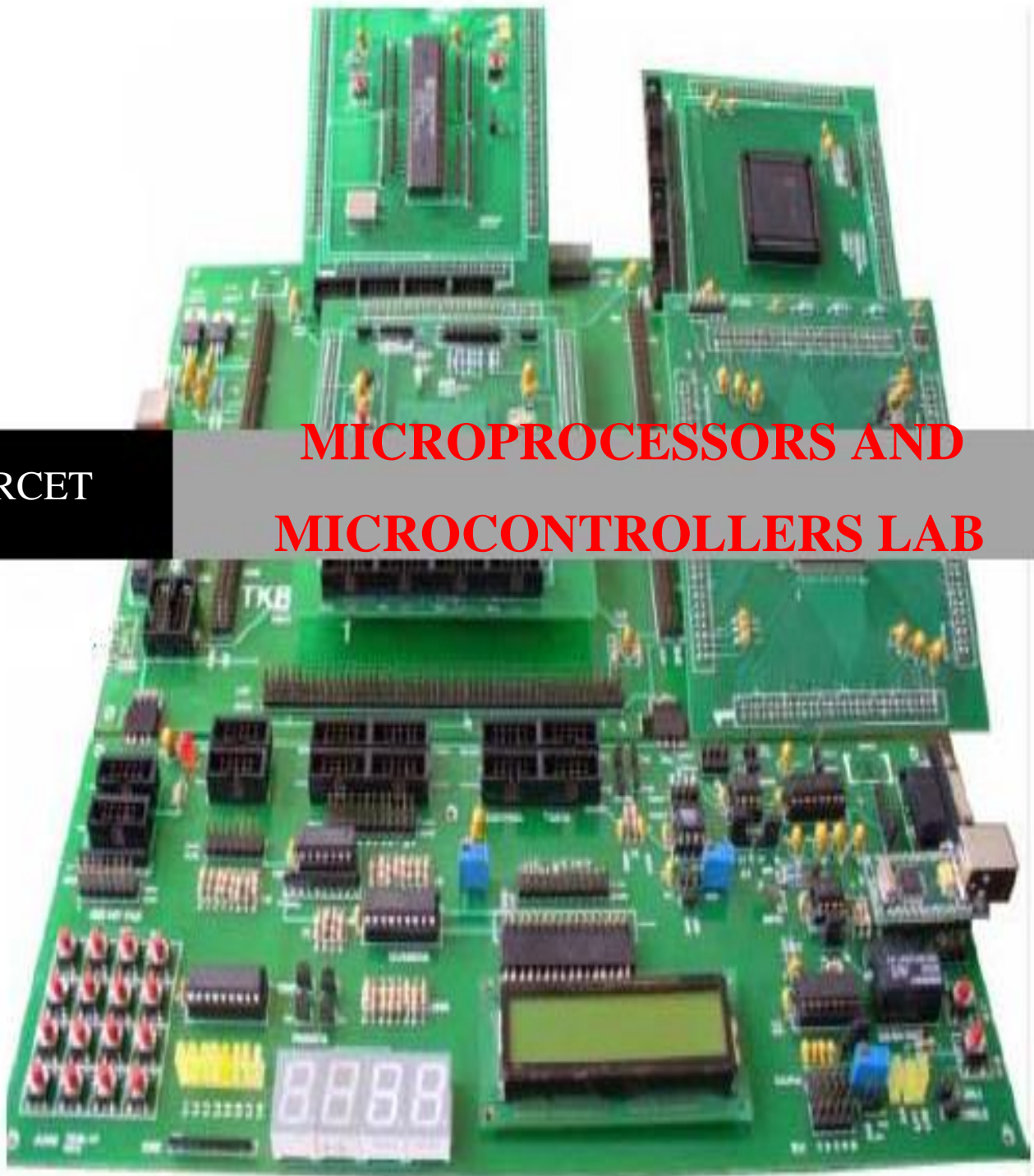**MRCET**

MICROPROCESSORS AND MICROCONTROLLERS LAB

ELECTRICAL AND ELECTRONICS ENGINEERING

VISION

To evolve into a center of excellence in Engineering Technology through creative and innovative practices in teaching-learning, promoting academic achievement & research excellence to produce internationally accepted competitive and world class professionals.

MISSION

To provide high quality academic programmes, training activities, research facilities and opportunities supported by continuous industry institute interaction aimed at employability, entrepreneurship, leadership and research aptitude among students.

QUALITY POLICY

❖ Impart up-to-date knowledge to the students in Electronics & Communication area to make them quality engineers.

❖ Make the students experience the applications on quality equipment and tools.

❖ Provide systems, resources and training opportunities to achieve continuous improvement.

❖ Maintain global standards in education, training and services.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO1: PROFESSIONALISM & CITIZENSHIP

To create and sustain a community of learning in which students acquire knowledge and learn to apply it professionally with due consideration for ethical, ecological and economic issues.

PEO2: TECHNICAL ACCOMPLISHMENTS

To provide knowledge based services to satisfy the needs of society and the industry by providing hands on experience in various technologies in core field.

PEO3: INVENTION, INNOVATION AND CREATIVITY

To make the students to design, experiment, analyze, interpret in the core field with the help of other multi disciplinary concepts wherever applicable.

PEO4: PROFESSIONAL DEVELOPMENT

To educate the students to disseminate research findings with good soft skills and become a successful entrepreneur.

PEO5: HUMAN RESOURCE DEVELOPMENT

To graduate the students in building national capabilities in technology, education and research.

PROGRAMME SPECIFIC OBJECTIVES (PSOs)

PSO1

To develop a student community who acquire knowledge by ethical learning and fulfill the societal and industry needs in various technologies of core field.

PSO2

To nurture the students in designing, analyzing and interpreting required in research and development with exposure in multi disciplinary technologies in order to mould them as successful industry ready engineers/entrepreneurs

PSO3

To empower students with all round capabilities who will be useful in making nation strong in technology, education and research domains.

PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

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 multi disciplinary 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.

Course Objectives:

- To develop and execute variety of assembly language programs of Intel 8086 including arithmetic and logical, sorting, searching, and string manipulation operations.
- To develop and execute the assembly language programs for interfacing Intel 8086 with peripheral devices.
- To develop and execute simple programs on 8051 micro controller.

Course Outcomes:

After going through this course the student will be able to

- The student will learn the internal organization of popular 8086/8051 microprocessors/microcontrollers.
- The student will learn hardware and software interaction and integration.
- To apply the concepts in the design of microprocessor/microcontroller based systems in real time applications

LABORATORY RULES

General Rules of Conduct in Laboratories:

1. You are expected to arrive on time and not depart before the end of a laboratory.

2. You must not enter a lab unless you have permission from a technician or lecturer.

3. You are expected to comply with instructions, written or oral, that the laboratory Instructor gives you during the laboratory session.

4. You should behave in an orderly fashion always in the lab.

5. You must not stand on the stools or benches in the laboratory.

6. Keep the workbench tidy and do not place coats and bags on the benches.

7. You must ensure that at the end of the laboratory session all equipment used is stored away where you found it.

8. You must put all rubbish such as paper outside in the corridor bins. Broken components should be returned to the lab technician for safe disposal.

9. You must not remove test equipment, test leads or power cables from any lab without permission.

10. Eating, smoking and drinking in the laboratories are forbidden.

11. The use of mobile phones during laboratory sessions is forbidden.

12. The use of email or messaging software for personal communications during laboratory sessions is forbidden.

13. Playing computer games in laboratories is forbidden.

Specific Safety Rules for Laboratories:

1. You must not damage or tamper with the equipment or leads.

2. You should inspect laboratory equipment for visible damage before using it. If there is a problem with a piece of equipment, report it to the technician or lecturer. DONOT return equipment to a storage area

3. You should not work on circuits where the supply voltage exceeds 40 volts without very specific approval from your lab supervisor. If you need to work on such circuits, you should contact your supervisor for approval and instruction on how to do this safely before commencing the work.

4. Always use an appropriate stand for holding your soldering iron.

5. Turn off your soldering iron if it is unlikely to be used for more than 10 minutes.

6. Never leave a hot soldering iron unattended.

7. Never touch a soldering iron element or bit unless the iron has been disconnected from the mains and has had adequate time to cool down.

8. Never strip insulation from a wire with your teeth or a knife, always use an appropriate wire stripping tool.

9. Shield wire with your hands when cutting it with a pliers to prevent bits of wire flying about the bench.

INDEX

PART-A

PART-B

PART-A

1. INTRODUCTION TO MASM

EDITOR

An editor is a program, which allows you to create a file containing the assembly language statements for your program. As you type in your program, the editor stores the ASCII codes for the letters and numbers in successive RAM locations. When you have typed in all of your programs, you then save the file on a floppy of hard disk. This file is called source file. The next step is to process the source file with an assembler. In the MASM /TASM assembler, you should give your source file name the extension, .ASM

ASSEMBLER

An assembler program is used to translate the assembly language mnemonics for instructions to the corresponding binary codes. When you run the assembler, it reads the source file of your program from the disk, where you saved it after editing on the first pass through the source program the assembler determines the displacement of named data items, the offset of labels and pails this information in a symbol table. On the second pass through the source program, the assembler produces the binary code for each instruction and inserts the offset etc that is calculated during the first pass. The assembler generates two files on floppy or hard disk. The first file called the object file is given the extension. OBJ. The object file contains the binary codes for the instructions and information about the addresses of the instructions. The second file generated by the assembler is called assembler list file. The list file contains your assembly language statements, the binary codes for each instructions and the offset for each instruction. In MASM/TASM assembler, MASM/TASM source file name ASM is used to assemble the file. Edit source file name LST is used to view the list file, which is generated, when you assemble the file.

LINKER

A linker is a program used to join several object files into one large object file and convert to an **exe** file. The linker produces a link file, which contains the binary codes for all the combined modules. The linker however doesn't assign absolute addresses to the program, it assigns is said to

be reloadable because it can be put anywhere in memory to be run. In MASM/TASM, LINK/TLIN8K source filename is used to link the file.

DEBUGGER

A debugger is a program which allows you to load your object code program into system memory, execute the program and troubleshoot are debug it the debugger allows you to look at the contents of registers and memory locations after your program runs. It allows you to change the contents of register and memory locations after your program runs. It allows you to change the contents of register and memory locations and return the program. A debugger also allows you to set a break point at any point in the program. If you inset a breakpoint the debugger will run the program up to the instruction where the breakpoint is set and stop execution. You can then examine register and memory contents to see whether the results are correct at that point. In MASM/TASM, td filename is issued to debug the file.

DEBUGGER FUNCTIONS:

1. Debugger allows looking at the contents of registers and memory locations.
2. We can extend 8-bit register to 16-bit register which the help of extended register option.
3. Debugger allows setting breakpoints at any point with the program.
4. The debugger will run the program up to the instruction where the breakpoint is set and then stop execution of program. At this point, we can examine registry and memory contents at that point.
5. With the help of dump we can view register contents.
6. We can trace the program step by step with the help of F7.
7. We can execute the program completely at a time using F8

The DOS -Debugger:

The DOS "Debug" program is an example of simple debugger that comes with MS-DOS. Hence it is available on any PC .it was initially designed to give the user the capability to trace logical errors in executable file.

Below, are summarized the basic DOS - Debugger commands

COMMAND	SYNTAX
Assemble	A [address]
Compare	C range address
Dump	D [range]
Enter	E address [list]
Fill	F range list
Go	G [=address] [addresses]
Hex	H value1 value2
Input	I port
Load	L[address] [drive][first sector][number]
Move	M range address
Name	N[pathname][argument list]
Output	O port byte
Proceed	P [=address][number]
Quit	Q
Register	R[register]
Search	S range list
Trace	T [=address][value]
Unassembled	u [range]
Write	W[address][drive][first sector][number]

MS-MASM:

Microsoft's Macro Assembler (MASM) is an integrated software package Written by Microsoft Corporation for professional software developers. It consists of an editor, an assembler, a linker and a debugger (Code View). The programmer's workbench combines these four parts into a user-friendly programming environment with built in on line help. The following are the steps used if you are to run MASM from DOS

MICROPROCESSOR LAB EXECUTION PROCEDURE

STEP1: **Opening the DOS prompt**

Click **start** menu button and click on **Run** and then type *cmd at* command prompt immediately DOS window will be appeared

STEP2: **Checking the masm installation**

To know MASAM is installed or not simply type **masm** at the command prompt upon that it replies masm version vendor (Microsoft), etc... If you get any error there is no masm in that PC

STEP3: Directory changing (create a folder with your branch and no in D drive)

Change the current directory to your won directory suppose your folder in **D** drive type the following commands to change the directory at command prompt type **D:** hit enter now you are in **D drive** type **cd folder name** hit the enter

Example: D cd ece10

Now we are in folder cse10

STEP4: **writing the program**

At the command prompt type the **edit programname.asm**

Example. Edit add.asm

Immediately editor window will open and there you have to write the program. Type the program in that window after completion save the Program, to save the program Go to file opt in the menu bar and select save opt now your code is ready to Assemble.

STEP5: A**ssembling, Linking and executing the program**

Go to *file* opt click *exit* opt now DOS prompt will be displayed to assemble the program type the following commands at the DOS prompt

Masm Program Name, Program Name, Program Name, Program Name hit the enter

Example: Masm add, add, add, add enter

OR

Example: Masm add.asm

If there are any errors in the program assembler reports all of them at the command prompt with line no's, if there are now bugs your ready to link the program. To link the program type the following line at command prompt Link program name,,,,, (5 commas)

Example: Link add,,,,,

OR

Example: link add.obj

After linking you are ready to execute the program. To execute the program type the following command

Debug program name.exe hit the enter

Example: Debug add.exe

Now you entered into the execution part of the program here you have to execute the program instruction by instruction (debugging) first of all press the *r* key(register) hit the enter key it'll displays all the registers and their initial values in HEXDECIMAL note down the values of all the register which are used in the program. To execute the next instruction press **t** key (TRACE) hit the enter it'll execute that instruction and displays the contents of all the register. You have to do this until you reach the last instruction of the program. After execution you have to observe the results (in memory or registers based on what you have written in the program).

STEP6: Copying list file (common for all programs):

A list file contains your code starting address and end address along with your program .For every program assembler generates a list file at your folder, programname.lst (ex. Add.lst) you should copy this to your lab observation Opening a list file

Edit programname.lst

Example. Edit add.lst

EXPERIMENT NO.2

16 BIT ARITHMETIC OPERATIONS

AIM: Write an ALP to 8086 to perform 16-bit arithmetic operations in various Addressing Modes

TOOLS: PC installed with MASM

<u>**ALGORITHM:**</u>

Step I	**:**	Initialize the Data segment memory.
Step II	**:**	Initialize the Extra segment memory.
Step III	**:**	Load the first number into AX register.
Step IV	**:**	Add two numbers.
Step V	**:**	Store the result in Extra segment.
Step VI	**:**	Terminate the program
Step VII	**:**	Stop.

FLOW CHART:
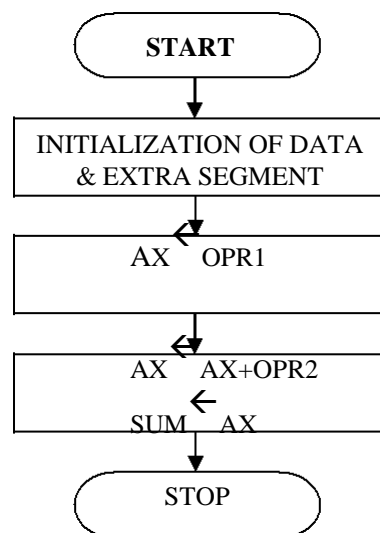
PROGRAM:

(A) 16-bit addition using different addressing modes

ASSUME CS: CODE, DS: DATA, ES: EXTRA

DATA SEGMENT

OPR1 DW 5169H

OPR2 DW 1000H

DATA ENDS

EXTRA SEGMENT

SUM DW ?

EXTRA ENDS CODE

SEGMENT START:

MOV AX, DATA

MOV DS, AX   ; REGISTER ADDRESING MODE

MOV AX, OPR1   ; DIRECT ADDRESSING MODE

ADD AX, OPR2   ; DIRECT ADDRESSING MODE

MOV SUM, AX   ; DIRECT ADDRESSING MODE

INT 03H

CODE ENDS

END START

END

(B) 16-bit subtraction using different addressing modes

ALGORITHM:

Step I	:	Initialize the data & extra segment memory.
Step II	:	Load the first number into AX register.
Step IV	:	Sub AX from OPR2.
Step V	:	Store result in extra segment
Step VI	:	verify the result.
Step VII	:	Stop.

FLOW CHART:

```
                    ┌─────────────┐
                    │    START    │
                    └──────┬──────┘
                           │
              ┌────────────▼────────────┐
              │  INITIALIZATION OF DATA │
              │        SEGMENT          │
              └────────────┬────────────┘
                           │
              ┌────────────▼────────────┐
              │      AX ← OPR1           │
              └────────────┬────────────┘
                           │
              ┌────────────▼────────────┐
              │     AX ← AX-OPR2         │
              │      DIFF ← AX           │
              └────────────┬────────────┘
                           │
                    ┌──────▼──────┐
                    │    STOP     │
                    └─────────────┘
```

PROGRAM:

ASSUME CS:CODE, DS: DATA,ES:EXTRA

        DATA SEGMENT

                OPR1 DW 5169H

                OPR2 DW 1000H

        DATA ENDS

        EXTRA SEGMENT

                 DIFF DW ?

        EXTRA ENDS

        CODE SEGMENT

        START: MOV AX, DATA

                MOV DS, AX   ; REGISTER ADDRESIING MODE

                MOV AX, EXTRA

                MOV ES, AX   ; REGISTER ADDRESIING MODE

                MOV BX, OFFSET OPR1   ; DIRECT ADDRESSING MODE

                MOV AX, [BX]   ; BASE ADDRESSING MODE/

                SUB AX, OPR2   ; DIRECT ADDRESSING MODE

                MOV DIFF, AX   ; DIRECT ADDRESSING MODE

                INT 03H

        CODE ENDS

        END START

        END

(C)16-bit Multiplication using different addressing modes

ALGORITHM:

Step I	:	Initialize the data & extra segment memory.
Step II	:	Load the first number into AX register.
Step III	:	Load the second number into BX register.
Step IV	:	Multiply AX with BX.
Step V	:	store lower word in accumulator into extra segment.
Step VI	:	Store Upper word in DX register into extra segment
Step VII	:	Verify the result.
Step VIII	:	Stop.
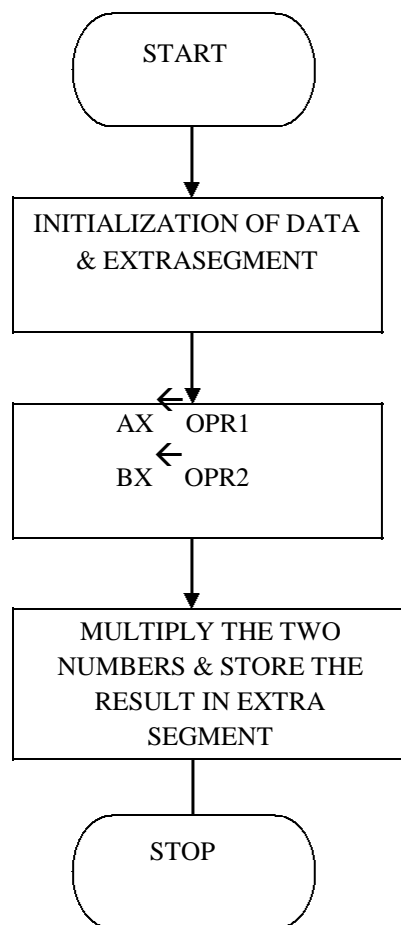
FLOW CHART:

```
                    ┌───────────────┐
                   (     START      )
                    └───────┬───────┘
                            │
                            ▼
            ┌───────────────────────────┐
            │  INITIALIZATION OF DATA   │
            │     & EXTRASEGMENT        │
            └───────────┬───────────────┘
                        │
                        ▼
            ┌───────────────────────────┐
            │      AX ← OPR1            │
            │      BX ← OPR2            │
            └───────────┬───────────────┘
                        │
                        ▼
            ┌───────────────────────────┐
            │    MULTIPLY THE TWO       │
            │  NUMBERS & STORE THE      │
            │    RESULT IN EXTRA        │
            │       SEGMENT             │
            └───────────┬───────────────┘
                        │
                        ▼
                 ┌─────────────┐
                (     STOP      )
                 └─────────────┘
```

PROGRAM:

ASSUME CS: CODE, DS: DATA, ES: EXTRA

DATA SEGMENT

    OPR1 DW 5169H

    OPR2 DW 1000H

DATA ENDS

EXTRA SEGMENT

    RES DW 2 DUP(0)

    EXTRA ENDS

CODE SEGMENT

    START:MOV AX,DATA

        MOV DS,AX   ; REGISTER ADDRESIING MODE

        MOV AX,EXTRA

        MOV ES, AX   ; REGISTER ADDRESIING MODE

        MOV SI,OFFSET OPR1

        MOV AX,[SI]   ; INDEXED ADDRESSING MODE

        MOV BX,OPR2   ; DIRECT ADDRESSING MODE

        MUL BX   ; REGISTER ADDRESSING MODE

        MOV RES, AX   ; DIRECT ADDRESSING MODE

        MOV RES+2, DX   ; DIRECT ADDRESSING MODE

        INT 03H

        CODE ENDS

        END START

        END

(D)16-bit Division using different addressing modes

ALGORITHM:

Step I	:	Initialize the data & extra segment memory.
Step II	:	Load the first number into DX:AX register pair.
Step III	:	Load the second number into BX register.
Step IV	:	Divide DX:AX pair by BX.
Step V	:	store the Quotient in AX register into extra segment.
Step VI	:	Store the reminder in DX register into extra segment.
Step VII	:	Verify the result.
Step VIII	:	Stop.

FLOW CHART:

```
                    ┌─────────────────────┐
                    │        START        │
                    └─────────────────────┘
                               │
                               ▼
        ┌──────────────────────────────────────────┐
        │   INITIALIZATION OF DATA &EXTRA SEGMENT   │
        │                                          │
        └──────────────────────────────────────────┘
                               │
                               ▼
        ┌──────────────────────────────────────────┐
        │     LOAD THE NUMBERS INTO RESPECTIVE      │
        │                REGISTERS                  │
        │                                          │
        │        DIVISION OF DX:AX BY BX            │
        └──────────────────────────────────────────┘
                               │
                               ▼
        ┌──────────────────────────────────────────┐
        │  STORE THE RESULT INTO THE EXTRA SEGMENT  │
        │                                          │
        └──────────────────────────────────────────┘
                               │
                               ▼
                    ┌─────────────────────┐
                    │        STOP         │
                    └─────────────────────┘
```
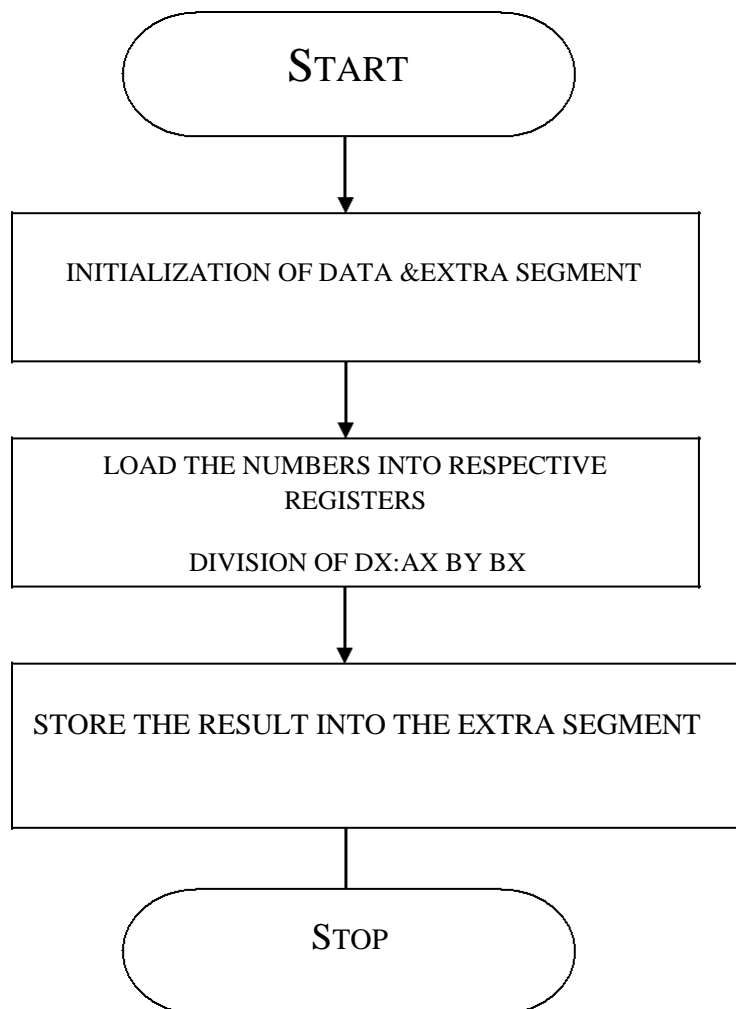
PROGRAM:

ASSUME CS: CODE, DS:DATA, ES:EXTRA

DATA SEGMENT

        OPR1 DD 74105169H

        OPR2 DW 7875H

DATA ENDS
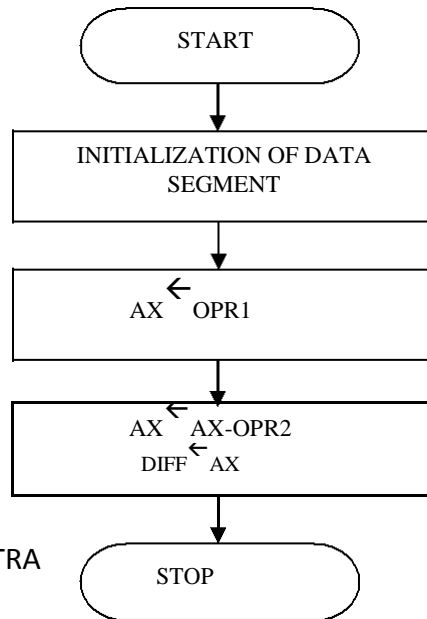
 EXTRA SEGMENT

        DIVQ DW ?

        DIVR DW ?

EXTRA ENDS

CODE SEGMENT

        START:MOV AX, DATA

        MOV DS, AX   ; REGISTER ADDRESIING MODE

        MOV AX, EXTRA

        MOV ES, AX   ; REGISTER ADDRESIING MODE

        MOV SI, OFFSET OPR1

        MOV AX, [SI]   ; INDEXED ADDRESSING MODE/

        MOV DX, [SI+2]   ; INDEXED ADDRESSING MODE

        MOV BX, OPR2   ; DIRECT ADDRESSING MODE

        DIV BX   ; REGISTER ADDRESSING MODE

        MOV   DIVQ , AX

        MOV   DIVR , DX

        INT 03H

CODE ENDS

END START

END

Result:

UNSIGNED NUMBERS

INPUT: OPR1 =

OPR2 =

OUTPUT: ALL RESULTS ARE STORED IN EXTRA SEGMENT (ES)

SUM =

DIFF=

MUL=

MUL+2=

DIVQ=

DIVR=

Exercise Questions:

1) Write an assembly language program for the expression ax+b

2) Write an assembly language program for the squaring of 16 bit Hexa Decimal number.

3) Write an assembly language program for the factorial of 8 bit Hexadecimal number.

Viva Question:

1) What is meant by microprocessor?

2) What is meant by accumulator?

3) What is meant by assembler directive?

4) What are segment Registers?

5) What is the use of INT 03H instruction ?

OBSERVATION:

EXPERIMENT NO.3

SORTING AN ARRAY FOR 8086

AIM: Write and execute an ALP to 8086 processor to sort the given 16-bit numbers in Ascending and Descending order.

TOOLS: PC installed with MASM 6.11

ALGORITHM:

Step I: Initialize the data segment memory.

Step II : Initialize the number of elements counter

Step III : Initialize the comparisons counter..

Step IV: Load the numbers into respective registers

Step V: Compare the elements. If first element < second element goto step **VII** Else go to next step.

Step VI: Swap the numbers in the memory..

Step VII: Increment memory pointer & Decrement the comparison counter.

Step VIII: Is count = 0 ? if yes go to next step else go to **step IV**.

Step IX: decrement the element counter.

Step X: Is count not 0 ? go **Step III** else go to next step

Step IX: Stop & terminate the program**.**

FLOW CHART:

```
                          ╭─────────────╮
                          │    START    │
                          ╰─────────────╯
                                 │
                                 ▼
                        ┌─────────────────┐
                        │ INITIALIZATION OF│
                        │  DATA SEGMENT   │
                        └─────────────────┘
                                 │
                                 ▼
                        ┌─────────────────┐
                        │   DX ← COUNT-1  │
                        └─────────────────┘
                                 │
                                 ▼
                        ┌─────────────────┐
                ┌──────▶│  BACK: CX ← DX  │
                │       └─────────────────┘
                │                │
                │                ▼
                │       ┌─────────────────┐
                │       │   SI ← OFFSET   │
                │       │ ADDRESS OF LIST │
                │       └─────────────────┘
                │                │
                │                ▼
                │       ┌─────────────────┐
                │  ┌───▶│    AX ← [SI]    │
                │  │    └─────────────────┘
                │  │             │
                │  │             ▼
                │  │          ╱IF╲         ┌──────┐
                │  │        ╱ AX <  ╲─────▶│ TRUE │
                │  │        ╲ [SI+2] ╱     └──────┘
                │  │          ╲  ╱             │
                │  │         FALSE            │
                │  │            │             │
                │  │            ▼             │
                │  │    ┌─────────────────┐   │
                │  │    │    EXCHANGE     │   │
                │  │    │   [SI] &[SI+2]  │   │
                │  │    └─────────────────┘   │
                │  │            │             │
                │  │            ▼             │
                │  │    ┌─────────────────┐   │
                │  │    │ INCREMENT SI BY 2│◀──┘
                │  │    └─────────────────┘
                │  │            │
                │  │            ▼
                │  │  FALSE   ╱IF╲
                │  └─────────╱ CX=0╲
                │            ╲    ╱
                │             ╲  ╱
                │             TRUE
                │               │
                │               ▼
                │       ┌─────────────────┐
                │       │  DECREMENT DX   │
                │       └─────────────────┘
                │               │
                │               ▼
                │   FALSE     ╱IF╲
                └───────────╱ DX=0╲
                            ╲    ╱
                             ╲  ╱
                             TRUE
                               │
                               ▼
                          ╭─────────────╮
                          │    STOP     │
                          ╰─────────────╯
```

PROGRAM:

ASCENDING ORDER

ASSUME CS: CODE, DS: DATA

DATA SEGMENT

      LIST DW 0125H,0144H,3001H,0003H,0002H

      COUNT EQU 05H

DATA ENDS

CODE SEGMENT

START:MOV AX,DATA

      MOV DS,AX

      MOV DX,COUNT-1

BACK: MOV CX,DX

      MOV SI, OFFSET LIST

AGAIN: MOV AX,[SI]

      CMP AX,[SI+2]

      JC GO

      XCHG AX,[SI+2]

      XCHG AX,[SI]

    GO:INC SI

      INC SI

      LOOP AGAIN

      DEC DX

      JNZ BACK

      INT 03H

CODE ENDS

END START

END

Result:

INPUT: (DS: 0000H) = 25H,01H,44H,01H,01H,30H,03H,00H,02H,00H

OUTPUT: (DS: 0000H) =

DESCENDING ORDER

ALGORITHM:

Step I: Initialize the data segment memory.

Step II : Initialize the number of elements counter

Step III : Initialize the comparisons counter..

Step IV: Load the numbers into respective registers

Step V: Compare the elements. If first element > second element go to step **VII**

   Else go to next step.

Step VI: Swap the numbers in the memory.

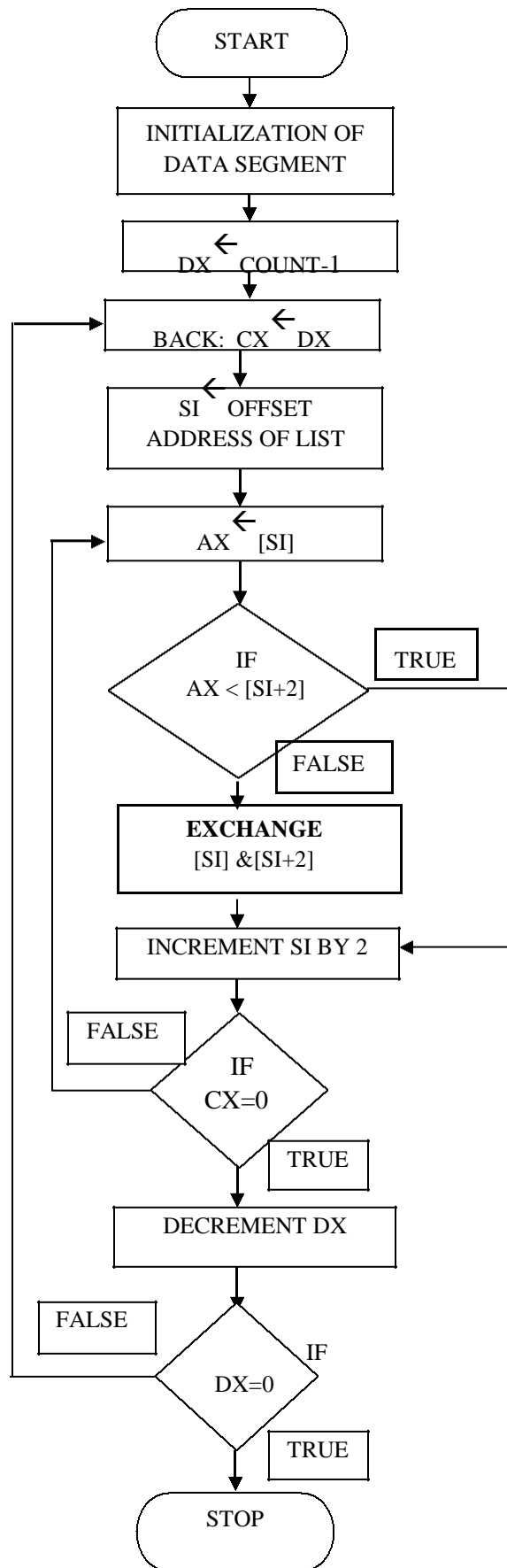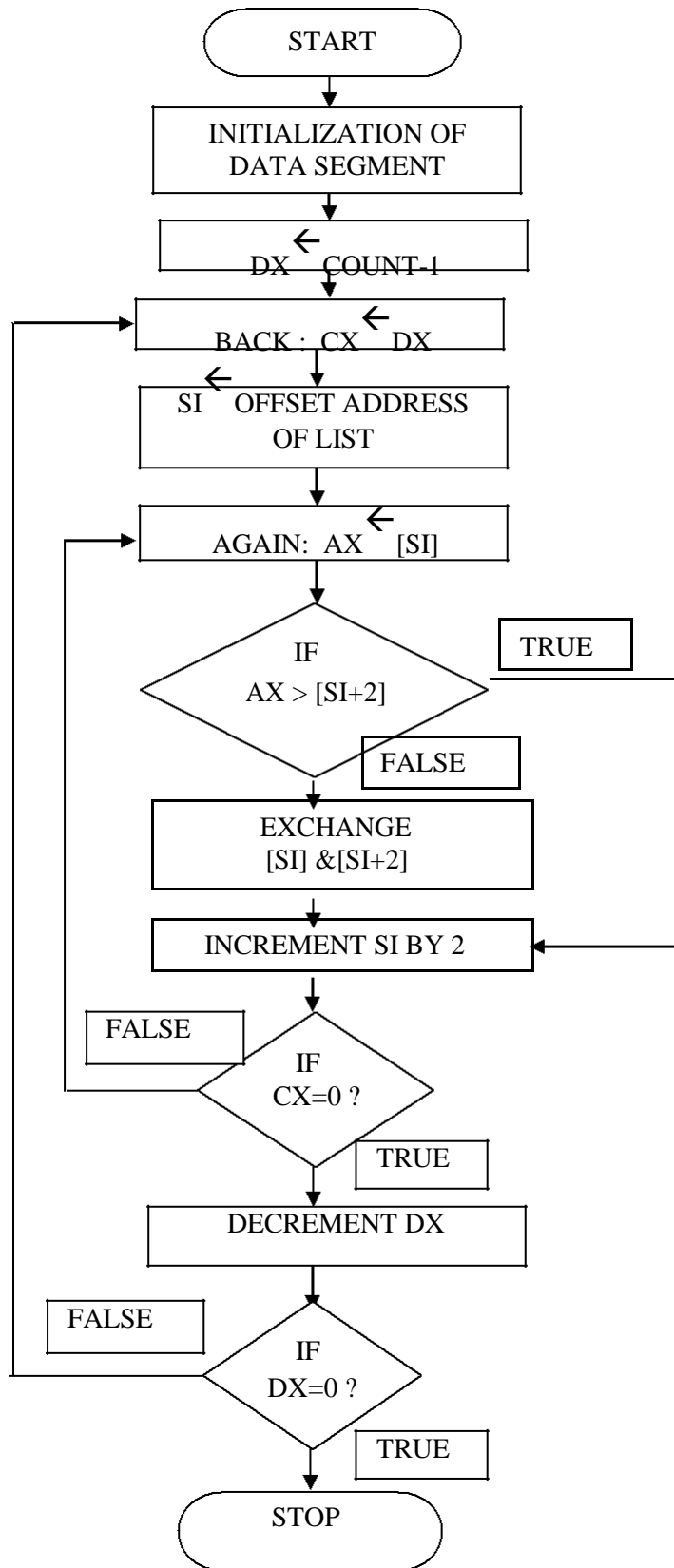**Step VII:** Increment memory pointer & Decrement the comparison counter.

Step VIII: Is count = 0? If yes go to next step else go to **step IV**.

Step IX: decrement the element counter.

 Step X: Is count not 0? go **Step III** else go to next step **Step IX:** Stop & terminate the program**.**

FLOW CHART:

```
                    ┌──────────┐
                    │  START   │
                    └────┬─────┘
                         ▼
              ┌────────────────────┐
              │ INITIALIZATION OF  │
              │   DATA SEGMENT     │
              └─────────┬──────────┘
                        ▼
              ┌────────────────────┐
              │  DX ← COUNT-1      │
              └─────────┬──────────┘
                        ▼
              ┌────────────────────┐
        ┌────▶│  BACK : CX ← DX    │
        │     └─────────┬──────────┘
        │               ▼
        │     ┌────────────────────┐
        │     │ SI ← OFFSET ADDRESS│
        │     │      OF LIST       │
        │     └─────────┬──────────┘
        │               ▼
        │     ┌────────────────────┐
        │ ┌──▶│ AGAIN: AX ← [SI]   │
        │ │   └─────────┬──────────┘
        │ │             ▼
        │ │           ◇ IF ◇        ──── TRUE ────┐
        │ │        AX > [SI+2]                      │
        │ │             │ FALSE                     │
        │ │             ▼                           │
        │ │   ┌────────────────────┐                │
        │ │   │    EXCHANGE        │                │
        │ │   │   [SI] &[SI+2]     │                │
        │ │   └─────────┬──────────┘                │
        │ │             ▼                           │
        │ │   ┌────────────────────┐◀───────────────┘
        │ │   │ INCREMENT SI BY 2  │
        │ │   └─────────┬──────────┘
        │ │             ▼
        │ │  FALSE    ◇ IF ◇
        │ └──────────  CX=0 ?
        │               │ TRUE
        │               ▼
        │     ┌────────────────────┐
        │     │   DECREMENT DX     │
        │     └─────────┬──────────┘
        │               ▼
        │  FALSE      ◇ IF ◇
        └──────────   DX=0 ?
                        │ TRUE
                        ▼
                   ┌──────────┐
                   │   STOP   │
                   └──────────┘
```

DESCENDING ORDER

PROGRAM:

ASSUME CS: CODE, DS:DATA

DATA SEGMENT

      LIST DW 0125H,0144H,3001H,0003H,0002H

      COUNT EQU 05H

DATA ENDS

CODE SEGMENT

      START:MOV AX,DATA

          MOV DS,AX

          MOV DX,COUNT-1

      BACK:MOV CX,DX

          MOV SI,OFFSET LIST

      AGAIN:MOV AX,[SI]

          CMP AX,[SI+2]

          JAE GO

          XCHG AX,[SI+2]

          XCHG AX,[SI]

      GO:INC SI

          INC SI

          LOOP AGAIN

          DEC DX

          JNZ BACK

          INT 03H

CODE ENDS

END START

END

Result:

INPUT: (DS: 0000H) = 25H,01H,44H,01H,01H,30H,03H,00H,02H,00H

OUTPUT: (DS: 0000H) =

Exercise Questions:

1) Write an assembly language program for finding the maximum number in array of five 16 bit hexadecimal numbers?

2) Write an assembly language program for finding the minimum number in array of five 16 bit hexadecimal numbers?

Viva Questions:

1) What is the use of SI Register?
2) What is the use of XCHG instruction?
3) What is the use of CX Register ?
4) What is the use of JNZ instruction?

OBSERVATION:

EXPERIMENT NO: 4

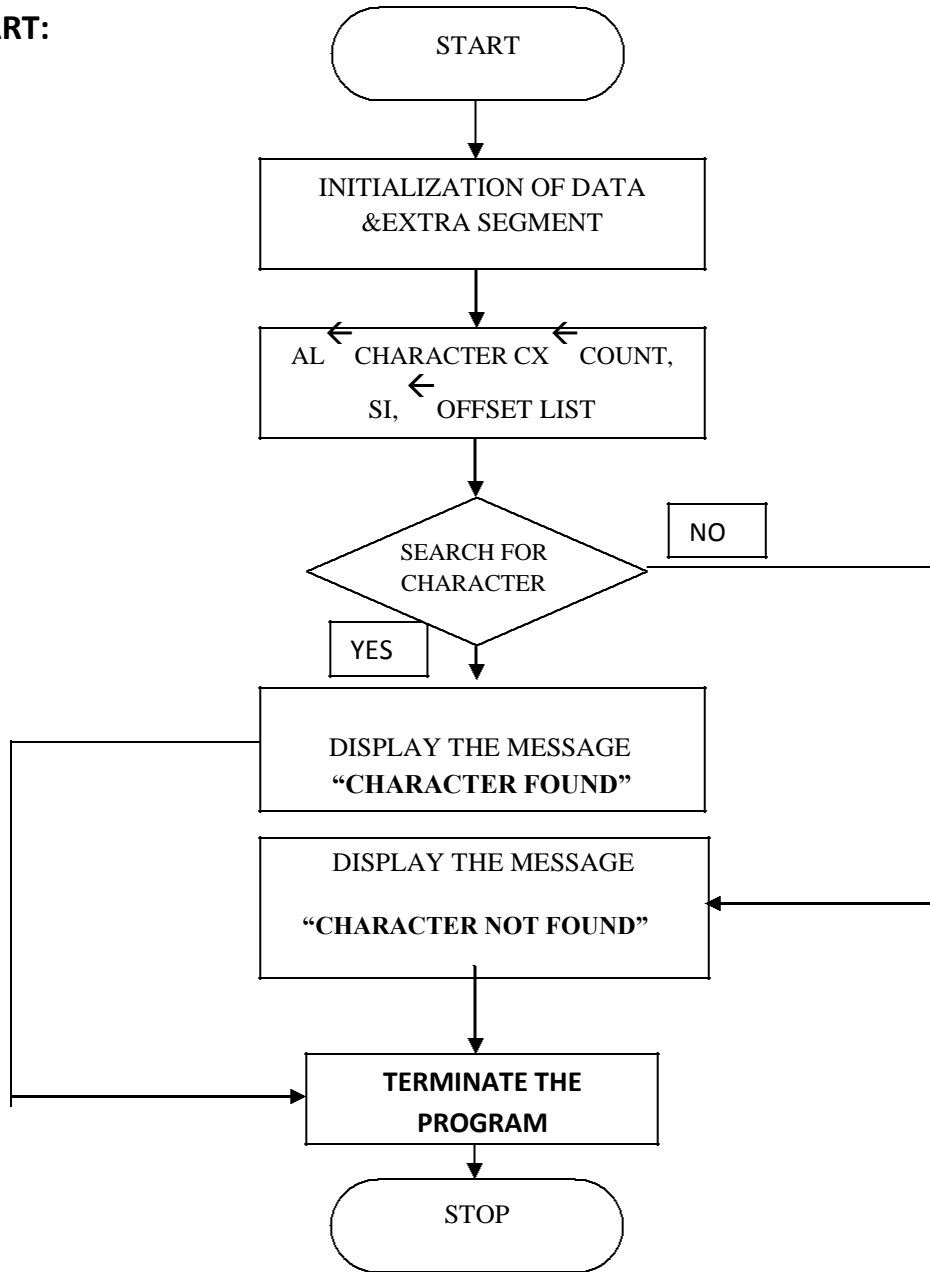## SEARCH ING FOR CHARACTER IN A STRING

AIM: Write an ALP for searching for a number or character in a string for 8086.

TOOLS: PC installed with MASM 6.11

ALGORITHM:

Step I   : Initialize the Data segment (DS) & Extra segment(ES)

Step II   : Load the offset address of the string into SI .

Step III   : Load the number of elements in the string into CX register

Step IV   : Move the character to be searched into the AL register

Step V   : Scan for the character in ES. If the character is not found go to step **VII** else go
               to next step.

 Step VI   : Display the message that character found and go to step **VIII**

Step VII   : Display the message that character not found

Step VII   : Stop.& Terminate the program

FLOW CHART:

```
                    ┌─────────────┐
                    │    START    │
                    └──────┬──────┘
                           │
                           ▼
              ┌────────────────────────┐
              │  INITIALIZATION OF DATA│
              │     &EXTRA SEGMENT     │
              └───────────┬────────────┘
                          │
                          ▼
              ┌────────────────────────┐
              │ AL ← CHARACTER CX ← COUNT, │
              │   SI, ← OFFSET LIST     │
              └───────────┬────────────┘
                          │
                          ▼
                      ╱───────╲            ┌─────┐
                     ╱ SEARCH  ╲───────────│ NO  │
                     ╲  FOR     ╱          └─────┘
                      ╲CHARACTER╱
                       ╲───────╱
              ┌─────┐      │
              │ YES │      │
              └─────┘      ▼
              ┌────────────────────────┐
              │  DISPLAY THE MESSAGE   │
              │  "CHARACTER FOUND"     │
              └───────────┬────────────┘
                          │
              ┌────────────────────────┐
              │  DISPLAY THE MESSAGE   │
              │  "CHARACTER NOT FOUND" │
              └───────────┬────────────┘
                          │
                          ▼
              ┌────────────────────────┐
              │     TERMINATE THE      │
              │       PROGRAM          │
              └───────────┬────────────┘
                          │
                          ▼
                    ┌─────────────┐
                    │    STOP     │
                    └─────────────┘
```

Program:

```
ASSUME CS: CODE, DS: DATA
              DATA SEGMENT

                      STRING DB 'MRCET$'

                      SLEN EQU ($-STRING)

                      CHAR DB 'E'

                      MSG1 DB 'THE CHARACTER IS FOUND$'

                      MSG2 DB 'THE CHARACTER IS NOT FOUND$'

               DATA ENDS

               CODE SEGMENT

               START: MOV AX, DATA

                      MOV DS, AX

                      MOV ES, AX

                      LEA SI, STRING

                      MOV CX, LEN

                      MOV AL, CHAR

                      CLD

               REPNE SCASB

                      JNZ EXIT

                      LEA DX, MSG1

                      MOV AH, 09H

                      INT 21H

                      JMP GOTOEND

               EXIT: LEA DX, MSG2

                      MOV AH, 09H

                      INT 21H

        GOTOEND: MOV AH, 4CH

                      INT 21H

               CODE ENDS

               END START

               END
```

Exercise Questions:

1) Write an assembly language program for the password verification?

Viva Questions:

1) What is the use of SCASB Register?

2) What is the use of REPNE instruction?

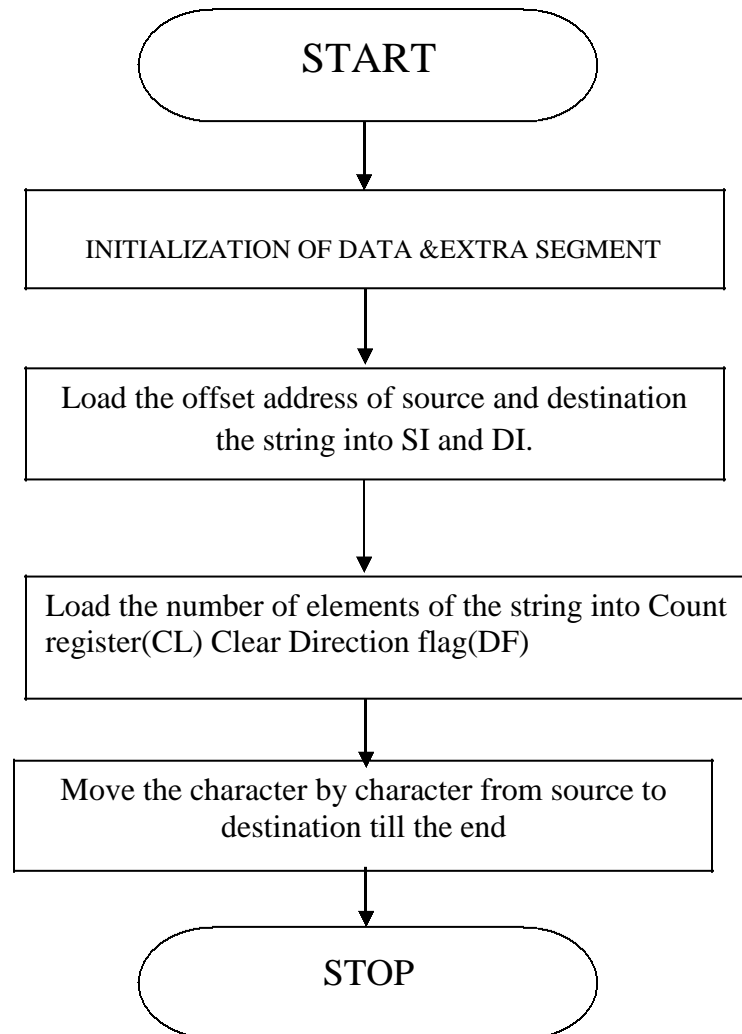3) What is the relation of CX Register with REPNE?

OBSERVATION:

EXPERIMENT NO.5

STRING MANIPULATIONS FOR 8086

AIM: To write an assembly language program to move the block of data from a source
BLOCK to the specified destination BLOCK.

TOOLS: PC installed with MASM 6.11

A) BLOCK TRANSFER

ALGORITHM:

Step I   : Initialize the Data segment (DS) & Extra segment (ES)

Step II   : Load the offset address of source and destination the string into SI and DI.

Step III   : Load the number of elements of the string into Count register(CL)

Step IV   : Clear Direction flag (DF) to make SI and DI into auto increment mode

Step V   : move the character by character from source to destination till the end

Step VI   : Stop & Terminate the program

FLOW CHART:

```
                    ┌─────────────────┐
                    │     START       │
                    └─────────────────┘
                             │
                             ▼
        ┌──────────────────────────────────────────┐
        │  INITIALIZATION OF DATA &EXTRA SEGMENT    │
        └──────────────────────────────────────────┘
                             │
                             ▼
        ┌──────────────────────────────────────────┐
        │  Load the offset address of source and   │
        │  destination the string into SI and DI.  │
        └──────────────────────────────────────────┘
                             │
                             ▼
        ┌──────────────────────────────────────────┐
        │  Load the number of elements of the      │
        │  string into Count register(CL) Clear    │
        │  Direction flag(DF)                      │
        └──────────────────────────────────────────┘
                             │
                             ▼
        ┌──────────────────────────────────────────┐
        │  Move the character by character from    │
        │  source to destination till the end      │
        └──────────────────────────────────────────┘
                             │
                             ▼
                    ┌─────────────────┐
                    │      STOP       │
                    └─────────────────┘
```

PROGRAM:

        ASSUME CS: CODE, DS: DATA

        DATA SEGMENT

                STRING DB 'MICROPROCESSOR$'

                COUNT EQU ($-STRING)

                ORG 0070H

        DATA ENDS

        EXTRA SEGMENT

                ORG 0010H

```
                STRING1 DB ?

        EXTRA ENDS

        CODE SEGMENT

        START:

                MOV AX,DATA

                MOV DS,AX

                MOV AX, EXTRA

                MOV ES,AX

                MOV SI,OFFSET STRING

                MOV DI,OFFSET STRING1

                MOV CL,COUNT

                CLD

            REP MOVSB

                INT 03H

        CODE ENDS

        END START

        END
```

RESULT:

INPUT: (DS: 0000H) = MICROPROCESSOR

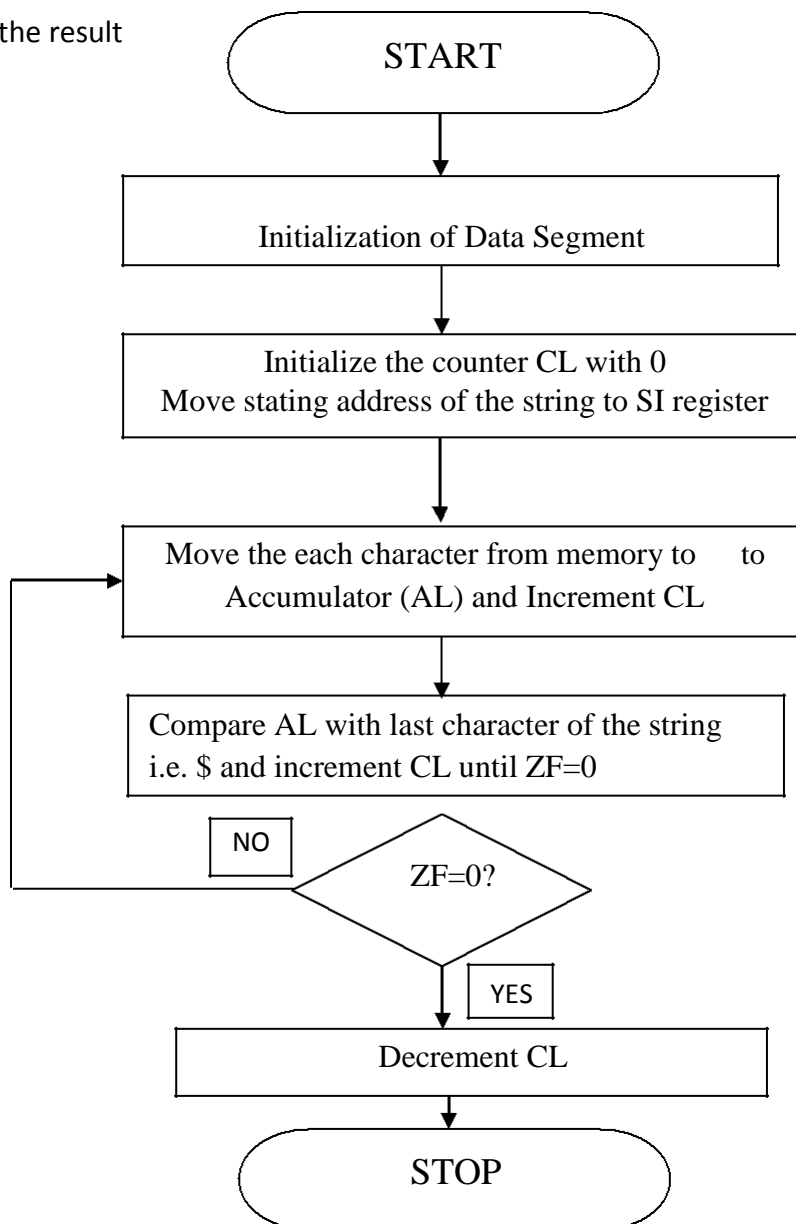OUTPUT: (ES: 0010H) = MICROPROCESSOR

B) REVERSE STRING

AIM: To write an assembly language program to reverse the given string.

TOOLS: PC installed with MASM 6.11

ALGORITHM:

Step I   **:** Initialize the Data segment (DS) & Extra segment (ES)

Step II   **:** Load the offset address of source and destination the string into SI and DI.

Step III   **:** Load the number of elements of the string into Count Register (CX)

Step IV   **:** Add CX to DI to Point to last location of the memory

Step V   **:** move the character by character from source to destination till the end

Step VI   **:** Stop & Terminate the program

FLOW CHART:

```
                        ┌─────────────────┐
                        │     START       │
                        └─────────────────┘
                                 │
                                 ▼
            ┌──────────────────────────────────────────┐
            │  INITIALIZATION OF DATA &EXTRA SEGMENT    │
            └──────────────────────────────────────────┘
                                 │
                                 ▼
            ┌──────────────────────────────────────────┐
            │  Load the offset address of source and   │
            │  destination the string into SI and DI.  │
            └──────────────────────────────────────────┘
                                 │
                                 ▼
            ┌──────────────────────────────────────────┐
            │  Load the number of elements of the      │
            │  string into Count register(CX)          │
            │  Add CX to DI to Point to last location  │
            │  of the                                  │
            └──────────────────────────────────────────┘
                                 │
                                 ▼
            ┌──────────────────────────────────────────┐
            │  Move the character by character from    │
            │  source to destination till the end      │
            └──────────────────────────────────────────┘
                                 │
                                 ▼
                        ┌─────────────────┐
                        │     STOP        │
                        └─────────────────┘
```
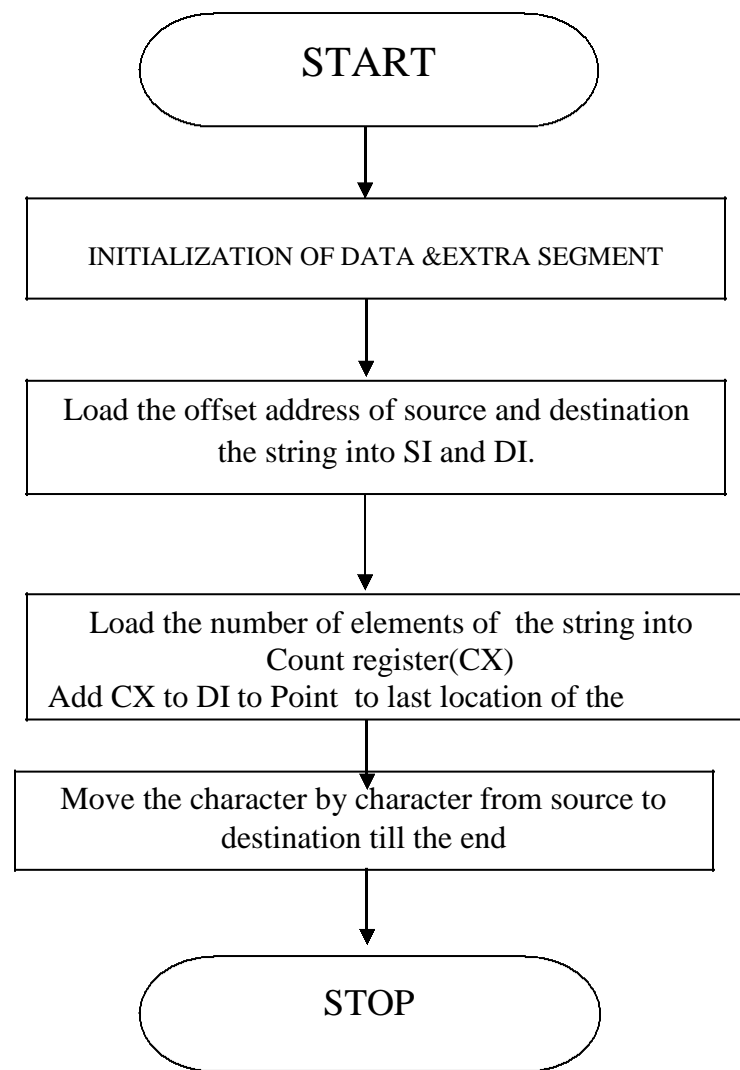
PROGRAM:

ASSUME CS: CODE, DS: DATA ,ES:EXTRA

DATA SEGMENT

STRING1 DB 'MICROPROCESSOR$'

STRLEN EQU ($-STRING1)

DATA ENDS

 EXTRA SEGMENT

 STRING2 DB ?

 EXTRA ENDS

CODE SEGMENT

START: MOV AX, DATA

      MOV DS, AX

      MOV AX, EXTRA

      MOV ES, AX

      MOV SI, OFFSET STRING1

      MOV DI, OFFSET STRING2

      MOV CX, STRLEN-1

      ADD DI, CX

      MOV DL,'$'

     MOV ES:[DI],DL

AGAIN: DEC DI

      MOV AL,DS:[SI]

      MOV ES:[DI],AL

      INC SI

      DEC CX

      JNZ AGAIN

      INT 3H

RESULT:

 INPUT: ' MICROPROCESSOR'

OUTPUT: 'ROSSECORPORCIM'

C) LENGTH OF THE STRING

AIM: To write an assembly language program to find the length of the given string.

TOOLS: PC installed with MASM 6.11

ALGORITHM:

Step I   : Initialize the data segment (DS)

Step II   : Initialize the counter CL with 0

Step III   : Move stating address of the string to SI register

Step IV   : Move the each character from memory to to Accumulator (AL)

Step V   : Compare AL with last character of the string i.e $ and increment CL until ZF=0

Step VII : Store the result

Step VIII : Stop.

FLOW CHART:

```
                    ( START )
                        |
        +-----------------------------------+
        |   Initialization of Data Segment  |
        +-----------------------------------+
                        |
        +-----------------------------------+
        |   Initialize the counter CL with 0 |
        |   Move stating address of the string to SI register |
        +-----------------------------------+
                        |
        +-----------------------------------+
        |   Move the each character from memory to    to |
        |      Accumulator (AL) and Increment CL    |
        +-----------------------------------+
                        |
        +-----------------------------------+
        |   Compare AL with last character of the string |
        |   i.e. $ and increment CL until ZF=0 |
        +-----------------------------------+
                        |
            [NO]    < ZF=0? >
                        |
                      [YES]
                        |
        +-----------------------------------+
        |          Decrement CL              |
        +-----------------------------------+
                        |
                    ( STOP )
```

Program:

ASSUME CS:CODE, DS:DATA

DATA SEGMENT

       STRING1 DB 'MICROPROCESSOR AND INTERFACING

       LAB$' SLENGTH DB 0

DATA ENDS

CODE SEGMENT

START: MOV AX, DATA

              MOV DS, AX

              SUB CL, CL

              MOV SI, OFFSET STRING1

              CLD

BACK: LODSB

              INC CL

              CMP AL,'$'

              JNZ BACK

              DEC CL

              MOV SLENGTH, CL

              INT 03H

CODE ENDS

END START

RESULT: INPUT: 'MICROPROCESSOR AND INTERFACING LAB

       OUTPUT:

D) STRING COMPARISON

AIM: Write an ALP to 8086 to compare the given strings.

TOOLS: PC installed with MASM 6.11

ALGORITHM:

Step I   : Initialize the data segment (DS) & extra Segment as per requirement

Step II   : Load the offset address of source and destination of the string into SI and DI.

Step III   : Initialize the counter register CX with length of source string

Step IV   : Clear Direction flag (DF) to make SI and DI into auto increment mode

Step V   : Compare source string with destination string until the characters are not equal
                 or up to last last character

Step VII : If ZF=0 the strings are equal or otherwise the strings are not equal

Step VIII : Stop.

FLOW CHART:

```
                          ┌─────────────┐
                          │    START     │
                          └─────────────┘
                                 │
                                 ▼
        ┌──────────────────────────────────────────────────┐
        │      Initialization of data & extra segment        │
        └──────────────────────────────────────────────────┘
                                 │
                                 ▼
        ┌──────────────────────────────────────────────────┐
        │   Load the offset address of source and destination the │
        │              string into SI and DI.                │
        └──────────────────────────────────────────────────┘
                                 │
                                 ▼
        ┌──────────────────────────────────────────────────┐
        │   Initialize the counter register CX with length of │
        │                 source string                      │
        └──────────────────────────────────────────────────┘
                                 │
                                 ▼
        ┌──────────────────────────────────────────────────┐
        │   Clear Direction flag(DF) to make SI and DI into   │
        │           auto increment mode                      │
        └──────────────────────────────────────────────────┘
                                 │
                                 ▼
        ┌──────────────────────────────────────────────────┐
        │   Compare source string with destination string until │
        │   the characters are not equal or up to last character │
        └──────────────────────────────────────────────────┘
                                 │
                                 ▼
                            ◇ ZF=0 ◇ ──── NO ─────┐
                                 │                 │
                                YES                │
                                 ▼                 │
        ┌──────────────────────────────────────────────────┐   │
        │             The strings are equal                  │   │
        └──────────────────────────────────────────────────┘   │
                                 │                              │
                                 ▼                              │
        ┌──────────────────────────────────────────────────┐   │
        │            The strings are NOT equal              │◄──┘
        └──────────────────────────────────────────────────┘
                                 │
                                 ▼
                          ┌─────────────┐
                          │    START     │
                          └─────────────┘
```

Program:

ASSUME CS: CODE, DS:DATA, ES:EXTRA

DATA SEGMENT

      STRING1 DB 'MRCET'

      STRLEN EQU ($-STRING1)

      SNOTEQUAL DB 'STRINGS ARE UNEQUAL$'

      SEQUAL DB 'STRINGS ARE EQUAL$'

DATA ENDS

EXTRA SEGMENT

      STRING2 DB 'MRCET'

EXTRA ENDS

CODE SEGMENT

    START: MOV AX,DATA

          MOV DS,AX

          MOV AX,EXTRA

          MOV ES,AX

          MOV SI,OFFSET STRING1

          MOV DI,OFFSET STRING2

          CLD

          MOV CX,STRLEN

     REPZ CMPSB

          JZ FORW

          MOV AH, 09H

          MOV DX, OFFSET SNOTEQUAL

          INT 21H

          JMP EXITP

    FORW: MOV AH,09H

          MOV DX, OFFSET SEQUAL

          INT 21H

    EXITP: MOV AH, 4CH

          INT 03H

CODE ENDS

END START

RESULT: INPUT: OUTPUT:

(E) STRING INSERTION

AIM: To Write and execute an Assembly language Program (ALP) to 8086 processor to insert or delete a character/ number from the given string.

TOOLS: PC installed with MASM 6.11

ALGORITHM:

Step I   : Initialize the data segment (DS) & extra segment (ES)

Step II : Load the offset address of source and destination of the string into SI and DI.

Step III : Initialize the counter register CX with length of first part of source string

Step IV : Copy the first part of STRING1 in to STRING3 of extra segment

Step V : Load the offset address of STRING2 in to SI

Step VI : Copy the STRING2 in to STRING3 of extra segment after first string of STRING1

Step VII: Load the new offset address of source of the STRING1 into SI

Step VIII: Copy the second part of STRING1 in to extra segment

Step IX: Stop

FLOW CHART:

```
                          ┌──────────────┐
                          │    START     │
                          └──────┬───────┘
                                 │
          ┌──────────────────────▼──────────────────────┐
          │      Initialization of data & extra segment  │
          └──────────────────────┬──────────────────────┘
                                 │
          ┌──────────────────────▼──────────────────────┐
          │  Load the offset address of source and       │
          │  destination string into SI and DI.          │
          └──────────────────────┬──────────────────────┘
                                 │
          ┌──────────────────────▼──────────────────────┐
          │  Initialize the counter register CX with     │
          │  length of  first part of source string      │
          └──────────────────────┬──────────────────────┘
                                 │
          ┌──────────────────────▼──────────────────────┐
          │  Clear Direction flag (DF) to make SI and    │
          │  DI into auto increment mode                 │
          └──────────────────────┬──────────────────────┘
                                 │
          ┌──────────────────────▼──────────────────────┐
          │  Copy the first part of STRING1 in to        │
          │  STRING3 of extra segment                    │
          └──────────────────────┬──────────────────────┘
                                 │
          ┌──────────────────────▼──────────────────────┐
          │  Load the offset address of STRING2 in to SI │
          └──────────────────────┬──────────────────────┘
                                 │
          ┌──────────────────────▼──────────────────────┐
          │  Copy the STRING2 in to STRING3 of extra     │
          │  segment after first string of STRING1       │
          └──────────────────────┬──────────────────────┘
                                 │
          ┌──────────────────────▼──────────────────────┐
          │  Load the new offset address of source of    │
          │  the STRING1 into SI                          │
          └──────────────────────┬──────────────────────┘
                                 │
          ┌──────────────────────▼──────────────────────┐
          │  Load the new offset address of source of    │
          │  the STRING1 into SI                          │
          └──────────────────────┬──────────────────────┘
                                 │
          ┌──────────────────────▼──────────────────────┐
          │  Copy the second part of STRING1 in to       │
          │  extra segment                               │
          └──────────────────────┬──────────────────────┘
                                 │
                          ┌──────▼───────┐
                          │     STOP     │
                          └──────────────┘
```

Program:

ASSUME CS:CODE,DS:DATA,ES:EXTRA

DATA SEGMENT

      STRING1 DB 'MICROPROCESSOR INTERFACING LAB$'

      STRING2 DB 'AND '

      STRLEN EQU ($-STRING1)

      ORG 0070H

DATA ENDS

EXTRA SEGMENT

      ORG 0010H

      STRING3 DB 38 DUP(0)

EXTRA ENDS

CODE SEGMENT

      START: MOV AX, DATA

            MOV DS, AX

            MOV AX, EXTRA

            MOV ES, AX

            MOV SI, OFFSET STRING1

            MOV DI, OFFSET STRING3

            CLD

            MOV CX, 15

           REP MOVSB

            CLD

            MOV SI, OFFSET STRING2

            MOV CX,4

          REP MOVSB

            MOV SI, OFFSET STRING1

            ADD SI,15

            MOV CX, 15

          REP MOVSB

            INT 3H

CODE ENDS

END START

RESULT:

INPUT: **STRING1:** 'MICROPROCESSOR INTERFACING LAB'

                    STRING2: 'AND '

OUTPUT: **STRING3:** 'MICROPROCESSOR AND INTERFACING LAB'

(F) STRING DELETION

ASSUME CS: CODE, DS:DATA, ES:EXTRA

   DATA SEGMENT

        STRING1 DB 'MICROPROCESSOR AND INTERFACING LAB$'

        ORG 0070

   DATA ENDS

EXTRA SEGMENT

        ORG 0010H

        STRING2 DB 40 DUP (0)

EXTRA ENDS

CODE SEGMENT

START: MOV AX, DATA

                MOV DS, AX

                MOV AX, EXTRA

                MOV ES, AX

                MOV SI, OFFSET STRING1

                MOV DI, OFFSET STRING2

                CLD

                MOV CX, 15

                REP MOVSB

                CLD

              MOV SI, OFFSET STRING1

                ADD SI, 19

MOV CX, 15

REP MOVSB

INT 03H

CODE ENDS

END START

RESULT:

INPUT: **STRING1:** MICROPROCESSOR AND INTERFACING LAB'

OUTPUT: **STRING2:** 'MICROPROCESSOR INTERFACING LAB'

Exercise Questions:

1) Write an assembly language program for the palindrome of a given string?

2) Write an assembly language program for the display of given string?

Viva Questions:

1) What are the string manipulation instructions?
2) What are the repeat instructions?
3) What is the use of DUP instruction?
4) What is the meaning of ORG assembler Directive?

OBSERVATION:

PART-B

INTRODUCTION TO HARDWARE EXPERIMENTS

8086 Programs can also executed by using ADM TK_µP Trainer kits. The Assembly language programs (ALP) can be executed by the following steps

1. UxAsm

2. TKµP

1. UxAsm: It is used to translate the Assembly language program in to Machine language (Hex File).The input file to the UxAsm is .asm and one of the output files is hex file

Procedure for UXASM:

1. Go to start and select UXAsm
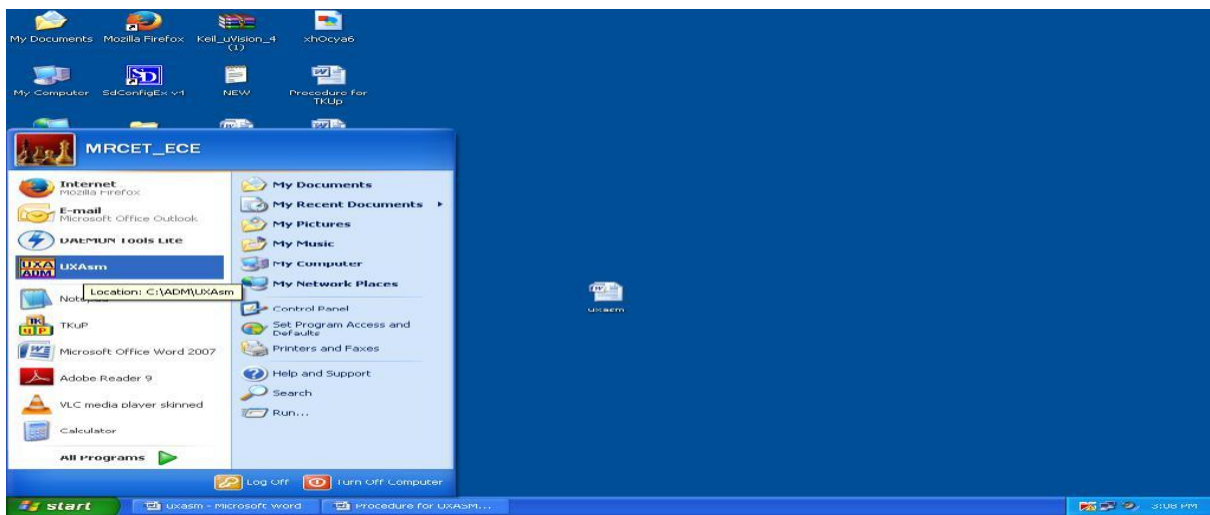
2. Verify the license by observing the following window and click "OK"
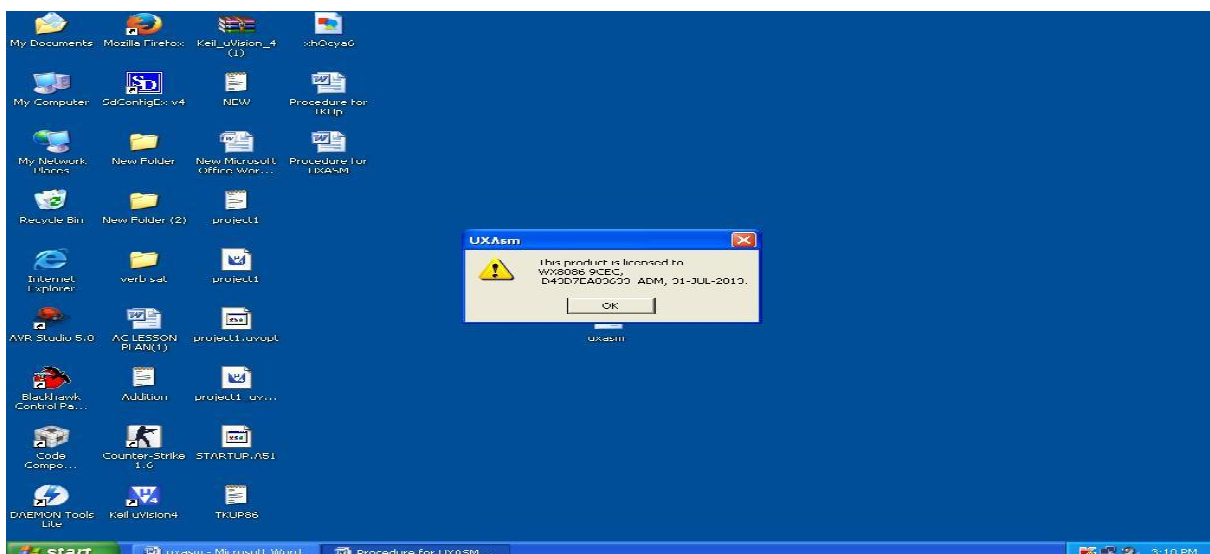
3. Go to file and select "open source" and browse the source file(.ASM file)
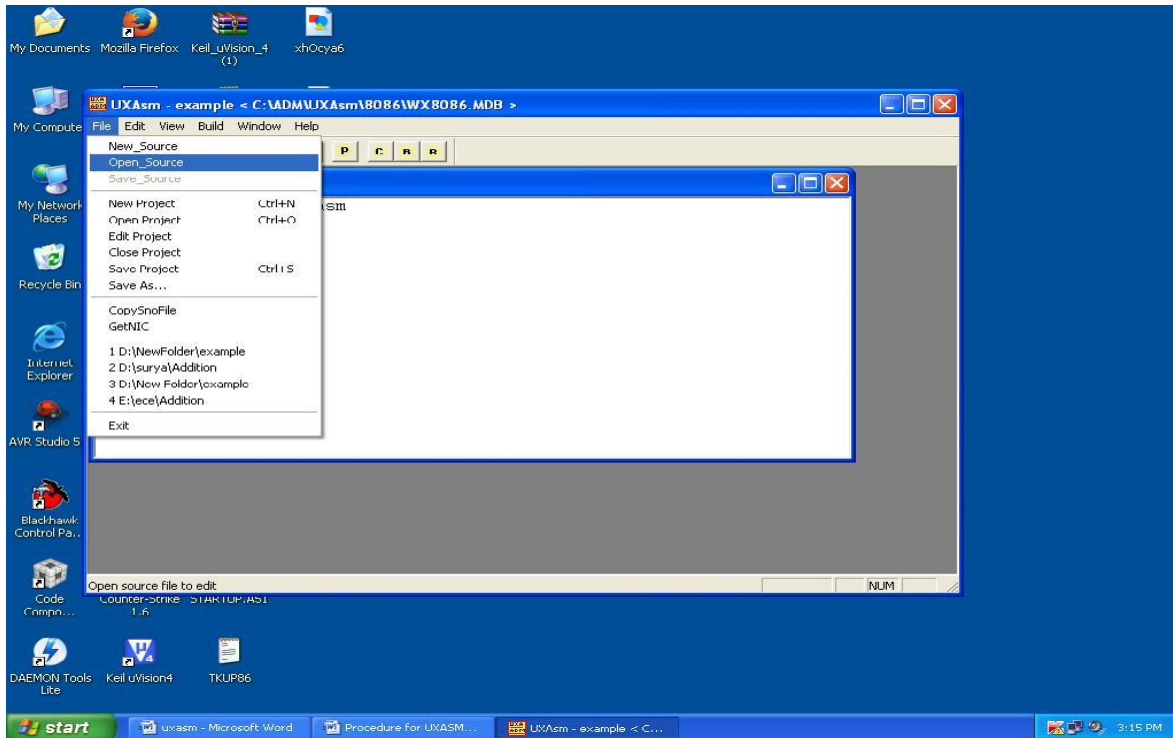
4. Observe the following window which shows the source code.
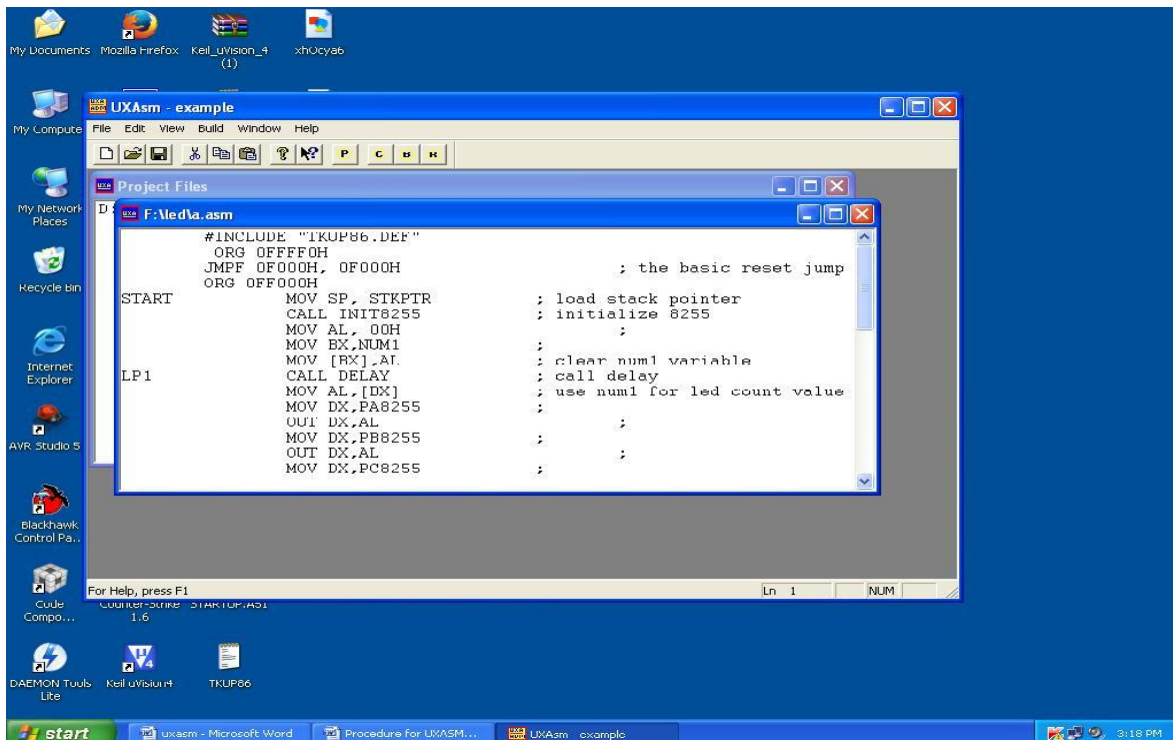
5. Again go to file select "New Project"
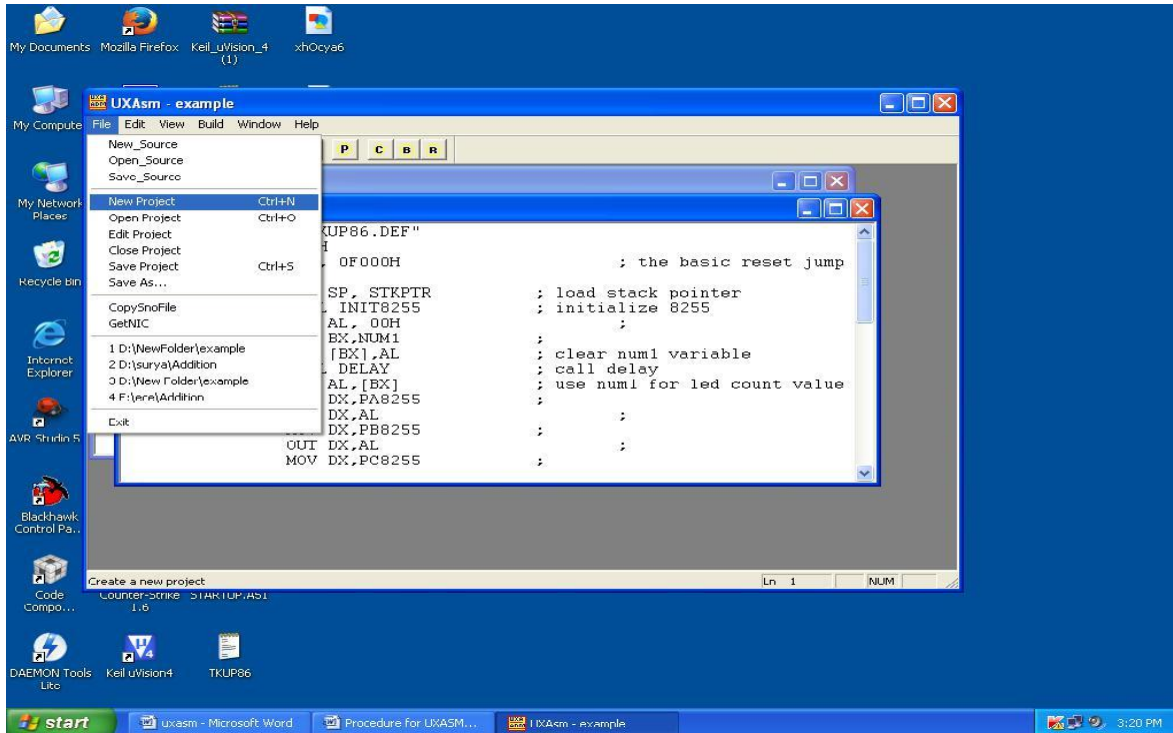
6. To add source file click on "Add File" and browse the source file and provide the source file path with .Uxa extension in the Project and press Tab and Press "Save" and click on "OK"
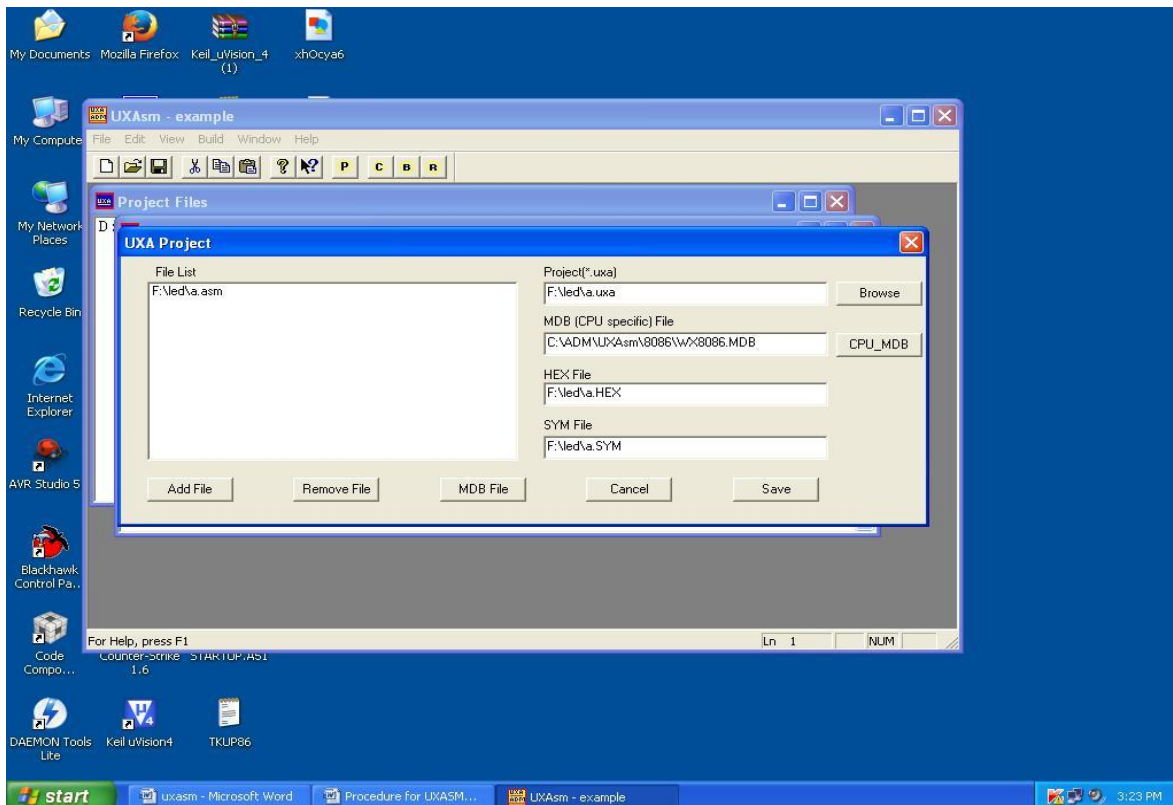
7. Observe the following window and double click on path of the File to view the program
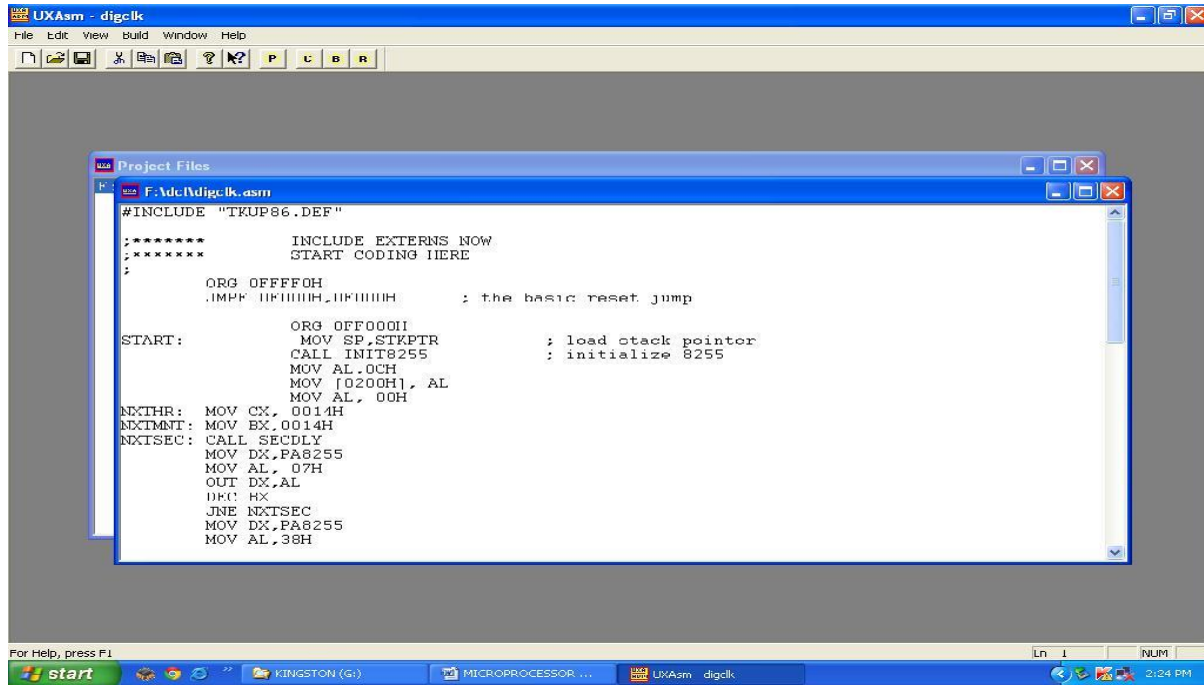
8. To save the project go to File select "save project"
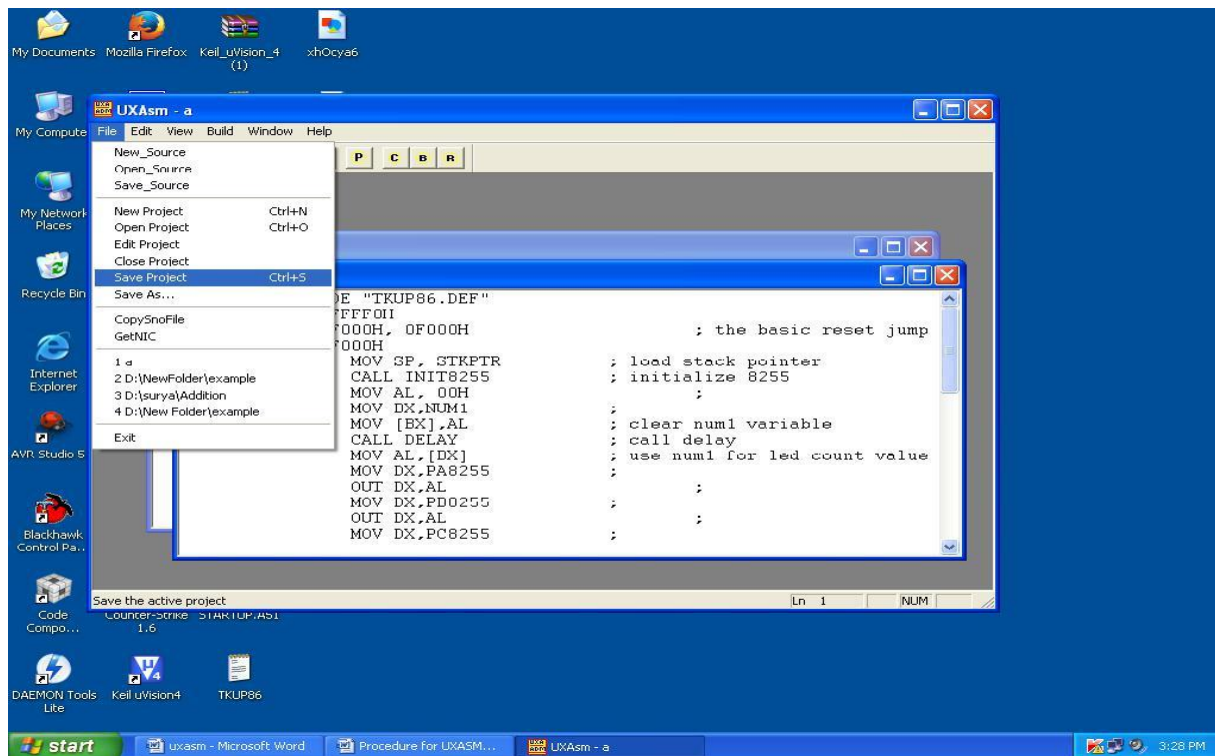
9. To compile the program, click on "C" and observe the following window
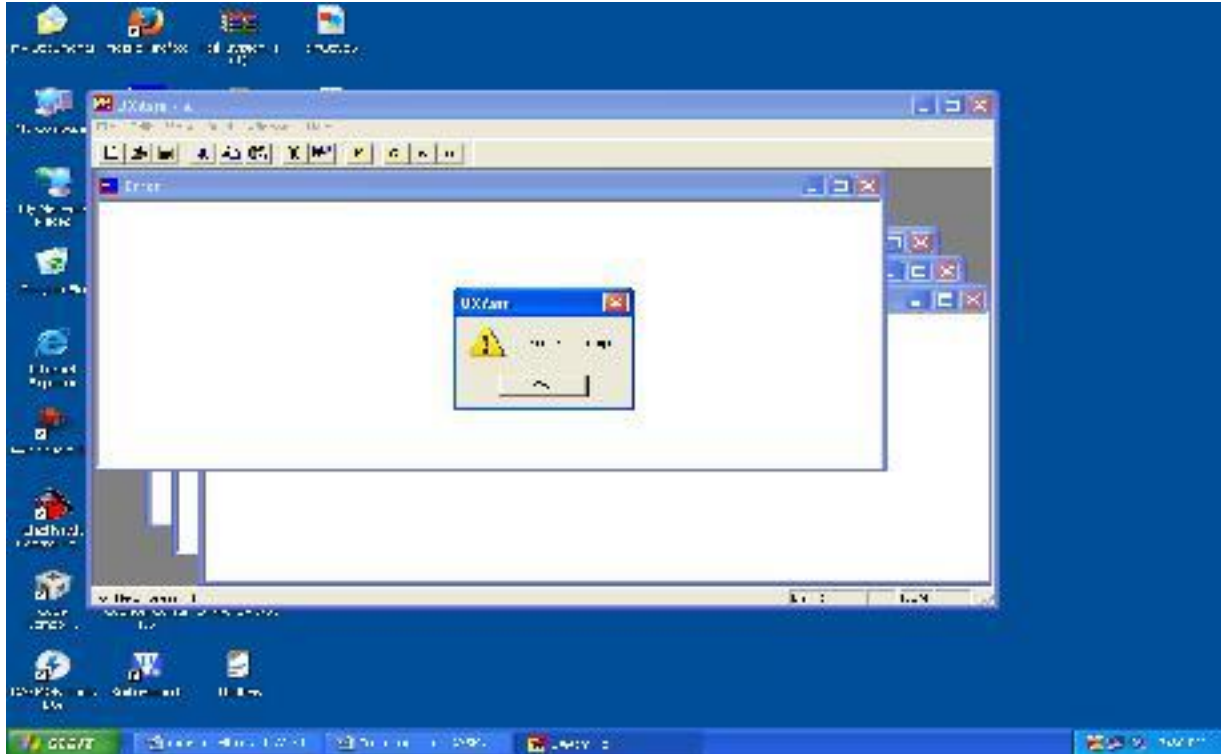
10. If any errors, Fix the errors, click on "OK" and click on "B" to build the program

TKµP
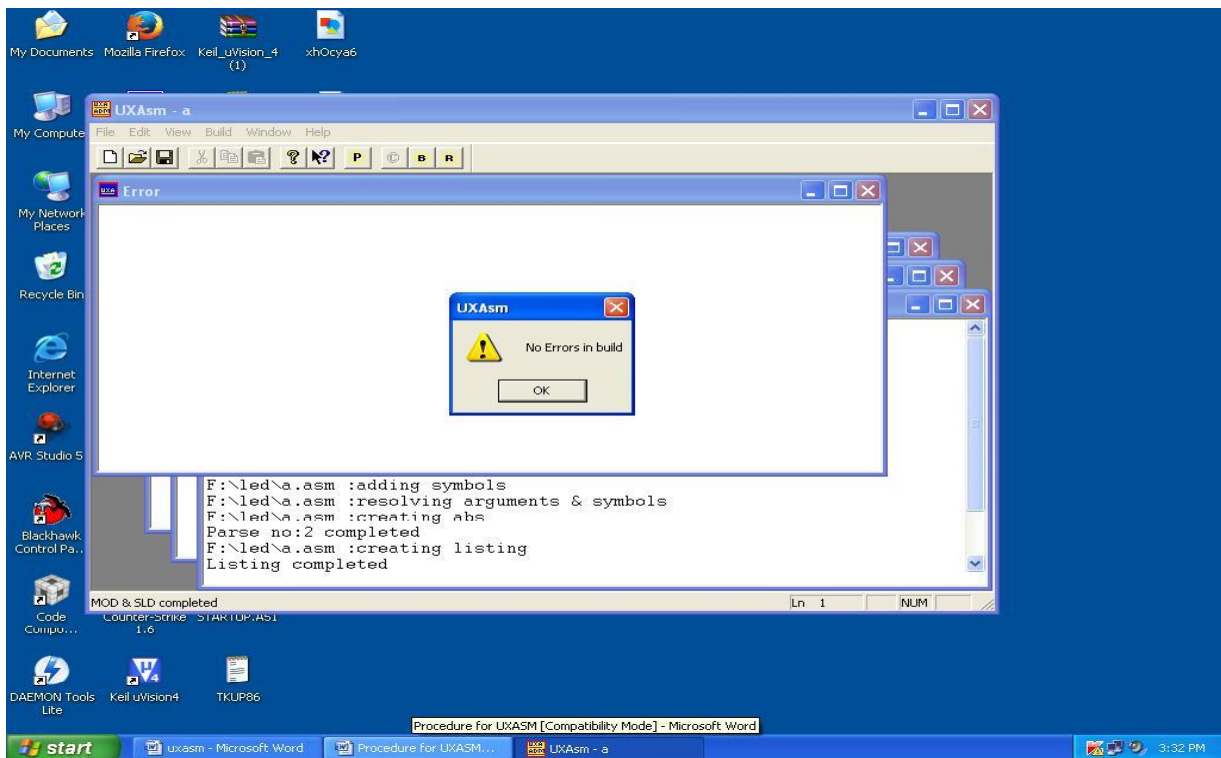
INTRODUCTION:

TKµP is an ideal trainer cum development boards for Microprocessors like Z80, 8032, 8085, 8088 and 8086. All interface is provided through 10 pin polarized Box Headers. TKµP user interface software communicates with the TKµP hardware through PC parallel port LPT1 and provides fast download of hex files. The PC user interface can open multiple windows for memory Dump and List. Multiple dump windows is also useful to study memory move operations and programs. TKµP is made up of three sections:

1. CPU specific daughter board.

2. Base board section: It has following features

➢ Four sockets for memory which can accommodate maximum 4x128KB.

➢ 8279 key board display controller.

➢ 8255 IO expander.

➢ 8155 IO expander with timer counter.

➢ 8251 Asynchronous serial Transmitter and Receiver.

3. User interface section: It has following features

- Hex keypad.

- 8-Leds indicator.

- Four multiplexed 7-Segment displays.

- LCD 16 characters x 2 lines.

- I2C NVRAM 24C1024.

- I2C RTC PCF8583.

- I2C ADC/DAC PCF8591.

- Serial port interface through MAX232.

Procedure for TKµP

1. Go to start and select TKµP
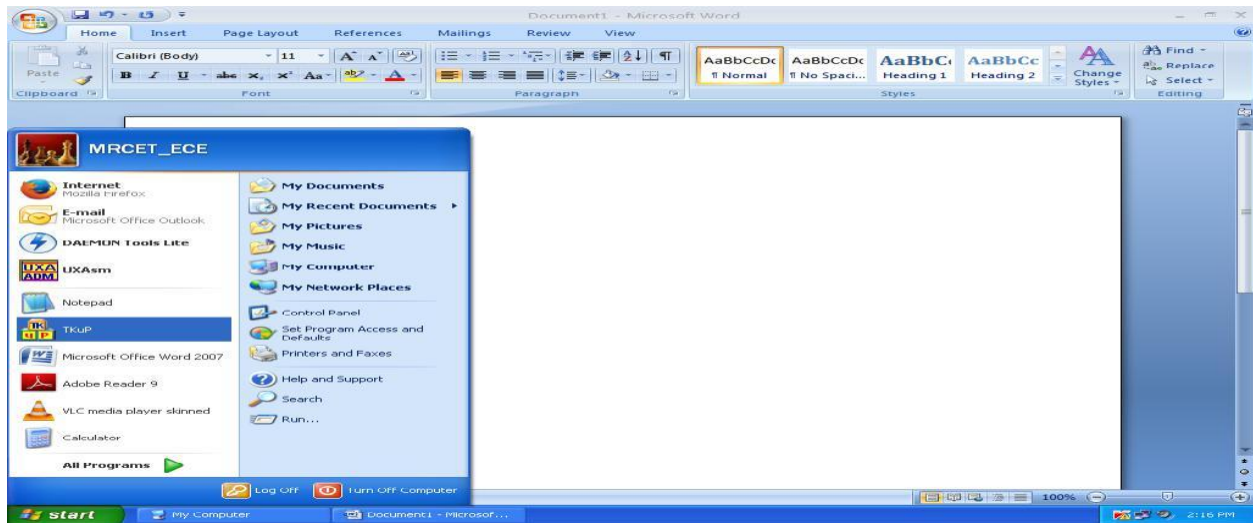
2. To Test the I/O connection clicks on Test I/O and click on OK
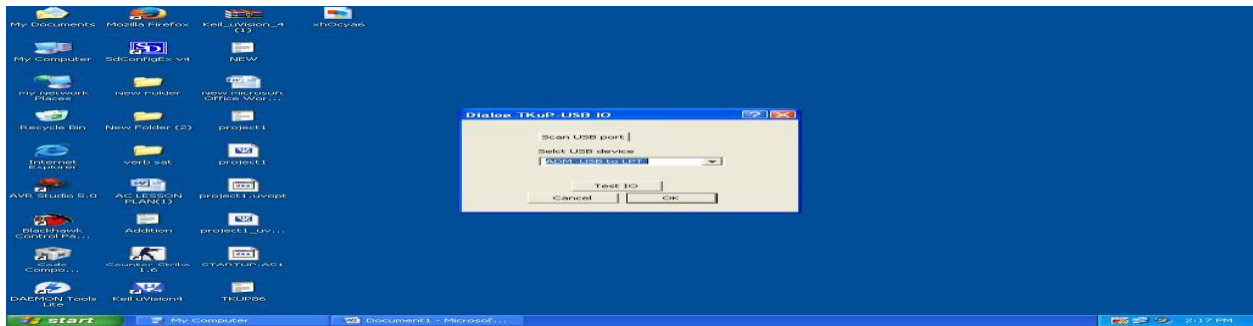
3. The following window will be displayed and go to window, select tile to avoid the overlap of windows
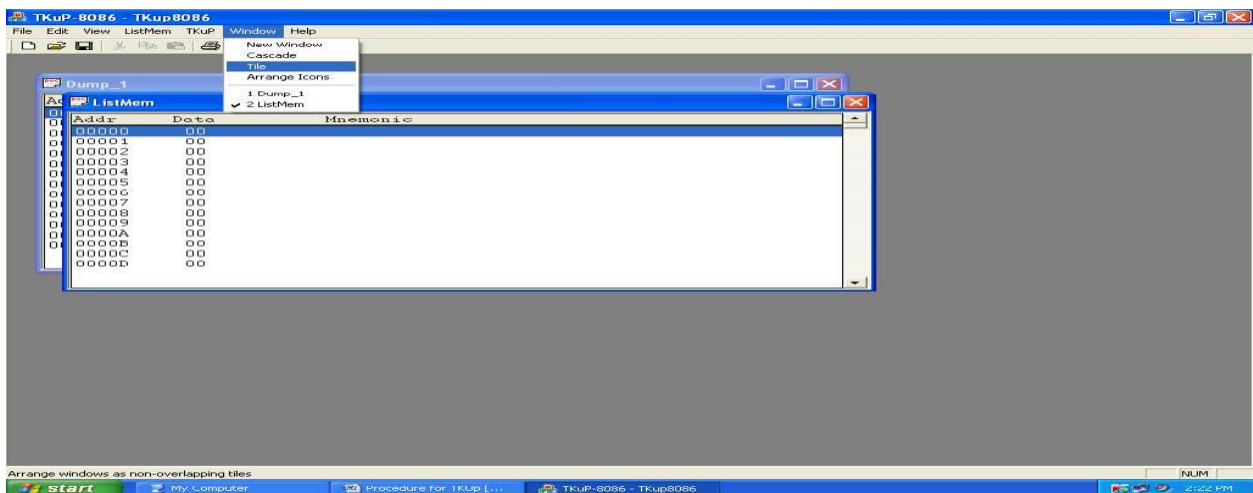
4 .To clear the garbage data from dump window, go to Listmem and select fillmem
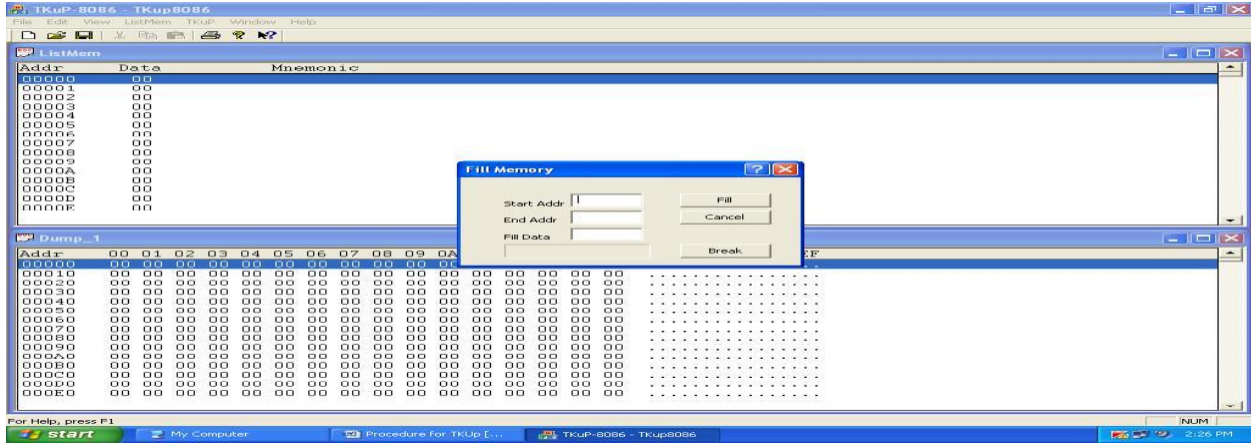
5. To clear the data from dump window, enter start and, end address and fill the data with 00
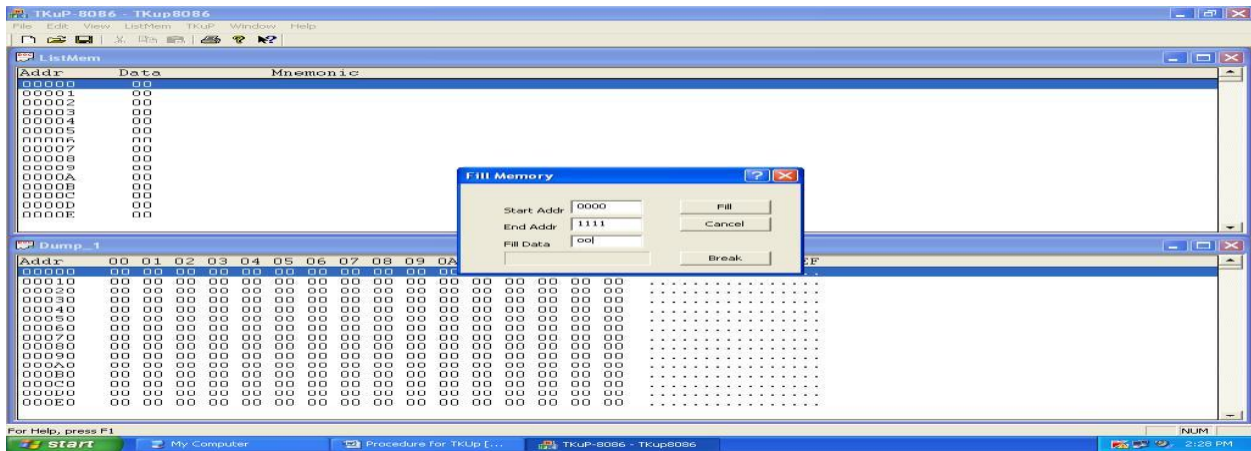
6. To load the hex file ,go to "Listmem" and select "HexLoad"
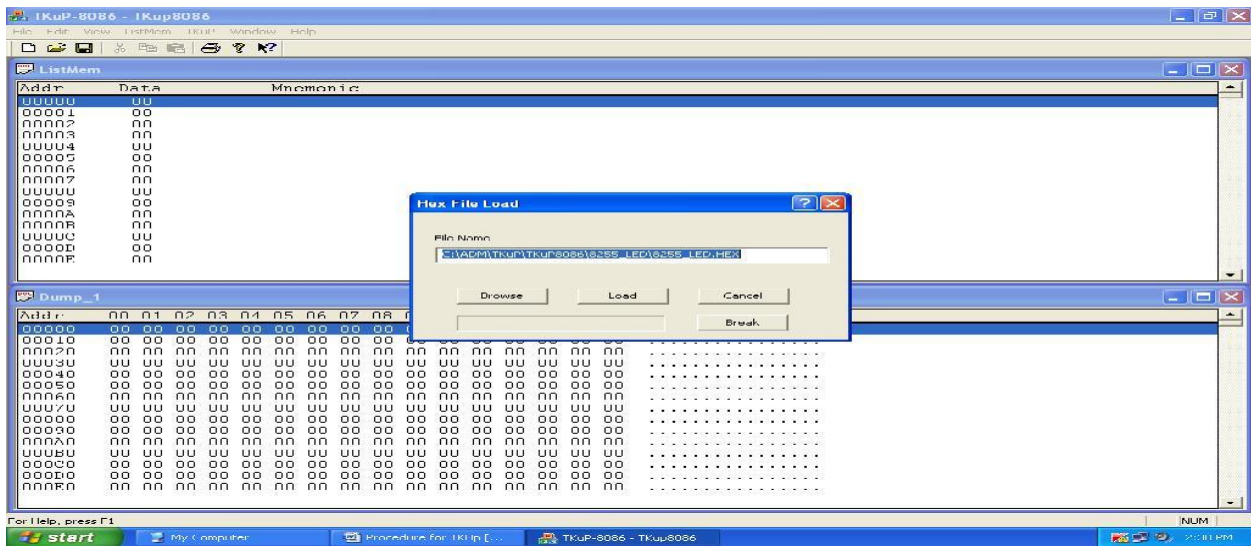
7. Browse the hex file from the source and click on "Load"
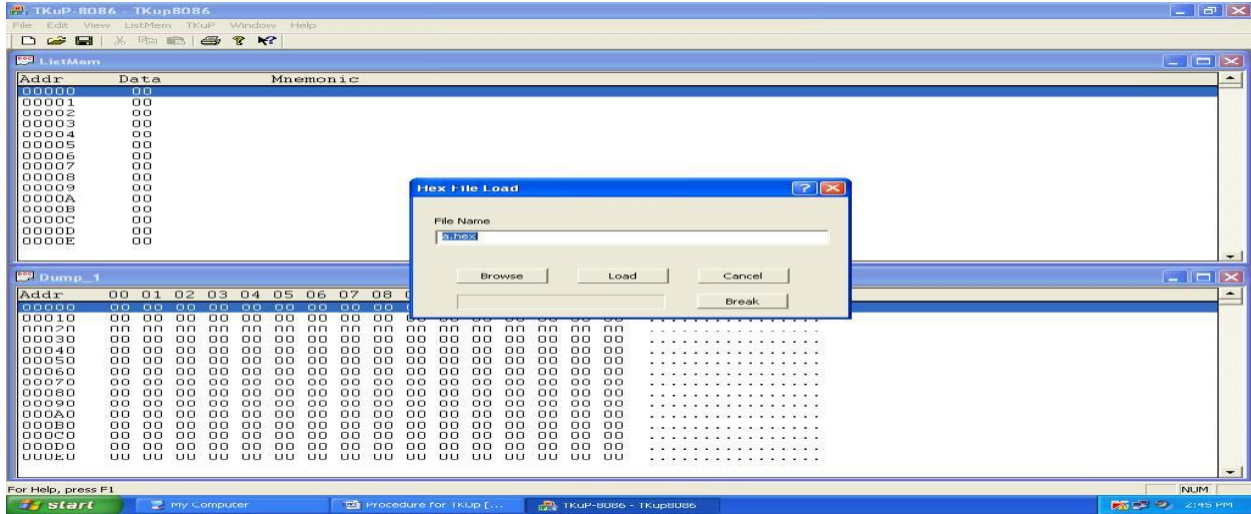
8. To view the program on the window, go to "Listmem" and select address
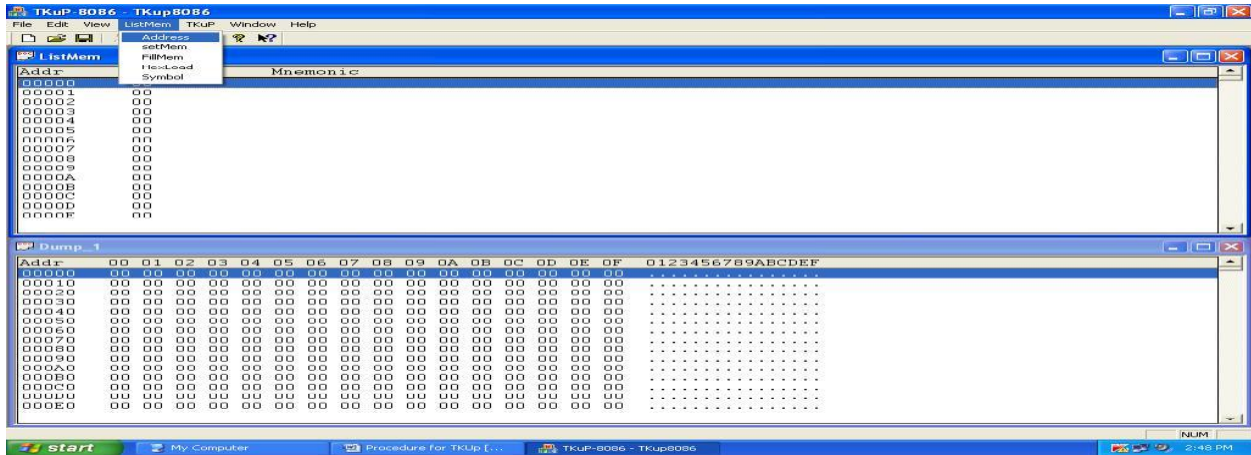
9. Enter the starting address of the program click "OK"
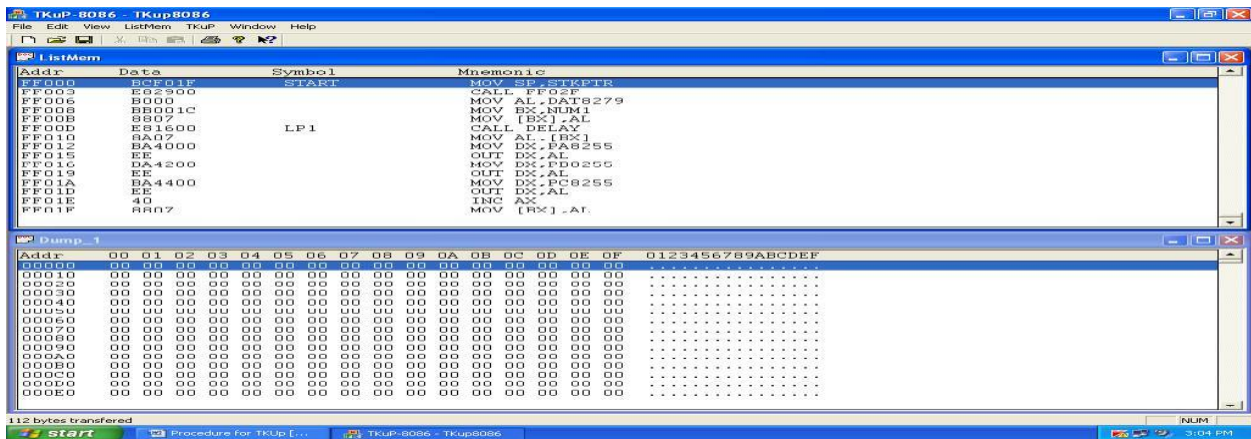
10. To verify the output, change the "SW-PP PROGRAM" switch to execution mode and verify output

EXPERIMENT NO: 6

DIGITAL CLOCK DESIGN USING 8086

AIM: Write an ALP for digital clock design using 8086

TOOLS:

i.   UXASM

ii.   TKUP

iii. TKUP86 KIT

iv.   FRC CABLE

PROGRAM:

; **CONNECT BH4 (PORT A) TO CNLED**

#INCLUDE "TKUP86.DEF"

;*******   INCLUDE EXTERNS NOW

;*******   START CODING HERE

     ORG 0FFFF0H

     JMPF 0F000H,0F000H ; the basic reset jump

        ORG 0FF000H

START: MOV SP,STKPTR   ; load stack pointer

        CALL INIT8255   ; initialize 8255

        MOV AL,0CH

        MOV [0200H], AL

        MOV AL, 00H

NXTHR: MOV CX, 003CH

NXTMNT: MOV BX,003CH

NXTSEC: CALL SECDLY

        MOV DX,PA8255

        MOV AL, 07H

        OUT DX,AL

        DEC BX

        JNE NXTSEC

```
                MOV DX,PA8255

                MOV AL,38H

                OUT DX,AL

                DEC CX

                JNE NXTMNT

                MOV DX, PA8255

                MOV AL, 0C0H

                OUT DX, AL

                MOV AX,[0200H]

                DEC AX

                MOV [0200H], AX

                JNE NXTHR

SECDLY:         PUSH AX

                PUSH BX

                PUSH CX

                PUSH DX

                MOV CX, 1234H

DLY:        NOP

                NOP

                LOOP DLY

                POP DX

                POP CX

                POP BX

                POP AX

                RET

;*******     initialize 8255

INIT8255

                MOV AL,080H

                MOV DX, CMD8255

                OUT DX,AL

                MOV AL,00H
```

MOV DX,PA8255

OUT DX,AL

MOV DX,PB8255

OUT DX,AL

MOV DX,PC8255

OUT DX,AL

RET

RESULT: INPUT:

OUTPUT:

Exercise Questions:

1) Write an assembly language program for the different clock rates to display the clock on the LCD.

Viva Questions:

1) What is the use of IN and OUT instructions?
2) What is meant by procedure?
3) What is meant by PPI?
4) What are the modes of 8255?

OBSERVATION:

EXPERIMENT NO: 7

PROGRAM FOR INTERFACING ADC&DAC TO 8086

AIM: Write an ALP for interfacing ADC to 8086

TOOLS:

   i.   UXASM

  ii.   TKUP

  iii.   TKUP86 KIT

  iv.   FRC CABLE

  v.   ADC KIT

PROGRAM:

```
; CONNECT BH4 (PORT A) TO DAC BH1A

; CONNECT BH5 (PORTB) TO DAC BH2B

; CONNECT CRO PROBES TO CND1_1 OF DAC


#INCLUDE "TKUP86.DEF"

DATA SEGMENT

        PORTA EQU 9000H

        PORTC EQU 9004H

        CNTLPRT EQU 9006H

        MEM DW 2000H

DATA ENDS

CODE SEGMENT

ASSUME CS: CODE, DS: DATA

START: MOV AX, DATA

            MOV DS, AX

            MOV DX, CNTLPRT

            MOV AL, 98H

            OUT DX, AL

            MOV AL, 01H

            OUT DX, AL
```

```
                    MOV AL, 00

                    OUT DX, AL

                    MOV DX, PORTC

CHK:                IN AL, DX

                    AND AL, 80H

                    JZ CHK

                    MOV DX, PORTA

                    IN AL, DX

                    MOV MEM, AL

                    INT 03H

CODE ENDS

END START
```

RESULT: INPUT   :

           OUTPUT :

INTERFACING DAC TO 8086

AIM: Write an ALP for interfacing DAC to 8086

TOOLS:
  i. UXASM
  ii. TKUP
  iii. TKUP86 KIT
  iv. FRC CABLE

PROGRAM:

; CONNECT BH4 (PORT A) TO DAC BH1A

; CONNECT BH5 (PORTB) TO DAC BH2B

; CONNECT CRO PROBES TO CND1_1 OF DAC

#INCLUDE "TKUP86.DEF"

        ORG 0FFFF0H

        JMPF 0F000H,0F000H

```
            ORG 0FF000H

            MOV AL,080H

            MOV DX,CMD8255

            OUT DX,AL

            MOV AL,00H

            MOV DX,PA8255

            OUT DX,AL

            MOV DX,PB8255

            OUT DX,AL

            MOV DX,PC8255

            OUT DX,AL

   RPT:     MOV AL,00H

            MOV AL,0FFH

 AGAIN:  MOV DX, PA8255

            OUT DX, AL

            CALL DELAY

            CALL DELAY

            CALL DELAY

            CALL DELAY

            CALL DELAY

            CALL DELAY

            INC AX

            JNE AGAIN

            JMP RPT

   DELAY: MOV CX, 0FF00H

   NXT2: MOV BX, 1234H

   NXT:   NOP

            NOP

            NOP

            NOP

            NOP
```

         DEC BX

         JNE NXT

         RET

RESULT: INPUT :

         OUTPUT :

Exercise Questions:

1) Write an assembly language program to convert a saw tooth wave into digital.
2) Write an assembly language program for the generation of triangular wave

Viva Questions:

1) What is the function of INC Instruction?
2) What is the function of NOP Instruction?
3) What is the size of the ports of 8255?
4) What is the function of the control word register of 8255?

OBSERVATION:

EXPERIMENT NO: 8

PARALLEL COMMUNICATION BETWEEN TWO MICROPROCESSORS USING 8255

AIM: Write an ALP for parallel communication between two microprocessors using 8255

TOOLS:

    i.   UXASM
   ii.   TKUP
  iii.   TKUP86 KIT
  iv.   FRC CABLE

PROGRAM: FOR DATA IN KIT

```
#INCLUDE "TKUP86.DEF"

        ORG 0FFFF0H

        JMPF 0F000H,0F000H

        ORG 0FF000H

        MOV AL,080H

        MOV DX,CMD8255

        OUT DX,AL

        MOV AL,00H

        MOV DX,PA8255

        OUT DX,AL

        MOV DX,PB8255

        OUT DX,AL

        MOV DX,PC8255

        OUT DX,AL

RPT:    MOV AL,47H

        MOV DX,PA8255

        OUT DX,AL

        MOV DX,PB8255

        OUT DX,AL

        MOV DX,PC8255

        OUT DX,AL

        JMP RPT
```

PROGRAM: FOR DATA OUT KIT

```
        #INCLUDE "TKUP86.DEF
         ORG 0FFFF0H
        JMPF 0F000H,0F000H
        ORG 0FF000H
        MOV AL,090H
        MOV DX,CMD8255
        OUT DX,AL
        MOV AL,00H
        MOV DX,PA8255
        OUT DX,AL
        MOV DX,PB8255
        OUT DX,AL
        MOV DX,PC8255
        OUT DX,AL
RPT:    MOV DX,PA8255
        IN AL,DX
        MOV [0200H],AL
        MOV DX,PB8255
        OUT DX,AL
        MOV DX,PC8255
        OUT DX,AL
`       JMP RPT
```

RESULT:

Exercise Questions:

1) Write an assembly language program to transfer MRCET string in between two 8255 kits.

Viva Questions:

1) What is the function of IN Instruction?
2) What is the function of OUT Instruction?
3) What is the size of the ports of 8255?
4) What is the function of the control word register of 8255?

   OBSERVATION:

EXPERIMENT NO: 9

PROGRAM FOR INTERFACING STEPPER TO 8086

(A) ROTATE THE STEPPER MOTOR IN ANTICLOCKWISE DIRECTION

; Connect 8255 Ports A to CNLED

;**** INCLUDE DEFINATION FILES NOW
#INCLUDE "TKUP86.DEF"

;****START CODING HERE

```
                 ORG 0FFFF0H
                 JMPF 0F000H,0F000H ; the basic reset jump
                 ORG 0FF000H
            START MOV SP,STKPTR        ; load stack pointer
                    CALL INIT8255       ; initialize 8255
            LP1     MOV AL,01H          ; use num1 for led count value
                    MOV DX,PA8255
                    OUT DX,AL           ;
                    CALL DELAY          ; call delay
                    MOV AL,02H          ; use num1 for led count value
                    MOV DX,PA8255
                    OUT DX,AL
                    CALL DELAY          ; call delay
                    MOV AL,04H          ; use num1 for led count value
                    MOV DX,PA8255
                    OUT DX,AL
                    CALL DELAY          ; call delay
                    MOV AL,08H          ; use num1 for led count value
                    MOV DX,PA8255
                    OUT DX,AL
                    CALL DELAY          ; call delay
                    JMP START           ; restart again

            ;*******      Delay module

            DELAY NOP                   ;
                    MOV CX,03500H            ; load Delay count = 0x3500
                    NOP                 ;
            DLY1    NOP                 ;
                    LOOP DLY1           ;
                    RET                 ; end of delay

            ;*******      initialize 8255
```

```
                    INIT8255
                            MOV AL,080H          ; make all ports output
                            MOV DX, CMD8255
                            OUT DX,AL            ; write to command register
                            MOV AL,00H           ; clear all ports
                            MOV DX,PA8255
                            OUT DX,AL            ;
                            MOV DX,PB8255
                            OUT DX,AL            ;
                            MOV DX,PC8255
                            OUT DX,AL            ;
                            RET                  ;
```

(B)ROTATE THE STEPPER MOTOR IN CLOCKWISE DIRECTION

;***** START CODING HERE

```
                    ORG 0FFFF0H
                    JMPF 0F000H,0F000H ; the basic reset jump


                    ORG 0FF000H
        START   MOV SP,STKPTR            ; load stack pointer
                    CALL INIT8255            ; initialize 8255
        LP1     MOV AL,08H               ; use num1 for led count value
                    MOV DX,PA8255
                    OUT DX,AL
                    CALL DELAY               ; call delay
                    MOV AL,04H               ; use num1 for led count value
                    MOV DX,PA8255
                    OUT DX,AL
                    CALL DELAY               ; call delay
                    MOV AL,02H               ; use num1 for led count value
                    MOV DX,PA8255                ;
                    OUT DX,AL
                    CALL DELAY               ; call delay
                    MOV AL,01H               ; use num1 for led count value
                    MOV DX,PA8255
                    OUT DX,AL
                    CALL DELAY               ; call delay
                    JMP START                ; restart again

            ;*******        Delay module

            DELAY NOP
```

```
                    MOV CX,03500H        ; load Delay count = 0x3500
                    NOP
            DLY1    NOP
                    LOOP DLY1
                    RET                   ; end of delay

            ;*******        initialize 8255
            INIT8255
                    MOV AL,080H          ; make all ports output
                    MOV DX, CMD8255
                    OUT DX,AL            ; write to command register
                    MOV AL,00H           ; clear all ports
                    MOV DX,PA8255
                    OUT DX,AL
                    MOV DX,PB8255
                    OUT DX,AL
                    MOV DX,PC8255
                    OUT DX,AL
                    RET
```

RESULT: INPUT:

OUTPUT:

Exercise Questions:

1) Write an assembly language program to rotate a stepper motor for 20 steps in clockwise direction?

Viva Questions:

1. Explain the principle of stepper motor.
2. How to calculate step angle?
3. What are the applications of stepper motor

OBSERVATION:

EXPERIMENT NO: 10

ARITHMETIC, LOGICAL AND BIT MANIPULATION INSTRUCTIONS OF 8051

AIM: Write an ALP for Arithmetic, logical and bit manipulation operations in 8051

TOOLS:

    i.   UXASM

   ii.   TKUP

  iii.   TKUP86 KIT

  iv.   FRC CABLE

A) PROGRAM: FOR ARITHMETIC INSTRUCTIONS OF

8051 ;Connect P1 to CNLED1

```
#INCLUDE "TKUP52.DEF"
        ORG 0000H
START: LJMP MAIN
        ORG 0150H
MAIN  MOV SP,#50H
        MOV R0,#20H
        MOV R1,#07H
        MOV A,R0
        ADD A,R1
        MOV P1,A
        LCALL DELAY
        MOV A,R0
        SUBB A,R1
        MOV P1,A
        LCALL DELAY
        MOV A,R0
        MOV 0F0H,R1
        MUL AB
        MOV P1,A
```

```
        LCALL DELAY

        MOV P1,0F0H

        LCALL DELAY

        MOV A,R0

        MOV 0F0H,R1

        DIV AB

        MOV P1,A

        LCALL DELAY

        MOV P1,0F0H

        LCALL DELAY

        LJMP MAIN

DELAY  NOP

        MOV R4,#020H

DLY3   MOV R3,#0FFH

DLY2   MOV R2,#0FFH

        NOP

DLY1   NOP

        NOP

        NOP

        DJNZ R2,DLY1

        DJNZ R3,DLY2

        DJNZ R4,DLY3

        RET           ;
```

B) PROGRAM: FOR LOGICAL INSTRUCTIONS OF 8051

i) **;Connect P1 to CNLED1**

```
        #INCLUDE "TKUP52.DEF"

        ORG 0000H

START: LJMP MAIN

        ORG 0150H

MAIN  MOV SP,#50H
```

```
            MOV A,#35H
            ANL A,#0FH
            MOV P1,A
            ACALL DLY
            MOV A,#04H
            ORL A,#30H
            MOV P1,A
            ACALL DLY
            MOV A,#54H
            XRL A,#78H
            MOV P1,A
            ACALL DLY
            MOV A,#55H
            CPL A
            MOV P1,A
            ACALL DLY
     DLY    NOP
            NOP
            MOV R4,#020H
     DLY3   MOV R3,#0FFH
     DLY2   MOV R2,#0FFH
            NOP
     DLY1   NOP
            NOP
            NOP
            NOP
            DJNZ R2,DLY1
            DJNZ R3,DLY2
            DJNZ R4,DLY3
            RET
```

ii) ;Connect P1 to CNLED1

```
    #INCLUDE "TKUP52.DEF"

                ORG 0000H

START:      LJMP MAIN

                ORG 0150H

MAIN:       MOV SP,#060H

              MOV A,#0A5H

              MOV P1,A

              LCALL SFTDL

              RR A

              MOV P1,A

              LCALL SFTDL

              SWAP A

              MOV P1,A

              LCALL SFTDL

              RL A

              MOV P1,A

              LCALL SFTDL

              SETB C

              RLC A

              MOV P1,A

              LCALL SFTDL

              RRC A

              MOV P1,A

              LCALL SFTDL

              LJMP MAIN

SFTDL MOV R4,#50H

DL3         MOV R5,#0FFH

DL2         MOV R6,#0FFH

DL1         DJNZ R6,DL1

              DJNZ R5,DL2
```

```
        DJNZ R4,DL3

        RET
```

C) PROGRAM: FOR BIT MANIPULATION INSTRUCTIONS OF 8051
; Connect P1 to CNLED1

#INCLUDE "TKUP52.DEF"

```
        ORG 0000H

START: LJMP MAIN

        ORG 0150H

MAIN  MOV SP,#50H

        MOV P1,#00H

        MOV C,00H

        SETB C

        MOV P1_7,C

        LCALL SFTDL

        CLR C

        ANL C,00H

        MOV P1_7,C

        LCALL SFTDL

        CPL C

        MOV P1_3,C

        LCALL SFTDL

        ORL C,00H

        MOV P1_7,C

        LCALL SFTDL

        LJMP MAIN

SFTDL MOV R4,#50H

DL3     MOV R5,#0FFH

DL2     MOV R6,#0FFH

DL1     DJNZ R6,DL1


        DJNZ R5,DL2
```

DJNZ R4,DL3

RET

RESULT: INPUT:

                 OUTPUT:

Exercise Questions:

1) Write an assembly language program for the addition of 012H and 376H in 8051?

Viva Questions:

1) What are the ports of 8051?
2) What is the use of DJNZ instruction?
3) What are the bit manipulation instructions of 8051?
4) What are the flags of 8051?

OBSERVATION:

EXPERIMENT NO: 11

TIMER/COUNTERS IN 8051

AIM: Write an ALP to verify timer/counter operation in 8051

TOOLS: i) UXASM

Ii) TKUP

Iii) TKUP86 KIT

IV) FRC CABLE

PROGRAM:

; Connect P1 to CNLED1

```
#INCLUDE "TKUP52.DEF"

              ORG 0000H
START:     LJMP MAIN
              ORG 0150H
MAIN:        MOV SP,#060H
             MOV TMOD,#01H
BACK:       MOV TL0,#075H
              MOV TH0,#0B8H
              MOV P1,#0AAH
              LCALL SFTDL
              ACALL DELAY
              MOV TL0,#00H
              MOV TH0,#00H
              MOV P1,#055H
              ACALL DELAY
              LCALL SFTDL
              SJMP BACK
              ORG 300H
DELAY:       SETB TCON4
AGAIN:       JNB TCON5,AGAIN
```

```
                    CLR TCON4

                    CLR TCON5

                     RET

SFTDL          MOV R4,#10H

DL3            MOV R5,#0FFH

DL2            MOV R6,#0FFH

DL1            DJNZ R6,DL1

               DJNZ R5,DL2

               DJNZ R4,DL3

               RET
```

RESULT: INPUT:

                         OUTPUT:

Exercise Questions:

1) Write a assembly language program for counting number of 1's and 0's in 34H?

Viva Questions:

1) What are timer/counter registers in 8051?
2) What is the size of timer/Counter?
3) When timer overflow occurs?
4) What are special functions registers of 8051?

OBSERVATION:

EXPERIMENT NO: 12
INTERRUPT HANDLING IN 8051

AIM: Write an ALP to verify the interrupt handling in 8051

TOOLS   i) UXASM

        ii)TKUP

        iii)TKUP86 KIT

        iv)FRC CABLE

PROGRAM:

```
#INCLUDE "TKUP52.DEF"

        ORG 0000H

START: LJMP MAIN

        ORG 0150H

MAIN    MOV SP,#50H

        MOV IE,#85H

HERE    MOV P1,#7EH

        SJMP HERE

        ORG 0003H        ; INT0 ISR

        MOV P1,#0AAH

        LCALL DELAY

        LCALL DELAY

        LCALL DELAY

        RETI

        ORG 0013H        ; INT1 ISR

        MOV P1,#0A5H

        LCALL DELAY

        LCALL DELAY

        RETI

DELAY    NOP
```

```
                    MOV R4,#020H

DLY3        MOV R3,#0FFH

DLY2        MOV R2,#0FFH

DLY1        NOP

            NOP

            DJNZ R2,DLY1

            DJNZ R3,DLY2

            DJNZ R4,DLY3

            RET
```

RESULT: INPUT:

                        OUTPUT:

Exercise Questions:

1) Write the program for interrupt handing of 8051 using PORT 0?

Viva Questions:

1) What are the interrupts of 8051?
2) What is the Priority among 8051 interrupts?
3) What are the interrupt registers of 8051?
4) What is the size of the interrupt registers of 8051?

OBSERVATION:

EXPERIMENT NO: 13

UART OPERATION IN 8051

AIM: To observe the UART operation in 8051

TOOLS: i) UXASM

     ii) TKUP

     iii) TKUP86 KIT

     iv) FRC CABLE

PROGRAM:

; **CONNECT THE RS232 FROM PC TO TKUP51 KIT**

; **CONNECT THE Tx PIN OF 8051 TO Rx OF MAX232 AND VICE VERSA**

; **CONNECT PORT1 TO CNLED**

```
        #INCLUDE "TKUP52.DEF"
                ORG 0000H
START:  LJMP MAIN
                ORG 0150H
MAIN:     MOV SP,#060H
              MOV IE,#85H
              MOV TMOD,#20H
              MOV TH1,#0FAH
              MOV SCON,#50H
              SETB TCON6
RPT:       MOV SBUF,#'Y'
HERE:     JNB SCON1,HERE
              CLR SCON1
              MOV A,#'A'
              MOV P1,A
              SJMP RPT
```

RESULT: INPUT:

       OUTPUT:

Exercise Questions:

1) Where do we prefer the serial communication & Why?

Viva Questions:

1) What is the full form of UART?
2) What is meant by Synchronous and Asynchronous communication?
3) What is the serial communication registers in 8051?
4) Which data communication method is supported by 8051?

OBSERVATION:

EXPERIMENT NO: 14

INTERFACING LCD TO 8051

AIM: Write an ALP for interfacing LCD to 8051

TOOLS:

I) UXASM

II) TKUP

III) TKUP86 KIT

 IV) FRC CABLE

PROGRAM:

```
;CONNECT BH4 TO CNLCDC

;CONNECT BH6 TO CNLCDD

#INCLUDE "TKUP52.DEF"
    ORG 0000H
START: LJMP MAIN
        ORG 0150H
        MAIN  MOV SP,#060H
        LCALL INIT8255
LOOP  MOV DPTR,#CMDTBL
        LCALL INIT_LCD
        MOV DPTR,#STRTBL
LP1     MOV A,#0
        MOVC A,@A+DPTR
        CJNE A,#00,LP2
        LCALL DELAY
        LCALL DELAY
        LCALL DELAY
        LJMP MAIN
LP2     LCALL WR_DAT
```

```
        INC DPTR

        LCALL SDELAY

        LJMP LP1

;*******LCD init module

INIT_LCD

        MOV A,#0

        MOVC A,@A+DPTR

        CJNE A,#00,IL2

        RET

IL2     LCALL WR_CMD

        INC DPTR

        LJMP INIT_LCD

;*******        LCD Write CMD module

WR_CMDPUSH DPH

        PUSH DPL

        MOV DPTR,#PB8255

        LCALL WRPORT

        LCALL SDELAY

        MOV A,#04

        MOV DPTR,#PA8255

        LCALL WRPORT

        LCALL SDELAY

        MOV A, #00

        MOV DPTR,#PA8255

        LCALL WRPORT

        LCALL SDELAY

        POP DPL

        POP DPH

        RET

;*******        LCD Write Data module

WR_DAT        PUSH DPH
```

```
                PUSH DPL

                MOV DPTR,#PB8255

                LCALL WRPORT

                LCALL SDELAY

                MOV A,#05H

                MOV DPTR,#PA8255

                LCALL WRPORT

                LCALL SDELAY

                MOV A,#01H

                MOV DPTR,#PA8255

                LCALL WRPORT

                LCALL SDELAY

                POP DPL

                POP DPH

                RET


;******Write Port

WRPORT          CLR P1_7

                MOVX @DPTR,A

                SETB P1_7

                RET

;******                Read Port

RDPORT          CLR P1_7

                MOVX A,@DPTR

                SETB P1_7

                RET

;*******       Delay module

SDELAY          NOP

                MOV R0,#0FFH

                MOV R1,#01H

                LJMP DLY1
```

NOP

DELAY NOP

        MOV R0,#0FFH

        MOV R1,#055H

        NOP

DLY1   DJNZ R0,DLY1

        MOV R0,#0FFH

        DJNZ R1,DLY1

        RET

;*******   initialize 8255

INIT8255

        MOV A,#080H

        MOV DPTR,#CMD8255

        LCALL WRPORT

        RET

        ORG 0500H

;*******   initialize seven segment table

CMDTBL HEX 38,0E,02,01,00

STRTBL ASCII   "HELLO ADM - TKUP"

ENDTBL HEX   00,00

RESULT: INPUT :

                OUTPUT :

Viva Questions:

1) What are the special function register of 8051?
2) What is the function of accumulator register?
3) What is the function of CJNE instruction?
4) What is the function of MOVX instruction?

OBSERVATION:

EXPERIMENT NO: 15

INTERFACING MATRIX/KEYBOARD TO 8051

AIM: Write an ALP for interfacing Matrix/keyboard to 8051

TOOLS: i) UXASM

ii) TKUP

iii) TKUP86 KIT

iv)FRC CABLE

PROGRAM:

```
;*******     8255_KBD

*******      INCLUDE DEFINATION FILES NOW

;       1. Connect 8255 PA0-7 to CNMUX of L1C peripheral board

;       2. Connect 8255 PC0-7 to CNKEY of L1C peripheral board

;       3. Connect 8255 PB0-7 to CNSEG of L1C peripheral board

;       4. Motor one segment showing 0000->0001->....->000F->0000 (key press)


#INCLUDE "TKUP52.DEF"

        ORG 0000H

START: LJMP MAIN

        ORG 0150H

MAIN  MOV SP,#060H

        LCALL INIT8255

        MOV DPTR,#NUM1

        LCALL CLRMEM

        MOV DPTR,#NUM2

        LCALL CLRMEM

        MOV DPTR,#NUM3

        LCALL CLRMEM

        NOP
```

```
LOOP    LCALL SCANKBD

        MOV DPTR,#NUM3

        MOVX A,@DPTR

        MOV DPTR,#SEGTBL

        MOVC A,@A+DPTR

        MOV DPTR,#PB8255

        LCALL WRPORT

        MOV A,#070H

        MOV DPTR,#PA8255

        LCALL WRPORT

        LJMP LOOP


;*******     MATRIX KBD SCAN module

;*******     Output either E0,D0,B0,70 for Row 1,2,3,4

;*******     Read PC port ended with 0x0F, expect 0F,0E,0D,0B,07


SCANKBD     MOV A,#00H

SKLOOP      MOV DPTR,#NUM1

        MOVX @DPTR,A

        MOV DPTR,#KBDTBL

        MOVC A,@A+DPTR

        CJNE A,#00,SKL1

        RET

SKL1    MOV DPTR,#PC8255

        LCALL WRPORT

        MOV DPTR,#PC8255

        LCALL RDPORT

        ANL A,#0FH

        CJNE A,#0FH,SKL2

        MOV DPTR,#NUM1

        MOVX A,@DPTR
```

```
                INC A

                LJMP SKLOOP

      SKL2    LJMP GETKEY

      ;*******        GETKEY module

      GETKEY          MOV DPTR,#NUM2

                MOVX @DPTR,A

                MOV DPTR,#RETTBL

                MOVC A,@A+DPTR

                MOV R0,A

                MOV DPTR,#NUM1

                MOVX A,@DPTR

                MOV DPTR,#ROWTBL

                MOVC A,@A+DPTR

                ADD A,R0

                MOV DPTR,#NUM3

                MOVX @DPTR,A

                RET

      ;******              Clear memory location

      CLRMEM        MOV A,#0

                MOVX @DPTR,A

                RET

      WRPORT CLR P1_7

                MOVX @DPTR,A

                SETB P1_7

                RET

      ;******              Read Port

      RDPORT        CLR P1_7

                MOVX A,@DPTR

                SETB P1_7

                RET
```

;*******   Delay module

SDELAY   NOP

          MOV R0,#0FFH

          MOV R1,#01H

          LJMP DLY1

          NOP

DELAY NOP

          MOV R0,#0FFH   ; load lsb of delay=0x34FF

          MOV R1,#055H   ; load msb

          NOP

DLY1   DJNZ R0,DLY1

          MOV R0,#0FFH   ; decrement msb count

          DJNZ R1,DLY1

          RET   ; end of delay

;*******   initialize 8255

INIT8255

          MOV A,#081H   ; make all ports output

          MOV DPTR,#CMD8255   ; write to command register

          LCALL WRPORT

          RET

          ORG 0500H

;*******   initialize seven segment table

SEGTBL   HEX   3F,06,5B,4F,66,6D,7D,07,7F,6F,77,7C,39,5E,79,71,00

KBDTBL   HEX   E0,D0,B0,70,00

RETTBLHEX   00,00,00,00,00,00,00,03,00,00,00,02,00,01,00,00,00

ROWTBL   HEX   00, 04, 08, 0C, 00

RESULT: INPUT:

                OUTPUT:

Exercise Question:

1) Write an assembly language program for the display of MRCET string on LCD by using 8051

Viva Questions:

1) What are the special function register of 8051?
2) What is the function of accumulator register?
3) How many no. of pins available for 8051?
4) What is the function of SP register

OBSERVATION: