

**DEPARTMENT OF COMPUTER OF SCIENCE & ENGINEERING  
R.L.JALAPPA INSTITUTE OF TECHNOLOGY  
KODIGEHALLI, DODDABALLAPUR - 561023**



**MICROPROCESSOR & MICROCONTROLLER  
LAB MANUAL**

**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)**

**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 gains 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:**

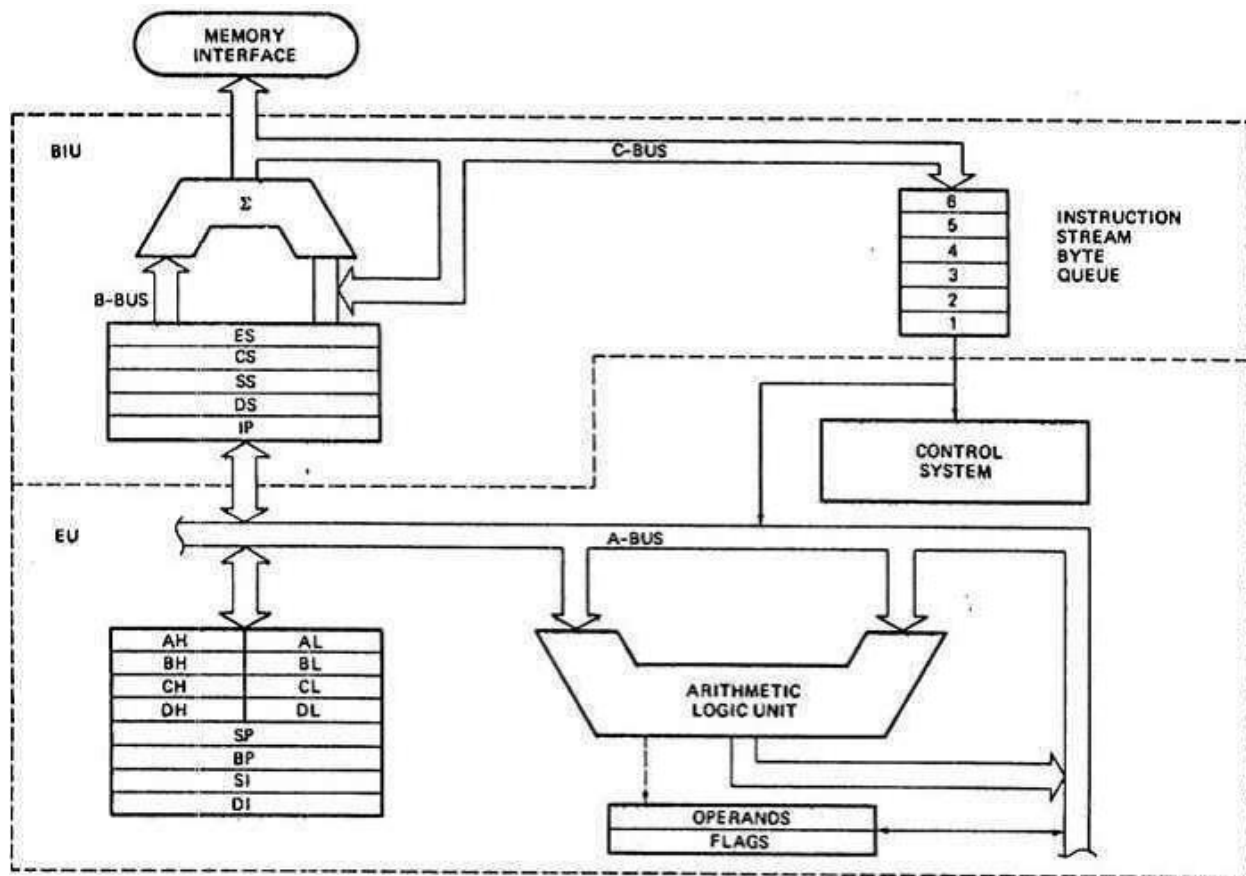
- All laboratory experiments (all 7 + 6 nos) are to be included for practical examination.
- Students are allowed to pick one experiment from each of the lot.
- Strictly follow the instructions as printed on the cover page of answer script for breakup of marks
- PART –A: Procedure + Conduction + Viva: 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.

**Steps to Edit a Program:**

1. Windows key + R
2. Type command
3. CD\
4. CD Folder\_Name
5. CD MASM
6. EDIT Filename.asm

**Commands to Run a Program**

1. MASM Filename.asm;
2. LINK Filename.obj;
3. filename (or ) AFDEBUG filename

Architecture of 8086:

Internal structure of microprocessor is divided into two sections

1. Bus Interface unit (BIU)
2. Execution Unit (EU)

Bus interface unit: it accesses memory and peripherals.

Execution unit: It executes the instructions previously fetched.

**Instruction Queue:**

- It contains pre-fetched instructions.
- Instruction queue in 8088 is 4 bytes long
- Instruction queue in 8086 is 6 bytes long.
- Fetching next Instruction while current instruction is being executed is called **Pipelining**.

**Registers:**

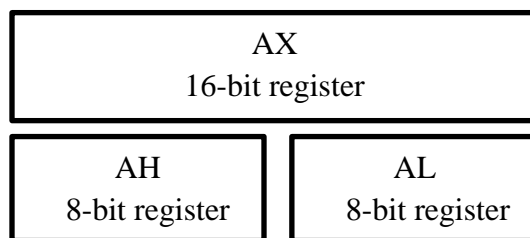
- In CPU, registers are used to store data temporarily.
- There are six types of registers in 8088/ 8086 as shown in below table.

**Registers of 8088/ 8086 by category**

| Category    | Bits | Register Names   |
|-------------|------|--|
| General     | 16   | AX, BX, CX, DX   |
|             | 8    | AH, AL, BH, BL, CH, CL, DH, DL   |
| Pointer     | 16   | SP (stack pointer), BP (Base Pointer)  |
| Index       | 16   | SI (source Index), DI (destination Index)                                    |
| Segment     | 16   | CS(code segment), DS (data segment)<br>SS (stack segment),ES (extra segment) |
| Instruction | 16   | IP (instruction pointer)   |
| Flag        | 16   | FR (flag register)   |

### General purpose registers:

- General purpose registers can be used as either 16-bit or 8-bit registers as shown below
- There are four general purpose registers AX, BX, CX, DX



AH-AL → AX register  
 BH-BL → BX register  
 CH-CL → CX register  
 DH-DL → DX register

### AX (Accumulator):

- AX is used to store the results of the arithmetic and logic operations.

### BX (Base Index register):

- BX register is used to hold the OFFSET address of DATA segment.
- BX is addressable as BH and BL

### CX (counter):

- CX is used to hold the count for various instructions.

### DX (data register):

- DX holds the part of the result from multiplication
- It holds the dividend for division

### Segment Registers:

- Four segment registers in BIU are used to hold the upper 16 bits of the starting addresses of a Segment.

CS, DS, ES, SS → Segment Registers.

- The segment will always start at an address with zero's in the lowest 4 bits.
- The part of the segment starting address stored in a segment register is often called Segment Base.

### Instruction Pointer:

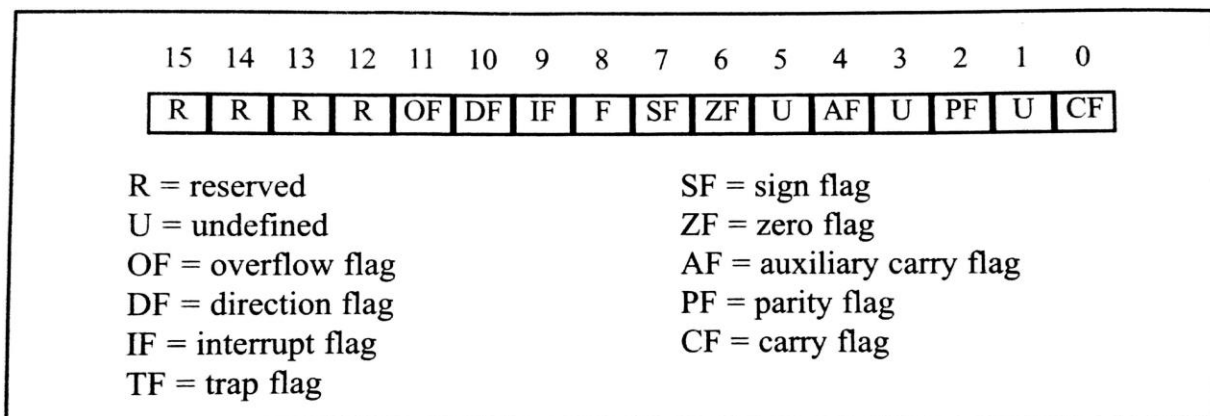
- The Instruction Pointer points to the next instruction

### Pointer and Index registers:

- SP, BP, SI, DI are the different Pointer and Index Registers.
- These registers can be used for temporary storage of data just as the general purpose registers described above.
- However, their main use is to hold the 16-bit offset of a data word in one of these segments.

### Flag register:

- A 16-bit flag register indicates some condition produced by the execution of an instruction or controls certain operations of the EU.
- It is a 16-bit register.
- It contains **9 active flags**.



**Figure 1-5. Flag Register**

### 8086 flag register:

- 8086 MP has 16-bit flag register.
- It has six conditional flags and three control flags



**Conditional flags:**

The six conditional flags are

1. Carry flag (CF)
2. Parity flag (PF)
3. Auxiliary carry flag (AF)
4. Zero flag (ZF)
5. Sign flag (SF)
6. Overflow flag (OF)

**Carry flag:**

- Carry flag holds the carry after addition or the borrow after subtraction.
- It also indicates the error conditions.

**Parity flag:**

- Parity is the number of ones in a binary number expressed as even or odd.
- Parity flag is 0 for odd parity and logic 1 for even parity.

**Auxiliary carry flag:**

- Auxiliary carry flag holds the half carry after addition or the borrow after subtraction.

**Zero flag:**

- The Zero flag shows that the result of an arithmetic or logic operation is zero.
- If Z=1, the result is zero, if Z=0 the result is not zero.

**Sign flag:**

- The sign flag holds the sign of the result
- If S=1 the sign is negative, if S=0 the sign is positive.

**Overflow flag:**

- Overflow flag indicates that the result of arithmetic operation has exceeded the capacity of the machine.

**Control flags:**

The three control flags are

1. Trap flag (TF)
2. Interrupt flag (IF)

### 3. Direction flag (DF)

**Trap flag:**

- Trap flag enables debugging feature.
- If T=1 debugging is enabled, if T=0 debugging is disabled.

**Interrupt flag:**

- Interrupt flag controls the operation of interrupt request.
- If I=1, interrupt request is enabled, if I=0 interrupt request is disabled.

**Direction flag:**

- Direction flag selects either the increment or decrement mode for the DI and SI registers during string instructions.
- If D=1, the registers are automatically decremented, if D=0 the registers are automatically incremented.

**Introduction to Assembly Language Programming:**

A program that consists of 0s and 1s is called machine language, it is difficult to write programs in machine language, to make programming easier assembly language was developed.

Assembly language:

- Assembly language provides mnemonic for the machine codes.
- Assembly language programs must be translated into machine language by a program called Assembler.
- Assembly language is low-level language.
- To write programs in assembly language, programmer must know the number of registers and their size and other details of the CPU.

**FORMAT of Assembly language instruction:**

**LABEL: Mnemonic      Destination, source; comments**

Label and comments are optional

**Introduction to Program Segments:**

An assembly language program has four segments

1. CODE Segment
2. DATA segment
3. STACK Segment
4. EXTRA segment

Code segment contains assembly language instructions

Data segment is used to store information (data)

Stack segment is used by CPU to store the data temporarily

**Segments:**

- In 8086, memory is divided into 16 segments.
- Each segment size is 64Kbytes.
- Starting of each segment ends with 0H e.g. 12340H
- Segment registers are used to hold upper 16-bits of the starting address of segment.  
e.g. if code segment starting address is 123F0H, CS holds 123FH

Physical Address:

- It is 20-bit address, it is an actual address of physical location in 1Mbyte memory.
- The range of physical address is 00000H-FFFFFFH.

e.g. 12AC3H is 20-bit physical address

Offset Address:

- It is 16-bit address, it an address of location within 64-Kbyte segment.
- The range of Offset address is 0000H-FFFFH  
e.g. 12FFH is 16-bit offset address.

Logical address:

- It consists of segment value and an Offset address.  
e.g. 1234H:0001H     1234H is segment value 0001H is offset address

**Logical and physical address in Code segment:**

- To execute an instruction, processor has to fetch an instruction from code segment.
- Logical address of an instruction always consists of a CS (code segment) and IP (instruction pointer) shown in CS: IP format.
- IP contains Offset address

**Data Segment:**

- In 8086 microprocessor, memory set aside for data is called Data segment.
- Data segment uses DS register and BX, SI, DI as Offset registers.
- Logical address of Data segment is shown as DS : OFFSET register

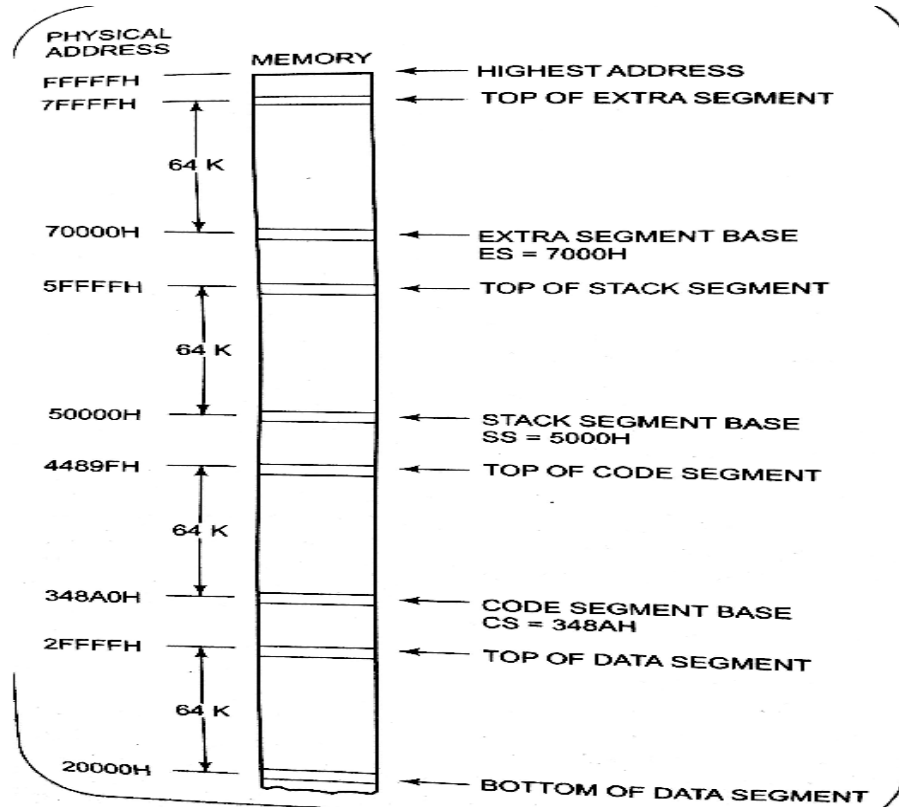
**Segment registers:**

At any given time 8086 works with only 4 segments, four segment registers are used to hold the 16-bits of starting addresses of four segments. The four segment registers are

1. Code segment register (CS)
2. Data segment register (DS)
3. Extra segment register (ES),
4. Stack segment register (SS)

Code segment register holds the starting address of the Code segment, DS register holds base address of data segment, ES register holds the base address of extra segment and SS register holds the base address of the stack segment.

Figure below shows how these four segments are positioned in memory at any given time.

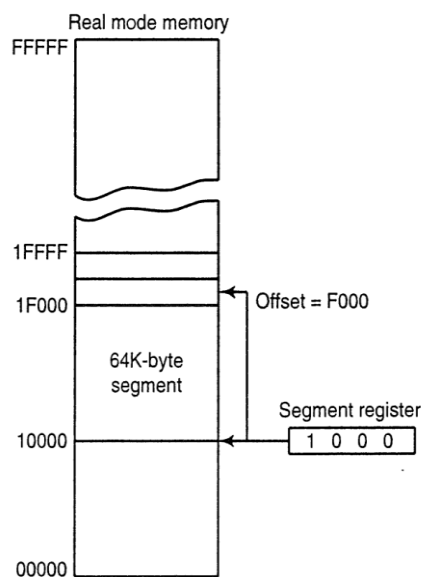


- OFFSET is the distance above the start of the segment.
- A combination of segment base address and OFFSET address is required to access memory location.
- In memory segmentation, if the beginning address of the segment is known the ending address is found by adding FFFFH.

**Example :** In the figure below segment address is 1000H and OFFSET is F000H,

$$\text{20bit address} = 1000\text{H} \times 10\text{H} + \text{F000H} = 1\text{F000H}$$

**1F000H is 20-bit address,**



### Model Definition:

- .Model is directive used to select the size of the memory.
- SMALL, MEDIUM, COMPACT, LARGE are the memory models.

e.g. .MODEL SMALL ; this directive defines memory model as SMALL

.MODEL MEDIUM ; in this memory model, data should fit in 64Kbytes , Code can exceed 64Kbytes

.MODEL COMPACT; data can exceed 64Kbytes, Code cannot exceed 64Kbytes

.MODEL SMALL; 64 Kbytes for Data , 64Kbytes for Code

.MODEL LARGE; both data and code can exceed 64Kbytes

.MODEL TINY; both data and code must fit in 64Kbytes

### Segment Definition

- 8086 has four segment registers CS, DS, SS, ES
- Segment definition uses three directive .CODE, .DATA, .STACK.

.CODE → indicates beginning of code segment

.DATA → indicates beginning of data segment

.STACK → indicates beginning of stack segment

e.g .STACK 64 ; this directive reserves 64K bytes of memory for stack segment

### Data Segment:

- In data segment, data items are declared
- Using directives DB, DW data items can be declared
- DB (Define Byte) declares data item of size byte
- DW (Define Word) declares data item of size word

Code segment:

- It is last segment in the program
- Instructions are written after .CODE directive

### Sample program using model definition method

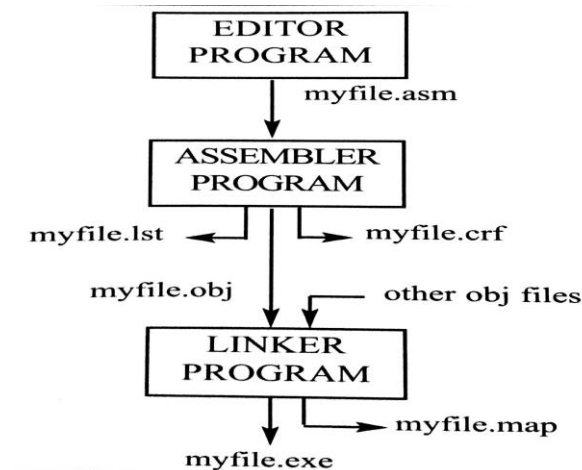
```
.MODEL SMALL
.DATA
MSG DB "RLJIT$"

.CODE
MOV AX, @DATA
MOV DS, AX
LEA DX, MSG
MOV AH, 09H
INT 21H
END
```

### Assemble Link and Run a program:

Three steps to create an executable assemble language program

1. Edit the Program
2. Assemble the program
3. Link the program



After editing the program, it must be assembled

### Assembling the program:

- Translating assembly program to machine program is called assembling
- Assemble is the tool used for translation
- Assembler generates three files
  1. .ASM file
  2. .OBJ file
  3. .LST file
  4. .CRF file

#### .ASM file:

- it is a file created with editor

#### .OBJ file:

- **MASM** assembler converts .asm file ( assembly language instructions) into .obj file ( machine language instructions)

#### .LST file:

- This file is optional
- It lists all the opcodes and OFFSET addresses of instructions
- To get LIST file **TYPE FILENAME.lst | more**

Note: MASM assumes that list file is not wanted (NUL.LST indicates no list file)

#### CRF file (cross reference file):

- This file contains list of all symbols and labels used in the program as well as program line numbers in which they are specified.

**Assembler directives (pseudo instructions):**

- Assembler directives are the instructions to the assembler.
- Assembler directives are not translated into the machine language.
- Directives control the generation of machine code and organisation of the program.

**Storing data in a memory segment:**

The following directives are used to store data in a memory segment.

1. DB
2. DW
3. DD
4. DQ
5. DT

**DB (define byte):**

- It defines the variable of type BYTE.
- It reserves one or more byte locations in memory.

**Format: variable-name DB Initialization values**

E.g. LIST DB 10H; this statement reserves one byte location for a variable name LIST and initialize the value 10.

|      |    |
|------|----|
| LIST | 10 |
|------|----|

DATA DB 1, 2, 3; this statement reserves 3 byte locations and initialize the values 1,2, 3.

|      |   |
|------|---|
| DATA | 3 |
|      | 2 |
|      | 1 |

CHAR DB 'A'; this statement reserves 1 byte location and initialized with the ASCII Value of A.

NUM DB ?; this statement reserves one byte location for variable NUM and not initialized.

**DW (define word):**

- It defines a variable of type word.
- Word ( two bytes).
- It reserves one or more word locations in memory.



**Format: variable-name    DW    initialization values**

E.g. DATA    DW    1234H; this statement reserves one word location and initializes the value 1234H.

|      |    |
|------|----|
|      | 12 |
| DATA | 34 |

**DUP:**

- It creates an array of size n.

e.g. LIST    DB    10 DUP (?)    → reserves 10 byte locations

DATA    DW    100 DUP (0)    → reserve 100 word locations and  
initializes with 0

**EQU:**

- This directive equates the constant value to the label.
- Each time the assembler finds the label in the program, it will replace the name with the value equated to that label.

e.g.            TEN        EQU    10  
                NINE      EQU    9

### Instruction Set:

**MOV:**

- It copies data from one location to another

**Format: MOV Destination, Source ; copy source operand to destination**

The above instructions moves source operand to destination, MOV is Mnemonic for Move operation.

e.g MOV AX, BX; it moves BX value to AX  
MOV CL, 55H; it moves 55H to CL  
MOV CH, BH ; it moves BH value to CH  
MOV AX, 1234H; it moves 1234H to AX

**ADD Instruction:**

This instruction adds the source and destination operands and puts the result in destination.

**Format: ADD Destination, Source**

**Operation: Destination = source + destination**

**E.g.** write instructions to add 25H and 34H, move numbers to any registers and then add together

```
MOV AL, 25H;    AL = 25H
MOV BL, 34H;    BL = 34H
ADD AL, BL;      AL = 59H (25H + 34H)
```

**SUB:**

- It subtracts the source from the destination and places result in the destination

Syntax: SUB destination, source

Operation: destination = destination – source

**Multiplication:**

1. BYTE × BYTE
2. WORD × WORD

**BYTE × BYTE:**

- In byte × byte multiplication, multiplicand must be in AL register and the second operand can be either in 8-bit register or 8-bit memory location and result will be stored in AX

**Syntax: MUL 8-bit register / memory location**

**Operation : AX = AL × 8-bit register/ memory location**

|       |              |       |                |
|-------|--------------|-------|----------------|
| e.g.1 | MUL BL       | e.g.2 | MUL NUM2       |
|       | AX = AL × BL |       | AX = AL × NUM2 |

**WORD × WORD:**

- In word×word multiplication, multiplicand must be in AX and multiplier can be in any 16-bit register or memory location and result will be stored in DX:AX
- DX contains higher word of the result and AX contains lower word of the result.

Syntax: MUL 16-bit register/memory location

DX:AX = AX \* 16-bit register /memory location

|       |        |       |                   |
|-------|--------|-------|-------------------|
| e.g.1 | MUL BX | e.g.2 | MUL WORD PTR [BX] |
|-------|--------|-------|-------------------|

$$DX:AX = AX * BX$$

$$DX:AX = AX * [BX]$$

[BX] is 16-bit memory location

### Division of unsigned numbers:

1. Byte over byte
2. Word over word
3. Word over byte

#### Byte / Byte division:

- In byte / byte division, numerator (dividend) must be in AL register and AH must be zero.
- The denominator (divisor) can be in any 8-bit register or memory location
- After division, quotient will be in AL and remainder will be in AH

Syntax: **DIV** 8-bit register / memory location

e.g. **DIV** BL;                      AX / BL    AL = quotient    AH = remainder

#### Word/Word Division:

- In Word/Word division, numerator (dividend) is in AX and DX must be zero
- Denominator (divisor) can be in any 16-bit register or memory location
- After division, AX contains quotient, and DX contains remainder

Syntax: **DIV** 16-bit register / memory location

e.g.            MOV    AX,9000  
                  MOV    DX,0  
                  MOV    BX,100  
                  DIV    BX;            DX:AX/BX → 9000/ 100    AX=90    DX=0

#### Word/Byte:

- Dividend is in AX, divisor in any 8-bit register or memory location
- After division, AL contains quotient AH contains remainder

e.g.    **DIV** BL ;            AX/BL    AL = quotient    AH = remainder

### Logic instructions:

#### AND:

- This instruction performs AND operation on source and destination and stores result in Destination.
- AND instruction automatically changes CF and OF to zero and PF, ZF and SF are set according to result.

Syntax: AND Destination, Source

Function: destination = destination AND Source

e.g. AND AL, BL ; AL = AL AND BL

### OR:

- This instruction performs OR operation on destination and source and stores result in destination.
- OR instruction automatically changes CF and OF to zero and PF, ZF and SF are set according to result.

### XOR:

- It performs exclusive-OR operation on destination and source and stores result in destination.
- XOR instruction clears CF and OF and SF, OF, PF are set according to the result.

Syntax: XOR destination, source

Functions: destination = destination XOR Source

#### Logical XOR Function

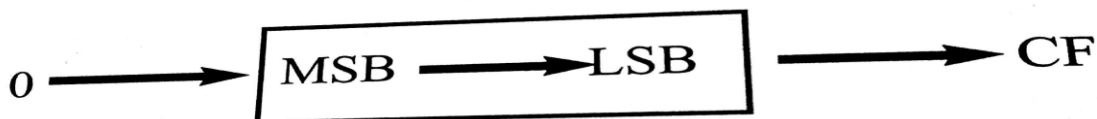
| Inputs |   | Output  |
|--------|---|---------|
| A      | B | A XOR B |
| 0      | 0 | 0       |
| 0      | 1 | 1       |
| 1      | 0 | 1       |
| 1      | 1 | 0       |

### SHR:

- SHR shifts the contents of the destination right by the number of bits count
- LSB bit is shifted to carry flag (CF)
- Zero is shifted to MSB
- If count is greater than 1, count should be in CL

Syntax: SHR destination, count

Operation:



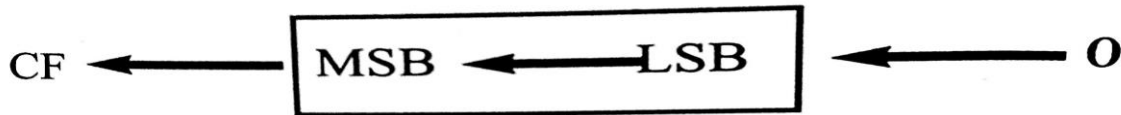
e.g. SHR AL, 1 ; the contents of AL are shifted left by one position

### SHL:

- SHR shifts the contents of the destination Left by the number of bits specified in the count
- MSB bit is shifted to carry flag (CF)
- Zero is shifted to LSB
- If count is greater than 1, count should be in CL

**Syntax:** SHL destination, count

**Operation:**



e.g SHL AL,1 ; contents of AL are shifted left by one position

**CMP (compare):**

- This instruction compares destination and source and changes flags CF, SF, ZF according to the result.
- Destination can be any register or memory location
- Source can be any register or memory location or immediate data.

**Table 3-3: Flag Settings for Compare Instruction**

| <b>Compare operands</b>        | <b>CF</b> | <b>ZF</b> |
|--------------------------------|-----------|-----------|
| <b>destination &gt; source</b> | 0         | 0         |
| <b>destination = source</b>    | 0         | 1         |
| <b>destination &lt; source</b> | 1         | 0         |

**Syntax:** CMP Destination, Source

**Operation:** destination – source

**Flags affected:** CF, SF, ZF

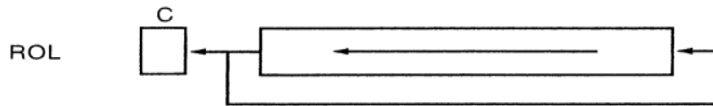
**Rotate Instructions:**

- Rotate instructions rotate the bits of any register or memory location from one end to another or through the carry.
- There are four types of rotate instructions
  1. Rotate left ( ROL)
  2. Rotate Right (ROR)
  3. Rotate through carry left (RCL)
  4. Rotate through carry Right (RCR)

**ROL (Rotate left):**

- This instruction rotates the contents of destination left by the number of bits specified in the count.
- MSB bits are rotated into the right most bit (LSB) and into the carry flag (CF).
- If count is more than 1, CL must contain count.

**Syntax:** ROL Destination,Count

**Operation:**

e.g. MOV BH,72H ; BH=0111 0010  
 ROL BH,1 ; CF=0 BH = 1110 0100

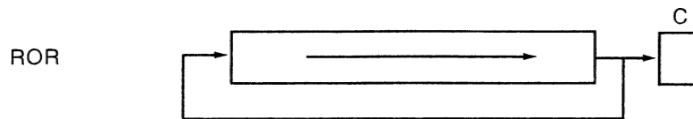
**To rotate the contents of BH by 4 positions, instructions are**

MOV BH, 72H; BH = 0111 0010  
 MOV CL, 4 CL=4 number of times to rotate  
 ROL BH, CL CF=1 BH = 0010 0111

**ROR (Rotate Right):**

- This instruction rotates the contents of destination right by the number of bits specified in the count.
- LSB bits are rotated into the left most bit (MSB) and into the carry flag (CF).
- If count is more than 1, CL must contain count.

**Syntax: ROR Destination, Count**

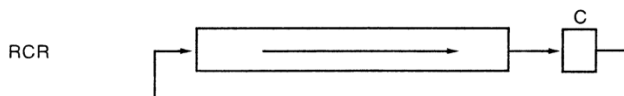
**Operation:**

e.g. MOV AL, 36H; AL = 0011 0110  
 ROR AL,1 AL = 0001 1011 CF=0

**RCR (Rotate right through carry):**

- This instruction rotates the contents of the destination right through carry flag bit by the number of positions specified in the count.
- LSB bit is shifted into CF, and CF bit is shifted to MSB bit.
- If count is more than 1, CL must contain count.

**Syntax: RCR Destination, Count**

**Operation:**

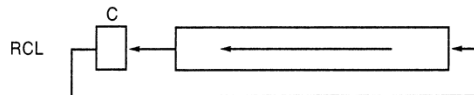
e.g. MOV AL,06H ; AL = 0000 0110 CF=1  
 RCR AL,1 ; AL = 1000 0011 CF=0

**RCL (Rotate Left through Carry):**

- This instruction rotates the contents of the destination left through carry flag bit by the number of positions specified in the count.
- MSB bit is shifted into CF, and CF bit is shifted to LSB bit.
- If count is more than 1, CL must contain count.

**Syntax:** RCL Destination, Count

**Operation:**



e.g. MOV AL, 90H;                      CF=0    AL = 1001 0000  
       RCL AL,1                        CF=1    AL= 0010 0000

**String Instructions:**

There is a group of instructions to perform operations on strings

| <b>Instruction</b>  | <b>Mnemonic</b> | <b>Destination</b> | <b>Source</b> |
|---------------------|-----------------|--------------------|---------------|
| move string byte    | MOVSB           | ES:DI              | DS:SI         |
| move string word    | MOVSW           | ES:DI              | DS:SI         |
| store string byte   | STOSB           | ES:DI              | AL            |
| store string word   | STOSW           | ES:DI              | AX            |
| load string byte    | LODSB           | AL                 | DS:SI         |
| load string word    | LODSW           | AX                 | DS:SI         |
| compare string byte | CMPSB           | ES:DI              | DS:SI         |
| compare string word | CMPSW           | ES:DI              | DS:SI         |
| scan string byte    | SCASB           | ES:DI              | AL            |
| scan string word    | SCASW           | ES:DI              | AX            |

**SI, DI Registers:**

- SI and DI registers are used by string instructions
- When DF=0 SI and DI automatically increment
- WHEN DF=1 SI and DI automatically decrement.

**CLD (clear direction Flag):** this instruction is used to clear the direction flag DF=0

**STD (set direction flag):** this instruction is used to set direction flag DF=1

**REPE (repeat while equal) or REPZ (repeat while zero) :**

- The REPE prefix is used for string instruction.
- **REPE** causes the string instruction to repeat until CX register reaches 0 or until unequal condition occurs (ZF=0)

**REPNE (repeat while not equal) or REPNZ (repeat while not zero) :**

- The REPNE prefix is used for string instruction.
- **REPNE** causes the string instruction to repeat until CX register reaches 0 or until equal condition occurs (ZF=1).

**CMPSB (compare string byte):**

- This instruction compares the byte in memory location (DS; SI) with the byte in memory location (ES:DI).
- It performs subtraction of the byte in ES:DI from the byte in DS:SI.
- CMPSB is normally used with REPE or REPNE prefix.

Syntax: CMPSB

Function: flags  $\leftarrow$  DS:SI – ES : DI; SI = SI  $\pm$  1; DI = DI  $\pm$  1

Flags affected: AF, CF, OF, PF, SF, ZF

**Example:** write a program to compare the string of length 10 in the Data Segment with the string of length 10 in the Extra Segment.

```
MOV  SI, OFFSET STR1
MOV  DI, OFFSET STR2
CLD
MOV  CX, 10
REPE CMPSB
```

**1.Clearing the Screen using INT 10H function 06H:**

The instructions to clear the screen are

```
MOV  AH,06H      ;AH=06H TO Select scroll Function
MOV  AL,00H      ;AL=00 TO Clear entire screen
MOV  BH,07       ;BH=07 Is normal attribute (white on black)
MOV  CH,00       ;CH=00 row value of start point (Upper left corner)
MOV  CL,00       ;CL=00 column value of start point
MOV  DH,24       ;DH=24 row value of end point (Lower right corner)
MOV  DL,79       ;DL=79 column value of end point
INT  10H         ;CALL BIOS interrupt
```

**2.Setting the Cursor Position to a Specific Location INT 10H Function 02H:**

Instructions to set the cursor position are

```
MOV  AH,02H
```



```

MOV BH,PAGE NUMBER(0-5)
MOV DH,ROW NUMBER(00-24)
MOV DL,COL NUMBER(00-79)
INT 10H

```

**Example: Write a code to set cursor position at ROW=15 and column = 25**

```

MOV AH,02H
MOV BH,0      ;PAGE 0
MOV DH,15
MOV DL,25
INT 10H

```

**Program: Write a program that 1)clears the screen 2) set cursor at center of the screen**

```

;To Clear the Screen
MOV AH,06H
MOV AL,00
MOV CH,00
MOV CL,00
MOV DH,24
MOV DL,79
INT 10H
;Setting the cursor at center of the screen
MOV AH,02H
MOV BH,00      ; page 0
MOV DH,12      ;12 row
MOV DL,39      ;39 column
INT 10H        ;call BIOS interrupt

```

### 1. Outputting a String of Data to monitor INT 21H Function 09H

Instructions to display the string of data on monitor

```

MOV AH,09H
MOV DX, OFFSET address of string
INT 21H;

```

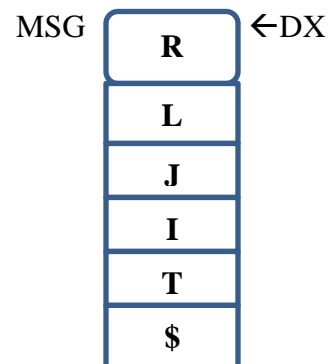
e.g. Write a code to Display string “welcome to RLJIT”

```
MSG DB “ Welcome to RLJIT$”
```

```

MOV AH,09H
MOV DX, OFFSET MSG or LEA DX, MSG
INT 21H

```



## 2. Outputting a single character on monitor INT 21H Function 02H

Instructions to display a character

```
MOV AH,02H
MOV AL, Character to be displayed
INT 21H
```

**E.g. Write a Code to Display a character 'J'**

```
MOV AH,02H
MOV AL,'J'
INT 21H
```

## 3. Inputting a Single Character with ECHO

Instructions:

```
MOV AH,01H
INT 21H
```

After reading a character from keyboard, it will be stored in AL

## 4. Inputting a single character without Echo

**Instructions:** MOV AH,07H  
INT 21H;

After reading a character from keyboard, it will be stored in AL

## 5. Inputting a String of Data from keyboard INT 21H Function 0AH

Steps:

1. Set AH=0AH
2. DX=OFFSET address at which the string of data is stored ( buffer area)
3. First byte in buffer contains size of the buffer
4. Second byte contains the length of the string
5. String will be stored from the third byte

**Instructions:**

```
MOV AH,0AH
MOV DX, OFFSET Variable-Name
INT 21H
```

## Carriage Return & Line Feed:

- Carriage Return & Line Feed are two ASCII characters
- ASCII of Carriage return = 13 or 0DH
- ASCII of Line Feed = 10 or 0AH
- Carriage return character returns a cursor to the beginning of current Line
- Line feed character moves cursor to next line.

**8255 Programmable peripheral Interface:**

- ▶ The 8255 is a popular interfacing component, that can interface any I/O device to a microprocessor.
- ▶ The parallel input-output port chip 8255 is also called as programmable *peripheral input-output port*.

**8255 Features:**

- ▶ The 8255 is a 40-pin chip.
- ▶ It has three ports.
- ▶ The ports are each 8-bit and are named A,B and C.
- ▶ The individual ports of the 8255 can be programmed to be input or output.

**PA0-PA7:**

The 8-bit port A can be programmed as all input, or as all output or all bits as bidirectional port.

**PB0-PB7:**

- ▶ The 8-bit port B can be programmed as all input or as all output.
- ▶ Port B cannot be used as a bidirectional port.

**PC0-PC7:**

- ▶ This 8 bit-port C can be all input or all output.
- ▶ It can also be split into two parts, CU (upper bits PC7-PC4) and CL (Lower bits PC3-PC0).
- ▶ In addition, any of bits PC0-PC7 can be programmed individually.
- ▶ D0-D7 data pin:
- ▶ The data pins of the 8255 are connected to the data pins of the microprocessor allowing it to send data back and forth between the microprocessor and the 8255 chip.

**Mode selection of the 8255:**

- ▶ The ports of the 8255 can be programmed in three different modes.
  - 1) Mode 0
  - 2) Mode 1
  - 3) Mode 2

**Programming the 8255:**

- ▶ **8255** is programmed through the internal command register.
- ▶ Command register is 8-bits wide.
- ▶ 8255 operates in two different modes.
  1. I/O mode
  2. BSR (bit set / reset mode)

**I/O MODE:**

Under the I/O mode of operation, further there are three modes of operation of 8255

**Mode 0:**

- ▶ In this mode, any of the ports A,B, CL and CU can be programmed as input or output.

- ▶ In this mode, all bits are out or all in.

### **Mode 1:**

- ▶ In this mode, ports A and B can be used as input or output ports with handshaking capabilities.
- ▶ Handshaking signals are provided by the bits of port C.

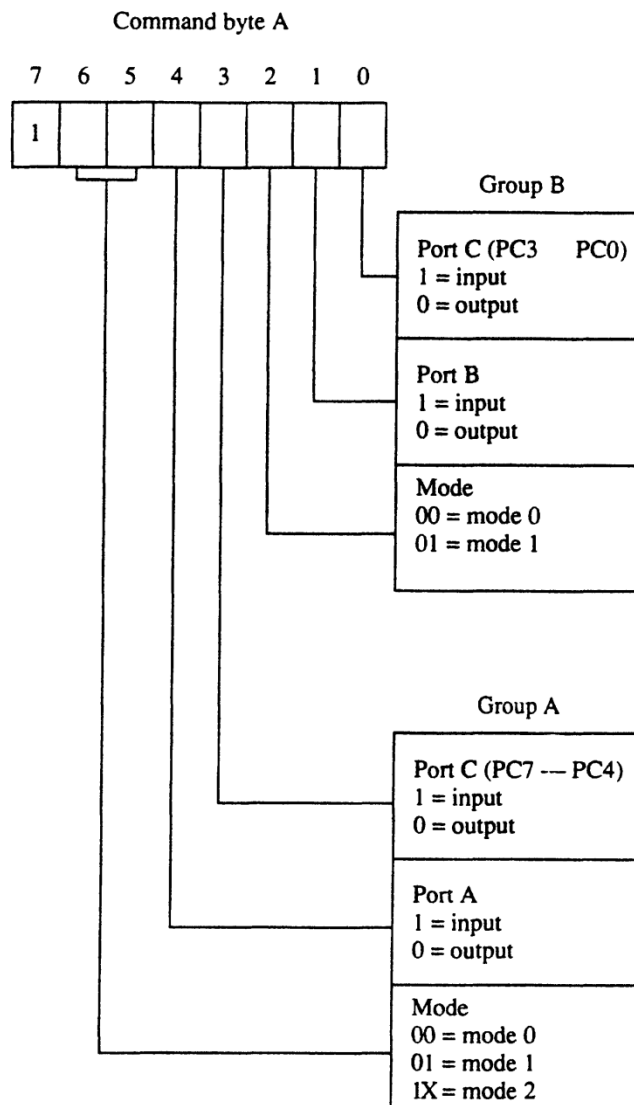
### **Mode 2:**

- ▶ In this mode, port A can be used as bidirectional port with handshaking capabilities whose signals are provided by port C.

### **BSR (bit set/reset) mode:**

- ▶ In BSR mode only port C ( $PC_0 - PC_7$ ) can be used to set or reset its individual port bits.

**FIGURE 10–15** The command byte of the command register in the 82C55. (a) Programs ports A, B, and C (b) Sets or resets the bit indicated in the select a bit field



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.

**Program:**

```
.MODEL SMALL
```

```
.DATA
```

```
ARR    DW    1,2,3,4,5,6,7
LEN     DW    ($-ARR)/2
MSG1    DB    "KEY IS FOUND$"
MSG2    DB    "KEY IS NOT FOUND$"
X       DW    6
```

```
.CODE
```

```
MOV     AX,@DATA
MOV     DS,AX
MOV     SI,0000H ; SI is LOW
MOV     DI,LEN
ADD     DI,DI
SUB     DI,2 ; DI is HiGH
RPT:    CMP     SI,DI ; Comparing SI with DI
        JA      KNF ; IF SI > DI jump to KNF
        MOV     AX,X ; AX= key
        MOV     BX,SI ;BX (MID) = (SI + DI)/2
        ADD     BX,DI
        SHR     BX,1
        CMP     AX,ARR[BX] ; Comparing AX (KEY) With middle element ARR[BX]
        JE      KF ; if key = middle element jump to KF
        JB      NEXT ; if KEY < middle element jump to NEXT
        MOV     SI,BX ; if KEY > middle element low (SI) = mid (bx)+2
        ADD     SI,2
        JMP     RPT

NEXT:    MOV     DI,BX ; if key < mid element high (di)= mid (bx)-2
        SUB     DI,2
        JMP     RPT
```

```
.MODEL SMALL
.DATA
    ARR DW 1,2,3,4
    LEN DW ($-ARR)/2 ;total 14 bytes ie 14/2 ie 7 elements
    MSG1 DB "KEY IS FOUND$"
    MSG2 DB "KEY IS NOT FOUND$"
    X DW 4 ; Key to search

.CODE
MAIN PROC
    MOV AX, @DATA
    MOV DS, AX

    ;setting SI as low ptr
    MOV SI, 0H
    ;setting DI as high ptr
    MOV CX, LEN ; now CX has 7
    DEC CX ; now CX has 6 ie last idx in array for "us"
    SHL CX, 1 ; now CS has 12 ie last idx in array for "memory"
    since we r using 2 byte numbers
    MOV DI, CX ; DI now points to last idx of array

    SEARCH_LOOP:
    CMP SI, DI
    JA NOT_FOUND ; If SI > DI, key not found, JA means Jump if
greater

    ; calculate mid ptr
    MOV BX, SI
    ADD BX, DI
    SHR BX, 1 ; BX = (SI + DI)/2 (word offset), BX now has mid ptr

    MOV AX, X
    CMP AX, ARR[BX] ; Compare key with middle element
    JE FOUND ; JE means jump if equal
    JB LESS_THAN ; If key < ARR[BX], search left, JB means jump
is less

    JA MORE_THAN

    MORE_THAN:
    MOV SI, BX
    ADD SI, 2 ; SI = mid + 1 (word offset)
    JMP SEARCH_LOOP

    LESS_THAN:
    MOV DI, BX
    SUB DI, 2 ; DI = mid - 1 (word offset)
    JMP SEARCH_LOOP

    FOUND:
    LEA DX, MSG1
    MOV AH, 09H
    INT 21H
    JMP EXIT

    NOT_FOUND:
    LEA DX, MSG2
    MOV AH, 09H
    INT 21H
    JMP EXIT

    EXIT:
    MOV AH, 4CH
    INT 21H
MAIN ENDP
END MAIN
```

```

KF: LEA    DX,MSG1 ; display the message "KEY is found"
      MOV    AH,09H
      INT    21H
      JMP    EXIT    ;      jump to exit

```

```

KNF:LEA    DX,MSG2; display the message "key is not found"
      MOV    AH,09H
      INT    21H

```

```

EXIT: MOV    AH,4CH
      INT    21H
      END

```

```

.MODEL SMALL
.DATA
  ARR DW 0004H,0003H,0003H,0002H,0001H
  LEN EQU ($-ARR)/2
.CODE
MAIN PROC
  MOV AX,@DATA
  MOV DS,AX

```

**Output: KEY IS FOUND**

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.

**Program:**

```

.MODEL SMALL
.DATA
  ARR DW 0005H,0004H,0003H,0002H,0001H
  LEN DB ($-ARR)/2
.CODE
  MOV AX,@DATA
  MOV DS,AX
  MOV CL,LEN
  DEC CL

```

```

LOOP2:MOV SI,OFFSET ARR
      MOV BL,CL

```

```

LOOP1:MOV AX,[SI] ; move first element to AX
      ADD SI,2 ; use SI as pointer to second element
      CMP AX,[SI] ; compare first element and second element
      JB NEXT ; if first element < second element jump to next
      MOV DX,[SI] ; SWAP Logic mov second element to DX
      MOV [SI],AX ; mov first element to second position
      MOV [SI-2],DX ; mov second element to first position
NEXT: DEC BL ; decrement no of comparisons
      JNZ LOOP1

```

```

MOV CL, LEN-1
LOOP2:
  MOV SI, OFFSET ARR
  MOV BL, LEN-1
  LOOP1:
    MOV AX, [SI] ; move first element to AX
    CMP AX, [SI+2] ; compare with next element
    JB NEXT ; if AX < [SI+2], no swap
    JA SWAP ; else swap
  SWAP:
    MOV DX, [SI+2] ; swap logic
    MOV [SI+2], AX
    MOV [SI], DX
  NEXT:
    ADD SI, 2 ; move to next pair
    DEC BL ; decrement comparison count
    JNZ LOOP1
    DEC CL ; decrement pass count
    JNZ LOOP2
  INT 3
MAIN ENDP
END MAIN

```

```

    DEC CL      ; decrement no of passes
    JNZ LOOP2
    INT 3
END

```

**Output:**

**Before Sorting:** 000A 000B 000C 000D 000E 000F 0010 0011 0012 0013

05 00 04 00 03 00 02 00 01 00

**After Sorting:** 01 00 02 00 03 00 04 00 05 00

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

**Program:**

```

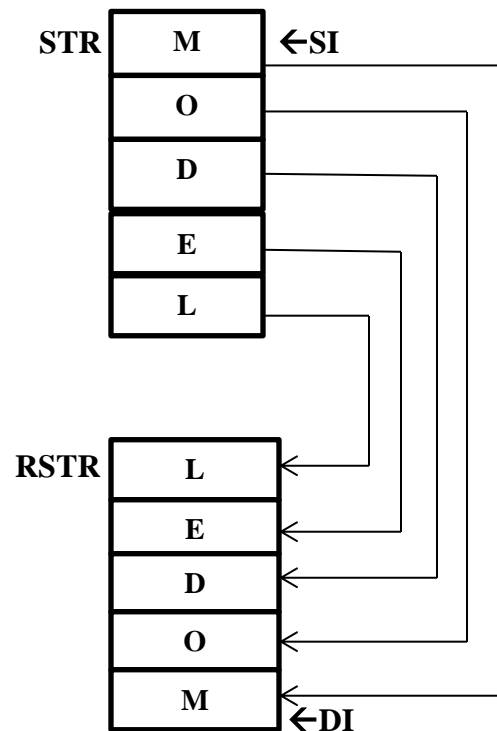
.MODEL SMALL
.DATA
STR DB "MODEL"
LEN EQU $-1-STR
RSTR DB 10 DUP('$')
MSG DB "Reverse string is:$"
MSG1 DB "String is Palindrome$"
MSG2 DB "String is Not Palindrome$"

.CODE
MOV AX,@DATA
MOV DS,AX
MOV ES,AX

LEA SI,STR      ; SI points to STR
LEA DI,RSTR     ; DI points to RSTR
ADD DI,LEN-1    ; adds DI and LEN-1
MOV CX,LEN      ; moves LEN to CX

RPT:MOV AL,[SI] ;moves the character pointed by SI to AL
      MOV [DI],AL ;moves AL value to location pointed by DI
      INC SI      ; increment SI to point to next character
      DEC DI      ; decrement DI to move next character
      LOOP RPT    ; jumps back to RPT if CX≠0

```



```
LEA    DX, STR
MOV    AH,09H
INT    21H
LEA    DX,MSG
MOV    AH,09H
INT    21H
LEA    DX,RSTR
MOV    AH,09H
INT    21H
LEA    SI,STR          ; SI points to STR
LEA    DI,RSTR         ; DI points to RSTR
MOV    CX,LEN          ; moves LEN to CX
REPE   CMPSB           ; Compares [SI] and [DI] and repeats if CX≠0 and ZF=1
JNE    NOTPAL          ; jumps to NOTPAL when unequal condition occurs
LEA    DX,MSG1          ; display the message "string is palindrome"
MOV    AH,09H
INT    21H
JMP    EXIT
```

```
NOTPAL:LEA    DX,MSG2 ; displays the message "string is not palindrome"
        MOV    AH,09H
        INT    21H
```

```
EXIT:MOV    AH,4CH    ; Exit to DOS
        INT    21H
        END
```

**Output:**

**MODEL**

**Reverse String is: LEDOM**

**String is not Palindrome**



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

**Program:**

MODEL SMALL

.DATA

```

N      DW      6
R      DW      2
NCR    DW      0

```

.CODE

```

MOV     AX, @DATA
MOV     DS, AX
MOV     AX, N
MOV     BX, R
CALL    NCR_PROC
MOV     AH, 4CH
INT     21H

```

ncr\_proc proc

```

CMP     AX, BX           ; Compare r and n
JZ      N1               ; If equal ADD 1 to result
CMP     BX, 0            ; If no, check if r = 0
JZ      N1               ; If yes, ADD 1 to result
CMP     BX, 1            ; If no, check if r = 1
JZ      N2               ; If yes, ADD n to result
MOV     CX, AX
DEC     CX
CMP     CX, BX           ; If no, check if r = n-1
JZ      N2               ; If yes, ADD n to result
PUSH    AX               ; Save n
PUSH    BX               ; Save r
DEC     AX               ; Compute n-1
CALL    ncr_proc         ; CALL ncr_proc
POP     BX               ; Restore r
POP     AX               ; Restore n
DEC     AX               ; Compute n-1
DEC     BX               ; Compute r-1
CALL    ncr_proc         ; CALL ncr_proc
JMP     LAST

```

**LOGIC**

$$N_Cr = (N-1)C_r + (N-1)C_{(r-1)}$$

If  $r = 0$  (OR)  $r = N$  then  $N_Cr = 1$

If  $r = 1$  (OR)  $R = N-1$  then  $N_Cr = N$

```

n1:ADD    ncr, 1          ; ADD 1 to result
      RET
n2:ADD    ncr, AX          ; ADD n to result
last:    RET
ncr_proc  ENDP
END

```

**OUTPUT:****Before:**

```

2  0 1 2 3 4 5
DS:0000 04 00 02 00 00 00

```

**After:**

```

2  0 1 2 3 4 5
DS:0000 04 00 02 00 06 00

```

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

**Program:**

```

.MODEL SMALL
.DATA
MSG1  DB    "TIME IS$"
MSG2  DB    10,13,"DATE IS$"

.CODE
MOV    AX,@DATA
MOV    DS,AX

LEA    DX,MSG1;      display msg1
MOV    AH,09H
INT    21H
MOV    AH,2CH;      Read system time CH = hours, CL = minutes, DH = seconds
INT    21H
MOV    AL,CH;      move hours to AL
CALL    DISPLAY;    call display procedure
MOV    AL,CL;      move minutes to AL
CALL    DISPLAY

```

```

MOV    AL,DH;          Move seconds to DH
CALL   DISPLAY

LEA     DX,MSG2;        display msg2
MOV     AH,09H
INT     21H

MOV     AH,2AH;         Read system date  AL = day (sun,mon...) DH =month DL =day CX= year
INT     21H
MOV     BX,DX;          save month and day in BX,  BH = month BL=day
CALL    DISPLAY

MOV     AL,BL;          move day from BL to AL
CALL    DISPLAY

MOV     AL,BH;          move  month from BH to AL
CALL    DISPLAY

MOV     DL',';          ;   display character ','
MOV     AH,02H
INT     21H

MOV     AX,CX;          move year to AX=2017
MOV     DX,0;           DX=0
MOV     BX,10;          BX=10
DIV     BX;             DX: AX / BX → 2017 / 10   AX =201 DL =07
MOV     CL,DL;          move 7 o CL  CL =7
MOV     DX,0;           DX=0
DIV     BX;             DX: AX / BX →  201 / 10   AX=20  DL =01
ADD     DL,30H;          DL + 30H = 01 + 30H = 31H display 1
MOV     AH,02H
INT     21H

MOV     DL,CL;          DL ← CL  DL=07
ADD     DL,30H ;         DL + 30H →   07 + 30H = 37H display  7
MOV     AH,02H
INT     21H

MOV     AH,4CH
INT     21H

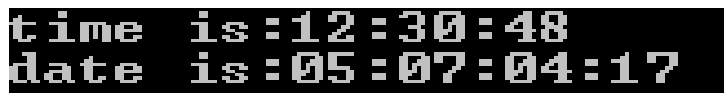
```

```

DISPLAY  PROC
PUSH  AX ;          save AX in stack
MOV   DL,':'      ;   display symbol ':'
MOV   AH,02H
INT   21H
POP   AX          ;   pop AX from stack
AAM           ;   convert packed BCD in AL to Unpacked eg. 12 → 01 02 =AX
MOV   DX,AX ;      DX= 01 02
ADD   DX,3030H;     DX + 3030H = 01 02 + 30 30 H = 31 32H → ASCII code of 1&2
XCHG  DH,DL       ;   display character in DH
MOV   AH,02H
INT   21H

MOV   DL,DH       ;   display character in DL
MOV   AH,02H
INT   21H
RET
DISPLAY  ENDP
END

```

**OUTPUT:**


```

time is:12:30:48
date is:05:07:04:17

```

**Steps to execute program in Keil:****Project → Close Project****Project → New µvision Project → give project name → save****NXP → LPC2148 → ok → yes****File → New → edit program → save file as .S****Target 1 → source group 1 → Add files to group “source group1” → select your .S file****Project → Build Target → if 0 errors → debug****Check registers or memory Locations to verify output.**

**PROGRAM 6a : Data Transfer Instructions**

```
        AREA PROG1, CODE, READONLY
        ENTRY
START
        LDR R2,=0X05
        LDR R0,=SOURCE
        LDR R1,=DEST
UP      LDR R3,[R0],#4
        STR R3, [R1],#4
        SUBS R2,#1
        BNE UP
STOP B STOP
        AREA SOURCE, DATA, READONLY
        DCD 0X10,0X20,0X30,0X40,0X50
        AREA DEST, DATA, READWRITE
        END
```

**6 b. Arithmetic Instructions:**

```
        AREA PROG2, CODE, READONLY
        ENTRY
START
        LDR R1,=0X00000006
        LDR R2,=0X00000001
        ADD R5,R1,R2
        SUB R6,R1,R2
        MUL R7,R1,R2
```

**STOP B STOP**

**END**

### 6.C Logical operations

**AREA PROG2, CODE, READONLY**

**ENTRY**

**START**

**LDR R1,=0X00000003**

**LDR R2,=0X00000007**

**AND R3,R1,R2**

**ORR R4,R1,R2**

**EOR R5,R1,R2**

**STOP B STOP**

**END**

**7. Write C programs for ARM microprocessor using KEIL (Demonstrate with help of suitable Program).**

#### **Program 1:**

```
#include<lpc21xx.h>
main()
{
int a=6,b=2,sum,sub,mul,div;
sum=a+b;
sub=a-b;
mul=a*b;
div=a/b;
}
```

#### **Program 2:**

```
#include<lpc21xx.h>
main()
{
int a=3, b=7, and, or, xor,not;
```

```

and=a&b;
or=a|b;
xor=a^b;
not=~a;
}

```

**8 a. Design and develop an assembly program to demonstrate BCD UP-Down Counter (00-99) on the Logic controller**



**Port A → Output Port**

**Control Word =**

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

**= 80H**

**Program:**

```
.MODEL SMALL
```

```
.DATA
```

```
    PA    EQU    0D800H
```

```
    CR    EQU    0D803H
```

```
.CODE
```

```
MOV  AX, @DATA
```

```
MOV  DS, AX           ; DATA segment Initialization
```

```
MOV  AL, 80h          ; Port A - Output
```

```
MOV  DX, CR
```

```
OUT  DX, AL           ; Initialize 8255
```

```
MOV  DX, PA           ; Move port A address to DX
```

```
MOV  AL, 00           ; AL=00
```

```
RPT: OUT  DX, AL       ; send AL value to Port A
```

```
CALL DELAY            ; call Delay Procedure
```

```
INC  AL               ; AL =AL +1
```

```
CMP  AL, 100          ; compare AL value with 100
```

```
JNE  RPT             ; if AL ≠100 jump to RPT
```

```
MOV  AH, 4CH          ; Exit to DOS
```

```
INT  21H
```

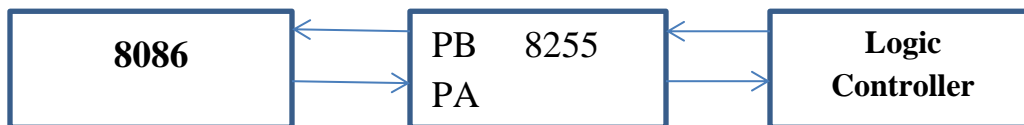
```
DELAY    PROC
```

```

PUSH CX
PUSH BX
MOV CX,0FFFFH
LOOP2:MOV BX,1000H
LOOP1:DEC BX
JNZ LOOP1
LOOP LOOP2
POP BX
POP CX
RET
DELAY ENDP
END

```

8 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.



**PB (Port B) → Input Port**

**PA (Port A) → Output Port**

Control Word = 

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

 = 82H

**Program:**

```
.MODEL SMALL
```

```
.DATA
```

```
PA EQU 0D800H
```

```
PB EQU 0D801H
```

```
CR EQU 0D803H
```

```
MSG_X DB 10,13,"Set value for X on logic contrl & press any key...$"
```

```
MSG_Y DB 10,13,"Set value for Y on logic contrl & press any key...$"
```

```
.code
```

```
MOV AX, @DATA ;
```

```
MOV DS, AX ; DATA segment Initialization
```

```
MOV AL, 82h ; Port A - Output, Port B - Input
```

```
MOV DX, CR ;
```



```

OUT    DX, AL           ; Initialize 8255

LEA    DX, MSG_X        ; Display message & wait for keyboard hit
MOV    AH, 09h           ;
INT    21h              ;
MOV    AH, 01h           ;
INT    21h              ;
MOV    DX, PB           ;
IN     AL, DX            ; Read DATA from Port B
MOV    CL, AL            ; Save read DATA IN CL

LEA    DX, MSG_Y        ; Display message & wait for keyboard hit
MOV    AH, 09h           ;
INT    21h              ;
MOV    AH, 01h           ;
INT    21h              ;

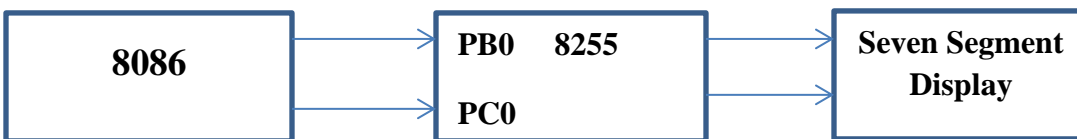
MOV    DX, PB           ;
IN     AL, DX            ; Read DATA from Port B
MUL    CL                ; Multiply first and second DATA

MOV    DX, PA           ;
OUT    DX, AL            ; SEND product to Port A

MOV    AH, 4CH           ;
INT    21h              ; Exit to DOS
END

```

**9. Design and develop an Assembly program to display messages “FIRE” & “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.**



**Port B → Output Port      Port C → Output Port**

**Control Word =**

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

**= 80H**

**Send data bit to PB0**

**Send clock signal to PC0**

**program:**

**.MODEL SMALL**

**.DATA**

```

    PB    EQU    0D801H
    PC    EQU    0D802H
    CR    EQU    0D803H
    FIRE  DB    86H,88H,0F9H,8EH
    HELP  DB    8CH,0C7H,86H,89H

```

**.code**

```

    MOV    AX, @DATA    ;
    MOV    DS, AX        ; DATA segment Initialization
    MOV    AL, 80H       ; Port A, B & C - Output
    MOV    DX, CR        ;
    OUT    DX, AL        ; Initialize 8255

```

```

    MOV    CX, 10        ; Display FIRE & HELP 10 times

```

```

RPT:  PUSH    CX
        LEA     SI, FIRE    ; Store offset address of message "DATA_fire" IN si
        CALL    DISPLAY    ; Display message "FIRE"
        CALL    DELAY
        LEA     SI, HELP    ; Store offset address of message "DATA_help" IN si
        CALL    DISPLAY    ; Display message "HELP"
        CALL    DELAY
        POP     CX
        LOOP    RPT
        MOV    AH, 4CH
        INT    21h        ; Exit to DOS

```

**DELAY PROC**

```

    PUSH    CX
    PUSH    BX
    MOV    CX, 1000H
    LOOP2: MOV    BX, 0FFFFH
    LOOP1: DEC    BX
            JNZ    LOOP1    ; Repeat INNER loop FFFF times
            LOOP    LOOP2    ; Repeat OUTER loop 1000h times
    POP     BX

```

```

    POP    CX
    RET
DELAY    ENDP

```

```

DISPLAY  PROC
    MOV    BL, 4          ; Four display codes to be sent
BACK2:   MOV    CL, 8      ; Eight bits IN each display code
    MOV    AL, [SI]
BACK1:   ROL     AL, 1
    MOV    DX, PB        ;
    OUT    DX, AL        ; DATA bit sent over PB0
    PUSH   AX
    MOV    AL, 1          ; SEND falling edge of the pulse over PC0
    MOV    DX, PC        ;
    OUT    DX, AL        ;
    DEC    AL            ;
    OUT    DX, AL        ;
    POP    AX
    DEC    CL
    JNZ    BACK1         ; Check if ALl 8 bits are sent or not
    INC    SI
    DEC    BL
    JNZ    BACK2         ; Check if ALl 4 display codes sent or not
    RET
DISPLAY  ENDP
END

```

10. Design and develop an assembly program to drive a stepper motor interface and rotate the motor in specified direction (Clock-wise or Counter – clock wise) by N steps. Introduce delay between successive steps.



Port A → output Port

Control Word = 

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

 = 80H

**Program:**

```

.MODEL SMALL
.DATA

```

```
PA EQU 0D800H
CR EQU 0D803H
```

```
.CODE
```

```
MOV AX, @DATA ;
MOV DS, AX ; DATA segment Initialization
MOV AL, 80H ; Port A, B & C - Output
MOV DX, CR ;
OUT DX, AL ; Initialize 8255

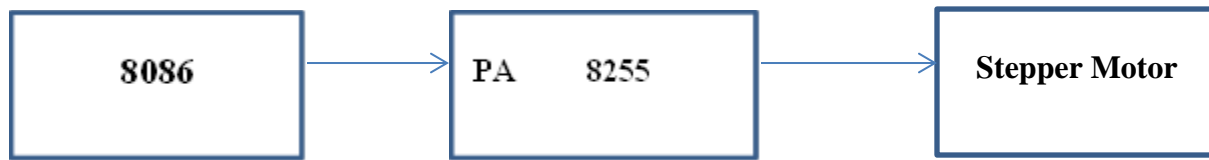
MOV CX, 10 ; Set LOOP counter to repeat clk times
MOV DX, PA
MOV AL, 11H ; Store bit Pattern IN AL
L1: OUT DX, AL
CALL DELAY
ROR AL, 1 ; Rotate AL right by one bit to get next Pattern
LOOP L1

MOV AH, 4CH ;
INT 21H ; Exit to DOS
```

```
DELAY PROC
```

```
PUSH CX
PUSH BX
MOV CX, 1000h
LOOP2: MOV BX, 0FFFFH
LOOP1: DEC BX
JNZ LOOP1 ; Repeat INNER LOOP FFFFH ×1000H times
LOOP LOOP2 ; Repeat OUTER LOOP FFFFH times
POP BX
POP CX
RET
DELAY ENDP
END
```

**11 a. Design and develop an assembly program to generate sine wave using DAC interface (output of DAC should be displayed on the CRO)**



**Port A → output Port**

**Control Word =** 

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

 **= 80H**

**Program:**

**.MODEL SMALL**

**.DATA**

```
VALUES DB 7FH,95H,0AAH,0BFH,0D1H,0E0H,0EDH,0F7H,0FDH,0FFH,0FDH,0F7H
        DB 0E0H,0D1H,0BFH,0A9H,95H,7FH,69H,53H,3FH,2DH,1DH,11H
        DB 7H,01H,00H,01H,07H,11H,1DH,2DH,3FH,53H,69H,7FH
```

```
PA EQU 0D800H
```

```
CR EQU 0D803H
```

**.code**

```
MOV AX, @DATA ;
MOV DS, AX ; DATA segment Initialization
```

```
MOV AL, 80H ; Port A - Output
MOV DX, CR ;
OUT DX, AL ; Initialize 8255
```

```
BEGIN: LEA SI, VALUES ; Si points to Values
```

```
MOV CX, 36 ; Initialize counter
```

```
LOOP1: MOV DX, PA
```

```
MOV AL, [SI] ; Copy DATA from memory
```

```
OUT DX, AL ; SEND to Port A
```

```
INC SI
```

```
LOOP LOOP1 ; If All 36 DATA are not sent goto label loop1 and repeat
```

```
JMP BEGIN ; Repeat from BEGIN to construct more waves
```

**END**

**11 b. Design and develop an assembly program to generate Half Rectified sine wave using DAC interface (output of DAC should be displayed on the CRO)**

**Program:**

```
.MODEL SMALL
.DATA
VALUES DB 7FH,95H,0AAH,0BFH,0D1H,0E0H,0EDH,0F7H,0FDH,0FFH,0FDH,0F7H
        DB 0E0H,0D1H,0BFH,0A9H,95H,7FH,7FH,7FH,7FH,7FH,7FH,7FH
        DB 7FH,7FH,7FH,7FH,7FH,7FH,7FH,7FH,7FH,7FH,7FH,7FH
PA EQU 0D800H
CR EQU 0D803H
.code
MOV AX, @DATA ;
MOV DS, AX ; DATA segment Initialization

MOV AL, 80H ; Port A - Output
MOV DX, CR ;
OUT DX, AL ; Initialize 8255

BEGIN: LEA SI, VALUES ; Si points to Values
      MOV CX, 36 ; Initialize counter
LOOP1: MOV DX, PA
      MOV AL, [SI] ; Copy DATA from memory
      OUT DX, AL ; SEND to Port A
      INC SI
      LOOP LOOP1 ; If All 36 DATA are not sent goto label loop1 and repeat
      JMP BEGIN ; Repeat from BEGIN to construct more waves

END
```

**Steps to execute Hardware programs in Keil:**

**Project → Close Project**

**Project → New µvision Project → give project name → save**

**NXP → LPC2148 → ok → yes**

**File → New → edit program → save file as .C**

**Target 1 → source group 1 → Add files to group “source group1” → select your .c file**

Target 1 → options for target 'target 1' → right click device → select LPC2148

Target 1 → options for target 1 → target → xtal (Mhz) = 12.0 → code generation = ARM mode  
→ select IRAM1 and no Init

Target 1 → options for target 'target 1' → output → create hex file

Target 1 → options for target 'target 1' → Listing → select C preprocessor listing

Target 1 → options for target 'target 1' → Linker → select use memory layout from target dialog

Project → Build Target → if 0 errors → Flash Magic

### FLASH MAGIC:

Click flash magic → device → ARM7 → LPC2148 → com port as COM 1 → baud rate as 9600  
→ interface none (ISP) → oscillator (MHz) = 12.0 → select erase of flash code Rd plot.

Browse → select your .hex file → start

## 12. To interface LCD with ARM Processor, Write and execute programs in C language for displaying text messages and numbers on LCD

### Program:

```
#include <LPC214x.H>                                // Header file for LPC2148
#include <STRING.H>                                  // Header file for string manipulations
#define LCD_DATA_PORT(DATA) (DATA << 16) // P1.16 to P1.23 configured as LCD data port.
#define LCD_CMD_PORT(CMD) (CMD << 8) // P0.9 to P0.11 as LCD control signals.
#define LEFT      0x18                          // Command for left shift.
#define RIGHT     0x1C                          // Command for right shift.
#define STABLE    0x06                          // Command for stable.

unsigned int ENABLE = 0x08; // Pin P0.11 configured as LCD enable signals. (high to low)
unsigned int RD_WR = 0x04; // Pin P0.10 as LCD read/write signals. (1-read, 0-write)
unsigned int RE_SE = 0x02; // Pin P0.9 as LCD register selection signals.(1-data, 0-command)

void DELAY_1M(unsigned int VALUE) // 1ms delay
{
    unsigned int i,j;
    for(i=0;i<VALUE;i++)
    {
        for(j=1;j<1200;j++);
    }
}
```

```

    }
void DELAY_1S(unsigned int n)                // 1s delay
{
    unsigned int i,j;
    for(i=1; i<=n; i++)
        for(j=0; j<=10000; j++);
}
void LCD_ENABLE()                          // Singal for LCD enable (high to low with 50ms)
{
    IO0SET = LCD_CMD_PORT(ENABLE);          // enable is set to 1
    DELAY_1M(50);                          // 50ms delay
    IO0CLR = LCD_CMD_PORT(ENABLE);          // enable is cleared to 0
}
void LCD_DATA(unsigned int LDATA)           // function to display the message
{
    IO1PIN = LCD_DATA_PORT(LDATA);          // moving the message to LCD data port
    IO0CLR = LCD_CMD_PORT(RD_WR);           // LCD to write mode(RD_WR = 0)
    IO0SET = LCD_CMD_PORT(RE_SE);           // LCD to data mode (RE_SE = 1)
    LCD_ENABLE();                          // Latching the data to display from buffer
}
void LCD_CMD(unsigned int LCMD)             // function to configure the LCD
{
    IO1PIN = LCD_DATA_PORT(LCMD);           // moving the command to LCD data port
    IO0CLR = LCD_CMD_PORT(RD_WR);           // LCD into write mode (RD_WR = 0)
    IO0CLR = LCD_CMD_PORT(RE_SE);           // LCD into command mode (RE_SE = 0)
    LCD_ENABLE();                          // Latching the data to display from buffer
}
void LCD_INIT()                            // initializing the LCD with basic commands
{
    LCD_CMD(0x38);                          // 8 Bit Mode, 2 Rows, 5x7 Font Style
    LCD_CMD(0x0C);                          // Display Switch Command : Display on, Cursor on, Blink off
    LCD_CMD(0x06);                          // Input Set : Increment Mode
    LCD_CMD(0x01);                          // Screen Clear Command , Cursor at Home
}
void DISPLAY_MESSAGE(unsigned char ADDRESS,char *MSG) //message display function
{
    unsigned char COUNT,LENGTH;
    LCD_CMD(ADDRESS);
    LENGTH = strlen(MSG);
    for(COUNT=0;COUNT<LENGTH;COUNT++)

```



```

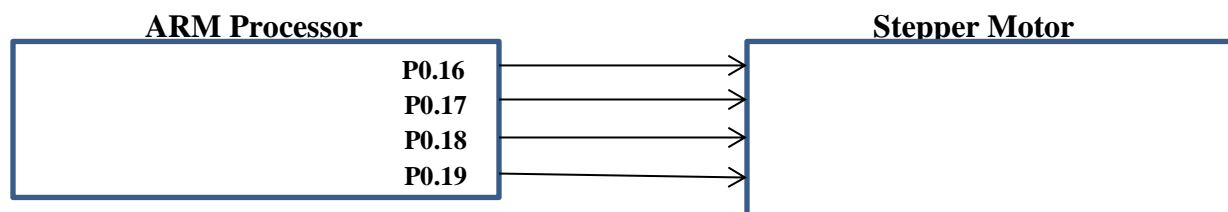
    {
        LCD_DATA(*MSG);
        MSG++;
    }
}

void SHIFT_MESSAGE(unsigned int SHIFT, unsigned int TIME) // Function for shift operations
{
    while(1)
    {
        LCD_CMD(SHIFT);
        DELAY_1S(TIME);
    }
}

int main() // Main function
{
    IO0DIR =0x00000E00; // Pin from P0.9 to P0.11 configured as output pins.
    IO1DIR =0X00FF0000; // Pin from P1.16 to P1.23 configured as output pins.
    LCD_INIT(); // LCD initialize
    DISPLAY_MESSAGE (0x80, " RL JALAPPA "); // Address of first row and message
    DISPLAY_MESSAGE (0xC0, "0123456789ABCDEF"); // Address of second row and message
    SHIFT_MESSAGE (LEFT,500); // Shift operations- Left, Right and no shift with speed
}

```

**13. To interface Stepper Motor with ARM Processor, Write a program to rotate stepper motor.**



**P0.16 – P0.19 are used as output ports**

**IO0DIR** → It is 32-bit register, it is used to set port 0 as either output pins or input pins.

**To set P0.16 – P0.19 as output ports IO0DIR value= 0x000F0000**

**IO0DIR**[illegible]

P0.31

p0.0

**IO0PIN → It is 32-bit register, this register is used to read or write values directly to the port 0**

**Values to rotate stepper motor**

| Clock-Wise | Anti-Clock Wise |
|------------|-----------------|
| 0x00030000 | 0x00060000      |
| 0x00090000 | 0x000C0000      |
| 0x00C0000  | 0x00090000      |
| 0x00060000 | 0x00030000      |

### Program:

```
#include <LPC214x.H>                                // Header file for LPC2148
unsigned char COUNT=0;
unsigned int j=0;
unsigned int CLOCK[4] = {0x00030000,0x00090000,0x000C0000,0x00060000}; // data for clockwise rotation
unsigned int ANTI_CLOCK[4] = {0x00060000,0x000C0000,0x00090000,0x00030000}; // data for anti-clockwise rotation

void DELAY_1S(unsigned int n)                        // 1s delay
{
    unsigned int i,j;
    for(i=1; i<=n; i++)
        for(j=0; j<=10000; j++);
}

void CLOCK_WISE_DIRECTION(unsigned int STEP, unsigned int TIME) //Function for clockwise
{
    for(j=0;j<=STEP;j++)
    {
        IO0PIN = CLOCK[COUNT];
        COUNT ++;
        DELAY_1S(TIME);
        if(COUNT==4) COUNT=0;
    }
}

void ANTI_CLOCK_WISE_DIRECTION(unsigned int STEP, unsigned int TIME) //Function for anti-
{
    for(j=0;j<=STEP;j++)
    {
        IO0PIN = (ANTI_CLOCK[COUNT]);
    }
}
```

```
        COUNT ++;
        DELAY_1S(TIME);
        if(COUNT==4) COUNT=0;
    }
}
int main()                                // Main function
{
    IO0DIR = 0x000F0000;                  // Pin from P0.16 to P0.19 configured as output pins.
    while(1)                              // infinite loop
    {
        CLOCK_WISE_DIRECTION    (10,500);    // clockwise direction with step and speed
        ANTI_CLOCK_WISE_DIRECTION(10,500);    // anti-clockwise direction with step and speed
    }
}
```