LABORATORY MANUAL

Microprocessor

Programming Lab

S.E (I.T) (Sem. IV)

INDEX

	LAB SESSION I (A)
Title:	TO ADD TWO 8 BIT NUMBERS
Objective:	Program involves storing the two 8-bit no in memory locations and adding them taking into consideration the carry generated. The objective of this program is to give an overview of arithmetic instructions of 8086 for 8-bit operands
Pre-Requisites:	1. Knowledge of TASM directives. 2. Knowledge of Arithmetic Instructions of 8086 for 8-bit operands
Theory:	The addressing modes used in program are: 1) Direct addressing mode: in this mode address of operand is directly specified in the instruction. This address is offset address of the segment being indicated by an instruction. E.g. MOV AL,[2000h] EA = DS x 10H + 2000H 2) Register Addressing Mode: In this mode operand are specified using registers. Instructions are shorter but operations cannot be identified looking at instruction. E.g. MOV CL, DL 3) Based Indexed Addressing Mode: The operand address is calculated using base register and index register. E.g. MOV DX, [BX + SI] moves word from address pointed by BX + SI in data segment to DX. EA = DS x 10H + BX + SI 4) Base indexed plus displacement: In this mode address of operand is calculated using base register, index register and displacement. E.g. MOV CX, [BX+DI+10h] This moves a word from address pointed by BX + DI +10h of data Segment to CX.
Algorithm:	1. Initialize the data segment. 2. Store two 8-bit numbers in memory locations. 3. Move the 1st number in any one of the general purpose register. 4. Move the 2nd number in any other general purpose register. 5. Add the 2 numbers. 6. Store the result in memory location. 7. Check for carry flag. If carry flag is set then store '1' as MSB of result. 8. Stop
Conclusion:	

	LAB SESSION 1 (B)
Title:	TO ADD TWO 16 BIT NUMBER
Objective:	Program involves storing the two 16-bit no in memory locations and adding them taking into consideration the carry generated. The objective of this program is to give an overview of arithmetic instructions of 8086 for 16-bit operands
Pre-Requisites:	1. Knowledge of TASM directives. 2. Knowledge of Arithmetic Instructions of 8086 for 16-bit operands
Algorithm:	1. Initialize the data segment. 2. Store two 16-bit numbers in memory locations. 3. Move the 1st number in any one of the general purpose register. 4. Move the 2nd number in any other general purpose register. 5. Add the 2 numbers. 6. Store the result in memory location. 7. Check for carry flag. If carry flag is set then store '1' as MSB of result. 8. Stop
Conclusion:	

	LAB SESSION 1 (C)
Title:	TO SUBTRACT TWO 8 BIT NUMBERS
Objective	Program involves storing the two 8-bit no in memory locations and subtracting them taking into consideration the carry generated. The objective of this program is to give an overview of arithmetic instructions of 8086 for 8-bit operands
Pre-Requisites	1. Knowledge of TASM directives. 2. Knowledge of Arithmetic Instructions of 8086 for 8-bit operands
Algorithm:	1. Initialize the data segment. 2. Store two 8-bit numbers in memory locations. 3. Move the 1st number in any one of the general purpose register. 4. Move the 2nd number in any other general purpose register. 5. Subtract 2nd number from the 1st number. 6. Store the result in memory location. 7. Stop
Conclusion	

	LAB SESSION 2 (A)
Title:	TO MULTIPLY TWO 8 BIT NUMBERS
Objective:	Program involves storing the two 8 bit numbers in memory locations and multiplying them the objective of this program is to give an overview of arithmetic instructions of 8086 for 8 bit operation
Pre-Requisites:	1. Instructions of microprocessor 8086 2. Addressing mode of microprocessor 8086. 3. Flag register of microprocessor 8086 4. Knowledge of TASM directories.
Theory:	MUL instruction This instruction multiplies byte/word present in source with AL/AX. a) When two 8 bit numbers are multiplied a 16 bit product is made available in AX register where AH stores higher byte and AL stores lower byte of the product. e.g. MUL BL b) When two 16 bit numbers are multiplied a 32 bit product is made available in DX and AX register pair. DX has higher word and AX has lower word of the product. e.g. MUL BX
Algorithm:	1. Initialize the data segment. 2. Store two 8 bit numbers in memory locations. 3. Move the 1st number in any one of the general purpose register. 4. Move the 2nd number in any other general purpose register. 5. Multiply the 2 numbers. 6. Store the result in memory location. 7. Stop.
Conclusion:	

	LAB SESSION 2 (B)
Title:	TO MULTIPLY TWO 16 BIT NUMBERS
Objective:	Program involves storing two 16 bit numbers in memory location and performing multiplication operation between them.
Pre-Requisites:	1. Instructions of microprocessor 8086 2. Addressing mode of microprocessor 8086. 3. Flag register of microprocessor 8086 4. Knowledge of TASM directories.
Algorithm:	1. Initialize the data segment. 2. Store the two 16 bit numbers in memory location. 3. Move one of the numbers in AX register. 4. Move another number in any general purpose register or use it directly from memory. 5. Perform multiplication using MUL instruction. 6. Store the product from DX, AX pair to two memory locations. 7. Stop
Conclusion:	

	LAB SESSION 2 (C)
Title:	TO PERFORM DIVISION OF TWO NUMBERS.
Objective:	Program involves storing two operands of size 32 bit and 16 bit in memory and performing division operation between them.
Pre-Requisites:	1. Instructions of microprocessor 8086 2. Addressing mode of microprocessor 8086. 3. Flag register of microprocessor 8086 4. Knowledge of TASM directories.
Theory:	DIV instruction This instruction is used to divide a word by a byte or to divide a double word (32 bits) by a word. a) When a word is divided by a byte, the word must be in the AX register. The divisor can be in a register or a memory location. After the division, AL will contain 8-bit quotient and AH will contain 8-bit remainder. e.g. DIV BL divides a word in AX by a byte in BL; Quotient in AL, remainder in AH b) When a double word is divided by a word, the most significant word of the double word must be in DX, and the least significant word of the double word must be in AX. After the division, AX will contain the 16-bit quotient and DX will contain the 16-bit remainder. e.g. DIV CX Divides double word in DX and AX by a word in CX; Quotient in AX, and remainder in DX. c) If an attempt is made to divide by 0 or if the quotient is too large to fit in the destination(greater than FFH / FFFFH), 8086 will generate a type 0 interrupt. All flags are undefined after a DIV instruction. d) If you want to divide a byte by a byte, you must first put the dividend byte in AL and fill AH with all 0’s. Likewise, if you want to divide a word by another word then put the dividend word in AX and fills DX with all 0’s.

Algorithm:	1. Initialize the data segment. 2. Store the two operands in memory location. 3. Move 32 bit number in DX, AX register pair. 4. Move 16 bit number in any general purpose register or use it directly from memory. 5. Perform division using DIV instruction. 6. Store the quotient and remainder from AX and DX into two memory locations. 7. Stop
Conclusion:	
	LAB SESSION 3 (A)
Title:	TO IMPLEMENT BLOCK TRANSFER
Objective	Program involves transferring source string from a particular location in source segment (Data Segment) to the desired location in destination segment (Extra Segment). The objective of this program is to give an overview of the String instructions of 8086.
Pre-Requisites	1. Knowledge of TASM directives. 2. Knowledge of String Instructions of 8086.
Algorithm:	1. Initialize the data segment. 2. Store the source string in consecutive memory location. 3. Initialize the extra segment. 4. Allocate consecutive memory locations for transfer. 5. Load the effective address of source string in SI register. 6. Load the effective address of destination string in DI register. 7. Initialize the Direction flag for Auto increment or Auto Decrement. 8. Store number of bytes to be transferred in any of the general Purpose registers. 9. Transfer the source string using appropriate string instructions (MOVSB / MOVSW) 10. Decrement count. 11. Check if count = 0.If yes then stop else repeat steps 9 - 11. 12. Stop
Conclusion:	

	LAB SESSION 3 (B)
Title:	TO FIND AVERAGE OF 10 NUMBERS
Objective	Program involves averaging of a ten numbers
Pre-Requisites	1. Knowledge of TASM directives. 2. Knowledge of CMP and Jump Instructions of 8086
Algorithm:	1. Initialize the data segment. 2. Clear carry flag. 3. Store initial address of list in a register. 4. Move 0A to register AL. 5. Store count in register AL. 6. Reset AL registers (content of AX) 7. Add one number from array to contents of AL 8. Decrement count 9. Increment SI 10. If count ≠ 0 go to step 8 11. Divide content of AX with that of BL. 12. Store answer in register. 13. Stop.
Conclusion:	

	LAB SESSION 4 (A)
Title:	ARRANGING NUMBER IN ASCENDING ORDER.
Objective	Program involves sorting an array in ascending order using Bubble sort algorithm. The objective of this program is to give an overview of the Compare and Jump instructions. Use of Indirect Addressing mode for array addressing is expected
Pre-Requisites	1. Knowledge of TASM directives. 2. Knowledge of CMP and Jump Instructions of 8086.
Theory:	1) compare instruction: CMP destination, source This instruction compares the source with destination. The source and destination must be of same size. Comparison is done by internally Subtracting source from destination. The result of this is not stored anywhere instead flag bits are affected. 2) JMP instruction: if condition is true then, it is similar to an intra segment branch and if false then branch does not take place and next sequential instruction is executed. The destination must be in range of -128 to 127 from address of instruction.
Algorithm:	1. Initialize the data segment. 2. Initialize the array to be sorted. 3. Store the count of numbers in a register. 4. Store count-1 in another register. 5. Load the effective address of array in any general purpose register. 6. Load the first element of the array in a register. 7. Compare with the next element of the array. 8. Check for carry flag. 9. If carry=0 first number > second number. Swap the 2 numbers.

	10. Increment to the contents of the SI register so that it points to the next element of the array. 11. Decrement (count-1) by 1. 12. Check if (count-1) =0. If no then repeat steps 7 to 11. 13. Decrement count by 1. 14. Check if count = 0.If no then repeat steps 6 through 15. Stop.
Conclusion:	

	LAB SESSION 4 (B)
Title:	ARRANGING NUMBER IN DESCENDING ORDER.
Objective	Program involves sorting an array in descending order using Bubble sort algorithm. The objective of this program is to give an overview of the Compare and Jump instructions. Use of Indirect Addressing mode for array addressing is expected
Pre-Requisites	1. Knowledge of TASM directives. 2. Knowledge of CMP and Jump Instructions of 8086.
Algorithm:	1. Initialize the data segment. 2. Initialize the array to be sorted. 3. Store the count of numbers in a register. 4. Store count-1 in another register. 5. Load the effective address of array in any general purpose register. 6. Load the first element of the array in a register. 7. Compare with the next element of the array. 8. Check for carry flag. 9. If carry=0 first number < second number. Swap the 2 numbers. 10. Increment to the contents of the SI register so that it points to the next element of the array. 11. Decrement (count-1) by 1. 12. Check if (count-1) =0. If no then repeat steps 7 to 11. 13. Decrement count by 1. 14. Check if count = 0.If no then repeat steps 6 through 15. Stop.
Conclusion:	

	LAB SESSION 5 (A)
Title:	TO COUNT EVEN AND ODD NUMBERS FROM AN ARRAY OF 10 NUMBERS.
Objective	Program involves counting even and odd numbers from a given array. The objective of this program is to give an overview of the string instructions of 8086.
Pre-Requisites	1. Knowledge of TASM directives. 2. Knowledge of CMP and Jump Instructions of 8086.
Theory:	A string is a series of bytes stored sequentially in the memory. string Instruction operates on such 'strings'. The SRC element is taken from the data segment using and SI register. The destination element is in extra segment pointed by DI register. These registers are incremented or decremented after each operation depending upon the direction flag in flag register. Some of the instructions useful for program are, 1) CLC - the instruction clears the carry flag. 2) RCR destination, count- Right shifts the bits of destination. LSB is shifted into CF. CF goes to MSB. Bits Are shifted counts no of times. 3) JC: jump to specified location. 4) INC/DEC destination: add/subtract 1 from the specified destination. 5) JMP label: The control is shifted to an instruction to which label is attached. 6) JNZ label: The control is shifted to an instruction to which label is attached if ZF = 0.
Algorithm:	1. Initialize the data segment. 2. Initialize the array. 3. Load the effective address of an array in any index register. 4. Load total number of elements of the array in any register. 5. Initialize any two registers as counter for even and odd numbers to

	zero. 6. Load first element of an array in any general purpose register. 7. Shift/rotate the contents of loaded register to right. 8. If CF=1 increment counter for odd numbers otherwise increment counter of even numbers. 9. Store the value of even and odd counter register to two memory locations. 10. Stop.
Conclusion:	

	LAB SESSION 5(B)
Title:	TO FIND SUM OF EVEN NUMBERS IN ARRAY OF 10 NUMBERS
Objective	Program involves averaging of a ten numbers
Pre-Requisites	1. Knowledge of TASM directives. 2. Knowledge of CMP and Jump Instructions of 8086
Algorithm:	1. Initialize the data segment. 2. Clear carry flag. 3. Store initial address of list in a register. 4. Move 0A to register AL. 5. Store count in register AL. 6. Reset AL registers (content of AX) 7. Add one number from array to contents of AL and rotate 8. Decrement count 9. Increment SI 10. If count ≠ 0 go to step 8 11. Divide content of AX with that of BL. 12. Store answer in register. 13. Stop.
Conclusion:	

	LAB SESSION 6 (A)
Title:	TO ADD TWO BCD NUMBERS
Objective	Add two 2 digit BCD numbers present in the registers.
Pre-Requisites	1. Knowledge of TASM directives. 2. Knowledge of Arithmetic Adjust Instructions of 8086.
Algorithm:	1. Initialize the data segment. 2 Get the first BCD number in AL 3. Get the second BCD number in BL. 4. Add the two BCD numbers. 5. Using DAA, adjust the result to valid BCD number. 6. Stop
Conclusion:	

	LAB SESSION 6 (B)
Title:	TO SUBTRACT TWO BCD NUMBERS
Objective	Add two 2 digit BCD numbers present in the registers.
Pre-Requisites	1. Knowledge of TASM directives. 2 Knowledge of Arithmetic Adjust Instructions of 8086.
Algorithm:	1. Initialize the data segment. 2 Get the first BCD number in AL 3. Get the second BCD number in BL. 4. Subtract the two BCD numbers. 5. Using DAS, adjust the result to valid BCD number. 6. Stop
Conclusion:	

	LAB SESSION 7 (A)
Title:	CONVERT BINARY CODE TO GRAY CODE
Objective	To write assembly language program to convert binary code to gray code
Pre-Requisites	1. Knowledge of TASM directives.
Algorithm:	1. Get the number whose equivalent is to be found. 2. Add number with itself 3. XOR this added result with the original number. 4. Shift the XORed number by 1 bit position to the right to get the Gray equivalent. 5. Display the result. 6. Stop.
Conclusion:	

	LAB SESSION 7 (B)
Title:	CONVERT BCD TO ASCII
Objective	Write a program to convert the BCD number into its equivalent ASCII form
Pre-Requisites	1. Knowledge of TASM directives.
Algorithm:	1. Start 2. Initialize the data segment 3. Load the BCD number in AX 4. Copy AL to AH 5. Shift the byte in AH by 4 bits 6. AND contents of AX by 0F0Fh 7. OR contents of AX by 3030h 8. End
Conclusion:	

	LAB SESSION 7 (C)
Title:	CONVERT PACKED TO UNPACKED BCD
Objective	To convert the packed BCD number in the unpacked BCD form
Pre-Requisites	1. Knowledge of TASM directives.
Algorithm:	1. Start 2. Initialize the data segment. 3. Initialize two variables digit1 and digit 2 to store unpacked BCD digits. 4. Load the packed BCD number into AL 5. Mask the upper nibble of number in AL using AND operation with 0FH 6. Move the lower nibble into variable digit1 7. Again load the packed BCD number into AL 8. Mask the lower nibble of number in AL using AND operation with F0H 9. Rotate the result towards left by 4 bits and store the upper nibble in digit 2. 10. End
Conclusion:	

	LAB SESSION 8(A)
Title:	DISPLAY A TO Z ON SCREEN.
Objective	To store A to Z Alphabets on an array and display them on user screen.
Pre-Requisites	1. Knowledge of TASM directives. 2. Knowledge of String instructions of 8086 and DOS interrupt.
Algorithm:	1. Initialize the data segment. 2. Store all Alphabets in array. 3. Initialize counter to 1AH. 4. Load starting Address of array in to SI. 5. Get each character in DL. 6. Display Character on user screen. 7. Increment SI. 8. Decrement counter. 9. Repeat step 5 to 8 until count becomes Zero. 10. Stop
Conclusion:	

	LAB SESSION 8(B)
Title:	DISPLAY CHARACTER FROM KEYBOARD UNTIL 0 IS ENTERED.
Objective	To Read Character from Keyboard and display on screen until 0 is pressed.
Pre-Requisites	1. Knowledge of TASM directives. 2. Knowledge of DOS Interrupt.
Theory	Instructions used in program are: MOV AH,08H INT 21H Read Input From Keyboard without echo and store at AL. **MOV AH,02H** **INT 21H** Display Character on screen. Character should be in DL register.
Algorithm:	1. Initialize the data segment. 2. Read input from keyboard. 3. Compare input with ASCII value of ZERO. 4. If result is 0, go to step 7. 5. Move content of AL to DL, to display it on screen. 6. Display character on screen. 7. Stop
Conclusion:	

	LAB SESSION 9(A)
Title:	TO WRITE A PROGRAM FOR 3 X 3 MATRIX MULTIPLICATION.
Objective	The objective is to multiply 3 X 3 matrix.
Pre-Requisites	1. Knowledge of TASM directives. 2. Knowledge of DOS interrupts. 3. Knowledge of string instruction and MACRO.
Algorithm:	1. Initialize Data segment. 2. Set the counter 3 .Multiply row1 of matrix1 with column1 of matrix2 and add all multiplication Answer. 4. Repeat step 3 for all other rows n columns. 5. Store final answer at the memory location defined for matrix3.
Conclusion:	

	LAB SESSION 9 (B)
Title:	TO WRITE A PROGRAM FOR 3 X 3 MATRIX ADDITION
Objective	The objective is to multiply 3 X 3 matrix.
Pre-Requisites	1. Knowledge of TASM directives. 2. Knowledge of DOS interrupts. 3. Knowledge of string instruction and MACRO
Algorithm:	1. Initialize the data segment. 2.Initialize counter = 9 3. Initialize pointer DI to matrix 1. 4. Initialize pointer Bx to matrix 2. 5. Initialize pointer SI to result matrix 3. 6. Get the number from matrix 1. 7. Add number from matrix 1 with matrix 1 number. 8. Save the carry if any. 9. Save the result in result matrix 3. 10. Increment DI, BX, and SI to point to next element. 11. Decrement count. 12.Check if count = 0,if not go to step VI else go to step XIII 13. Display the result. 14. Stop.
Conclusion:	

	LAB SESSION 10
Title:	PASSWORD VERIFICATION
Objective	The objective is to make use of string instruction and MACRO, to check whether the entered password by the user is correct or not.
Pre-Requisites	1. Knowledge of TASM directives. 2. Knowledge of DOS interrupts. 3. Knowledge of string instruction and MACRO.
Algorithm:	1. Store Initial password into Array. 2. Write Macro for printing output message. 3. Write Macro to display '*'. 4. Initialize the data segment. 5. Set the counter value=no. of character present in password. 6. Load Effective address of stored password in BX. 7. Take input from the keyboard. 8. Compare input with the password string. 9. If zero=0, both value are equal. Go to step 10. .If zero is not equal to 0. Go to step 15. 10. display '*' Macro. 11. Increment BX. 12. Decrement counter by 1. 13. Check if counter=0.If not, Repeat step 7 to 12. 14. Display Macro message for correct password, Go to step 16. 15. Display '*' macro and Macro message for wrong password. 16. End
Conclusion:	

	LAB SESSION 11
Title:	TO CHECK IF THE ENTERED STRING IS PALINDROME OR NOT.
Objective	Read a string from user and check whether it is palindrome or not and display appropriate message.
Pre-Requisites	1. Knowledge of TASM directives. 2. Knowledge of DOS interrupts. 3. Knowledge of string instruction and MACRO.
Theory:	String instruction/ data transfer a) LEA register, Source - Loads effective address (offset Address) of source in given register. e.g. LEA BX, Total; b) STOSB/STOSW (Store string) – It is used to store AL (or AX) into a memory location pointed by ES:DI. DI is incremented or decremented after transfer depending upon DF. c) LODSB/LODSW: Used to copy the contents of memory location pointed by DS: SI and store it in AL registers. SI is incremented /decremented after transfer depending upon DF.
Algorithm:	1. Start. 2. Initialize Data segment (DS). 3. Initialize Extra segment (ES). 4. Ask user to enter a string. 5. Read each character of a string using function value 01h or 08h. 6. Store individual character read in a string S. 7. Make SI register point to first element of a string and DI resister to point to last element of a string. 8. Initialize count register to number of comparisons required. 9. Move contents pointed by SI to AL. 10. Compare character in AL with character pointed by DI. 11. If there is a mismatch (ZF=0) display a message “Not a palindrome” and stop. 12. If two characters are matching then increment SI, decrement DI, decrement count register. 13. If count register becomes zero display a message “String is palindrome” and stop.

Conclusion:	

	LAB SESSION 12 (A)
Title:	TO WRITE A PROGRAM TO OBTAIN DIGITAL VOLATGE CORRESPONDING TO GIVEN ANALOG INPUT VOLTAGE USING 8255 AND ADC
Objective	The objective is to obtain digital voltage for the corresponding analog input by interfacing ADC with microprocessor using 8255.
Pre-Requisites	1. Knowledge of TASM directives. 2. Knowledge of DOS interrupts. 3. Knowledge of interfacing 8255
Algorithm:	Port A of 8255 used as an input port to ADC card. Port B as output port and port C as input port. 1. Start 2. Initialize the ports of 8255 using CWR 3. Give SOC and clock by making PB0=1 and PB@=1 4. Remove SOC 5. Check PC0 by reading port C for EOC 6. If PC0=1 then given output enable by making PB1=1 else go to step 5 7. Read port A to get digital data. 8. End
Conclusion:	

	LAB SESSION 12 (B)
Title:	TO WRITE A PROGRAM TO GENERATE TRIANGULAR, SQUARE, STAIRCASE WAVEFORMS USING 8255 AND DAC
Objective	The objective is to generate different waveforms by interfacing DAC with microprocessor using 8255.
Pre-Requisites	1. Knowledge of TASM directives. 2. Knowledge of DOS interrupts. 3. Knowledge of interfacing 8255
Algorithm:	Port A of 8255 used as an output port which is connected to DAC card. Port B and C are not used. I). SQUARE WAVEFORM 1. Initialize control word register with port A and port B working as output and port C as Input. 2. Port B, PB0 bit make 01h to enable DAC card. 3. Give immediate value 00 to port A to produce low voltage at its output which is given to DAC card. 4. Call delay subroutine to maintain voltage constant for some time. 5. Move FFh to port A to produce high voltage at its output. 6. Call delay to maintain high voltage. 7. Continue from step 3 for infinite times to produce continuous square wave. II). TRIANGULAR WAVEFORM 1. Initialize control word register with port A and port B working as output and port C as Input. 2. Port B, PB0 bit make 01h to enable DAC card. 3. Give immediate value 00 to port A to produce low voltage at its output which is given to DAC card. 4. The voltage at port A is incremented by 1 till reaches FF. 5. This produces incrementing voltage for up slope of a wave. 6. Now keep on decrementing voltage at Port A till it becomes 0 produce down slope of triangular wave.

	7. Continue from step 3 for infinite times to produce continuous triangular wave. III). STAIRCASE WAVEFORM 1. Initialize control word register with port A and port B working as output and port C as Input. 2. Port B, PB0 bit make 01h to enable DAC card. 3. Give immediate value 00 to port A to produce low voltage at its output which is given to DAC card. 4. Call delay subroutine to maintain voltage constant for some time. 5. Add contents of Port A with 3F that is increase its voltage by 3F. 6. Display the incremented voltage. 7. Continue step 5 to produce staircase effect till voltage at port A reaches FFh. 8. Bring voltage of DAC card back to 0 volts by initializing port A with 00h. 9. Continue from step 3 for infinite times to produce continuous wave.
Conclusion:	

	LAB SESSION 12(C)
Title:	IMPLEMENT UP AND DOWN COUNTER
Algorithm:	A) To implement UP counter 1. Start. 2. Initialize Port A as output and port B as input working in mode 0. 3. Move 00h in AL register. 4. Output AL on port A. 5. Call delay subroutine. 6. Increment AL. 7. Repeat steps 4 to 6 till AL becomes FF. 8. End. B) To implement DOWN counter 1. Start. 2. Initialize Port A as output and port B as input working in mode 0. 3. Move FFh in AL register. 4. Output AL on port A. 5. Call delay subroutine. 6. Decrement AL. 7. Repeat steps 4 to 6 till AL becomes 00. 8. End. C) Algorithm for delay subroutine 1. Move FFFFh to CX register. 2. Decrement CX. 3. Repeat step 2 till CX becomes zero. 4. Return to main program.
Conclusion:	