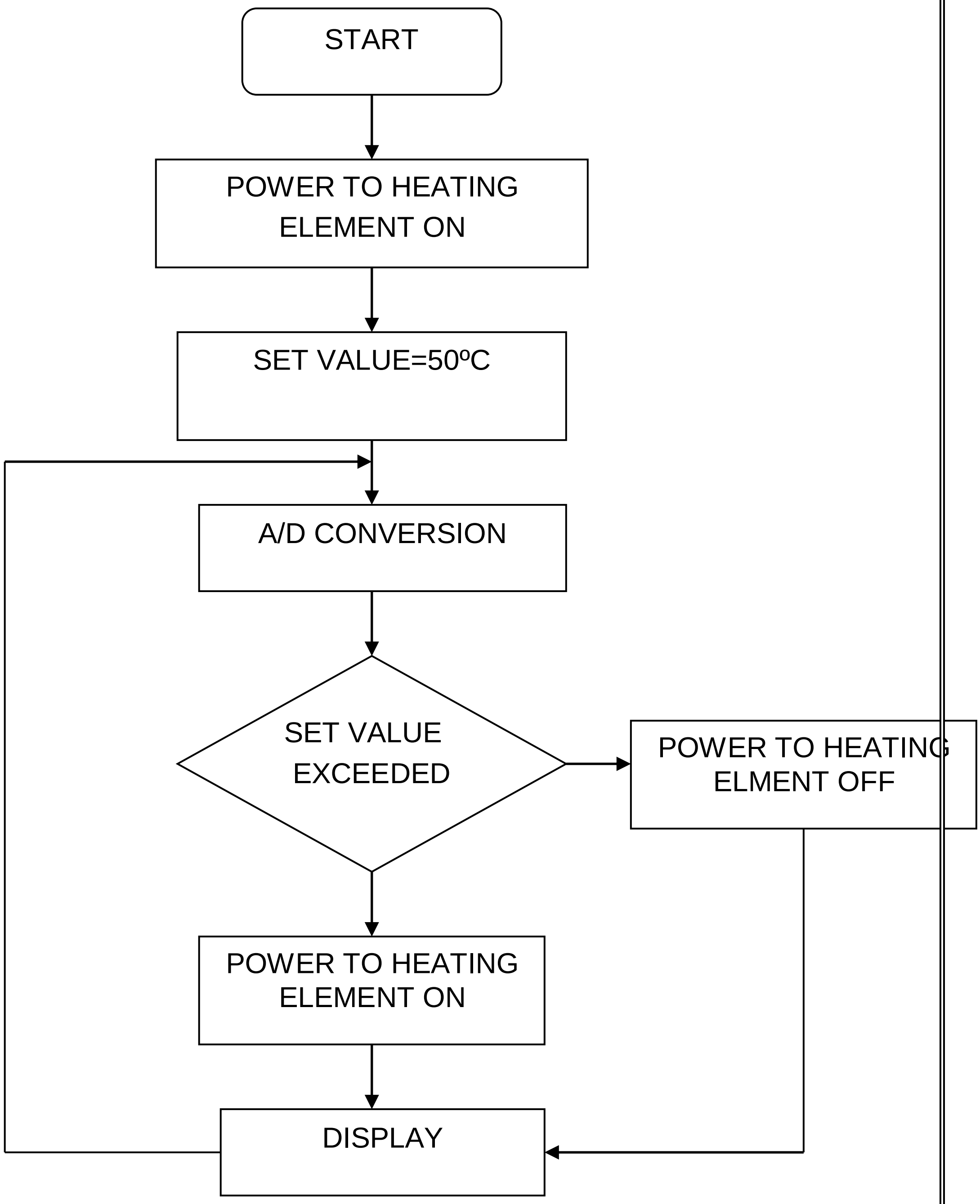
DESCRIPTION OF THE CIRCUIT:

The bridge has a resistance of 100Ω in each arm (one of the arms of the resistor bridge is replaced by the Temperature Transducer –PT100). When the temperature is 0º C, the PT 100 will have a resistance of 100Ω. So the bridge will be in balanced condition and its output will be zero. As the temperature increases, the resistance of the transducer increases, thus creating an imbalance in the resistance bridge. As a result, a voltage difference will be generated between the points A & B in the bridge, which is fed to a differential amplifier. The single ended output of the differential amplifier is fed to an inverting amplifier, whose output is fed to a comparator. The other input of the comparator is connected to the output of 8-bit DAC 0800 whose input is controlled through 8255 port A.

The comparator output is monitored using port line PCO. The comparator output will be high as long as the DAC output is lesser than the Analog input. This line will go low when the DAC input represents the digital equivalent of the analog input voltage. Then the read value is compared with a preset temperature and the temperature of the heating element is controlled by switching OFF the power to the heating element using PBO till the temperature reduces to the preset value. The software initiates the next conversion process after the previous conversion result is displayed.

The software may employ either successive approximation or the counter method for the conversion process. In the first method, the value for each bit (0/1) is determined, starting with the most significant bit, in 8 successive steps. In the counter method, values from 00 to FFH are tried (in succession) successively. The digital equivalent of the Analog input would be that at which the DAC output matches the Analog input (indicated by the comparator output going low).

FLOWCHART:


START
POWER TO HEATING ELEMENT ON
SET VALUE=50ºC
A/D CONVERSION
SET VALUE EXCEEDED
POWER TO HEATING ELMENT OFF
POWER TO HEATING ELEMENT ON
DISPLAY

PROGRAM:

```
                OUTPUT 2500AD
                ORG 2000H
                MOV AX, 00H
                MOV CS, AX
                MOV SS, AX
                MOV SP, 4000H
                MOV DX, 0FFE6H
                MOV AL, 81H
                OUT DX, AL
                JMP START
        MES0    DB 0AH, 0DH, ‘SET VALUE = 50.00’, 0F8H, 43H, 0AH, 00H
        MES1    DB 0DH, ‘TEMPERATURE =’, 00H
        MES2    DB 0F8H, 43H, 20H, 20H, 00H
        START   LEA DX, MES0
                MOV AX, DX
                CALL FAR 0FE00: 1B55H
        MAIN    MOV CL, 00H         ; COUNT INITIALISATION
        LOOP3   MOV AL, CL
                MOV DX, 0FFE0H      ; SEND TO PORT A
                OUT DX, AL
                CALL DELAY
                MOV DX, 0FFE4H      ; READ FROM PORT C
                IN AL, DX
                AND AL, 01H         ; CHECK COMPARATOR O/P
                JZ FINISH
                INC CL              ; INCREMENT THE COUNT
                JMP LOOP3
        FINISH  CLC
                MOV DX, 0FFE2H
                MOV AL, CL          ; PUT THE CONVERTED DATA IN
MEMORY
                CMP AL, 2BH         ; COMPARE WITH SET VALUE
                JG SPLYOFF
        SPLYON  MOV AL, 00H
```

```
                OUT DX, AL
                JMP CONV
   SPLYOFF      MOV AL, 0FFH
                OUT DX, AL
                CALL DELAY
   CONV         MOV AL, CL        ; GET THE CONVERTED DATA
                MOV BX, 75H       ; MULTIPLICATION FACTOR
                MUL BL
                CALL BIN2BCD16
                CALL DISPVAL
                JMP MAIN
   DELAY        PUSH CX
                MOV CX, 400H
     UP1        LOOP UP1
                POP CX
                RET
BIN2BCD16       PUSH AX
                PUSH BX
                PUSH CX
                MOV SI, 3000H
                MOV BX, 2710H
                CALL BCONV
                MOV BX, 3E8H
                CALL BCONV
                MOV BX, 64H
                CALL BCONV
                MOV BX, 0AH
                CALL BCONV
                MOV [SI], AL
                POP CX
                POP BX
                POP AX
                RET
   BCONV        MOV CL, 00H
   BNEXT        INC CL
                SUB AX, BX
                JNC BNEXT
```

```
                DEC CL
                ADD AX, BX
                CLC
                MOV [SI], CL
                INC SI
                RET
DISPLAY         LEA DX, MES1
                MOV AX, DX
                CALL FAR FE00: 1B55H
                MOV SI, 3001H
                CALL DBYTE
                MOV AL, '.'
                CALL FAR FE00: 1B50H
                INC SI
                CALL DBYTE
                LEA DX, MES2
                MOV AX, DX
                CALL FAR FE00: 1B50H
                CALL DELAY
                RET
DBYTE           MOV AL, [SI]
                MOV CL, 04H
                ROL AL, CL
                INC SI
                MOV AH, [SI]
                OR AL, AH
                CALL FAR FE00: 1B64H
                RET
CODE            ENDS
                END
```

29. TRAFFIC LIGHTS INTERFACE

Introduction:

The 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 LEDs can be controlled by a port line.

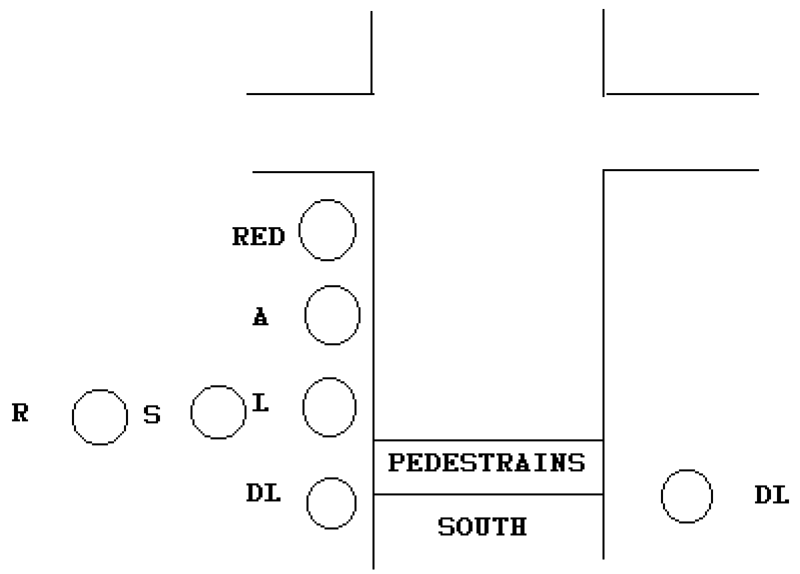
Thus this interface allows user to simulate a variety of traffic situations. using appropriate software routines. The sample programs provided in section 4 of this manual simulate some interesting traffic movement sequences.

Description of the Circuit:

Please refer to the schematics of this interface presented at the end of this manual. As already mentioned, the interface provides a set of six LEDs at each of the four corners of a four road junction.

The organization of these LEDs is identical at each of the four corners. Hence, for simplicity, the organization is described below with reference to the LEDs at SOUTH-WEST corner only.

The LEDs at SOUTH-WEST corner are organized as follows:



RED: Referred to as SOUTH RED henceforth

A: Referred to as SOUTH AMBER henceforth

L: Referred to as SOUTH LEFT henceforth

S: Referred to as SOUTH STRAIGHT henceforth

R: Referred to as SOUTH RIGHT henceforth

DL: Referred to as SOUTH PEDESTRAIN henceforth

please note that DL refers to a set of two LEDs, one on either side of the road.)

Of these, the first five LEDs will be ON or OFF depending on the state of the corresponding port line (LED is ON if the port line is logic HIGH and LED is OFF if the port line is logic LOW.) The last one marked DL is a set of two dual-colour LEDs 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 LEDs and these are Controlled by 24port lines. Each port line is inverted and buffered using 7406 (open collector inverter buffers) and is used to control an LED. Dual-colour LEDs and controlled by a port line and its complement.

The 24 LEDs and their corresponding port lines are summerised below:

	LED	Port line
SOUTH	RED	PA3
	AMBER	PA2
	LEFT	PA0
	STRAIGHT	PC3
	RIGHT	PA1
	PEDESTRIAN	PC6
EAST	RED	PA7
	AMBER	PA6
	LEFT	PA4

	STRAIGHT	PC2
	RIGHT	PA5
	PEDESTRIAN	PC7
NORTH	RED	PB3
	AMBER	PB2
	LEFT	PB0
	STRAIGHT	PC1
	RIGHT	PB1
	PEDESTRIAN	PC4
WEST	**RED**	PB7
	AMBER	PB6
	LEFT	PB4
	STRAIGHT	PC0
	RIGHT	PB5
	PEDESTRIAN	PC5

User can assign any meaningful interpretation to these LEDs and then develop software accordingly. Usually, the interpretation would be as follows:

Vehicles coming from one direction are controlled by the LEDs at the opposite corner**.** For example, vehicles coming from **NORTH** are controlled by the set of LEDs at the **SOUTH WEST** corner, as shown below:

Vehicles from **NORTH** can Go left (i.e to **EAST**) if **SOUTH LEFT LED** is ON

Go right (i.e to **WEST**) if **SOUTH RIGHT LED** is ON

Go straight (i.e to **SOUTH**) if **SOUTH-STRAIGHT LED** is ON

Further, the above movements are allowed only if

SOUTH RED LED is OFF. If **SOUTH RED LED** is ON, no movement is allowed for vehicles from north. Pedestrian crossing on south is allowed when **SOUTH PEDESTRIAN** is green and disallowed when it is red. It is obvious that, logically some combinations cannot be allowed. For example, **SOUTH RED**=OFF, **SOUTH STRAIGHT**=ON and **SOUTH PEDESTRIAN**=**GREEN** cannot be allowed. (Vehicles are allowed to go from **NORTH** to **SOUTH** and pedestrians are allowed to cross on **SOUTH**). **SOUTH AMBER** can be ON to indicate that **SOUTH RED** is about to change its status from off to ON.

The movement of vehicles and pedestrians on other road scan be controlled in a similar way.

As already noted, user can assign a different interpretation if desired. However, the sample programs presented in the next section are based on the above simple interpretation.

Example:

Determine port values for the following traffic situation:

Vehicles from WEST are allowed to go **NORTH** or **EAST.**

Vehicles from EAST are allowed to go west.

Pedestrian crossing is allowed on **SOUTH.**

NO other vehicle movements/pedestrian crossings are allowed.

Now, as per the above interpretation, the status of the LEDs should be as shown below:

	RED	**AMBER**	**LEFT**	**STRAIGHT**	**RIGHT**	**PEDESTRIAN**
SOUTH	ON	OFF	OFF	OFF	OFF	GREEN
EAST	OFF	OFF	ON	ON	OFF	RED
NORTH	ON	OFF	OFF	OFF	OFF	RED
WEST	OFF	OFF	OFF	ON	OFF	RED

From the correspondence already described between the Port lines and LEDs, we can now determine the logic values for each port line, for example, **PA0** should be **LOGIC LOW** as **SOUTH LEFT LED** is OFF. Determining the values of the other port lines in a similar fashion, we can arrive at the following port values:

A=18H **PB=08H** **PC=B5H** User can work out the port values for other situations in a similar way.

TRAFFIC CONTROLLER WITH 8086

```
START       MOV AL,80H
            MOV DX,0FFE6H
            OUT DX,AL
AGAIN               MOV SI,2038H
NEXT        MOV AL,[SI]
            MOV DX,0FFE0H
            OUT DX,AL
            INC SI
            ADD DX,2
            MOV AL,[SI]
            OUT DX,AL
            INC SI
            ADD DX,2
            MOV AL,[SI]
            OUT DX,AL
            INC SI
            CALL DELAY
            CMP SI,2056H
            JNZ NEXT
            JMP AGAIN
DELAY               MOV CX,OFFH
DLY5        PUSH CX
            MOV CX,03FFH
DLY10             NOP
            LOOP DLY10
            POP CX
            LOOP DLY5
            RET
CODE        ENDS
            END
            DB 10H, 81H, 7AH
            DB 44H, 44H, F0H
```

```
DB 08H, 11H, E5H
DB 44H, 44H, F0H
DB 81H, 10H, DAH
DB 44H, 44H, F0H
DB 11H, 08H, B5H
DB 44H, 44H, F0H
DB 88H, 88H, 00H
DB 44H, 44H, F0H
 DB 00H
```

CYCLE – III

80C31/51 Programming

30. a) ADDITION OF TWO UNSIGNED BYTES

AIM: To write an ALP to add two unsigned bytes stored in internal RAM and store back the result in internal RAM.

PROGRAM:

```
ORG 8000H
MOV R0, #12H
MOVX R1, #45H
MOV A, R1
ADD A,R0
MOV 80, A
LJMP 03H
```

OBSERVATIONS AFTER EXECUTION OF CODE:

DATA1:	DATA2:
R0: 12H	R0: 2DH
R1: 45H	R1: 01H
RESULT:	RESULT:
80: 57H	80: 2EH

30. b) ADDITION OF TWO UNSIGNED BYTES

AIM: To write an ALP to add two unsigned 8 bit numbers stored in external RAM and store back the result in external memory.

PROGRAM:

```
ORG 8000H
MOV DPTR, #9000H
MOVX A, @DPTR
MOV B,A
INC DPTR
MOVX A, @DPTR
ADD A,B
INC DPTR
MOVX @DPTR, A
LJMP 03H
```

OBSERVATIONS AFTER EXECUTION OF CODE:

DATA1:	DATA2:
9000: 11H	9000: 2EH
9001: 22H	9001: 49H
RESULT:	RESULT:
9003: 33H	9003: 77H

31. SUBTRACTION OF TWO UNSIGNED BYTES

AIM: To write an ALP to subtract two unsigned 8 bit numbers stored in external RAM and store back the result in external RAM.

PROGRAM:

```
ORG 8000H
MOV DPTR, #9000H
MOVX A, @DPTR
MOV B,A
INC DPTR
MOVX A, @DPTR
CLR C
SUBB A,B
INC DPTR
MOVX @DPTR, A
LJMP 03H
```

OBSERVATIONS AFTER EXECUTION OF CODE:

DATA1:	DATA2:
9000: 01H	9000: 0AH
9001: 10H	9001: 14H
RESULT:	RESULT:
9003: 0FH	9003: 0AH

32. MULTIPLICATION OF TWO UNSIGNED BYTES

AIM: To write an ALP to MULTIPLY two unsigned 8 bit numbers stored in external RAM and store back the result in external RAM.

PROGRAM:

```
ORG 8000H
MOV DPTR, #9000H
MOVX A, @DPTR
MOV B,A
INC DPTR
MOVX A, @DPTR
MUL AB
INC DPTR
MOVX @DPTR, A
MOV B,A
INC DPTR
MOVX @DPTR, A
LJMP 03H
```

OBSERVATIONS AFTER EXECUTION OF CODE:

DATA1:
9000: 11H
9001: 02H
RESULT:
9003: 22H
9004: 00H

DATA2:
9000: 12H
9001: 12H
RESULT:
9003: 44H
9004: 01H

33. DIVISION OF TWO UNSIGNED BYTES

AIM: To write an ALP to DIVIDE two unsigned 8 bit numbers stored in external RAM and store back the result in external RAM.

PROGRAM:

```
ORG 8000H
MOV DPTR, #9000H
MOVX A, @DPTR
MOV B,A
INC DPTR
MOVX A, @DPTR
DIV AB
INC DPTR
MOVX @DPTR, A
MOV B,A
INC DPTR
MOVX @DPTR, A
LJMP 03H
```

OBSERVATIONS AFTER EXECUTION OF CODE:

DATA1:	DATA2:
9000: 02H	9000: 02H
9001: 08H	9001: 23H
RESULT:	RESULT:
9003: 04H	9003: 11H
9004: 00H	9004: 01H

34. ADDITION OF 'N' UNSIGNED BYTES

AIM: To write an ALP to add 'N' unsigned bytes stored in external RAM and store back the result in external RAM.

PROGRAM:

```
            ORG 8000H
            MOV DPTR, #9000H
            MOVX A, @DPTR
            MOV B,A
            MOV R1, #00H
            MOV R2, #00H
UP          INC DPTR
            MOVX A, @DPTR
            ADD  A,R1
            JNC DOWN
            INC R2
DOWN        MOV R1,A
            DJNZ B,UP
            INC DPTR
            MOV A,R1
            MOVX @DPTR, A
            INC DPTR
            MOV A,R2
            MOVX A, @DPTR
            LJMP 03H
```

OBSERVATIONS AFTER EXECUTION OF CODE:

DATA1:
9000: 04H
9001: 0AH
9002: FFH
9003: 02H
9004: AEH
RESULT:
9005: B9H
9006: 01H

DATA2:
9000: 03H
9001: 01H
9002: A0H
9003: 15H
RESULT:
9004: B6H
9004: 00H

35. BINARY TO BCD

AIM: To write an ALP to convert Binary byte to BCD equivalent and store the result in external RAM location (9000H).

PROGRAM:

```
ORG 8000H
MOV DPTR, #9000H
MOV B, #10
MOV A, #00010010B
DIV AB
SWAP A
ADD A,B
MOVX @DPTR, A
LJMP 03H
```

OBSERVATIONS AFTER EXECUTION OF CODE:

DATA1:	DATA2:
00010010B	01000110B
RESULT:	RESULT:
9000: 18	9000: 70

CYCLE – IV

80C31/51 INTERFACING

36. INTERFACING OF DAC TO 80C51

AIM: TO Write an ALP to generate Triangular and Square Waves using DAC 0800 interfaced to 80C51 Micro Controller.

PROGRAM:

TRIANGULAR WAVE GENERATION

```
                OUTPUT 2500AD
                SYMBOLS
CODE            SEGMENT
                ASSUME CS:CODE
                ORG 8000H
                MOV A,#80H
                MOV DPTR,#E803H
                MOVX @DPTR,A
UP2             MOV A,#00H
                MOV DPTR,#E800H
UP              MOVX @DPTR,A
                INC A
                CJNE A,#FFH,UP
UP1             DEC A
                MOVX @DPTR,A
                CJNE A,#OOH,UP1
                JMP UP2
                LJMP 03
```

SQUARE WAVE GENERATION

```
                OUTPUT 2500AD
                SYMBOLS
CODE            SEGMENT
                ASSUME CS:CODE
                ORG 8000H
                MOV A,#80H
                MOV DPTR,#E803H
                MOVX @DPTR,A
UP1             MOV A,#00H
                MOV DPTR,#E800H
                MOVX @DPTR,A
                CALL DELAY
                MOV  A,#FFH
                MOV DPTR,#E800H
                MOVX @DPTR,A
                CALL DELAY
                JMP UP1
DELAY           MOV R0,#FFH
UP              NOP
                DJNZ R0,UP
                RET
                LJMP 03
```

37. INTERFACING STEPPER MOTOR TO 80C51

AIM: To interface stepper motor of half step size = 0.9 degrees to 80C51 and to rotate in clockwise and Anti clock wise direction for given angle of coverage.

Program:

(ANTICLOCK WISE ROTATION)

```
        ORG 8000H
        MOV A, #80H          ; COMMAND FOR 8255A TO MARK ALL PORTS
                                AS  OUTPUT
        MOV DPTR,#E803H      ;CREG ADD FOR 8255A
        MOVX @DPTR,A
        MOV RO,#19H          ; DECIDES ANGLE OF COVERAGE = NO. OF
                               COMMANDS* STEP SIZE
        MOV R1,#04           ; INNER LOOP TO ELIMINATE MOTOR STOP
                               AFTER EVERY FOUR ROTATIONS
UP2:    MOV A,#01H           ; INTITIAL COMMAND FOR ANTICLOCK WISE
                               ROTATION
                             ;08H FOR CLOCKWISE ROTATION
UP1:    MOV DPTR,#E800H      ;8255A PORT A ADD. WHERE STEPPER IS
                               CONNECTED
        MOVX @DPTR,A
        CALL DE
        RL A                 ;RRA  FOR CLOCK WISE ROTATION
        DJNZ  R1,UP1
        DJNZ R0,UP2
DE:     MOV R3, #F0H
UP3:    MOV R4,#FFH
UP4:    NOP
        NOP
        DJNZ R4,UP4
        DJNZ R3,UP3
        RET
```

Program:

(CLOCK WISE ROTATION)

```
        ORG 8000H
        MOV A, #80H          ; COMMAND FOR 8255A TO MARK ALL PORTS AS
                                OUTPUT
        MOV DPTR,#E803H      ;CREG ADD FOR 8255A
        MOVX @DPTR,A
        MOV RO,#19H          ; DECIDES ANGLE OF COVERAGE = NO. OF
                               COMMANDS* STEP SIZE
        MOV R1,#04           ; INNER LOOP TO ELIMINATE MOTOR STOP
                               AFTER EVERY FOUR
                                  ROTATIONS
UP2:    MOV A,#08H           ; INTITIAL COMMAND FOR CLOCK WISE
                               ROTATION
```

```
                                          ;01H FOR ANTICLOCKWISE ROTATION
UP1:    MOV DPTR,#E800H   ;8255A PORT A ADD. WHERE STEPPER IS
                                          CONNECTED
        MOVX @DPTR,A
        CALL DE
        RRA                        ;RL A  FOR CLOCK WISE ROTATION
        DJNZ  R1,UP1
        DJNZ R0,UP2
DE :     MOV R3, #F0H
UP3:     MOV R4,#FFH
UP4:    NOP
        NOP
        DJNZ R4,UP4
        DJNZ R3,UP3
        RET
```

RESULT: Stepper motor interfaced to 80C51 using 8255A Port A lower nibble to rotate clock wise and anti-clock wise directions for desired angle of coverage.

38. Generation of Square wave at P1.1 of 8051 using Timers in Mode1 & Mode 2

AIM: To generate square wave at P1.1 using 8051 Timers in Mode 1 & Mode2 using Polling

Timer o in Mode-1: TIMER 0 is a 16 bit (TH0-TL0) timer when operated in mode1 acts like timer with preloaded 16-bit initial count . TMOD is the SFR to program to operated Timer 0 in mode1. Mode 1 is single shot timer which is needed to be reloaded each time explicitly.

Polling TF0 (Timer 1 over flow Flag) at regular interval & complementing P1.1 using bit addressable instruction generates square wave of required frequency on CRO connected at P1.1

Program:

```
        ORG 8000H
        MOV TMOD, #01H
L1:     MOV TL0,#1AH
        MOV TH0,#FFH
        SETB TR0
BK:     JNB TF0,BK
        CLR TR0
        CPL P1.1
        CLR TF0
        SJMP L1
```

Timer 1 in Mode-2: Timer 1 when operated in Mode-2 reloads the 8-bit initial value loaded in TH1 repeatedly each time the value expires (counts down to zero).

Polling TF1 (Timer 1 over flow Flag) at regular interval & complementing P1.1 using bit addressable instruction generates square wave of required frequency on CRO connected at P1.1

Program:

```
        ORG 8000H
        MOV TMOD,#20h
        MOV TH1,#1AH
L1:     SETB TR1
Bk:     JNB TF1,Bk
        CLR TR1
        CPL P1.1
        CLR TF1
        SJMP L1
```

RESULT: Square wave of desired frequency decided by preloaded value into timer is observed on CRO. Its amplitude and frequency were analyzed.

80C31/51 INTERFACING
(Beyond Syllabi)

39. ELEVATOR INTERFACE WITH 8051

AIM: To write an ALP to interface Elevator with 80C51

PROGRAM :

```
            SYMBOLS
SEG         EQU 0E8H
P2          EQU 0A0H
DPL         EQU 82H
DPH         EQU 83H
            ORG 8000H
START       MOV P2,#SEG
            MOV R0,#03H
            MOV A,#82H
            MOVX @R0,A
            CLR A
            MOV R0,A
            MOV R1,A
LOOP1       MOV A,R1
            ORLA,#0F0H
            MOV R0,#00H
            MOV X @R0,A
            MOV DPTR,#FLOOR
            INC R0
LOOP2       MOVX A,@R0
            ORL A,#0F0H
            MOV R2,A
            INC A
            JZ LOOP2
LOOP3       MOV A,R2
            RRC A
            MOV R2,A
            JNC DECIDE
            INC DPTR

            SJMP LOOP3
DECIDE      LCALL DELAY
            CLR A
            MOV C A,@A+DPTR
            CJNE A,1,L1
            SJMP RESET
L1          JC DOWN
                UP          INC R1
            MOV A,R1
            ORL A,#0F0H
```

```
                MOV R0,#00H
                MOV X@R0,A
                SJMP DECIDE
DOWN            DEC R1
                MOV A, R1
                ORL A,#0F0H
                MOV R0,#00H
                MOV X@R0,A
                SJMP DECIDE
RESET           MOV A,#05H
                ADD A,DPL
                MOV DPL,A
                CLE A
                MOV R0,A
                MOV C A@A+DPTR
                MOV X@R0,A
                SJMP LOOP1
DELAY           PUSH DPH
                PUSH DPL
                MOV DPTR,#00H
DELAY1          INC DPTR
                MOV A,DPH
                ORL A,DPL
                JNZ DELAY1
                POP DPL
                POP DPH
                RET
FLOOR           DB 00H,03H,06H
                DB 09H,00H,0E0H
                DB 0D3H,0B6H,79H
```

40. TRAFFIC CONTROLLER using 8051

AIM: To write an ALP to interface Traffic Controller using 8051

PROGRAM :

```
                ORG 8000H
                MOV DPTR,#CWR
                MOV A,#80H
                MOVX @DPTR,A
AGAIN MOV DPTR,#0000H
NEXTST          MOVX A,@DPTR
                PUSH DPL
                PUSH DPH
                MOV DPTR,#PORT_A
                MOVX @DPTR,A
                POP DPH
                POP DPL
                INC DPTR
                MOVX A,@DPTR
                PUSH DPL
                PUSH DPH
                MOV DPTR,#POR_B
                MOVX @DPTR,A
                POP DPH
                POP DPL
                INC DPTR
                MOVX A,@DPTR
                PUSH DPL
                PUSH DPH
                MOV DPTR,#PORT_C
                MOVX @DPTR,A
                POP DPH
                POP DPL
                INC DPTR
                CALL DELAY
                MOV A,DPL
                CJNE   A,#1EH,NEXTST
                SJMP AGAIN
DELAY           MOV R2,#07H
```

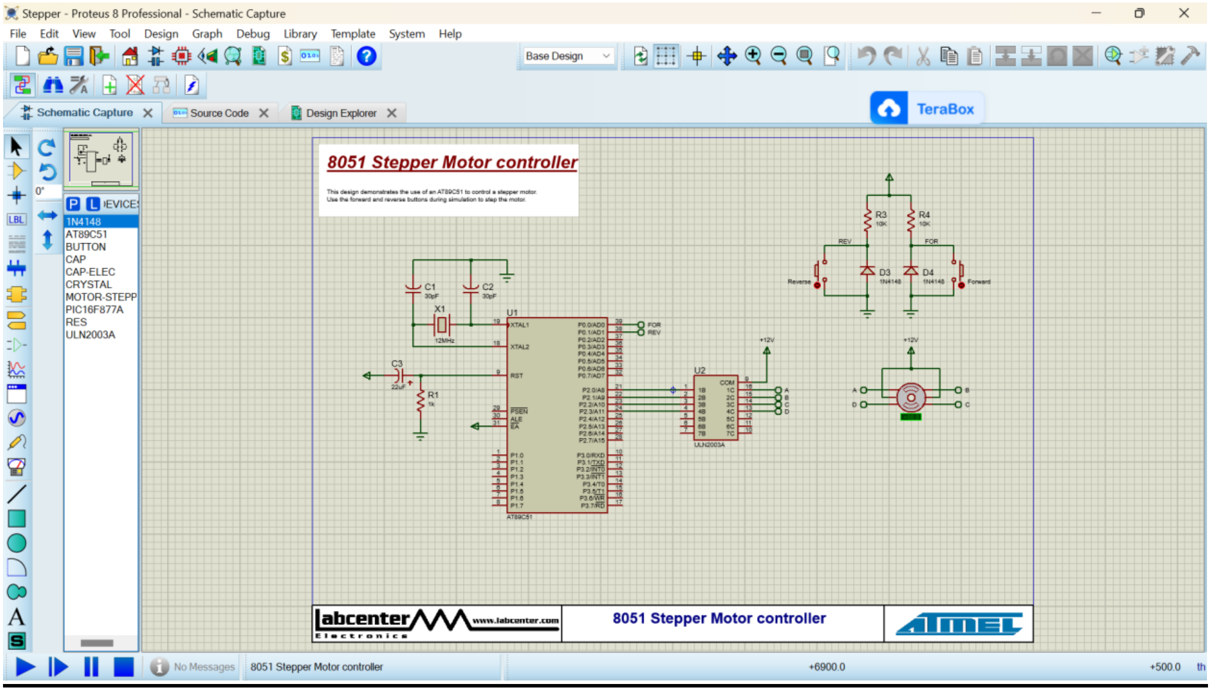
```
LOOP3    MOV R4,#0FFH
LOOP2    MOV R3,#0FFH
LOOP1    DEC R3
         CJNE R3,#00,LOOP1
         DEC R4
         CJNE   R4,#00,LOOP2
         DEC R2
         CJNE R2,#00,LOOP3
         RET
         DB 10H,81H,7AH
         DB 44H,44H,0F0H
         DB 08H,11H,0E5H
         DB 44H,44H,0F0H
         DB 81H,10H,0DAH
         DB 44H,44H,0F0H
         DB 11H,08H,0B5H
         DB 44H,44H,0F0H
         DB 88H,88H,00H
         DB 44H,44H,0F0H
         DB 00H
```

INTERFACING STEPPER MOTOR TO 80C51

AIM: To interface stepper motor of half step size = 0.9 degrees to 80C51 and to rotate in clockwise and Anti clock wise direction for given angle of coverage.

TOOLS REQUIRED: Proteus 8 Professional

Schematic:



ALP Program:

ANTICLOCK WISE ROTATION

```
            ORG 0000H
            SJMP START
            ORG 3000H
START:      MOV RO,#19H      ; ANGLE OF COVERAGE = 19h*4*0.9=90 DEGREES
            MOV R1,#04       ; INNER LOOP TO ELIMINATE MOTOR STOP
                               AFTER EVERY FOUR ROTATIONS
UP2:         MOV A,#01H      ; INTITIAL COMMAND FOR ANTICLOCK WISE
                                  ROTATION 08H FOR CLOCKWISE ROTATION
UP1:         MOV P1,A         ;AFTER RESET ALL PORTS ARE OUTPUT
            CALL DE
            RL A              ;RRA  FOR CLOCK WISE ROTATION
            DJNZ  R1,UP1
            DJNZ R0,UP2

DE:     MOV R3, #F0H
UP3:    MOV R4,#FFH
UP4:    NOP
```

```
NOP
DJNZ R4,UP4
DJNZ R3,UP3
RET
```

Source Code:

```
typedef unsigned char   uchar;
typedef unsigned int    uint;

void delayms(uint);

// Array of Stepping Sequences
uchar const sequence[8] = {0x02,0x06,0x04,0x0c,0x08,0x09,0x01,0x03};

void main(void)
 { uchar i;
   out_port = 0x03;
   while(1)
    { // Has the forward key been pressed ?
      if (!key_for)
       { i = i<8 ? i+1 : 0;
         out_port = sequence[i];
         delayms(50);
       }
      // Has the reverse key been pressed ?
      else if (!key_rev)
       { i = i>0 ? i-1 : 7;
         out_port = sequence[i];
         delayms(50);
       }
    }
 }

void delayms(uint j)
 { uchar i;
   for(; j>0; j--)
    { i = 120;
      while (i--);
    }
 }
```

<u>RESULT:</u> Stepper motor interfaced to 80C51 using 8255A Port A lower nibble to rotate clock wise and anti-clock wise directions for desired angle of coverage.

Generation of Square wave at P1.1 of 8051 using Timers in Mode1 & Mode 2

AIM: To generate square wave at P1.1 using 8051 Timers in Mode 1 & Mode2 using Polling

Timer o in Mode-1: TIMER 0 is a 16 bit (TH0-TL0) timer when operated in mode1 acts like timer with preloaded 16-bit initial count . TMOD is the SFR to program to operated Timer 0 in mode1. Mode 1 is single shot timer which is needed to be reloaded each time explicitly.

Polling TF0 (Timer 1 over flow Flag) at regular interval & complementing P1.1 using bit addressable instruction generates square wave of required frequency on CRO connected at P1.1

Program:

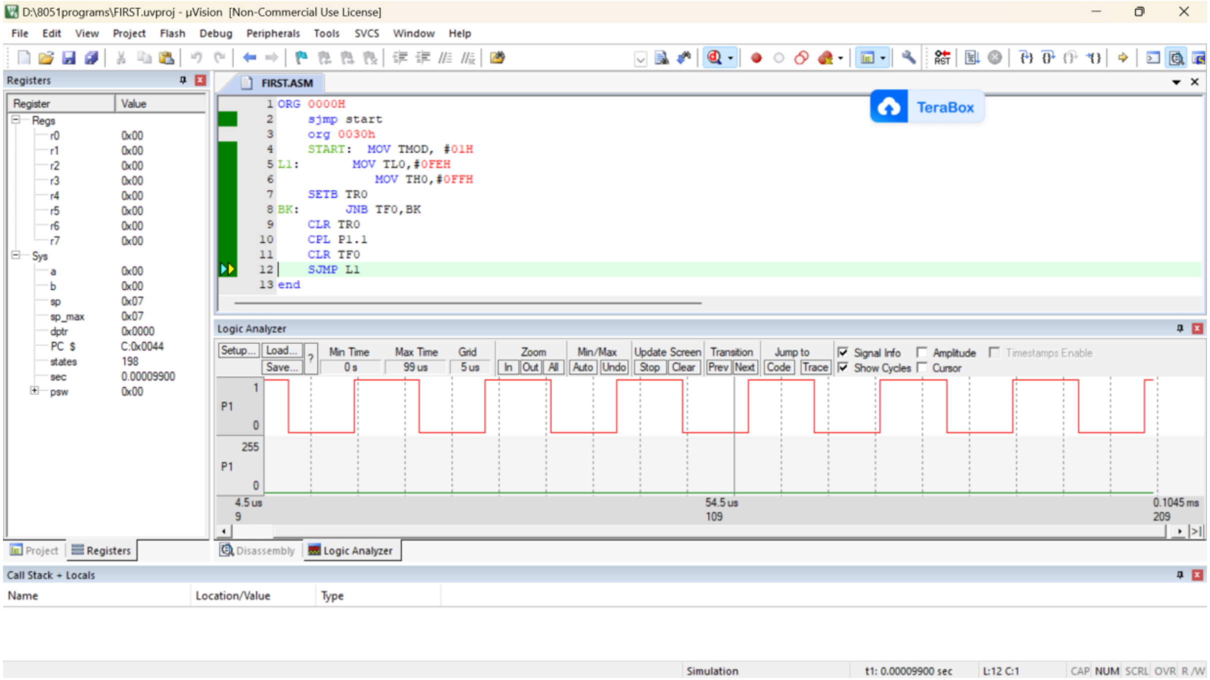
```
        ORG 8000H
        MOV TMOD, #01H
L1:     MOV TL0,#1AH
        MOV TH0,#FFH
        SETB TR0
BK:     JNB TF0,BK
        CLR TR0
        CPL P1.1
        CLR TF0
        SJMP L1
```

Timer 1 in Mode-2: Timer 1 when operated in Mode-2 reloads the 8-bit initial value loaded in TH1 repeatedly each time the value expires (counts down to zero).

Polling TF1 (Timer 1 over flow Flag) at regular interval & complementing P1.1 using bit addressable instruction generates square wave of required frequency on CRO connected at P1.1

Program:

```
        ORG 8000H
        MOV TMOD,#20h
        MOV TH1,#1AH
L1:     SETB TR1
Bk:     JNB TF1,Bk
        CLR TR1
        CPL P1.1
        CLR TF1
        SJMP L1
```



RESULT: Square wave of desired frequency decided by preloaded value into timer is observed on CRO. Its amplitude and frequency were analyzed.