Gopalan College
of Engineering and Management
ISO 9001 : 2008

APPROVED BY AICTE NEW DELHI, AFFILIATED TO VTU BELGAUM

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

MICROPROCESSOR AND MICROCONTROLLER LABORATORY
LAB MANUAL - 15CSL48

As per Choice Based Credit System (CBCS) scheme

Effective from the academic year 2016 -2017

Prepared by:	Reviewed by:	Approved by:
Smruthi Nair	N.S.Saradha Devi	Dr. A.A. Powly Thomas
Assistant Professor	Head of the Department	Principal
Dept. of CSE	Dept. of CSE	GCEM
GCEM	GCEM	

181/1, 182/1, Hoodi Village, Sonnenahalli, K.R. Puram,, Bengaluru, Karnataka 560048

Computer Lab DO's and DON'TS

Do's

1. Know the location of the fire extinguisher and the first aid box and how to use them in case of an emergency.
2. Read and understand how to carry out an activity thoroughly before coming to the laboratory.
3. Report fires or accidents to your lecturer/laboratory technician immediately.
4. Report any broken plugs or exposed electrical wires to your lecturer/laboratory technician immediately.

Don'ts

1. Do not eat or drink in the laboratory.
2. Avoid stepping on electrical wires or any other computer cables.
3. Do not open the system unit casing or monitor casing particularly when the power is turned on. Some internal components hold electric voltages of up to 230 volts, which can be fatal.
4. Do not insert metal objects such as clips, pins and needles into the computer casings. They may cause fire.
5. Do not remove anything from the computer laboratory without permission.
6. Do not touch, connect or disconnect any plug or cable without your lecturer/laboratory technician's permission.
7. Do not misbehave in the computer laboratory.

MICROPROCESSOR AND MICROCONTROLLER LABORATORY

[As per Choice Based Credit System (CBCS) scheme] (Effective from the academic year 2016 -2017)

SEMESTER – IV

Subject Code	15CSL48	IA Marks	20
Number of Lecture Hours/Week	01 I + 02 P	Exam Marks	80
Total Number of Lecture Hours	40	Exam Hours	03

CREDITS – 02

Course objectives: This course will enable students to

- To provide practical exposure to the students on microprocessors, design and coding knowledge on 80x86 family/ARM. To give the knowledge and practical exposure on connectivity and execute of interfacing devices with 8086/ARM kit like LED displays, Keyboards, DAC/ADC, and various other devices.

Description

Demonstration and Explanation hardware components and Faculty in-charge should explain 8086 architecture, pin diagram in one slot. The second slot, the Faculty in-charge should explain instruction set types/category etc. Students have to prepare a write-up on the same and include it in the Lab record and to be evaluated.

Laboratory Session-1: Write-up on Microprocessors, 8086 Functional block diagram, Pin diagram and description. The same information is also taught in theory class; this helps the students to understand better.

Laboratory Session-2: Write-up on Instruction group, Timing diagrams, etc. The same information is also taught in theory class; this helps the students to understand better.

Note: These TWO Laboratory sessions are used to fill the gap between theory classes and practical sessions. Both sessions are evaluated as lab experiments for 20 marks.

Experiments

- Develop and execute the following programs using 8086 Assembly Language. Any suitable assembler like MASM/TASM/8086 kit or any equivalent software may be used.
- Program should have suitable comments.
- The board layout and the circuit diagram of the interface are to be provided to the student during the examination.
- Software Required: Open source ARM Development platform, KEIL IDE and Proteus for simulation

SOFTWARE PROGRAMS: PART A

1. Design and develop an assembly language program to search a key element "X" in a list of 'n' 16-bit numbers. Adopt Binary search algorithm in your program for searching.
2. Design and develop an assembly program to sort a given set of 'n' 16-bit numbers in ascending order. Adopt Bubble Sort algorithm to sort given elements.
3. Develop an assembly language program to reverse a given string and verify whether it is a palindrome or not. Display the appropriate message.
4. Develop an assembly language program to compute nCr using recursive procedure. Assume that 'n' and 'r' are non-negative integers.

5. Design and develop an assembly language program to read the current time and Date from the system and display it in the standard format on the screen.
6. To write and simulate ARM assembly language programs for data transfer, arithmetic and logical operations (Demonstrate with the help of a suitable program).
7. To write and simulate C Programs for ARM microprocessor using KEIL (Demonstrate with the help of a suitable program)

Note : To use KEIL one may refer the book: Insider's Guide to the ARM7 based microcontrollers, Hitex Ltd.,1st edition, 2005

HARDWARE PROGRAMS: PART B

8. a. Design and develop an assembly program to demonstrate BCD Up-Down Counter (00-99) on the Logic Controller Interface.

b. Design and develop an assembly program to read the status of two 8-bit inputs (X & Y) from the Logic Controller Interface and display X*Y.

9. Design and develop an assembly program to display messages "FIRE" and "HELP" alternately with flickering effects on a 7-segment display interface for a suitable period of time. Ensure a flashing rate that makes it easy to read both the messages (Examiner does not specify these delay values nor is it necessary for the student to compute these values).
10. Design and develop an assembly program to drive a Stepper Motor interface and rotate the motor in specified direction (clockwise or counter-clockwise) by N steps (Direction and N are specified by the examiner). Introduce suitable delay between successive steps. (Any arbitrary value for the delay may be assumed by the student).
11. Design and develop an assembly language program to
    a. Generate the Sine Wave using DAC interface (The output of the DAC is to be displayed on the CRO).
    b. Generate a Half Rectified Sine waveform using the DAC interface. (The output of the DAC is to be displayed on the CRO).
12. To interface LCD with ARM processor-- ARM7TDMI/LPC2148. Write and execute programs in C language for displaying text messages and numbers on LCD
13. To interface Stepper motor with ARM processor-- ARM7TDMI/LPC2148. Write a program to rotate stepper motor

Study Experiments:

1. Interfacing of temperature sensor with ARM freedom board (or any other ARM microprocessor board) and display temperature on LCD
2. To design ARM cortex based automatic number plate recognition system
3. To design ARM based power saving system

Course Outcomes: After studying this course, students will be able to

- Learn 80x86 instruction sets and gins the knowledge of how assembly language works.
- Design and implement programs written in 80x86 assembly language
- Know functioning of hardware devices and interfacing them to x86 family
- Choose processors for various kinds of applications.

Graduate Attributes

- Engineering Knowledge
- Problem Analysis
- Modern Tool Usage
- Conduct Investigations of Complex Problems
- Design/Development of Solutions

Conduction of Practical Examination:

- All laboratory experiments (all 7 + 6 nos) are to be included for practical examination.
- Students are allowed to pick one experiment from each of the lot.
- Strictly follow the instructions as printed on the cover page of answer script for breakup of marks
- PART –A: Procedure + Conduction + Viva: 10 + 25 +05 (40)
- PART –B: Procedure + Conduction + Viva: 10 + 25 +05 (40)
- Change of experiment is allowed only once and marks allotted to the procedure part to be made zero.

TABLE OF CONTENTS

SL.NO	TITLE	PAGE NO
	SOFTWARE PROGRAMS:PART A	
1A	Binary Search	11-12
2A	Bubble Sort	13-14
3A	Palindrome	15-16
4A	Compute nCr using recursive procedure	17-18
5A	Current Time and Date of System	19-20
6A	ARM assembly language programs	21-22
7A	C Programs for ARM microprocessor using KEIL	23
	HARDWARE PROGRAMS:PART B	
7B	BCD Up-Down Counter (00-99) on the Logic Controller Interface.	24-26
8B	X&Y from the Logic Controller Interface and display X*Y.	27-28
9B	7-segment display	29-33
10B	Stepper Motor	34-36
11A	Sine Wave using DAC interface	37-39
11B	Half Rectified Sine wave form using the DAC interface	40-42
12B	Interface LCD with ARM processor	43-45
13B	To interface Stepper motor with ARM processor	46-47
	STUDY EXPERIMENTS	

Introduction to 8086 and Microsoft assembler

8086 Internal Block diagram

8086 is a 16-bit processor having 16-bit data bus and 20-bit address bus. The block diagram of 8086is as shown. (Refer figures 1A & 1B). This can be subdivided into two parts; the Bus Interface Unit (BIU) and Execution Unit (EU).

Bus Interface Unit:

The BIU consists of segment registers, an adder to generate 20 bit address and instruction prefetch queue. It is responsible for all the external bus operations like opcode fetch, mem read, mem write, I/O read/write etc. Once this address is sent OUT of BIU, the instruction and data bytes are fetched from memory and they fill a 6-byte First in First out (FIFO) queue.

Execution Unit:

The execution unit consists of: General purpose (scratch pad) registers AX, BX, CX and DX; Pointer registers SP (Stack Pointer) and BP (Base Pointer); index registers source index (SI) & destination index (DI) registers; the Flag register, the ALU to perform operations and a control unit with associated internal bus. The 16-bit scratch pad registers can be split into two 8-bit registers. AX AL, AH ; BX BL, BH; CX CL, CH; DX DL, DH.

Figure 1A

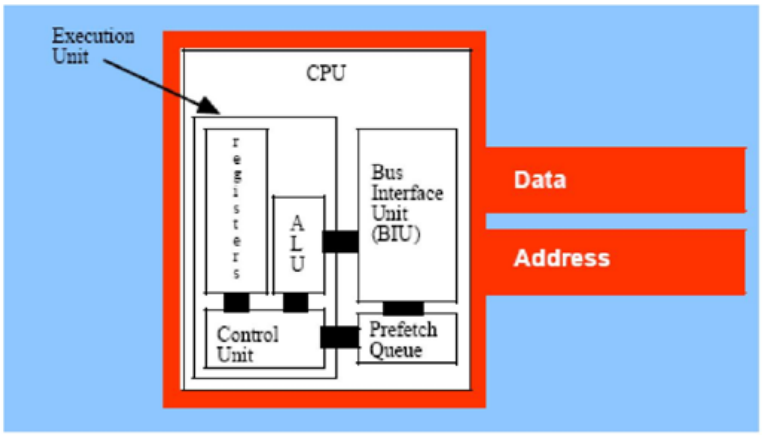
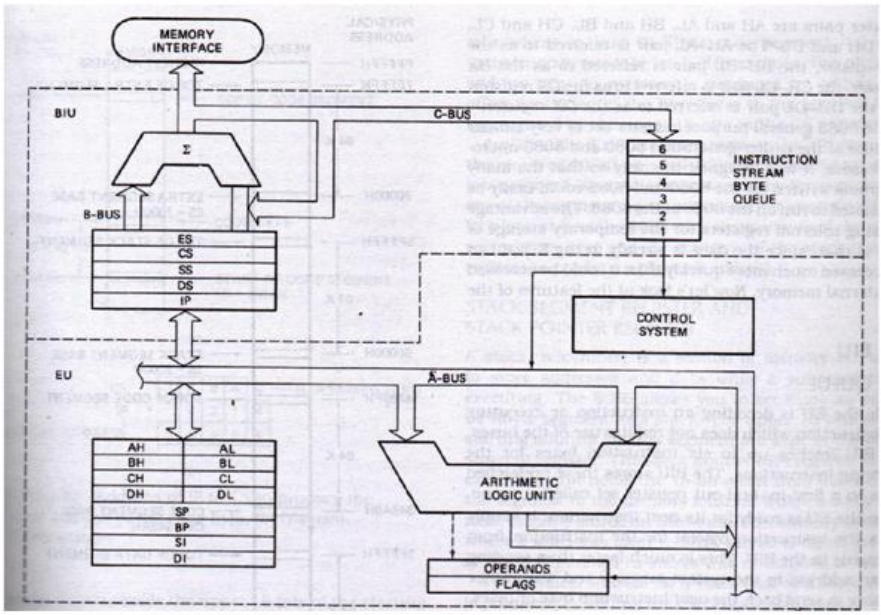


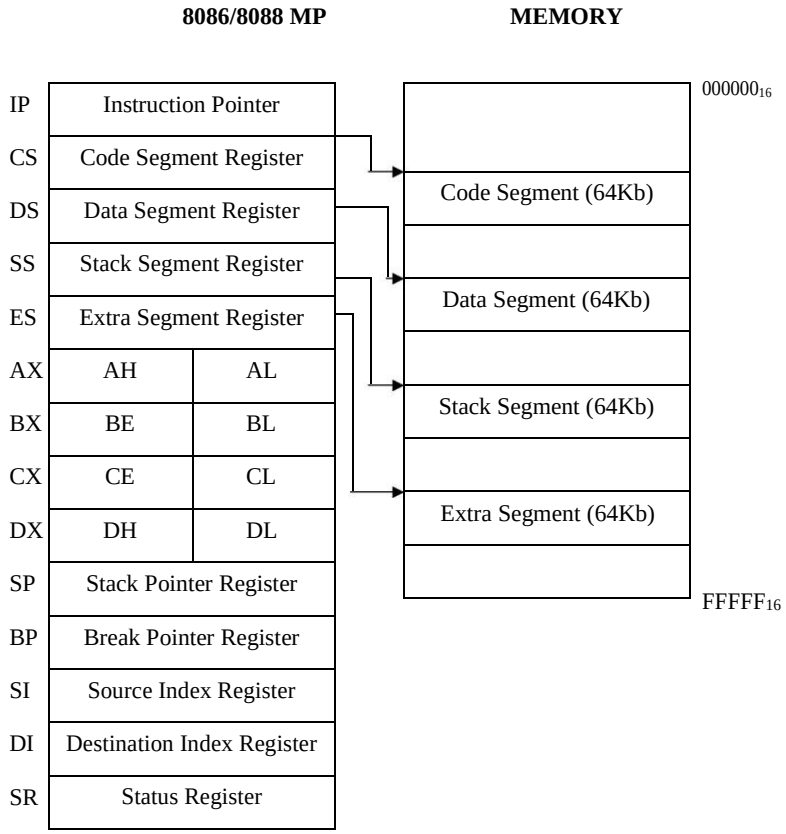
Figure 1B



Note: All registers are of size 16-bits.

Different registers and their operations are listed below:

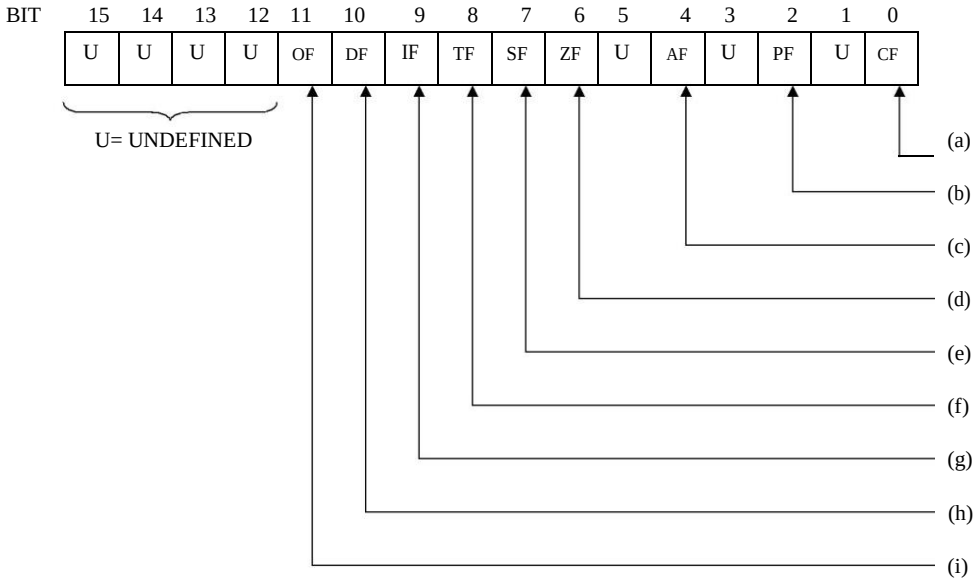
Register	**Uses/Operations**
AX	As accumulator in Word multiply & Word divide operations, Word I/O operations
AL	As accumulator in Byte Multiply, Byte Divide, Byte I/O, translate, Decimal Arithmetic
AH	Byte Multiply, Byte Divide
BX	As Base register to hold the address of memory
CX	String Operations, as counter in Loops
CL	As counter in Variable Shift and Rotate operations
DX	Word Multiply, word Divide, Indirect I/O


8086/8088 MP
MEMORY
IP
Instruction Pointer
CS
Code Segment Register
DS
Data Segment Register
SS
Stack Segment Register
ES
Extra Segment Register
AX
AH
AL
BX
BE
BL
CX
CE
CL
DX
DH
DL
SP
Stack Pointer Register
BP
Break Pointer Register
SI
Source Index Register
DI
Destination Index Register
SR
Status Register
Code Segment (64Kb)
Data Segment (64Kb)
Stack Segment (64Kb)
Extra Segment (64Kb)
000000_{16}
$FFFFF_{16}$

Execution of Instructions in 8086:

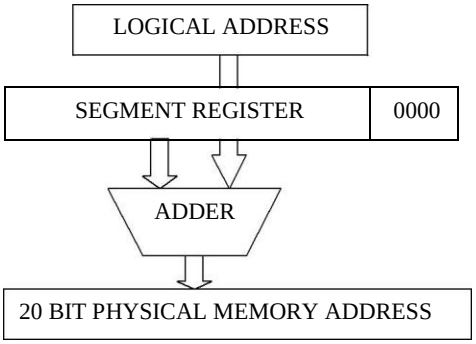
The microprocessor sends OUT a 20-bit physical address to the memory and fetches the first instruction of a program from the memory. Subsequent addresses are sent OUT and the queue is filled up to 6 bytes. The instructions are decoded and further data (if necessary) are fetched from memory. After the execution of the instruction, the results may go back to memory or to the output peripheral devices as the case may be.

8086 Flag Register format



(a) : CARRY FLAG – SET BY CARRY OUT OF MSB
(b) : PARITY FLAG – SET IF RESULT HAS EVEN PARITY
(c) : AUXILIARY CARRY FLAG FOR BCD
(d) : ZERO FLAG – SET IF RESULT = 0
(e) : SIGN FLAG = MSB OF RESULT
(f) : SINGLE STEP TRAP FLAG
(g) : INTERRUPT ENABLE FLAG
(h) : STRING DIRECTION FLAG
(i) : OVERFLOW FLAG

Generation of 20-bit Physical Address:



Programming Models:

Depending on the size of the memory the user program occupies, different types of assembly language models are defined.

TINY All data and code in one segment

SMALL one data segment and one code segment

MEDIUM one data segment and two or more code segments

COMPACT one code segment and two or more data segments

LARGE any number of data and code segments

To designate a model, we use ".MODEL" directive.

Assembly Language Development Tools:

1. EDITOR:

- It's a system software (program) which allows users to create a file containing assembly instructions and statements. Ex: Wordstar, DOS Editor, Norton Editor
- Using the editor, you can also edit/delete/modify already existing files.
- While saving, you must give the file extension as ".asm".
- Follow the AL syntax while typing the programs
- Editor stores the ASCII codes for the letters and numbers keyed in.
- Any statement beginning with semicolon is treated as comment.

When you typed your entire program, you have to save the file on the disk. This file is called "source" file, having an '.asm' extension. The next step is to convert this source file into a machine executable '.obj' file.

2. ASSEMBLER:

- An "assembler" is a system software (program) used to translate the assembly language mnemonics for instructions to the corresponding binary codes.
- An assembler makes two 'passes' thro' your source code. On the first pass, it determines the displacement of named data items, the offset of labels etc., and puts this information in a symbol table. On the second pass, the assembler produces the binary code for each instruction and inserts the offsets, etc., that is calculated during the first pass. The assembler checks for the correct syntax in the assembly instructions and provides appropriate warning and error messages. You have to open your file again using the editor to correct the errors and reassemble it using assembler. Unless all the errors are corrected, the program cannot be executed in the next step.
- The assembler generates two files from the source file; the first file, called the object file having an extension ".obj" which contains the binary codes for instructions and information about the addresses of the instructions. The second file is called "list file" with an extension "'.lst". This file contains the assembly language statements, the binary codes for each instruction, and the offset for each inst. It also indicates any syntax errors or typing errors in the source program.

Note: The assembler generates only offsets (i.e., effective addresses); not absolute physical addresses.

3. LINKER:

- It's a program used to join several object files into one large object file. For large programs, usually several modules are written and each module is tested and debugged. When all the modules work, their object modules can be linked together to form a complete functioning program.
- The LINK program must be run on ".obj" file.
- The linker produces a link file which contains the binary codes for all the combined modules. The linker also produces a link map file which contains the address information about the linked files.
- The linker assigns only relative addresses starting from zero, so that this can be put anywhere in physical primary memory later (by another program called 'locator' or 'loader'). Therefore, this file is called relocatable. The linker produces link files with ".exe" extension.

- Object modules of useful programs (like square root, factorial etc.) can be kept in a "library", and linked to other programs when needed.

4. LOADER:

- It's a program used to assign absolute physical addresses to the segments in the ".exe" file, in the memory. IBM PC DOS environment comes with EXE2BIN loader program. The ".exe" file is converted into ".bin" file.
- The physical addresses are assigned at run time by the loader. So, assembler does not know about the segment starting addresses at the time program being assembled.

5. DEBUGGER:

- If your program requires no external hardware, you can use a program called debugger to load and run the ".exe" file.
- A debugger is a program which allows you to load your object code program into system memory, execute the program and troubleshoot or debug it. The debugger also allows you to look at the contents of registers and memory locations after you run your program.
- The debugger allows you to change the contents of registers & memory locations and rerun the program. Also, if facilitates to set up "breakpoints" in your program, single step feature, and other easy-to-use features.
- If you are using a prototype SDK 86 board, the debugger is usually called "monitor program".

We would be using the development tool MASM 5.0 or higher version from Microsoft Inc. MASM stands for Microsoft Macro Assembler. Another assembler TASM (Turbo Assembler) from Borland Inc., is also available.

8255 Programmable Peripheral Interface:

8255 is a programmable peripheral IC which can be used to interface computer (CPU) to various types of external peripherals such as: ADC, DAC, Motor, LEDs, 7-segment displays, Keyboard, Switches etc. It has 3 ports A, B and C and a Control word register. User can program the operation of ports by writing appropriate 8-bit "control word" into the control word register.

Control Word format

Bits →	D7	D6	D5	D4	D3	D2	D1	D0
	1 for I/O	PA mode: 00 – mode 0, 01 – mode1, 10/11 – mode 2		PA direction 0 – output 1 – input	PCU direction 0 – output 1 – input	PB mode 0 – mode 0 1 – mode 1	PB direction 0 – output 1 – input	PCL direction 0 – output 1 – input

How to Write and execute your ALP using MASM?

Steps to be followed:

1. Type EDIT at the command prompt (C :\> \MASM\). A window will be opened with all the options like File, Edit etc., In the workspace, type your program according to the assembly language syntax and save the file with a ".asm" extension. (say test.asm)
2. Exit the Editor using File menu or pressing ALT + F + X.
3. At the prompt, type the command MASM followed by filename.asm (say, test.asm). Press Enter key 2 or 3 times. The assembler checks the syntax of your program and creates ".obj" file, if there are no errors. Otherwise, it indicates the error with line numbers. You have to correct the errors by opening your file with EDIT command and changing your instructions. Come back to DOS prompt and again assemble your program using MASM command. This has to continue until MASM displays "0 Severe Errors". There may still be "Warning Errors". Try to correct them also.
4. Once you get the ".obj" file from step 3, you have to create the".exe" file. At the prompt, type the command LINK followed by "filename.obj" (say, test.obj) and press Enter key. (Note that you have to give the extension now as ".obj" and not as ".asm"). If there are no linker errors, linker will create ".exe" file of your program. Now, your program is ready to run.

5. There are two ways to run your program.

a) If your program accepts user inputs thro' keyboard and displays the result on the screen, then you can type the name of the file at the prompt and press Enter key. Appropriate messages will be displayed.

b) If your program works with memory data and if you really want to know the contents of registers, flags, memory locations assigned, opcodes etc., then type CV test (file name) at the prompt. Another window will be opened with your program, machine codes, register contents etc., Now, you also get a prompt > sign within CV window. Here you can use "d" command to display memory contents, "E" command to enter data into memory and "g" command to execute your program. Also, you can single step through your program using the menu options. In many ways, CV (Code View) is like Turbo C environment.

Once you are familiar with the architecture and basics of assembly language tools, you can start typing and executing your program.

Instructions for Laboratory Exercises:

1. The programs with comments are listed for your reference. Write the programs in observation book.
2. Create your own subdirectory in the computer. Edit (type) the programs with program number and place them in your subdirectory. Have a copy of MASM.EXE, CV.EXE and LINK.EXE files in your subdirectory. You can write comments for your instructions using Semicolon (;) symbol.
3. Execute the programs as per the steps discussed earlier and note the results in your observation book.
4. Make changes to the original program according to the questions given at the END of each program and observe the outputs.
5. For part A programs, input-output is through computer keyboard and monitor or through memory.
6. For part B programs, you need an external interface board. Connect the board to the computer using the FRC available. Some boards may require external power supply also.
7. Consult the Lab In-charge/Instructor before executing part B experiments.
8. The assembler is not case sensitive. However, we have used the following notation: uppercase letters to indicate register names, mnemonics and assembler directives; lowercase letters to indicate variable names, labels, segment names, and models.

SOFTWARE PROGRAMS: PART A

EXP NO: 1

Binary Search

Aim:
Design and develop an assembly language program to search a key element "X" in a list of 'n' 16-bit numbers. Adopt Binary search algorithm in your program for searching.

Algorithm:

Step 1 : Declare the array
Step 2 : Input the array elements in the sorted order
Step 3 : Input the search element
Step 4 : Assign left as 0 and right as n-1
Step 5 : Find mid index = (left +right)/2
Step 6 : Compare mid element with search element
Step 7 : If search element <mid element assign right as mid-1
Step 8 : If search element >mid element, assign left as mid+1
Step 9 : If search element = mid element , the search is successful so display the location of the search element, go to step -12
Step 10 : Repeat step 5 to 9 until the search is successful
Step 11 : If search element is not available, display "element not available"
Step 12 : Terminate the program

Program:

```
.MODEL SMALL
.DATA
A1 DW 010H,020H,030H,040H ;Array declaration
LEN DW (LEN-A1)/2 ; Finding no of elements
KEY DW 20H; Key value
M1 DB 'KEY FOUND AT  '
RES DB ?
M3 DB '  POSITION  $'
M2 DB 'KEY NOT FOUND $'
.CODE
MOV AX,@DATA ; Initialize data segment
MOV DS,AX
MOV BX,01; Initialize left
MOV DX,LEN; Initialize right
MOV CX,KEY
L2:CMP BX,DX ; compare left and right
  JA FAIL
  MOV AX,BX ; Find mid index
  ADD AX,DX
  SHR AX,01
```

```
  MOV SI,AX
  DEC SI
  ADD SI,SI
  CMP CX,A1[SI] ; Compare mid element with key value
  JAE L1
  MOV DX,AX ;If Element< key  assign right is mid -1
  DEC DX
  JMP L2
L1: JE SUCCESS
   MOV BX,AX If Element> key  assign left is mid +1
   INC BX
   JMP L2
FAIL:LEA DX,M2
    JMP L3
SUCCESS: ADD AX,30H ; Display found message and position
       MOV RES,AL
       LEA DX,M1
L3: MOV AH,09H ; Display not fount message
   INT 21H
   MOV AH,04CH ; Terminate the program
   INT 21H
   END
```

Expected Output:

The message "the search element is available at the particular position" is displayed if "the search element is available" else it displays search element is "not available"

Result:

The program used the binary search algorithm to find a particular element from an array of elements and at a specific location.

Input	Output
Enter the number of elements in the array:5 Enter the array elements: 15 20 30 45 50 Search element: 20	Element available at location 2
Enter the number of elements in the array:6 Enter the array elements: 115 120 130 140 150 160 Search element: 15	Element not available

EXP No: 2

Bubble Sort

Aim:
Design and develop an assembly program to sort a given set of 'n' 16-bit numbers in ascending order. Adopt Bubble Sort algorithm to sort given elements.

Algorithm:

Step 1	:	Declare the array with the numbers that need to be sorted.
Step 2	:	Initialize iteration count (n-1)
Step 3	:	Initialize comparison counter
Step 4	:	Compare num1 and num2
Step 5	:	Num1<=num2 do not exchange
Step 6	:	Num1>=num2 then exchange the number positions
Step 7	:	Decrement iteration counter, comparison counter
Step 8	:	Terminate the program

Program:

```
.MODEL SMALL
.STACK
.DATA
NUM DB 54H,98H,77H,18H
COUNT DW 0004H
.CODE
MOV AX,@DATA
MOV BX,COUNT
DEC BX
MOV CX,BX
UP2: LEA SI,NUM
UP1 MOV AL,[SI]
CMP AL,[SI+1]
JC SKIP
XCHG AL,[SI+1]
MOV [SI],AL
SKIP:INC SI
DEC CX
JNZ UP1
DEC BX
MOV CX,BX
JNZ UP2
INT 03H
END
```

Expected Output:

Trace the program after the debug command –t to get the location of SI then type the command d ds:000c to find the declared array use the debug command –g for executing the program go to the same location to find the sorted array

Result: The 8086 assembly program sorts the declared array using bubble sort algorithm in ascending

order.

At source before execution	58	98	77	18

-g (to execute)

At source after execution	18	58	77	98

At source before execution	10	45	2	8

-g (to execute)

At source after execution	2	8	10	45

~~Result: The array is sorted in ascending order using bubble sort algorithm~~

EXP No: 3

Palindrome

Aim:

Develop an assembly language program to reverse a given string and verify whether it is a palindrome or not. Display the appropriate message.

Algorithm:

Step 1 : Create display macro to display the message
Step 2 : Declare the string
Step 3 : Declare the message to display
Step 4 : Find the reverse of string and store in string1
Step 5 : Is string=string1,display it is a palindrome
Step 6 : Else if display not a palindrome
Step 7 : Terminate the program

Program:

```
DISPLAY MACRO M ; Display macro
MOV AH,09
LEA DX,M
INT 21H
ENDM

.MODEL SMALL
.DATA
  STR DB 'MADAM' ; Declare the string
  LEN DW LEN-STR
  STR1 DB 10 DUP('$')
  M1 DB 'POLINDROME$' ; Declare the message to display
  M2 DB 'NOT POLINDROME$'
.CODE
   MOV AX,@DATA
   MOV DS,AX
   MOV ES,AX
   LEA DI,STR1  ;Find the reverse string and store in str1
   LEA SI,STR
   ADD SI,LEN
   DEC SI
   MOV CX,LEN
 L2:MOV AL,[SI]
   MOV [DI],AL
   INC DI
   DEC SI
   LOOP L2
   MOV CX,LEN
   LEA SI,STR
   LEA DI,STR1
   CLD
   REPE CMPSB ; Compare original and reverse string
   JNE L1
   DISPLAY M1 ; Display palindrome if equal
   JMP L3
 L1:DISPLAY M2 ; Display not palindrome if not equal
```

```
 L3: MOV AH,4CH
    INT 21H
END,
```

Expected output: The string declared is checked with the original and reverse of the string and if the original and the reverse is equal then it is palindrome else it is not a palindrome.

Result: The entered string is reversed and compared with the original string to see if it is a palindrome or not, appropriate messages are displayed

INPUT	**OUTPUT**
MADAM	PALINDROME
HELLO	NOT A PALINDROME

EXP No: 4

Compute ncr using recursive procedure

Aim:

Develop an assembly language program to compute nCr using recursive procedure. Assume that 'n' and 'r' are non-negative integers.

Algorithm:

Step 1	:	Initialize the values for n,r,res.
Step 2	:	Call ncr procedure
Step 3	:	If r=0,res=1 goto step
Step 4	:	Else r=r-1
Step 5	:	Subtract n-r
Step 6	:	Multiply (n-r)*res
Step 7	:	Res=(n-r)*res/2
Step 8	:	Return to step 2
Step 9	:	Save the result in res
Step 10	:	Terminate the program

Program:

```
MODEL SMALL
.DATA ; Initialize the values for n, r, res
  N DW 5
  R DW 3
  RES DW 0

.CODE
    MOV AX,@DATA; Initialize data segment
    MOV DS,AX
    MOV BX,N ; sent n value to bx
    MOV CX,R ; Set r value to cx
    CALL NCR ; Call ncr procedure
    MOV AH,4CH ; Terminate the program
    INT 21H
NCR PROC NEAR
   CMP CX,0 ; Check the value of cx is zero
   JE EXIT
   PUSH CX ; Push the cx value to stack and decrement cx
   DEC CX
   CALL NCR ; call ncr recursive
   POP CX
   MOV AX,BX
   INC AX
   SUB AX,CX ; subtract cx from ax
   MOV DX,00
   MUL RES ; multiply ax and res
   DIV CX
  MOV RES,AX ; move the value to res
  RET
```

```
EXIT: MOV RES,1; cx is zero set res is 1
   RET
NCR ENDP
END
```

Expected Result: The nCr is calculated using the recursive procedure. N and R are non-negative integers.

Result: For the value N=5, R=3 the result is =0AH

Input	**Output**
N=5 R=3(nCr)	**0AH**
N=6 R=3(nCr)	**0DH**

EXP NO: 5

Read the Current Time and Date from the System and Display

Aim:

Design and develop an assembly language program to read the current time and Date from the system and display it in the standard format on the screen.

Algorithm:

Step1	:	Create display message macro to display the message
Step2	:	Create dis macro for displaying two digits after converting to ascii.
Step 3	:	Create main program.
Step 4	:	Call display message.
Step 5	:	Use INT 21H function 02ch to get the system time.
Step 6	:	Call dis macro to display hours
Step 7	:	Call dis macro to display minutes
Step 8	:	Call dis macro to display seconds.
Step 9	:	The system date is displayed on the screen
Step 10		Terminate the program.

Program:

```
DISPLAY MACRO M1 ; Display message macro
MOV AH,09
LEA DX,M1
INT 21H
ENDM

DIS MACRO M ; Display macro for two digit numbers
    MOV AL,M
    AAM
    MOV BX,AX
    ADD BX,3030H ; Convert to ASCII
    MOV DL,BH
    MOV AH,02 ; Display left digit
    INT 21H ;Display right digit
    MOV DL,BL
    INT 21H
ENDM

.MODEL SMALL
.DATA
     STR DB ' CURRENT SYSTEM TIME IS  $'
.CODE
     MOV AX,@DATA
     MOV DS,AX
     DISPLAY STR
     MOV AH,02CH ; Read time form system
     INT 21H
     DIS CH    ; CH HOLDING HOURS
```

```
    MOV DL,':'
    INT 21H
    DIS CL   ; CL HOLDING MINUTES
    MOV DL,':'
    INT 21H
    DIS DH    ; DH HOLDING SECONDS
    MOV AH,4CH
    INT 21H
END
```

Expected output:

The current time from the system is displayed in the standard format on the screen.

Result:

CURRENT SYSTEM TIME IS 08:30:30

Input(current time)	**Output(time displayed)**
12:10:06	12:10:06
08:30:30	08:30:30

EXP NO: 6

ARM ASSEMBLY LANGUAGE PROGRAMS

Aim:
To write and simulate ARM assembly language programs for data transfer, arithmetic And logical operations (Demonstrate with the help of a suitable program).

Program:

a. Data Transfer

```
; PROGRAM TO TRANSFER DATA FROM CODE AREA TO DATA AREA
; COMPILE AND DEBUG PROGRAM
; SET BREAKPOINT AT NOP INSTRUCTION
; PRESS F5 TO RUN THE PROGRAM
; HAVE MEMORY1 WINDOW OPENED AND SET ADDRESS AT 0X40000000
; AND AFTER EXECUTION CHECK 0X44444444 APPEAR AT MEMORY1 WINDOW
AREA LARGEST , CODE, READONLY
ENTRY ;Mark first instruction to execute
START
LDR R1,=VALUE1 ; LOADS THE ADDRESS OF FIRST VALUE
LDR R2,[R1],#4 ; WORD ALIGN T0 ARRAY ELEMENT
LDR R4,=RESULT ; LOADS THE ADDRESS OF RESULT
STR R2,[R4] ; STORES THE RESULT IN R2
NOP
NOP
NOP
; ARRAY OF 32 BIT NUMBERS(N=7)
VALUE1
DCD 0X44444444 ;
AREA DATA2,DATA,READWRITE ; TO STORE RESULT IN GIVEN ADDRESS
RESULT DCD 0X0
END ; Mark end of file
```

b. Arithmetic

```
; program to add two words
; R1,=0X43210010 + R3,=0X43212102 = R4 = 0x8642212
; R0,=0X1234E640 + R2,=0X12348900 = R5 = 0x24696f40
EXPORT ADD64
AREA ADDITION , CODE , READONLY
ADD64
LDR R0,=0X1234E640
LDR R1,=0X43210010
LDR R2,=0X12348900
LDR R3,=0X43212102
ADDS R4,R1,R3
ADC R5,R0,R2
NOP
NOP
BX LR
END //Mark end of file
```

c. Logical

```
; PROGRAM TO DEMONSTRATE LOGICAL OR INSTRUCTION
```

```
; COMPILE AND DEBUG PROGRAM
; SET BREAKPOINT AT NOP INSTRUCTION
; PRESS F5 TO RUN THE PROGRAM
; STEP THROUGH THE PROGRAM AND FIND
; FIRST OPERNAD AT R2= 55AAAA55
; SECOND OPERNAD AT R3= 5555AA55
; AND AFTER EXECUTION CHECK REGISTER R2 = 5500AA55
; AND AT 0X40000000 5500AA55 AT MEMORY 1 WINDOW
AREA LARGEST , CODE, READONLY
ENTRY ;Mark first instruction to execute
START
LDR R1,=VALUE1 ; LOADS THE ADDRESS OF FIRST VALUE
LDR R2,[R1] ; WORD ALIGN T0 ARRAY ELEMENT
LDR R1,=VALUE2 ; LOADS THE ADDRESS OF FIRST VALUE
LDR R3,[R1]
AND R2,R3
LDR R4,=RESULT ; LOADS THE ADDRESS OF RESULT
STR R2,[R4] ; STORES THE RESULT IN R2 AND ATT MEMORY
0X40000000
NOP
NOP
NOP
; ARRAY OF 32 BIT NUMBERS(N=7)
VALUE1
DCD 0X55AAAA55 ;
VALUE2
DCD 0X5555AA55 ;
AREA DATA2,DATA,READWRITE ; TO STORE RESULT IN GIVEN ADDRESS
RESULT DCD 0X0
END ; Mark end of file
```

EXP NO: 7

C PROGRAMS FOR ARM MICROPROCESSOR USING KEIL

AIM: To write and simulate C Programs for ARM microprocessor using KEIL (Demonstrate with the help of a suitable program).

Program:

```
#include <lpc214x.h>
void main(void)
{
int a,b,c;
a=4;
b=5;
c=a+b;
a=5;
}
```

PART B

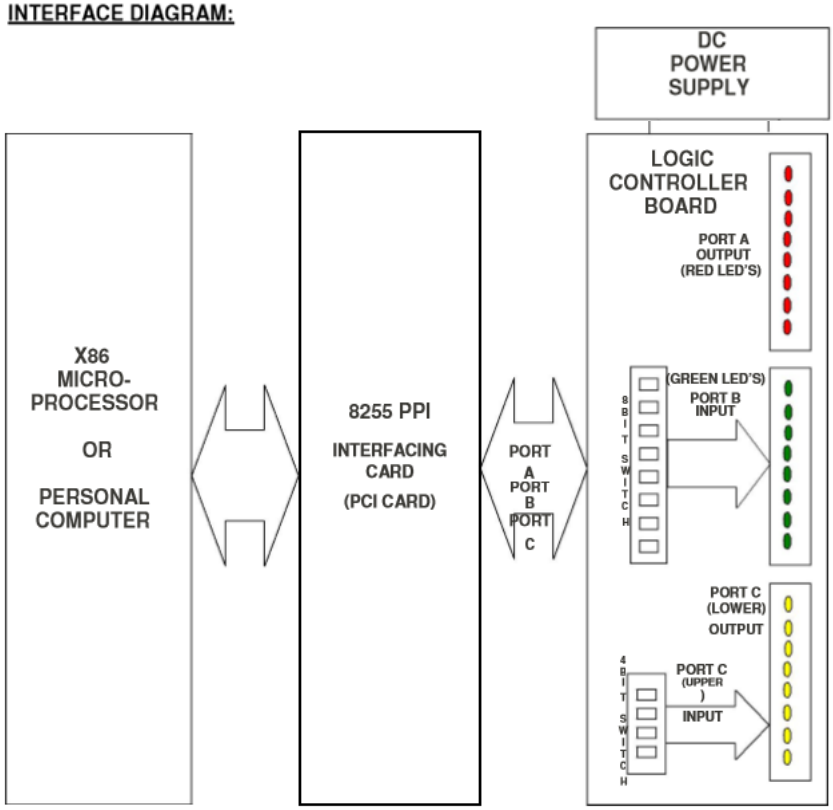
EXP 8A:

BCD Up-Down Counter (00-99) on the Logic Controller Interface.

Aim:
Design and develop an assembly program to demonstrate BCD Up-Down Counter (00-99) on the Logic Controller Interface.

PARITY CHECK USING LOGIC CONTROLLER

INTERFACE DIAGRAM:



Algorithm:

Step 1 : Create a delay macro
Step 2 : Create a up procedure to count form 0-9
Step 3 : Send the initial value 0 to the logic controller interface, sense a key

from the keyboard to stop
Step 4 : Call the up procedure
Step 5 Increment the value and send it to output until it reaches a value greater than 9
Step 6 : Call the down procedure to count from 9-0
Step 7 : Decrement the value to be sent to the output until it reaches the value 0
Repeat the steps from step2 ,until any key is pressed from the keyboard
Step 8 : Terminate the program

Program:

```
DELAY MACRO ; Delay macro
LOCAL D1,D2
PUSH BX
PUSH CX
MOV BX,0FFFFH ; Count of outer loop
D1: MOV CX,0FFFH count of Inner loop
D2: LOOP D2
DEC BX
JNZ D1
POP CX
POP BX
ENDM

.MODEL SMALL
.DATA
  PA EQU 0DOC0H ; Initialize port values
  PB EQU 0D0C1H
  CT EQU 0DOC3H
.CODE
   MOV AX,@DATA
   MOV DS,AX
   MOV DX,CT        ;Initialize 8255
   MOV AL,82H
   OUT DX,AL

   L2:CALL UP ; call up Procedure
   CALL DOWN        ; Call down procedure
   JMP L2
STOP:MOV AH,04CH        ;Terminate the program
    INT 21H
UP PROC NEAR
   MOV AL,00H
  L3:MOV DX,PA      ; Send the value to port A from 0-9
   OUT DX,AL
   INC AX
   PUSH AX
   MOV AH,0BH       ; Sense the key is pressed
   INT 21H
   OR AL,AL
   JNZ STOP
   DELAY
```

```
  POP AX
  CMP AL,0AH        ;repeat the loop if value less than 10
  JNE L3
  RET
UP ENDP

DOWN PROC NEAR
  MOV AL,09H
L4:MOV DX,PA        ; send the values to port A from 9-0
  OUT DX,AL
  DEC AX
  PUSH AX
  MOV AH,0BH        ; Sense the key is pressed
  INT 21H
  OR AL,AL
  JNZ STOP
  DELAY
  POP AX
  CMP AL,00H ;Repeat the loop the value is greater than 0
  JNE L4
  RET
DOWN ENDP
   END
```

Expected Result: The LED of the logic controller glows form 0-9 and -9-0 and so on until key is pressed on the keyboard to stop.

Result: The output shows the behavior of a BCD UP-DOWN Counter.

EXP 8B:

Display X*Y on the Logic Controller Interface.

Aim:

Design and develop an assembly program to read the status of two 8-bit inputs (X & Y) from the Logic Controller Interface and display X*Y.

Algorithm:

Step1 : Give the input value of X from the interface
Step 2 : Read the input from the logic controller
Step 3 : Repeat step1 and step2 for y value
Step 4 : Multiply X*Y
Step 5 : The result is shown on the LED of the interface, first the LSB is shown on the led then the MSB is seen on the led
Step 6 : Sense any key on the keyboard
Step 7 : Terminate the program

Program:

```
DISMSG MACRO M ; display message macro
LEA DX,M
MOV AH,09
INT 21H
ENDM
WAIT1 MACRO ; waiting for setting  input
MOV AH,01
INT 21H
ENDM

.MODEL SMALL
.DATA
.STACK
M1 DB 'SET X VALUE AND ENTER$' ; Initialize strings
M2 DB 'SET Y VALUE AND ENTER$'
M3 DB 'LOWER BYTE FORM PORTA AND ENTER FOR HIGHER BYTE$'
M4 DB 'HIGHER BYTE FROM PORTA$'
PORTA EQU 0E880H          ; Initialize port numbers
PORTB EQU 0E881H
CTRL EQU 0E883H
.CODE
MOV AX,@DATA; Initialize data segment
MOV DS,AX
MOV DX,CTRL ; Initialize 8255 control word
MOV AL,82H
OUT DX,AL
DISMSG M1 ;Display message macro
WAIT1          ;wait macro
MOV DX,PORTB ; Read x value from port B
```

```
IN AL,DX
MOV BL,AL
DISMSG M2
WAIT1
MOV DX,PORTB ;Read y value from port B
IN AL,DX
MOV AH,00
MUL BL        ; Multiply  x and y
PUSH AX
DISMSG M3
POP AX
MOV DX,PORTA     ; Display lower byte from port A
OUT DX,AL
MOV BL,AH
WAIT1
DISMSG M4
MOV DX,PORTA     ; Display higher byte from port A
MOV AL,BL
OUT DX,AL
MOV AH,04CH       ; Terminate program
INT 21H
END
```

Expected Result:
The input of x and y is given from the input port by switching on/off the 8 switches e.g. x=00000010 y=00000100, x*y=00001000 that is x=2,y=4 x*y=8. The result is specified by the equivalent glowing of LED'S

Result: The result is seen at the output port of the interface ,the LED'S glow according the result of multiplication the lower byte is displayed first, press enter the higher byte is displayed.

Input given to the logic controller	Output displayed on the logic controller
X=00000101, Y=00000010	X*Y=00010000
X=00000011 ,Y=00000101	X*Y=00010101

EXP 9:

Display Messages Alternatively

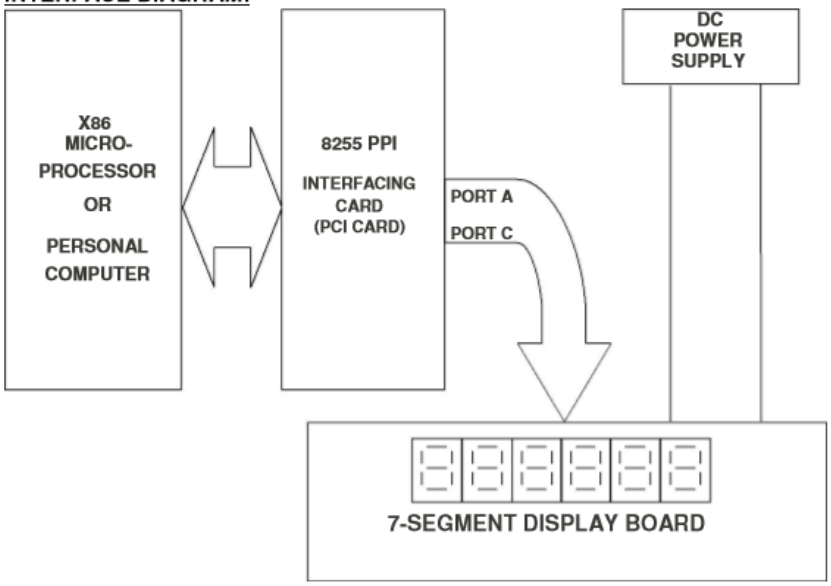
Aim: Design and develop an assembly program to display messages "FIRE" and "HELP" alternately with flickering effects on a 7-segment display interface for a suitable period of time. Ensure a flashing rate that makes it easy to read both the messages (Examiner does not specify these delay values nor is it necessary for the student to compute these values).

Algorithm:

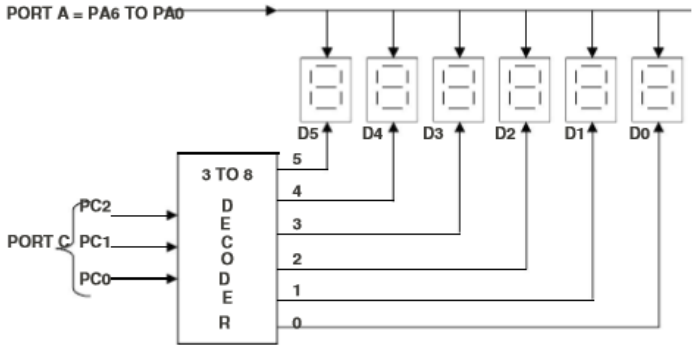
Step1	:	Create delay macro, initialize count to display no of times and the number of characters.
Step 2		Declare the 7 segment codes of the characters that is to be displayed(FIRE BLANK HELP)
Step3	:	Call display procedure to display FIRE
Step4	:	Call display procedure to display BLANK
Step5	:	Call display procedure to display HELP
Step 6		Select the 7-segment position, send data to be displayed
Step7	:	Sense if any key is pressed on the keyboard
Step8	:	Call delay
Step 9		Repeat to display all characters step 2 to step 6

7-SEGMENT DISPLAY EXPERIMENTS

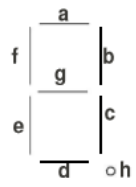
INTERFACE DIAGRAM:



7-SEGMENT DISPLAY BOARD INTERNAL CONNECTIONS:



7-SEGMENT LED DISPLAY:



7-SEGMENT CODES

Character	h	g	f	e	d	c	b	a	In Hex
F	0	1	1	1	0	0	0	1	71H
I	0	0	0	0	0	1	1	0	06H
R	0	1	1	1	0	1	1	1	77H
E	0	1	1	1	1	0	0	1	79H
H	0	1	1	1	0	1	1	0	76H
E	0	1	1	1	1	0	0	1	79H
L	0	0	1	1	1	0	0	0	38H
P	0	1	1	1	0	0	1	1	73H

CONTROL WORD:

D7	D6	D5	D4	D3	D2	D1	D0	
1	GA	GA	PA	PC_U	GB	PB	PC_L	
1	0	0	0	0	0	0	0	= 80H

All ports are configured as I/O ports in Mode-0.
Port A & Port C are output ports.

Program:

```
DELAY MACRO             ; Delay macro
PUSH BX
PUSH CX
MOV BX,02                       ; set outer loop
D1: MOV CX,0FFFFH               ; Set inner loop
D2: LOOP D2
DEC BX
JNZ D1
POP CX
POP BX
ENDM

.MODEL SMALL
.STACK 64
.DATA
N1 DB 071H,06H,050H,079H        ; Values for FIRE
N2 DB 076H,079H,038H,073H ; Values for HELP
N3 DB 00H,00H,00H,00H    ; Values for clear
PORTA EQU 0D0C0H        ; Initialize port values
PORTB EQU 0DOC1H
PORTC EQU 0DOC2H
CTRL EQU 0DOC3H
.CODE
    MOV AX,@DATA
    MOV DS,AX
    MOV AL,80H    ; Initialize 8255
    MOV DX,CTRL
    OUT DX,AL
```

```
L3:   MOV BX,OFFSET N1
      CALL DISPLAY          ; Display FIRE
      MOV BX, OFFSET N3
      CALL DISPLAY          ; Clear LEDs
      MOV BX,OFFSET N2
      CALL DISPLAY
      MOV BX, OFFSET N3 ; Display HELP
      CALL DISPLAY
      JMP L3
STOP: MOV AH,4CH        ; Terminate Program
      INT 21H

DISPLAY PROC NEAR
      MOV CX,0CFH ;Display no.of times
L2:   PUSH CX
      PUSH BX
      MOV CX,04H
      MOV AL,05H
L1:   MOV DX,PORTC ;Select the seven segment position
      OUT DX,AL
      DEC AX
      PUSH AX
      MOV AL,[BX]
      MOV DX,PORTA ; send data to display
      OUT DX,AL
      INC BX
      MOV AH,0BH ; Sense the keyboard
      INT 21H
      OR AL,AL
      JNZ STOP ; key pressed go to stop
      PUSH CX
DELAY ; call delay for 2ms
      POP CX
      POP AX
      DEC CX
      JNZ L1 ; repeat to display all characters
      POP BX
      POP CX
      DEC CX
      JNZ L2 ; display the specific times
      RET
    DISPLAY ENDP
      END
```

Expected result: The 7 segment interface needs to display FIRE and HELP blinking alternately.

Result:

Input data to be displayed	Output data to be displayed
FIRE HELP on the 7 segment display(BLINKING)	FIRE HELP on the 7 segment display(BLINKING)

EXP 10:

Stepper Motor in Both Directions

Aim:

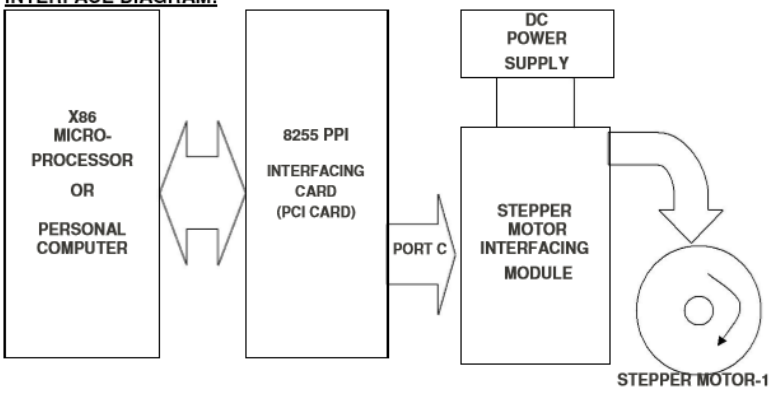
Design and develop an assembly program to drive a Stepper Motor interface and rotate the motor in specified direction (clockwise or counter-clockwise) by N steps (Direction and N are specified by the examiner). Introduce suitable delay between successive steps. (Any arbitrary value for the delay may be assumed by the student).

Algorithm:

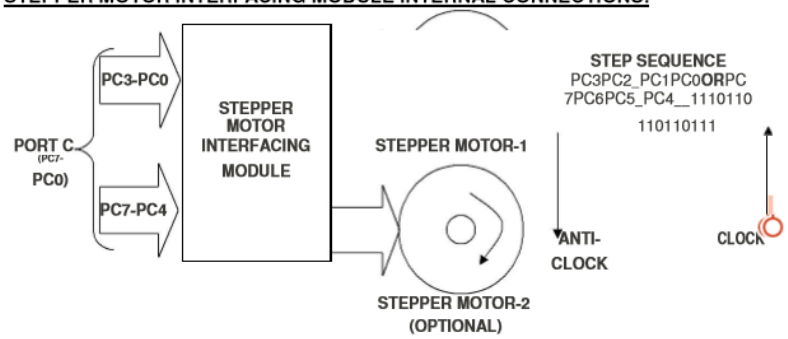
Step1 : Create delay macro
Step2 : Initialize the steps in which the stepper motor should rotate.
Step2 : Set the phase value
Step3 : Send the values to the motor to rotate
Step4 : Call delay between steps
Step5 : Set the direction by rotating the value of phase
Step 7 : Repeat step 2 to step 5 until steps=0
Step 6 : Terminate the program

STEPPER MOTOR EXPERIMENTS

INTERFACE DIAGRAM:



STEPPER MOTOR INTERFACING MODULE INTERNAL CONNECTIONS:



Program:

```
DELAY MACRO; Delay macro for few seconds
PUSH BX
PUSH CX
MOV BX,09FFH ; Outer loop value
D1: MOV CX,0FFFFH ; Inner loop value
D2: LOOP D2 inner loop
DEC BX
JNZ D1; Repeat outer loop
POP CX
POP BX
ENDM
.MODEL SMALL
.STACK 20
.DATA
NSTEPR DB 20; initialize the steps
PORTC EQU 0E882H ; Initialize the port nos.
CTR EQU 0E883H
```

```
.CODE
MOV AX,@DATA ; Initialize data segment
MOV DS,AX
MOV DX,CTR
MOV AL,80H ; Initialize 8255 control word value
OUT DX,AL
MOV AL,77H ; set the phase value
L1:MOV DX,PORTC ; send the values to rotate
  OUT DX,AL
DELAY ; Delay between the steps
  ROL AL,01 ; ROR AL,01 FOR CLOCK DIRECTION
  DEC NSTEPR
  JNZ L1 ; Repeat for no of steps
  MOV AH,4CH ; Terminate the program
  INT 21H   END
```

Expected Result:

The stepper motor rotates in clockwise or anti-clock wise direction in specified number of steps.

Result:

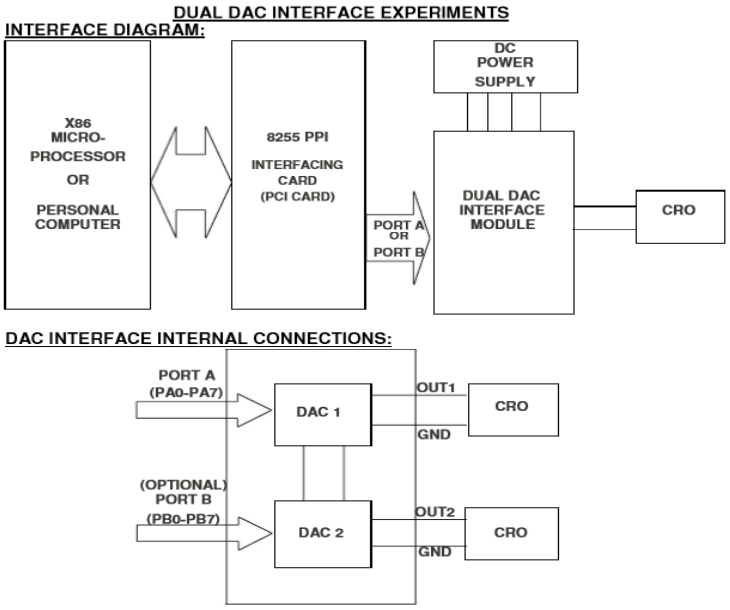
Input no of steps the motor rotates	output
5	The motor rotates in steps of 5
6	The motor rotates in steps of 6

EXP 11A:

Generate Sine-Wave

Aim:

Design and develop an assembly language program to generate the Sine Wave using DAC interface (The output of the DAC is to be displayed on the CRO).



Algorithm:

Step1	:	Initialize the values for generating Sine Wave.
Step2	:	Initialize count to the number of values to be plotted.
Step3	:	Send the values to DAC and plot on CRO.
Step4	:	Decrement count.
Step5	:	Repeat step3 until count becomes zero.
Step6	:	Sense the keyboard input to stop the program
Step7	:	Terminate the program

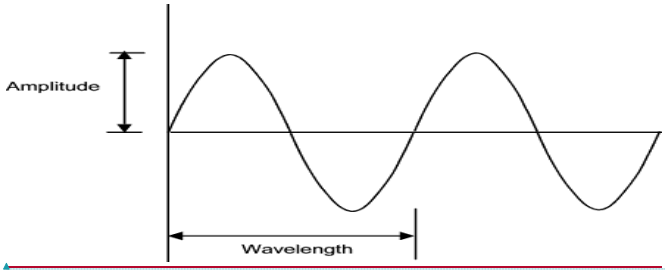
Program:

```
.MODEL SMALL
.DATA
PORTA EQU 0D0C0H
PORTB EQU ODOC1H
PORTC EQU 0D0C2H
```

```
CWR EQU 0D0C3H
SINES DB 00,11,22,33,43,53,63,72,81,89,97,104,109,115,119,122,125,126,127
MSG DB 10,13,'OBSERVE SINE WAVE ON CRO; PRESS ANY KEY TO EXIT',10,13,'$'
.CODE
MOV AX,@DATA
MOV DS,AX
MOV DX,CWR
MOV AL,80H
OUT DX,AL
LEA DX,MSG
MOV AH,9H
INT 21H
MOV DX,PORTA
FULL_WAVE:MOV SI,OFFSET SINES
MOV CX,13H
FIRST_QUART: MOV AL,7FH
MOV BL,BYTE PTR[SI]
ADD AL,BL
OUT DX,AL
INC SI
LOOP FIRST_QUART
MOV CX,12H
DEC SI
SECOND_QUART:
MOV AL,7FH
MOV BL,BYTE PTR[SI]
ADD AL,BL
OUT DX,AL
DEC SI
LOOP SECOND_QUART
MOV SI,OFFEST SINES
MOV CX,13H
THIRD_QUART:
MOV AL,7FH
MOV BL,BYTE PTR[SI]
SUB AL,BL
OUT DX,AL
DEC SI
LOOP FOURTH_QUART
MOV AH,1
INT 16H
JNZ STOP
JMP FULL_WAVE
STOP:MOV AH,4CH
INT 21H
END
```

Expected Result: Sine wave is generated using DAC interface and output is observed on the CRO

Result:



Sine wave is generated using the DAC interface.

Formatted: Font: (Default) Times New Roman, Font color: Blue

EXP 11B:

HALF RECTIFIED SINE WAVEFORM USING THE DAC

Aim:

Generate a Half Rectified Sine waveform using the DAC interface. (The output of the DAC is to be displayed on the CRO).

Algorithm:

Step 1 : Initialize the values for generating Half Rectified Sine Wave.
Step 2 : Initialize count to the number of values to be plotted.
Step 3 : Send the values to DAC and plot on CRO.
Step 4 : Decrement count.
Step 5 : Repeat step3 until count becomes zero.
Step 6 : Sense the keyboard input to stop the program
Step 7 : Terminate the program

Program:

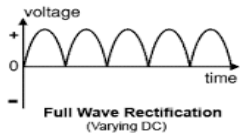
```
.MODEL

.DATA

SINES DB

00,22,44,66,87,108,127,146,164,180,195,209,221,231,240,246,251,254,255

MSG DB 10,13,10,'OBSERVE HALF RECTIFIED WAVE ON CRO.PRESS ANY KEY TO EXIT$'

PORTA EQU 0D0C0H

PORTB EQU ODOC1H

PORTC EQU ODOC2H

CTRL EQU 0D0C3H

.STACK

.CODE

MOV AX,@DATA

MOV DS,AX

LEA DX,MSG

MOV AH,9

INT 21H

MOV AL,80H

MOV DX,CTRL

OUT DX,AL
```

```
CALL DELAY
HALF_WAVE
MOV DX,PORTA
MOV CX,13H
MOV SI,OFFSET SINES
FIRST_QUART:
MOV AL,BYTE PTR[SI]
OUT DX,AL
CALL DELAY
INC SI
LOOP FIRST_QUART
DEC SI
MOV CX,12H
SECOND_QUART:
MOV AL,BYTE PTR[SI]
OUT DX,AL
CALL DELAY
DEC SI
LOOP SECOND_QUART
MOV CX,25H
NO_WAVE
MOV AL,00H
OUT DX,AL
CALL DELAY
LOOP NO_WAVE
MOV AH,1
STOP
INT 16H
JNZ STOP
JMP HALF_WAVE
STOP:MOV AH,4CH
INT 21H
```

```
DELAY PROC NEAR
PUSH CX
MOV CX, 2FFFH
BACK: NOP
LOOP BACK
POP CX
RET
DELAY ENDP
END
```

Expected output: The Half Rectified Sine waveform is observed in the CRO.

Result:



Formatted: Font: (Default) Times New Roman, 13.5 pt, Font color: Blue

EXP NO: 12

INTERFACE LCD WITH ARM PROCESSOR

Aim:

To interface LCD with ARM processor-- ARM7TDMI/LPC2148. Write and execute programs in C language for displaying text messages and numbers on LCD.

Algorithm:

Program:

```
#include<lpc214x.h>
#include<stdio.h>

//Function prototypes void
    lcd_init(void); void
    wr_cn(void); void
    clr_disp(void);
void delay(unsigned int); void
    lcd_com(void);
void wr_dn(void); void
    lcd_data(void);
    unsigned char temp1;
unsigned long int temp,r=0;
unsigned char *ptr,disp[] = "GCEM BENGALURU",disp1[]="LCD
                                        INTERFACING";

int main()
{
PINSEL0 = 0X00000000;           // configure P0.0 TO P0.15 as GPIO
IO0DIR = 0x000000FC;            //configure o/p lines for lcd [P0.2-
                                P0.7]
lcd_init();                     //lcd initialisation
delay(3200);                    // delay 1.06ms
clr_disp();                     //clear display
delay(3200);                    // delay 1.06ms
    temp1 = 0x81;               //Display starting address of first line 2nd pos
lcd_com();                      //function to send command to lcd
        ptr = disp;             // pointing data

while(*ptr!='\0')
{
temp1 = *ptr;
lcd_data();                     //function to send data to lcd ptr ++;
}

 temp1 = 0xC0;                  // Display starting address of second line 1st pos
    lcd_com();                  //function to send command to lcd

    ptr = disp1;                // pointing second data

        while(*ptr!='\0')
    {
```

```
temp1 = *ptr;
lcd_data();                                  //send data to lcd ptr ++;
}

while(1);
} //end of main()


// lcd initialisation routine. void lcd_init(void)
{
temp = 0x30;                        //command to test LCD voltage level wr_cn();
delay(3200);
temp = 0x30;                        //command to test LCD voltage level  wr_cn();
delay(3200);
temp = 0x30;                        //command to test LCD voltage level wr_cn();
delay(3200);
temp = 0x20;                        // change to 4 bit mode from default 8 bit
                                    mode
wr_cn(); delay(3200);
temp1 = 0x28;                       // load command for lcd function setting
                                      with lcd in 4 bit mode, lcd_com();       // 2 line and
     5x7 matrix display delay(3200);
temp1 = 0x0C;                       // load a command for display on, cursor on
                                          and blinking off
lcd_com(); delay(800);
temp1 = 0x06;                       // command for cursor increment after data
                                    dump
lcd_com(); delay(800);
temp1 = 0x80;                       // set the cursor to beginning of line 1 lcd_com();
delay(800);
}

void lcd_com(void)
{
temp = temp1 & 0xf0; //masking higher nibble first wr_cn();
temp = temp1 & 0x0f; //masking lower nibble temp = temp << 4;
wr_cn();
delay(500);                         // some delay
}

// command nibble o/p routine
```

```
void wr_cn(void)                    //write command reg
{
IO0CLR = 0x000000FC; // clear the port lines.
IO0SET = temp; // Assign the value to the PORT lines IO0CLR = 0x00000004; // clear bit RS = 0
IO0SET = 0x00000008; // E=1
delay(10);
IO0CLR = 0x00000008; //E=0
}
                                    // data nibble o/p routine
void wr_dn(void)                    ////write data reg
{
IO0CLR = 0x000000FC; // clear the port lines.
IO0SET = temp;                      // Assign the value to the PORT lines IO0SET =
    0x00000004; // set bit RS = 1

IO0SET = 0x00000008; // E=1
delay(10);
IO0CLR = 0x00000008; //E=0
}

// data o/p routine which also outputs high nibble first
// and lower nibble next void
    lcd_data(void)
{
temp = temp1 & 0xf0; //masking higher nibble first temp = temp ;
wr_dn();
temp= temp1 & 0x0f; //masking lower nibble temp= temp << 4;
                                    //shift 4bit to left wr_dn();
delay(100);
}

void clr_disp(void)                 // function to clear the LCD screen
{
temp1 = 0x01; lcd_com();
    delay(500);
}

void delay(unsigned int r1) // delay function using for loop
{
for(r=0;r<r1;r++);
}
```

EXP NO: 13

TO INTERFACE STEPPER MOTOR WITH ARM PROCESSOR

Aim:

To interface Stepper motor with ARM processor-- ARM7TDMI/LPC2148. Write a program to rotate stepper motor.

```
// the coils. Port lines : P1.20 to P1.23
#include <LPC21xx.h>

void clock_wise(void) ;
void anti_clock_wise(void) ;
    unsigned int var1 ;
unsigned long int i = 0 , j = 0 , k = 0 ;

int main(void)
{
        PINSEL2 = 0x00000000;                       //P1.20 to P1.23 GPIO
            IO1DIR |= 0x00F00000 ;                  //P1.20 to P1.23 made as output
    while(1)
{
        for( j = 0 ; j < 50 ; j++ )                 // 50 times in Clock wise Rotation
            clock_wise() ;                          // rotate one round clockwise
        for( k = 0 ; k < 65000 ; k++ ) ; // Delay to show anti_clock Rotation for( j=0 ; j < 50 ; j++ )   // 50
            times in Anti Clock wise Rotation anti_clock_wise() ;   // rotate one round anticlockwise
        for( k = 0 ; k < 65000 ; k++ ) ; // Delay to show ANTI_clock Rotation
}
}   // End of main

void clock_wise(void)
{
        var1 = 0x00080000;                          //For Clockwise

for( i = 0 ; i <= 3 ; i++ )                         // for A B C D Stepping
{
        var1 <<= 1 ;
        IO1CLR =0x00F00000 ;                        //clearing all 4 bits
            IO1SET = var1 ;                         // setting particular bit
            for( k = 0 ; k < 3000 ; k++ ); //for step speed variation
}
}
void anti_clock_wise(void)
{
        var1 = 0x00800000 ;                         //For
            Anticlockwise IO1CLR =0x00F00000 ;      //clearing all 4
            bits IO1SET = var1 ;
        for( k = 0 ; k < 3000 ; k++ ) ;
```

```
for( i = 0 ; i < 3 ; i++ )                                  // for A B C D Stepping
{
        var1 >>=1; //rotating bits
        IO1CLR =0x00F00000 ;                                // clear all bits before setting
             IO1SET = var1 ;                                // setting particular bit
        for( k = 0 ; k < 3000 ; k++ ) ; //for step speed variation
}}
```