18ECC203J Microprocessor, Microcontroller and Interfacing Techniques

LABORATORY

ACADEMIC YEAR: 2021-2022

NAME: Pushpal Das

REG.NO: RA1911004010565

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
(Under SECTION 3 of the UGC Act, 1956)
S.R.M. NAGAR, KATTANKULATHUR – 603203.
KANCHEEPURAM DISTRICT

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

COLLEGE OF ENGINEERING & TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

BONAFIDE CERTIFICATE

Register No.:RA1911004010565

Certified to be the Bonafide record of work done by Pushpal Das of **ELECTRONICS AND COMMUNICATION ENGINEERING (ECE)** **B.Tech.**Degree course in the Practical **18ECC203J Microprocessor, Microcontroller and Interfacing Techniques** in **SRM Institute of Science and Technology**, Kattankulathur during the academic year 2021-2022

Staff
In-Charge

Date: **Head of the Department**

Submitted for University Examination held on_____at **SRM Institute of Science and Technology**, Kattankulathur.

Date: **Internal Examiner I** **Internal Examiner II**

TABLE OF CONTENT

Name: **Class:**

Reg.No: **Branch:**

Exp. No.	Date of Performance	Title of Experiment	Marks Obtained (30)

SRM Institute of Science and Technology
Faculty of Engineering and Technology
Department of Electronics and Communication Engineering

18ECC203J Microprocessor, Microcontroller & Interfacing Techniques Lab

Fifth Semester, 2020-21 (odd semester)

Name : Pushpal Das

Register No. : RA1911004010565

Day / Session : 2

Venue : Online

Title of Experiment : Data Transfer and Logical Operation using 8086 Date

of Conduction : 20/07/2021

Date of Submission : 15/08/2021

Particulars	Max. Marks	Marks Obtained
Pre-lab	5	
Lab Performance	10	
Post-lab	15	
Total	**30**	

REPORT VERIFICATION

Date : 20/07/2021

Staff Name : Prithiviraj R

Signature :

Experiment 1.
Data Transfer and Logical Operation using 8086.

1.1 Aim(s) / Objective(s) / Purpose.

The purpose of this experiment is to learn about the registers, instruction sets, and arithmetic operators of 8086 by addition , subtraction ,multiplication and division in the given two 16 bit numbers and store them in a memory location.

1.2 Introduction / Background

The purpose of this experiment is to learn about the registers, instruction sets, data transfer operation and logical operation of 8086 by using AND , OR in the given two 16 bit numbers and store them in a memory location.

Tips: Any information copied directly or verbatim from Lab manuals or other references should be stated within quotes and referred, otherwise, it is considered plagiarism.

1.3 Materials / Equipment

1. *K. M. Bhurchandi and A. K. Ray, "Advanced Microprocessors and Peripherals-with ARM and an Introduction to Microcontrollers and Interfacing ", Tata McGraw Hill, 3rd edition 2015*

2. *Doughlas.V.Hall, "Microprocessor and Interfacing : Programming and Hardware", 3rd edition, McGraw Hill, 2015*

1.4 Hardware Requirement:

The 8086 Microprocessor kit, Power Supply.

Software Requirement :

8086 Emulator (EMU8086)

1.5 Procedure

a. Load the program in the emulator.
b. Load the input values to the corresponding address location or directly into the registers.
c. Save, Compile and Emulate the program.
d. Execute the program using Single run or Run until termination process. e. Store the results of the process in the destination register
f. Terminate the program

1.6 Program Logic:

The logical AND instruction is used for masking off bits. The bits which have to be cleared are to be AND ed with a logical zero and other bits are to be high.hence to achieve the above objective,.AND with 0F0F .

The logical OR of a bit with procedure a result of 1.hence bits can be set selectively by ORing the particular bit with a 1 .

Program 1(a)

ADDRESS		MNEMONICS	OPCODE	COMMENTS
01000 01001 01002		MOV AX, [1100h]	A1,00,11	MOVE DATA FROM ADDRESS 1100H TO AX
01003 01004 01005 01006		MOV BX, [1102h]	8B,1E,02,11	MOVE DATA FROM ADDRESS 1102H TO BX
01007 01008		AND AX, BX	23,C3	AND OPERATION IS DONE
01009 0100A 0100B		MOV [1200h],AX	A3,00,12	THE RESULT OF AND OPERATION IN AX IS MOVED TO ADDRESS 1200H
0100C		HLT	F4	STOP THE SIMULATION

Observation

IN PUT ADDRESS	D
1100	

1101 1102 1103

83
0F
0F

Calculations:

OUT PUT ADDRESS	DATA

1200	07
1201	03

8317 – 1000 0011 0001 0111
0F0F – 0000 1111 0000 1111

0307 – 0000 0011 0000 0111

Program 1(b)

ADDRESS		MNEMONICS	OPCODE	COMMENTS
01000 **01001** **01002**		MOV AX, [1100h]	**A1,00,11**	**Move data from address 1100H to AX**
01003 **01004** **01005** **01006**		MOV BX,[1102H]	**8B,1E,02,11**	**Move data from address 1100H to BX**
01007 **01008**		OR AX,0F0Fh	**0B,C3**	**OR operation is done**
01009 **0100A** **0100B**		MOV [1200h],AX	**A3,00,12**	**The result of OR operation in AX is moved to address 1200H**
0100C		HLT	**F4**	**Stop the simulation**

Observation

INPUT ADDRESS	DATA
1100	**17**
1101	**83**
1102	**0E**

1103	**0E**

8F1F- 1000 1111 0001 1111

OUT PUT ADDRE **SS DATA**

8317 – 1000 0011 0001 0111

0E0E– 0000 1110 0000 1110 **Program 1(c):**
1200 1F 1201 8F

Calculations:

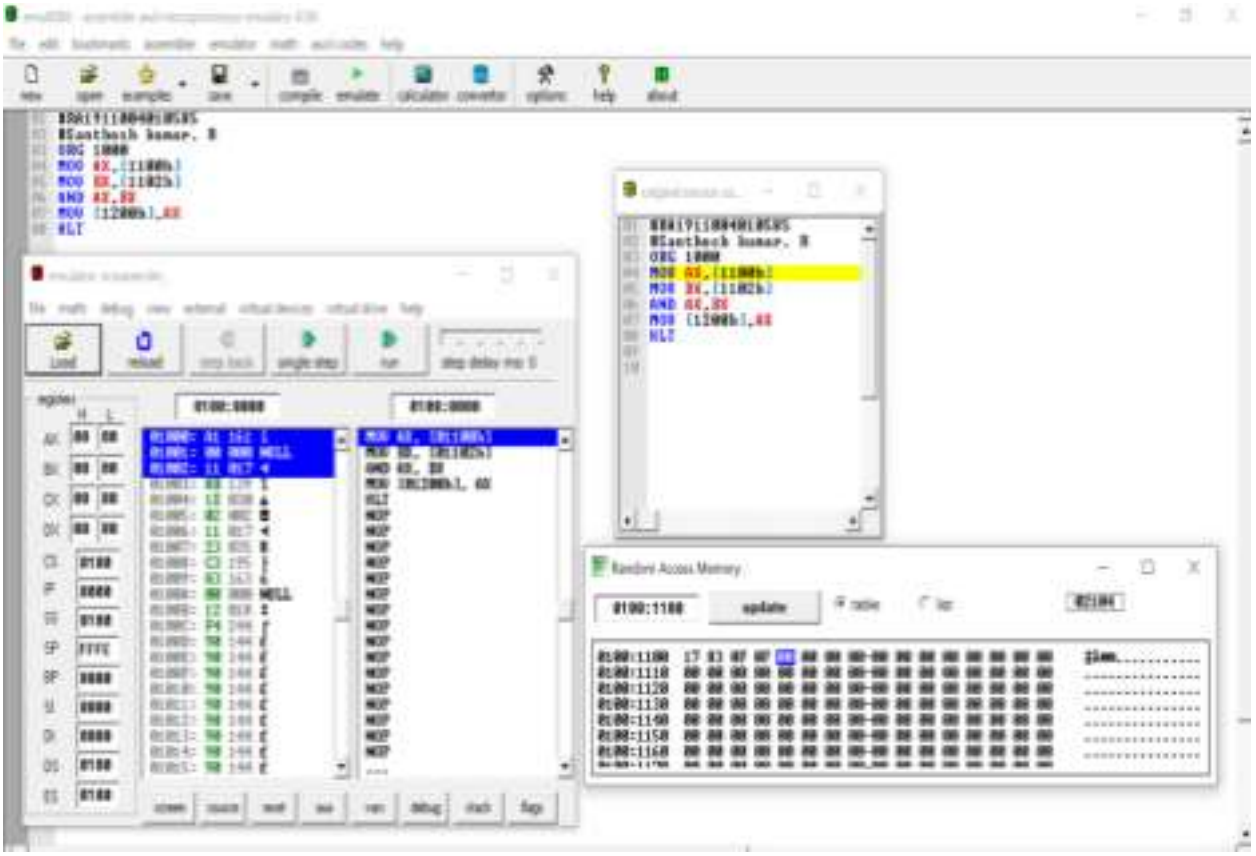
ADDRESS		MNEMONICS	OPCODE	COMMENTS

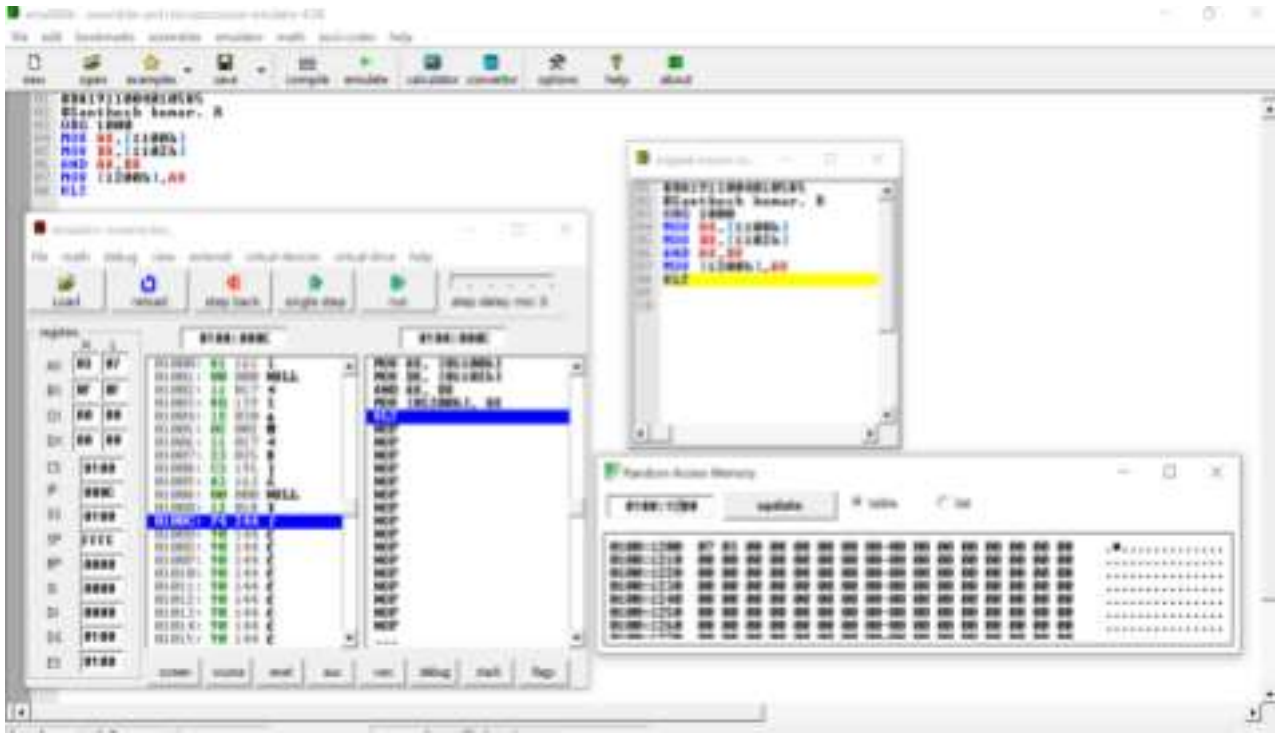
01000 **01001** **01002**		MOV AX, [1100h]	**A1,00,11**	**Move data from address 1100H to AX**
01003 **01004**		NOT AX	**F7,D0**	**NOT operation is done**
01005 **01006** **01007**		MOV [1200h],AX	**A3,00,12**	**The result of NOT operation in AX is moved to address 1200H**

01008 HLT **F4 Stop the simulation**

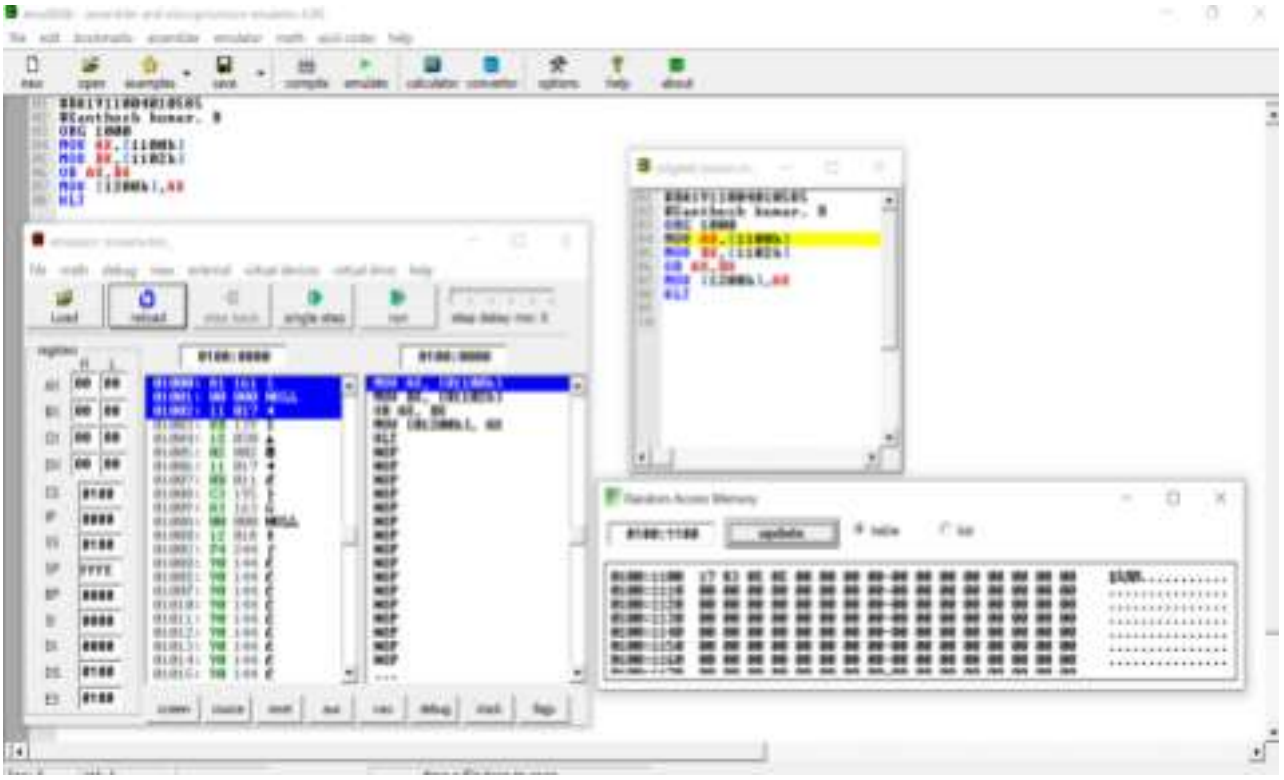
Emu8086 SIMULATION:

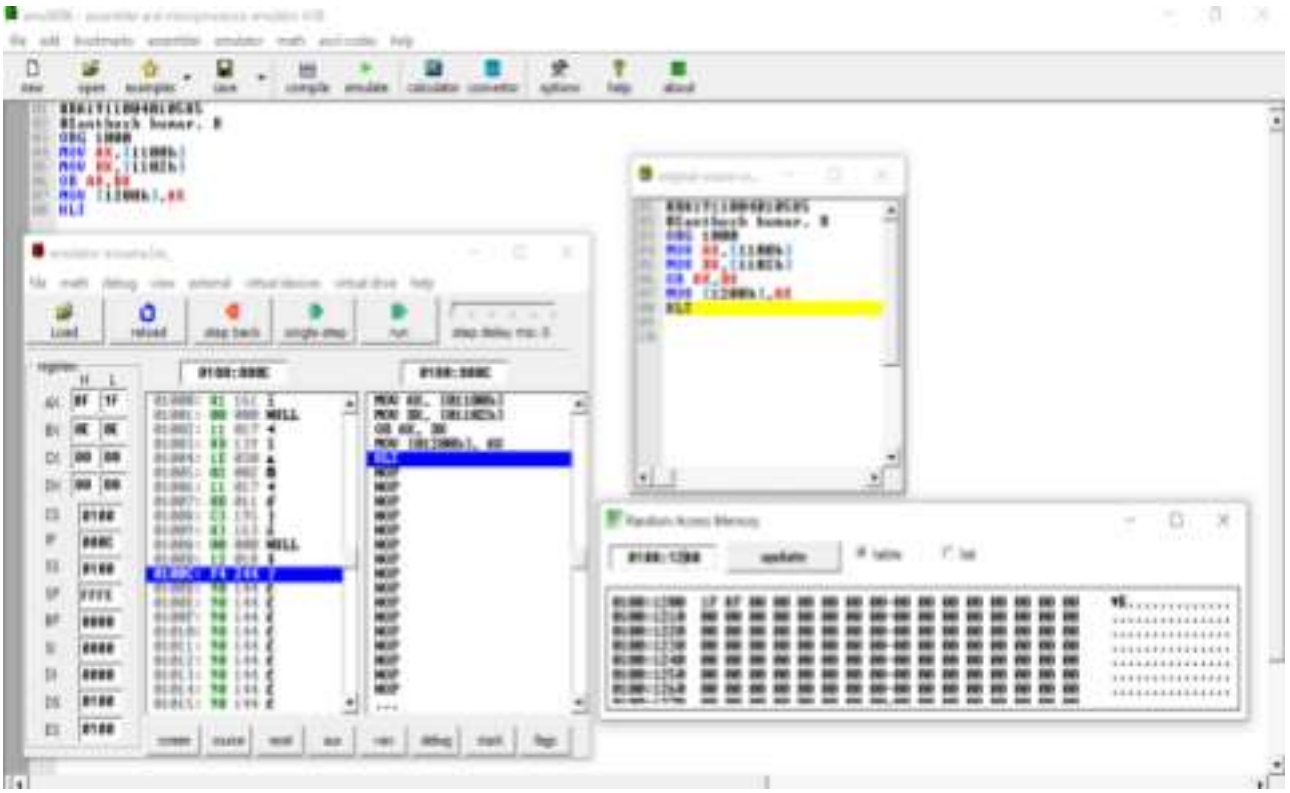
AND logic:



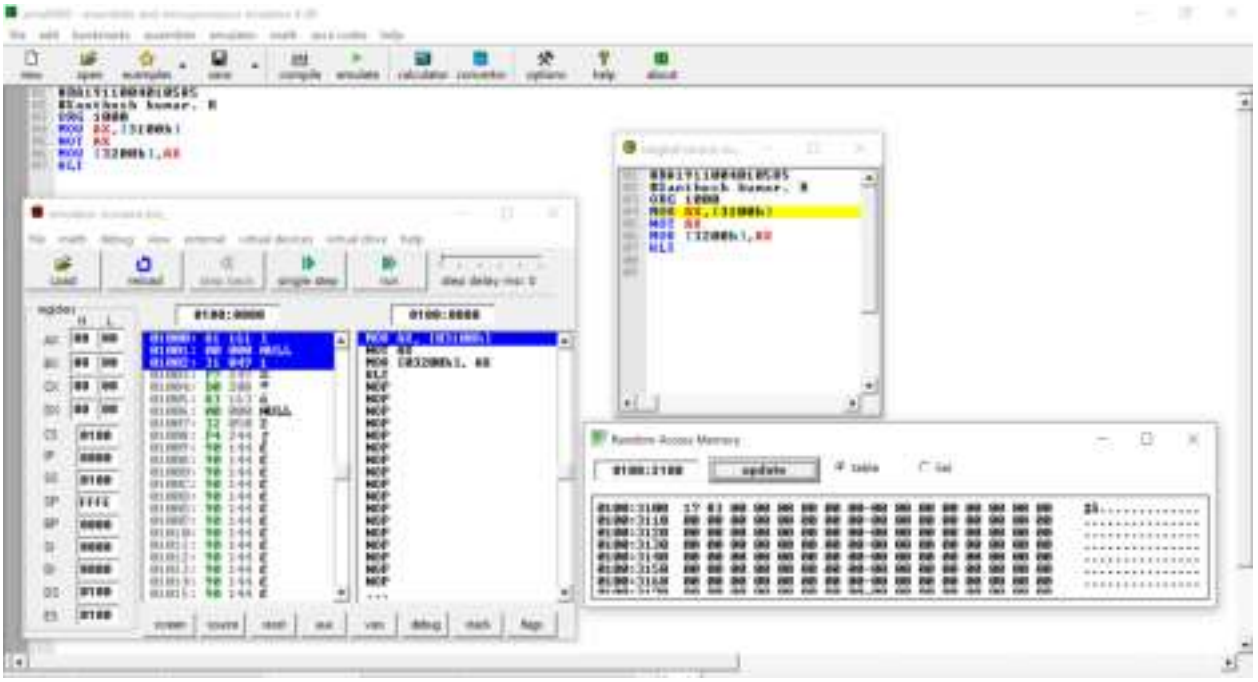


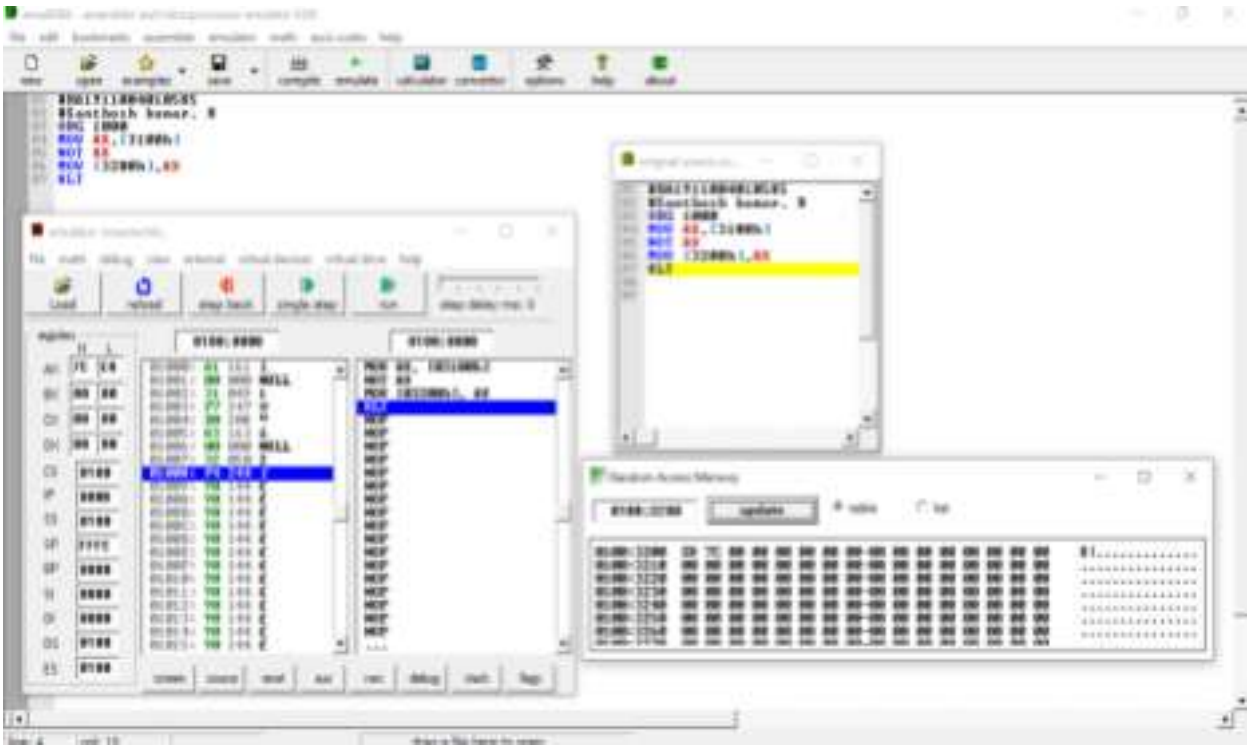
OR Logic:





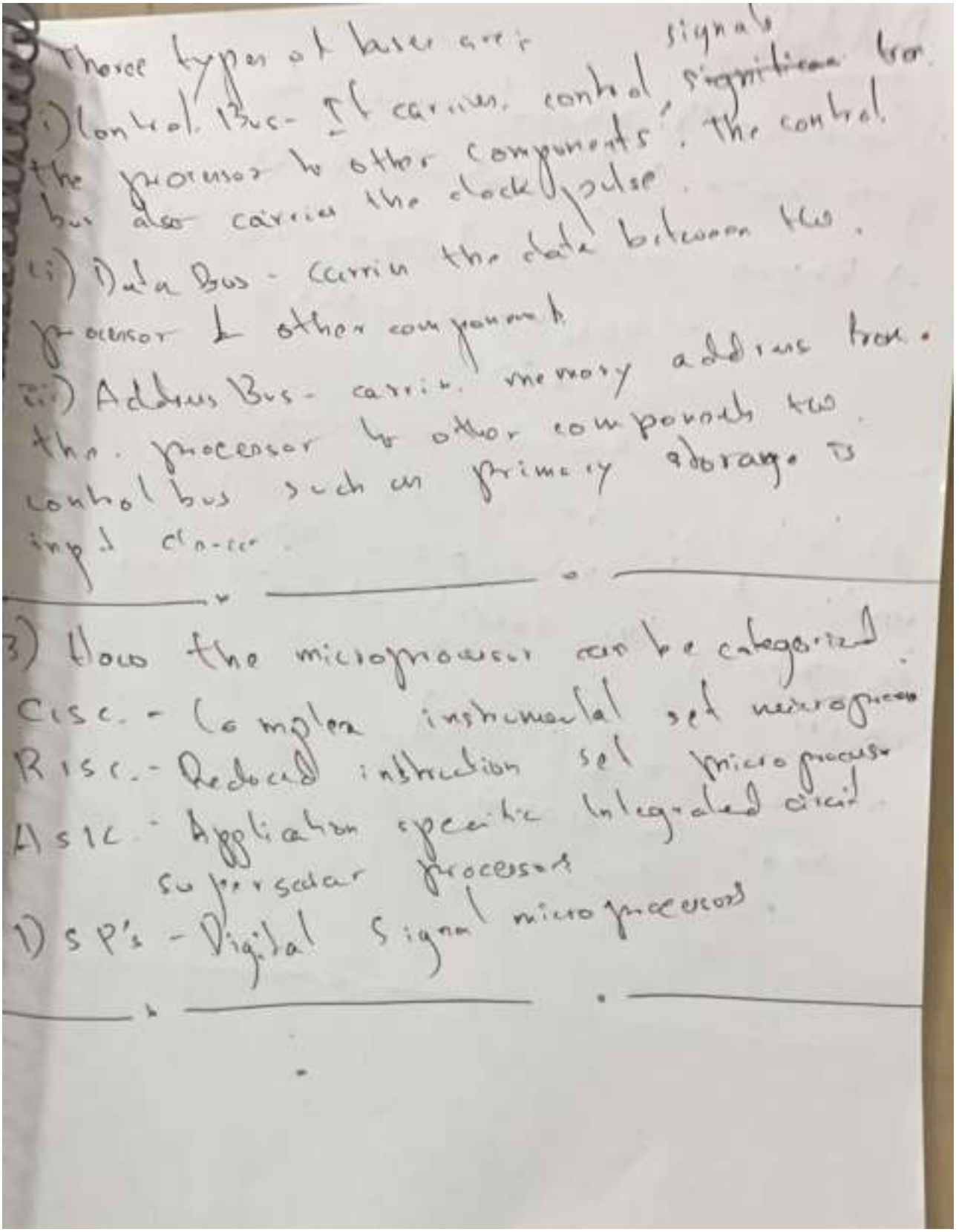
NOT Logic:





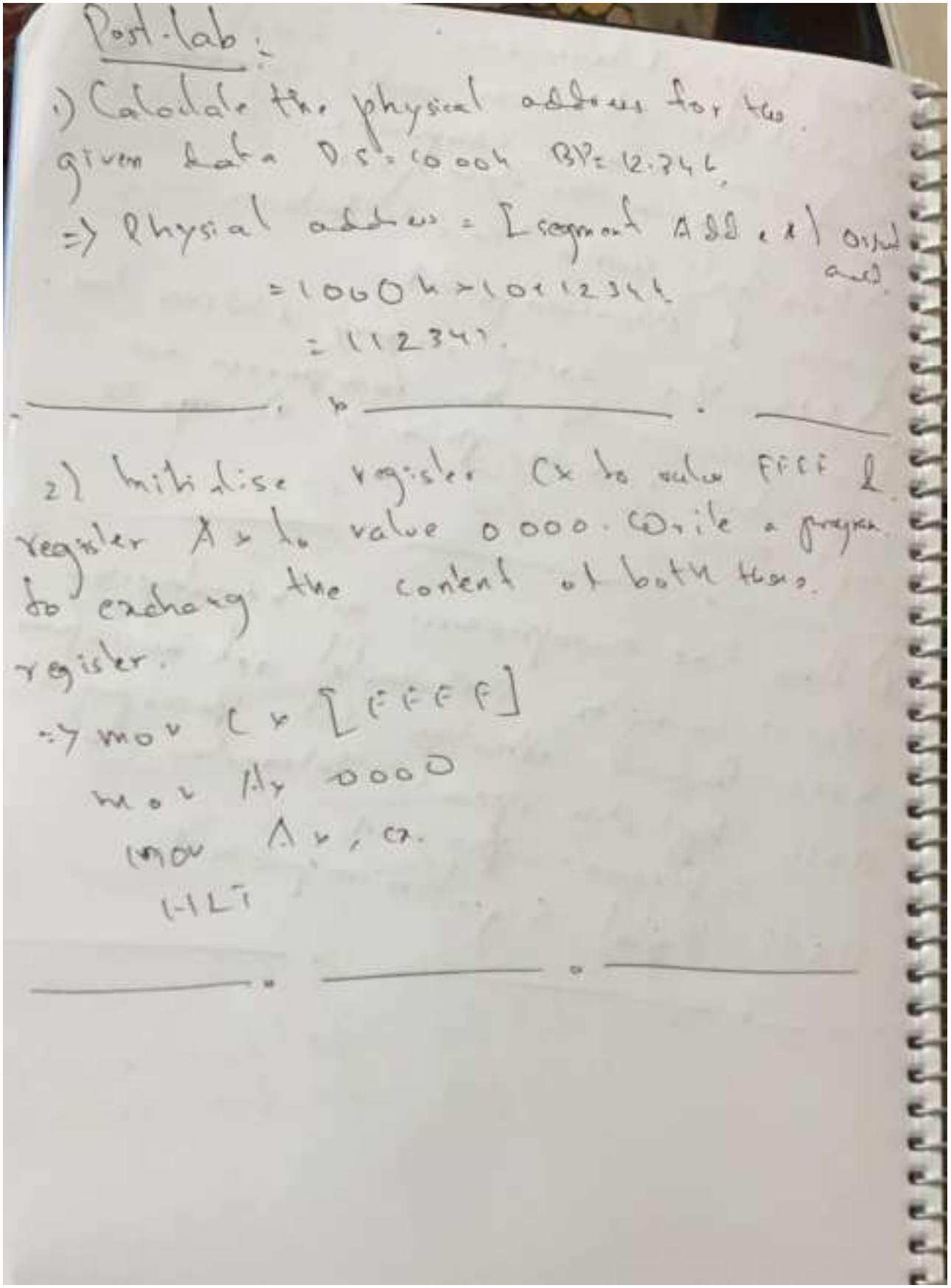
1.7 Pre Lab Questions

1. Difference between Microprocessor & Microcontroller?
2. Define BUS and give the classification of Buses
3. How the Microprocessors can be categorized?



1.8 Post Lab Questions

1. Calculate the physical address for the given data. DS=1000h, BP=1234h 2. Initialise register CX to value FFFF and register AX to value 0000. Write a program to exchange the content of both these registers.



1.9 Results and Conclusion

Thus, the registers, instruction sets, data transfer operation and logical operation of 8086 by using AND , OR in the given two 16 bit numbers and store them in a memory location was experimented.

SRM Institute of Science and Technology
Faculty of Engineering and Technology
Department of Electronics and Communication Engineering

18ECC203J Microprocessor, Microcontroller & Interfacing Techniques Lab

Fifth Semester, 2020-21 (odd semester)

Name : Pushpal Das **Register No.** RA1911004010565

: Day / Session :

Venue : Online

Title of Experiment : Data transfer and Arithmetic operations using 8086

Date of Conduction : 28/07/2021

Date of Submission : 02/08/2021 15/08/2021

Post-lab 15

Total 30

Particulars **Max.**

REPORT VERIFICATION

Marks
Pre-lab 5

Lab Performance 10

Marks Obtained

Staff Name :

Signature :

28/07/2021

Prithiviraj R

Date :

Experiment 2.
Data Transfer and Arithmetic Operation using 8086.

2.1. Aim(s) / Objective(s) / Purpose.

The purpose of this experiment is to learn about the registers, instruction sets, and arithmetic operators of 8086 by addition , subtraction ,multiplication and division in the given two 16 bit numbers and store them in a memory location.

2.2 Introduction / Background

The purpose of this experiment is to learn about the registers, instruction sets, data transfer operation and logical operation of 8086 by using ADD, SUB, MUL & DIV in the given two 16 bit numbers and store them in a memory location.

Tips: Any information copied directly or verbatim from Lab manuals or other references should be stated within quotes and referred, otherwise, it is considered plagiarism.

2.3 Materials / Equipment

1. *K. M. Bhurchandi and A. K. Ray, "Advanced Microprocessors and Peripherals-with ARM and an Introduction to Microcontrollers and Interfacing ", Tata McGraw Hill, 3rd edition 2015*

2. *Doughlas.V.Hall, "Microprocessor and Interfacing : Programming and Hardware", 3rd edition, McGraw Hill, 2015*

2.4 Hardware Requirement:

The 8086 Microprocessor kit, Power Supply. **Software Requirement :**

8086 Emulator (EMU8086)

2.5 Procedure

a. Load the program in the emulator.
b. Load the input values to the corresponding address location or directly into the registers.
c. Save, Compile and Emulate the program.
d. Execute the program using Single run or Run until termination process.
e. Store the results of the process in the destination register
f. Terminate the program

2.6 Program Logic:

The add instruction requires either the addend or the augend to be in a register, unless the source operand is immediate since the addressing modes permitted for the source and destination are register-register, memory to register, register to memory, register to immediate, and finally memory to immediate.

Hence one of the operands is initially moved to AX. Then using the add instruction, 16-bit addition is performed

The next arithmetic primitive is SUB. As discussed in ADD it permits the same modes of addressing. Hence moving the minuend to a register pair is necessary. Then the result is moved to a location in memory.

The 8086 Processor provides both signed and unsigned multiply in their instruction set to overcome the loss of efficiency in performing the repeated addition. The MUL instruction can have both 16 and 8 bit operands and the multiplicand is AX or AL, accordingly the result

for a byte multiply is a 16 bit number in AX while that for a word multiply is a 32 bit number, the lower word of which is in AX and the higher word in DX.

Program 2(a)

ADDRESS LABEL MNEMONICS OPCODE COMMENTS

01000 01001 01002 01003 01004 01005
01006 01007 01008 01009 0100A 0100B

MOV AX, [1100h] **A1,00,11 Move the data from memory 1100h to AX**

ADD AX,BX **03,C3 Add the data in AX and BX**

MOV BX, [1102h] **8B,1E,02,11 Move the data from memory 1102h to BX**

MOV [1200h],AX **A3,00,12 Move the result in Ax to memory 1200h**

0100C HLT **F4 Stop the simulation Observation**

INPUT ADDRESS	**DATA OUTPUT**	**ADDRESS DATA**
1100	**47**	
1101	**86**	
1102	**32**	
1103	**32**	

8647 – 1000 0110 0100
0111 3232 – 0011 0010
0011 0010
_____
____ B879 – 1011 1000
0111 1001
1200 79 1201 B8

Calculations:

Program 2(b)

ADDRESS LABEL MNEMONICS OPCODE COMMENTS

01000 01001 01002 01003 01004 01005
01006 01007 01008 01009 0100A 0100B

MOV AX, [1100h] **A1,00,11 Move the data from memory 1100h to AX**

SUB AX,BX **2B,C3 Subtraction operation is done**

MOV [1200h],AX **A3,00,12 Move the result from AX to memory 1200h**

MOV BX, [1102h] **8B,1E,02,11 Move the data from memory 1102h to BX**

0100C HLT **F4 Stop the simulation Observation**

INPUT ADDRESS	**DATA OUTPUT**	**ADDRESS DATA**
1200	**15**	
1201	**54**	

1100 32 1101 32 1102 47 1103 86

Calculations:

3232 – 0011 0010 0011 0010
8647 – 1000 0110 0100 0111

Take 2's complement of 8647

0111 1001 1011 1000
1

0111 1001 1011 1001 – 2s' complement

Add 3232 and 2s' complement

0011 0010 0011 0010
0111 1001 1011 1001

1010 1011 1110 1011
Invert the bits

0101 0100 0001 0100 – 5415

Program 2(c)

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
01000 01001 01002		MOV AX, [1100h]	**A1,00,11**	**Move the data from memory 1100h to AX is done**
01003 01004 01005 01006 01007 01008 01009 0100A 0100B 0100C 0100D 0100E 0100F		MOV [1200h],AX	**A3,00,12**	**The result in AX is moved to memory 1200h**
		MOV BX, [1102h]	**8B,1E,02,11**	**Move the data from memory 1102h to BX**
		MOV [1202h],DX	**89,16,02,11**	**The results in the DX are moved to memory 1202h**
		MUL BX	**F7,E3**	**Multiplication operation**
01010		HLT	**F4**	**Stop the simulation Observation**

INPUT	**ADDRESS**	**OUTPUT**	**ADDRESS**
	DATA		**DATA**
		1102	**32**
		1103	**32**
1100	**47**		
1101	**86**		

Calculations: 8647- 34375
1200 DE 1201 17 3232- 12850
1202 54 1203 1A and 3232 into decimal

Convert 8647
Multiply the decimal values

34375 x 12850 = 441718750

Convert the decimal value to binary

44178750 = 1A5417DE

Program 2(d)

ADDRESS LABEL MNEMONICS OPCODE COMMENTS

01000 01001 01002 01003
01004 01005 01006 MOV BX, [1102h] **8B,1E,02,11 Move the data**
MOV AX, [1100h] **A1,00,11 Move the data fromfrom memory 1102h to BX**
memory 1100h to AX
from AX to memory 1200h

01007 01008 01009 0100A 0100B 0100C MOV [1202h],DX **89,16,02,11 Move the**
0100D 0100E 0100F **result from DX to memory 1202h**
DIV BX **F7,F3 Division operation is done**
MOV [1200h],AX **A3,00,12 Move the result**

01010 HLT **F4 Stop the simulation Observation**

INPUT	**ADDRESS DATA**	**OUT PUT**	**ADDRESS DATA**
		1103 32	
1100 47	**1101**	**1200 02**	**1201 00**
86	**1102 32**	**1202 E3**	**1203 21**

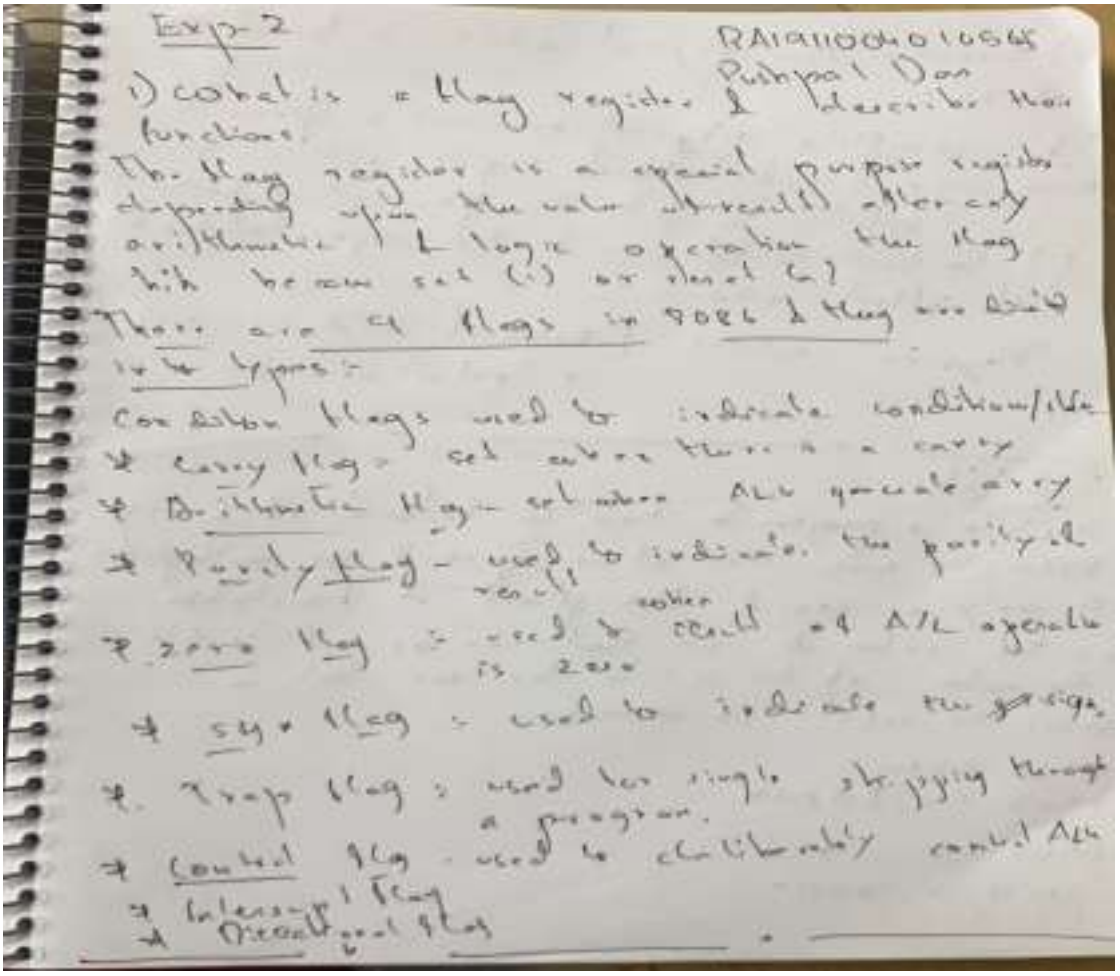
Emu8086 simulation:

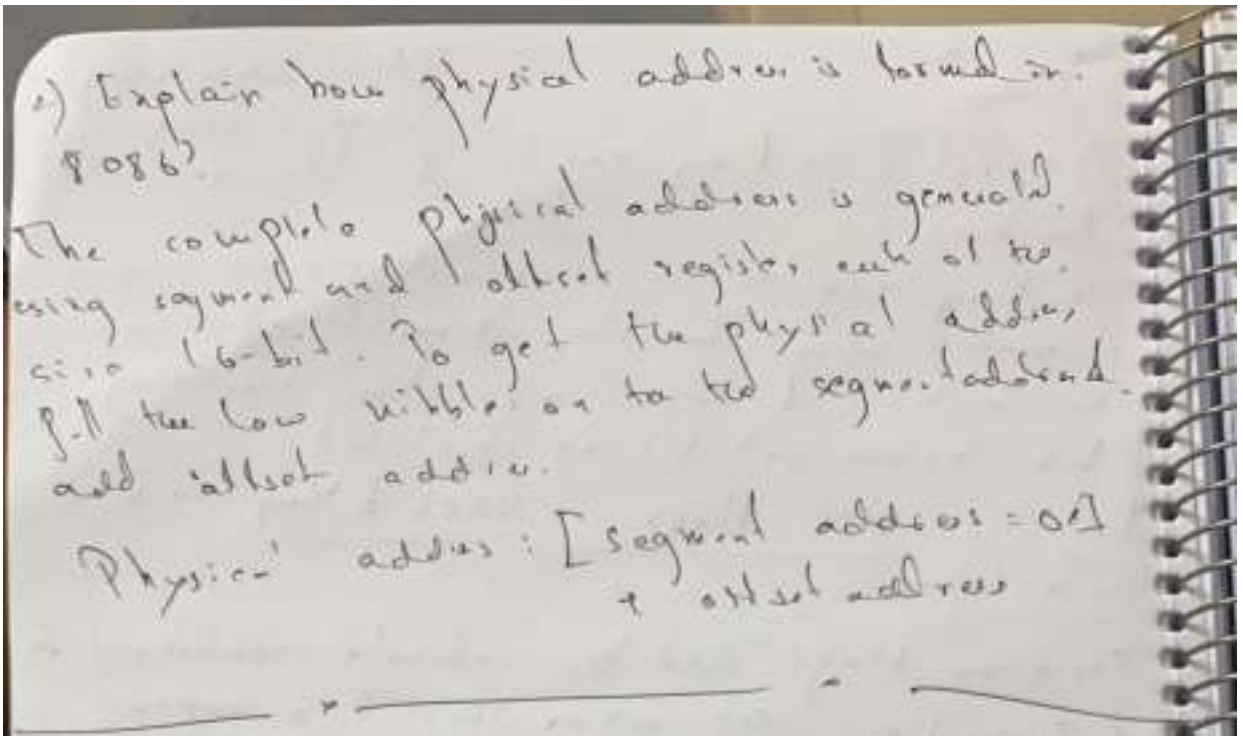
ADD:

SUB:
MUL:
DIV:

2.7 Pre Lab Questions

1. What is a Flag register? List and describe their functions.
2. Explain how physical Address is formed in 8086?

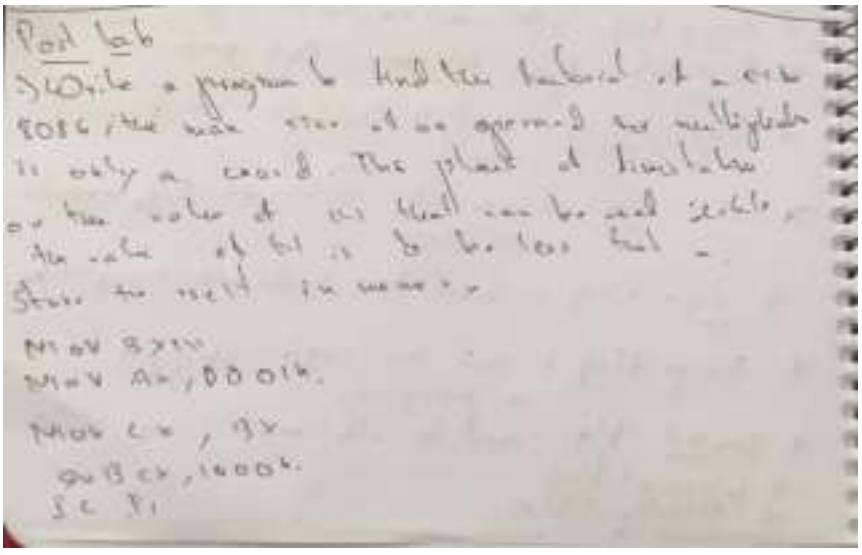


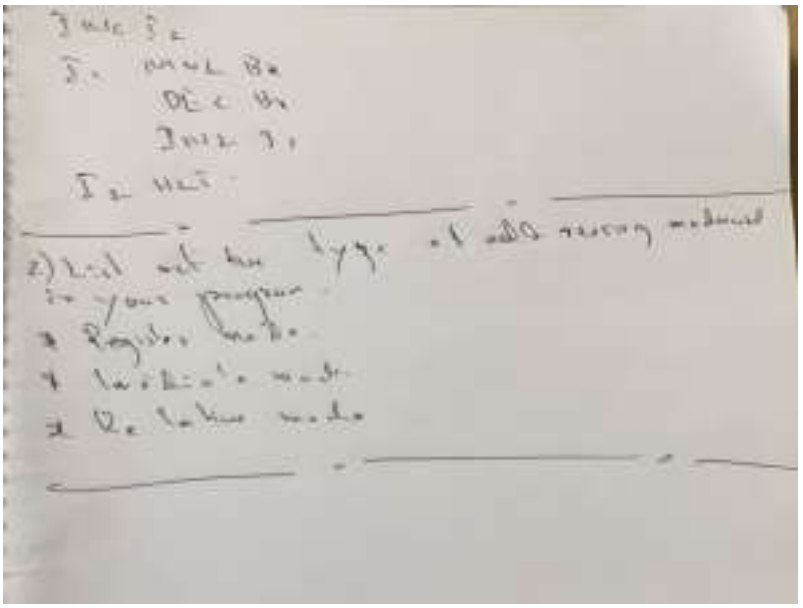


2.8 Post Lab Questions

1. Write a program to find the factorial of a number N. for 8086, the maximum size of an operand for multiplication is only a word. This places a limitation on the value of N that can be used. Hence the value of N is to be less than 9. Store the result in memory

2. List out the type of addressing modes used in your program





2.9 Result.

Thus, the registers, instruction sets, and arithmetic operators of 8086 by addition , subtraction , multiplication and division in the given two 16 bit numbers and store them in a memory location was experimented.

SRM Institute of Science and Technology

College of Engineering and Technology

Department of Electronics and Communication Engineering

18ECC203J Microprocessor, Microcontroller & Interfacing Techniques Lab

Fifth Semester, 2021-22 (Odd semester)

Name : Pushpal Das

Register No. : RA1911004010565

Day / Session : 1/ 3&4

Venue : Online (Google Meet)

Title of Experiment : **Decision making and looping operation using 8086**

Date of Conduction : 20 JULY 2021

Date of Submission : 23 AUG 2021

Particulars	Max. Marks	Marks Obtained
Pre-lab	5	
Lab Performance	10	
Post-lab	15	
Total	**30**	

REPORT VERIFICATION

Faculty Name : Mr.R.Prithiviraj

Signature :

Lab 3

Experiment 3: Decision making and looping operation using 8086

3.1 Introduction / Background

The purpose of this experiment is to learn about the registers, instruction sets, general-purpose registers, logical operators, indirect addressing, and loop instructions, compare instruction, exchange instruction, increment & decrement instruction of 8086 by

a) Finding Sum of given 'N' natural numbers

b) sorting the sequence of numbers from the array stored in a memory location into ascending order.

3.2 Hardware Requirement:

The 8086 Microprocessor kit, Power Supply.

Software Requirement:

8086 Emulator (EMU8086)

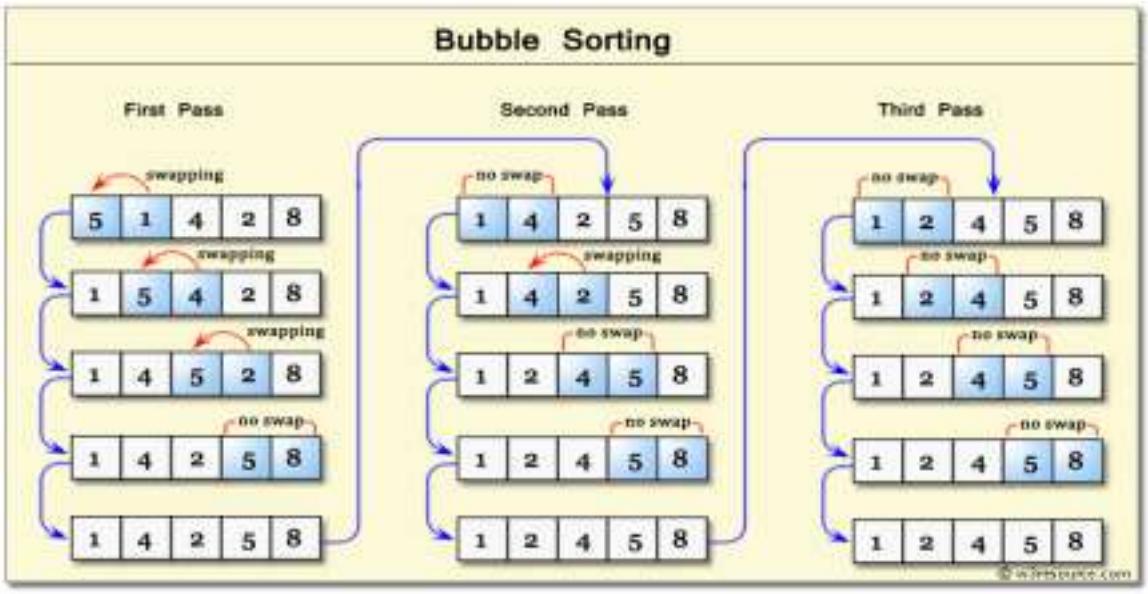
3.3 Program Logic:

a) To find the sum of 'N' natural numbers

In this program input value is entered in the memory location pointed by the source index (SI). Move content of address pointed by source index to Counter (CL). Clear AX to store sum and move 01 to BL as it is the first number. Add content of BL to AL, increment BL and decrement the count. These three steps are repeated till the count becomes zero. Store content of Register AX to destination, which is the total sum.

b) Ascending order:

To arrange the given numbers in ascending and descending order, the bubble sorting method is used. Initially the first number of the series is compared with the second one. If the first number is greater than second, exchange their positions in the series otherwise leave the position unchanged. Then compare the second number in the recent form of the series with third and repeat the exchange part that you are carried out for the first and second number, and for all the remaining number of the series. Repeat this procedure for complete series (n-1) times. After n-1 iterations you will get the largest number at the end of the series. Again, start from the first number of the series. Repeat the same procedure right from the first element to the last element. After n-2 iteration you will get the second highest number at the last but one place in the series. Repeat this till the complete series is arranged in ascending order.



3.4 Program

Sum of 'N' number:

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
01000 **01001** **01002**		MOV SI,1100h	**BE,00,11**	**Move data in 1100h to Stack Index**
01003 **01004**		MOV CL,[SI]	**8A,0C**	**Move the address of SI to CL**
01005 **01006**		MOV AX,0000h	**B0,00**	**The data in 0000h to AX**
01007 **01008**		MOV BL, 01h	**B3,01**	**Move 01h to BL**
01009 **0100A**	Back	ADD AL, BL	**02,C3**	**Data in AL and BL are added**
0100B **0100C**		INC BL	**FE,C3**	**Increment BL**
0100D **0100E**		DEC CL	**FE,C9**	**Decrement CL**
0100F **01010**		JNZ Back	**75,F8**	**If CL is not equal to 0 then the loop jumps to 'Back'**
01011 **01012** **01013**		MOV DI, 1200h	**BF,00,12**	**Move data in 1200h to DI**
01014		MOV [DI], AX	**89,05**	**Move the result from**

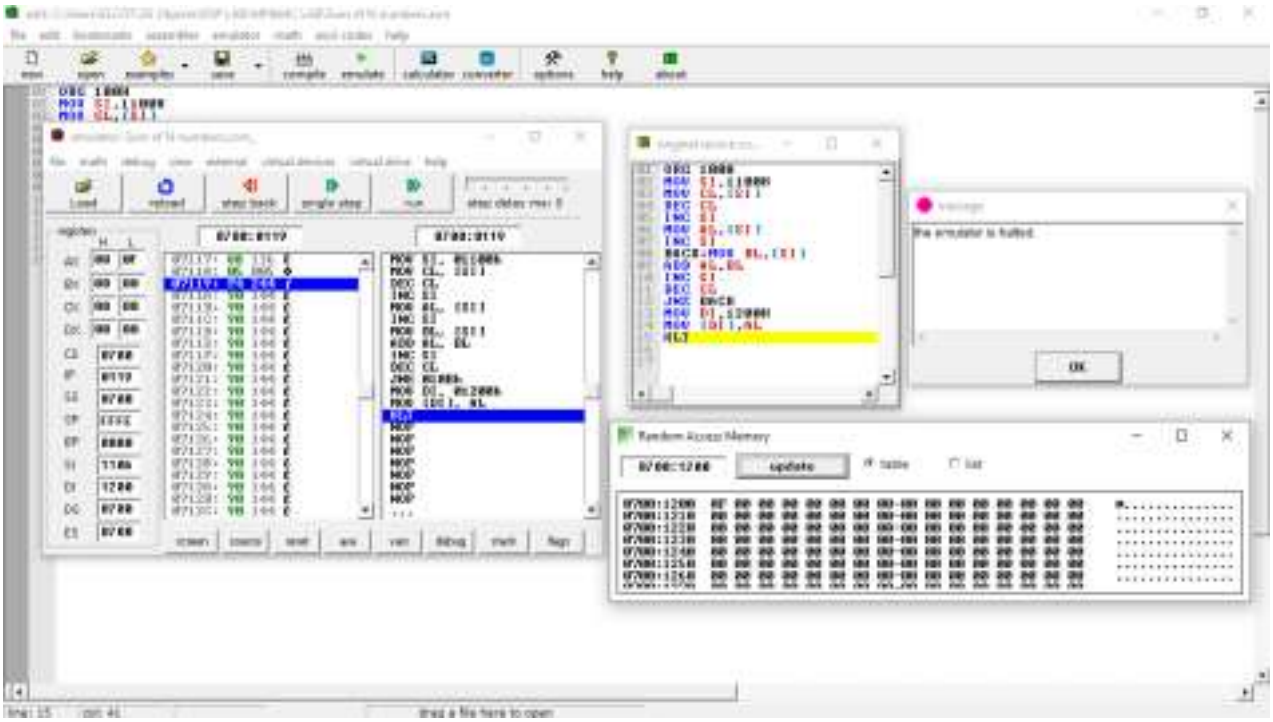
01015				**AX to address DI**
01016		HLT	**F4**	**Stop the simulation**

Observation

IN PUT ADDRESS	**DATA**
1100	**07**

OUT PUT ADDRESS	**DATA**
1200	**1C**

CODE & SIMULATION:-



Calculation:

1+2+3+4+5+6+7 = 28 = 1C (HEXADECIMAL)

Ascending order:

ADDRESS	**LABEL**	**MNEMONICS**	**OPCODE**	**COMMENTS**

01000 01001 01002		MOV SI,1200h	**BE,00,12**	**Move the data in 1200h to Stack Index**
01003 01004		MOV CL,[SI]	**8A,0C**	**Move the address SI to CL**
01005 01006		DEC CL	**FE,C9**	**Decrement CL**
01007 01008 01009	LOOP2	MOV SI,1200h	**BE,00,12**	**Move 1200h to SI**
0100A 0100B		MOV CH,[SI]	**8A,2C**	**Move address SI to CH**
0100C 0100D		DEC CH	**FE,CD**	**Decrement CH**
0100E		INC SI	**46**	**Increment SI**
0100F 01010	LOOP1	MOV AL,[SI]	**8A,04**	**Move address SI to AL**
01011		INC SI	**46**	**Increment SI**

01012 01013		CMP AL,[SI]	**3A,04**	**Compare AL and address SI**
01014 01015		JC Ahead	**72,05**	**If carry exists, jump to Loop 1**
01016 01017		XCHG AL,[SI]	**86,04**	**Exchange AL and address SI**
01018 01019 0101A		XCHG [SI 1],AL	**86,44,FF**	**Exchange address SI-1 to AL**
0101B 0101C	Ahead	DEC CH	**FE,CD**	**Decrement CH**
0101D 0101E		JNZ LOOP1	**75,F0**	**Jump to LOOP 1 if CH not equal to 0**
0101F 01020		DEC CL	**FE,C9**	**Decrement CL**
01021		JNZ LOOP2	**75,E4**	**Jump to LOOP 2 if CL**

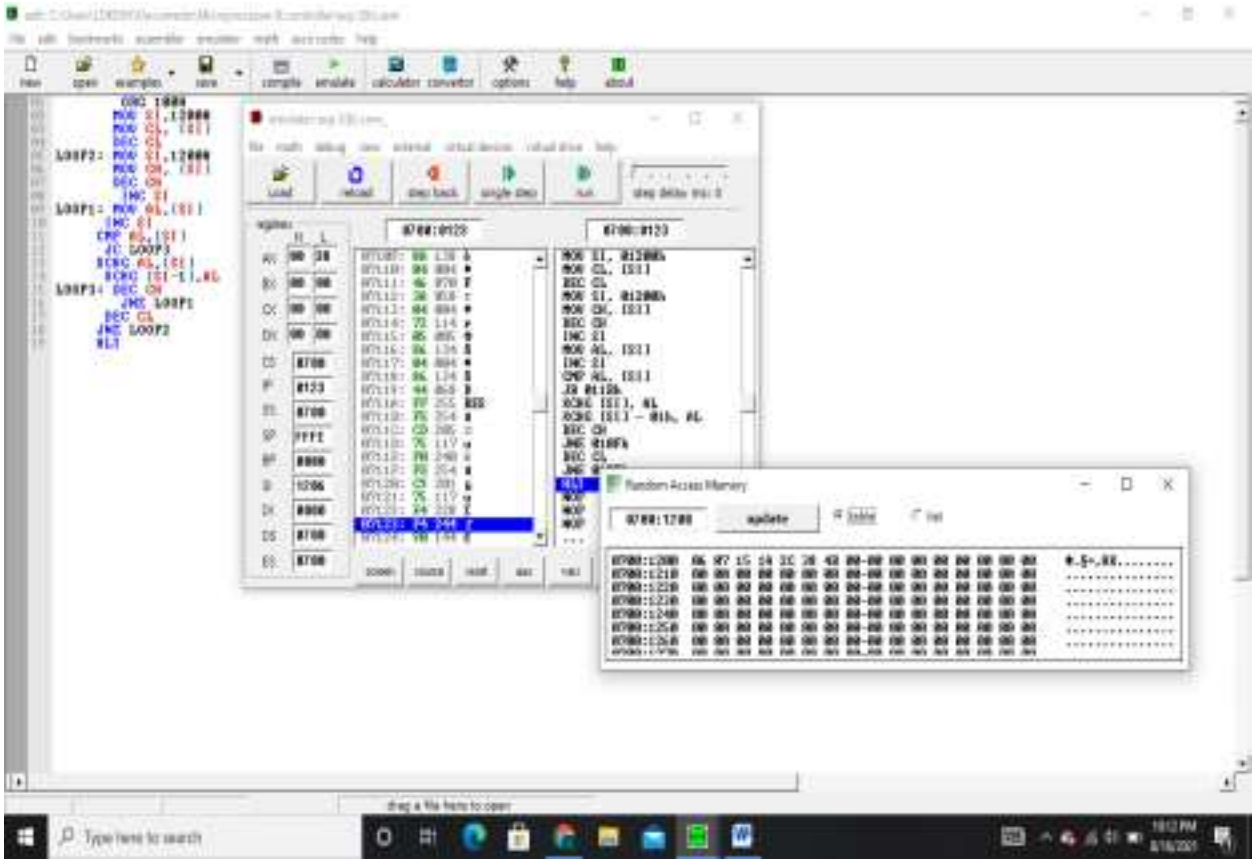
01022				**not equal to 0**

01023		HLT	**F4**	**Stop the simulation**

Observation:

1200	No. of Elements	5
1201	DATA 1	05
1202	DATA 2	8F
1203	DATA 3	02
1204	DATA 4	1E
1205	DATA 5	09
After Sorting		
1201	DATA 1	02
1202	DATA 2	05
1203	DATA 3	09
1204	DATA 4	1E
1205	DATA 5	8F

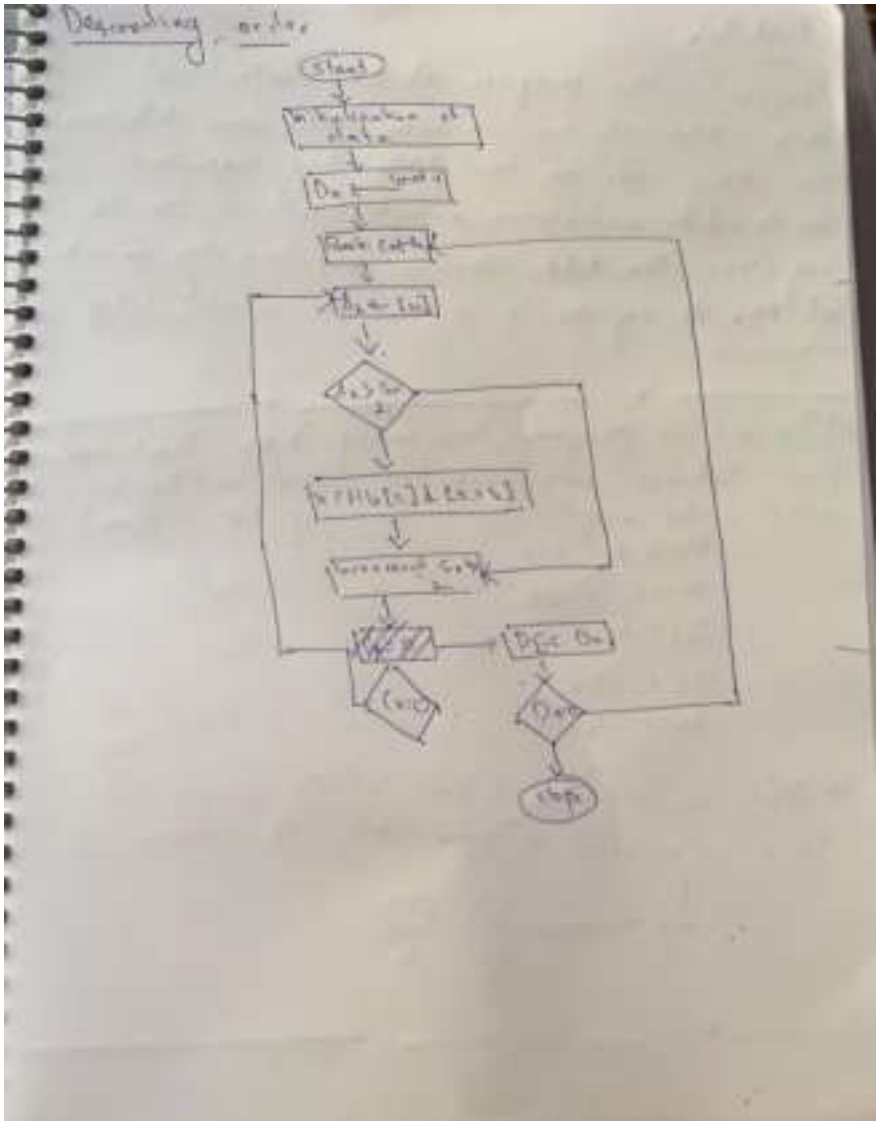
CODE & SIMULATION:-



3.5 Pre-Lab Questions:

1. What is the purpose of XCHG instruction?
2. Draw the flow chart to arrange a given series of numbers in ascending and descending order. **PRE**

LAB:-

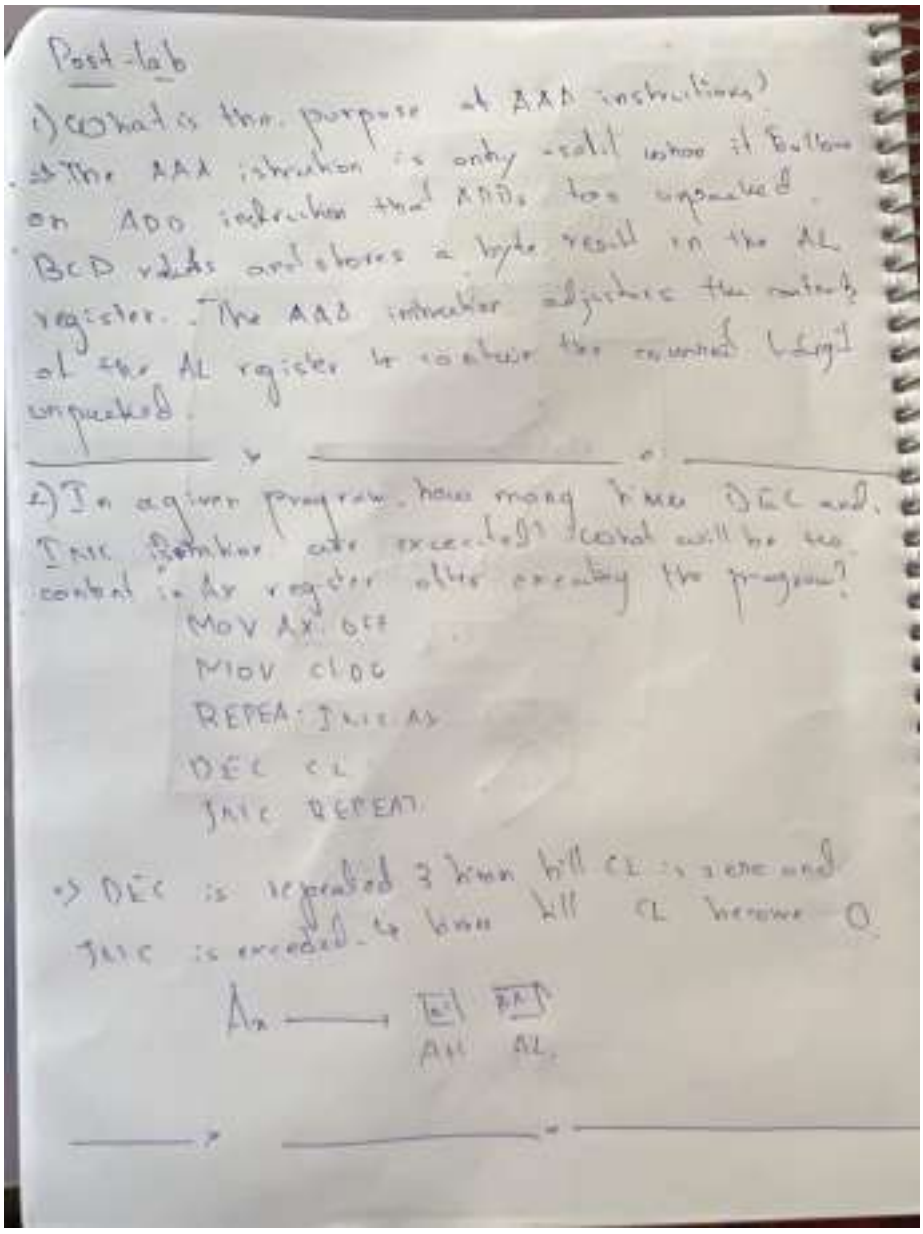


3.6 Post-Lab Questions:

1. What is the purpose of AAA instruction?
2. In a given program how many times DEC and JNZ instructions are executed? What will be content

in AX register after executing the program?

```
        MOV AX, 00FF

        MOV CL, 05
REPEAT: INC AX
        DEC CL
        JNZ REPEAT
```



3.7 Result:

Thus, the decision making and looping instructions of 8086 carried out through the sum of ‘N’ numbers and ascending order sorting operations.

Laboratory Report Cover Sheet

SRM Institute of Science and Technology
Faculty of Engineering and Technology
Department of Electronics and Communication Engineering

18ECC203J Microprocessor, Microcontroller & Interfacing Techniques Lab

Fifth Semester, 2020-21 (odd semester)

Name : **Pushpal Das**

Register No. : **RA1911004010565**

Day / Session : **01/ FN**

Venue : **ONLINE(G Meet)**

Title of Experiment : **4.PROGRAM TO DEMONSTRATE STRING OPERATION**

Date of Conduction : **09 AUG 2021**

Date of Submission : **17 SEP 2021**

Particulars	Max. Marks	Marks Obtained
Pre-lab	5	
Lab Performance	10	
Post-lab	15	
Total	**30**	

REPORT VERIFICATION

Date : **17 SEP 2021**

Staff Name : **Mr.R.Prithiviraj**

Signature :

Experiment 4.

Program to Demonstrate String Operation.

4.1 Aim(s) / Objective(s) / Purpose.

The purpose of this experiment is to learn about the instruction sets, addressing modes and to perform string operation by counting the number of characters stored in the memory.

4.2 Introduction / Background

The purpose of this experiment is to find the number of character in a string

Tips: Any information copied directly or verbatim from Lab manuals or other references should be stated within quotes and referred, otherwise, it is considered plagiarism.

4.3 Materials / Equipment

1. *K. M. Bhurchandi and A. K. Ray, "Advanced Microprocessors and Peripherals-with ARM and an Introduction to Microcontrollers and Interfacing ", Tata McGraw Hill, 3rd edition 2015*

2. *Doughlas.V.Hall, "Microprocessor and Interfacing : Programming and Hardware", 3rd edition, McGraw Hill, 2015*

4.4 Hardware Requirement:

The 8086 Microprocessor kit, Power Supply.

Software Requirement :

8086 Emulator (EMU8086)

4.5 Procedure

a. Load the program in the emulator.
b. Load the input values to the corresponding address location or directly into the registers.
c. Save, Compile and Emulate the program.
d. Execute the program using Single run or Run until termination process.
e. Store the results of the process in the destination register
f. Terminate the program

4.6 Program Logic:

Addressing the string is done using SI register, and the DX register is used to store the number of character. End of string is detected using FF. Hence each character is fetched from memory and is compared with FF.

If the zero flag is set, then it denotes end of string, the count have been stored in DX, by incrementing it after each comparison.

Program 4

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
1000		MOV SI,1200	**BE 00 12**	Moves data from 1200H memory location to SI
1003		MOV DX,FFFF	**BA FF FF**	Moves 0FFFFH to DX register
1006		MOV AH,FF	**B4 FF**	Moves 0FFH to AH register
1008	LOOP	INC DX	**42**	Increments DX by 1
1009		MOV AL,[SI]	**8A 04**	Data in SI moved to AL
100B		INC SI	**46**	Increment SI by 1
100C		CMP AH,AL	**3A E0**	Compares data in AH and AL
100E		JNZ LOOP	**78 F8**	Checks for zero condition
1010		MOV [1100],DX	**89 16 00 11**	Contents of DX register are moved to 1100H memory location
1014		HLT	**F4**	Emulator is halted

Observation

IN PUT ADDRESS	DATA
1200	**23**
1201	**01**
1202	**05**
1203	**38**

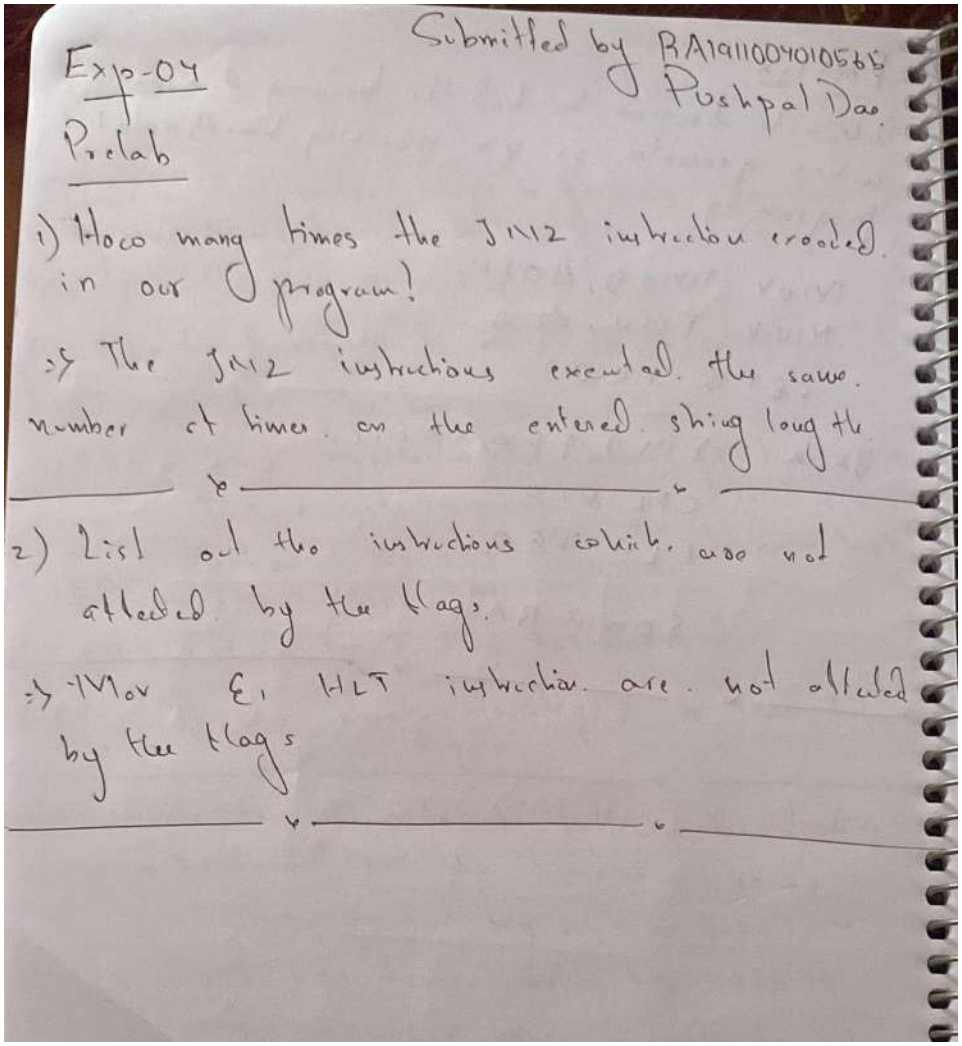
1204	**FF**

OUT PUT ADDRESS	**DATA**
1100	**04**
1101	**00**

4.7 Pre Lab Questions

1. How many times the JNZ instruction executed in our program?
2. In our program, List out the instructions which are not affected by the flags.

PRE LAB:-



4.8 Post Lab Questions

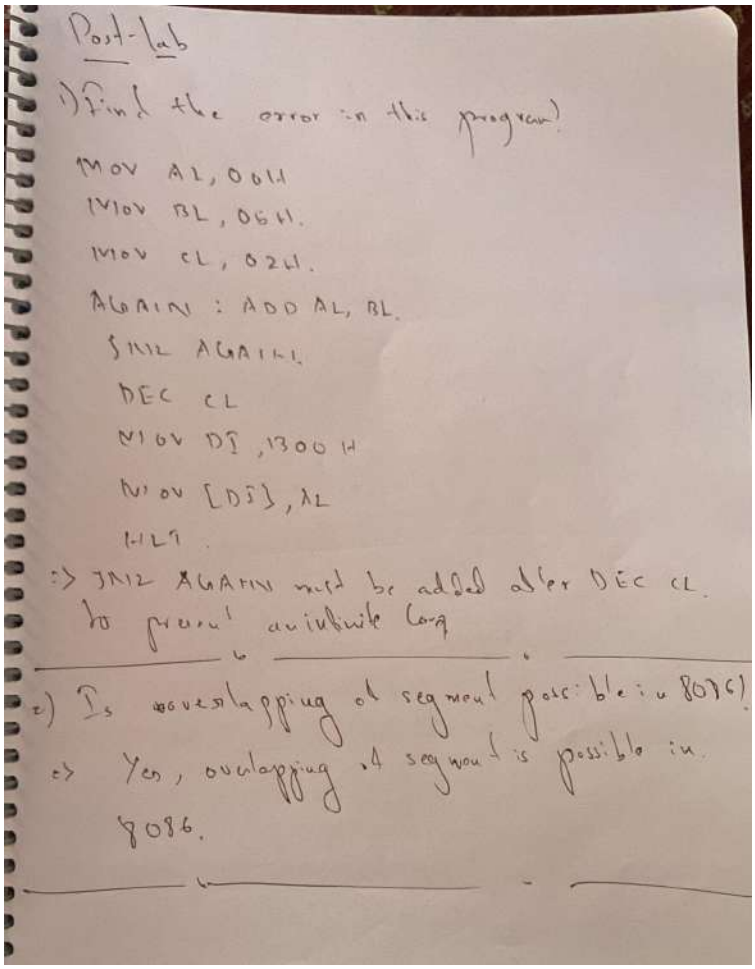
1. Find the error in this program?

```
MOV AL, 00H
MOV BL, 05H
MOV CL, 02H
AGAIN:  ADD AL, BL
JNZ AGAIN
DEC CL
MOV DI, 1300H
MOV [DI], AL
```

HLT

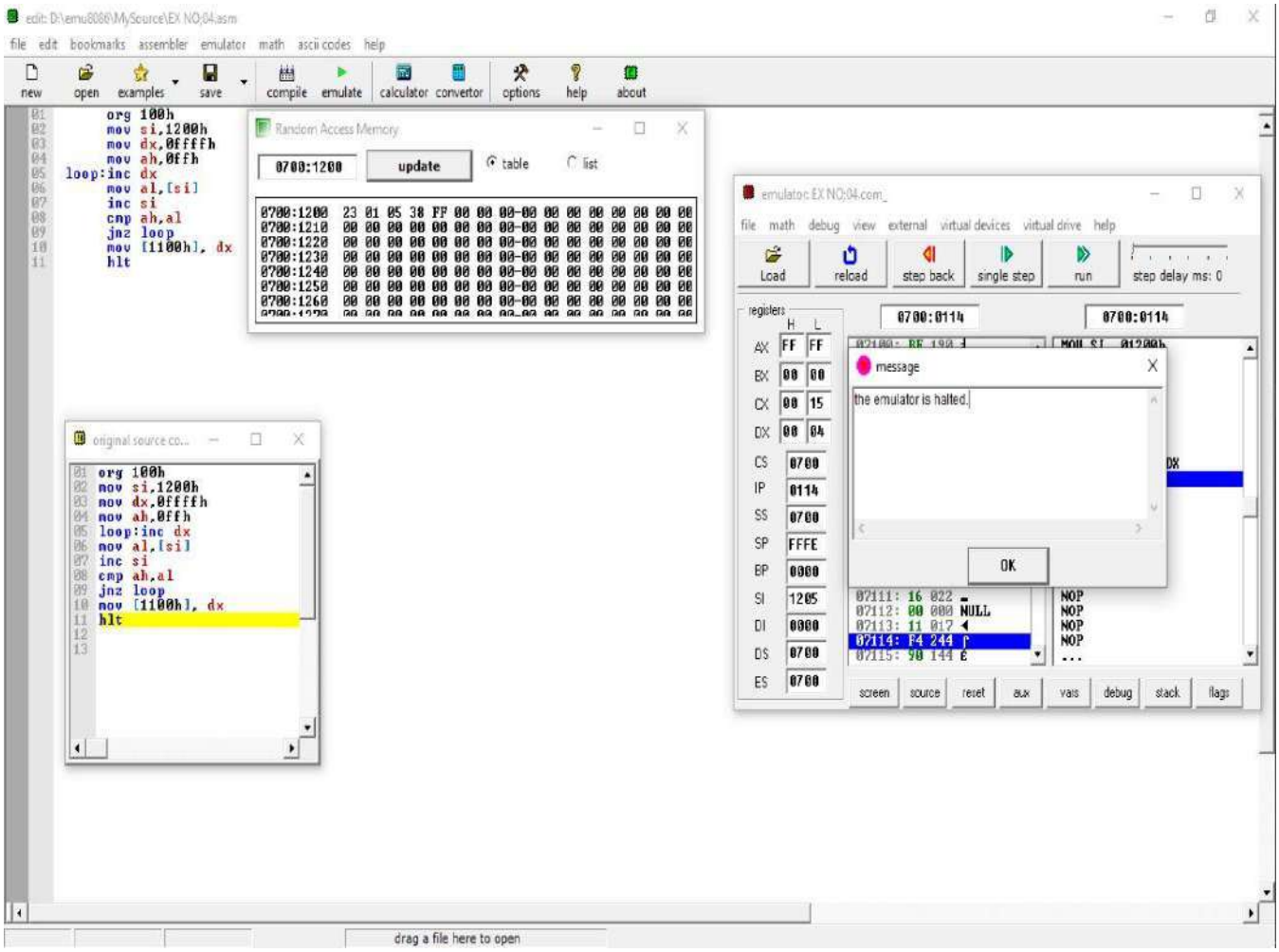
2. Is overlapping of segment possible in 8086?

POST LAB:-

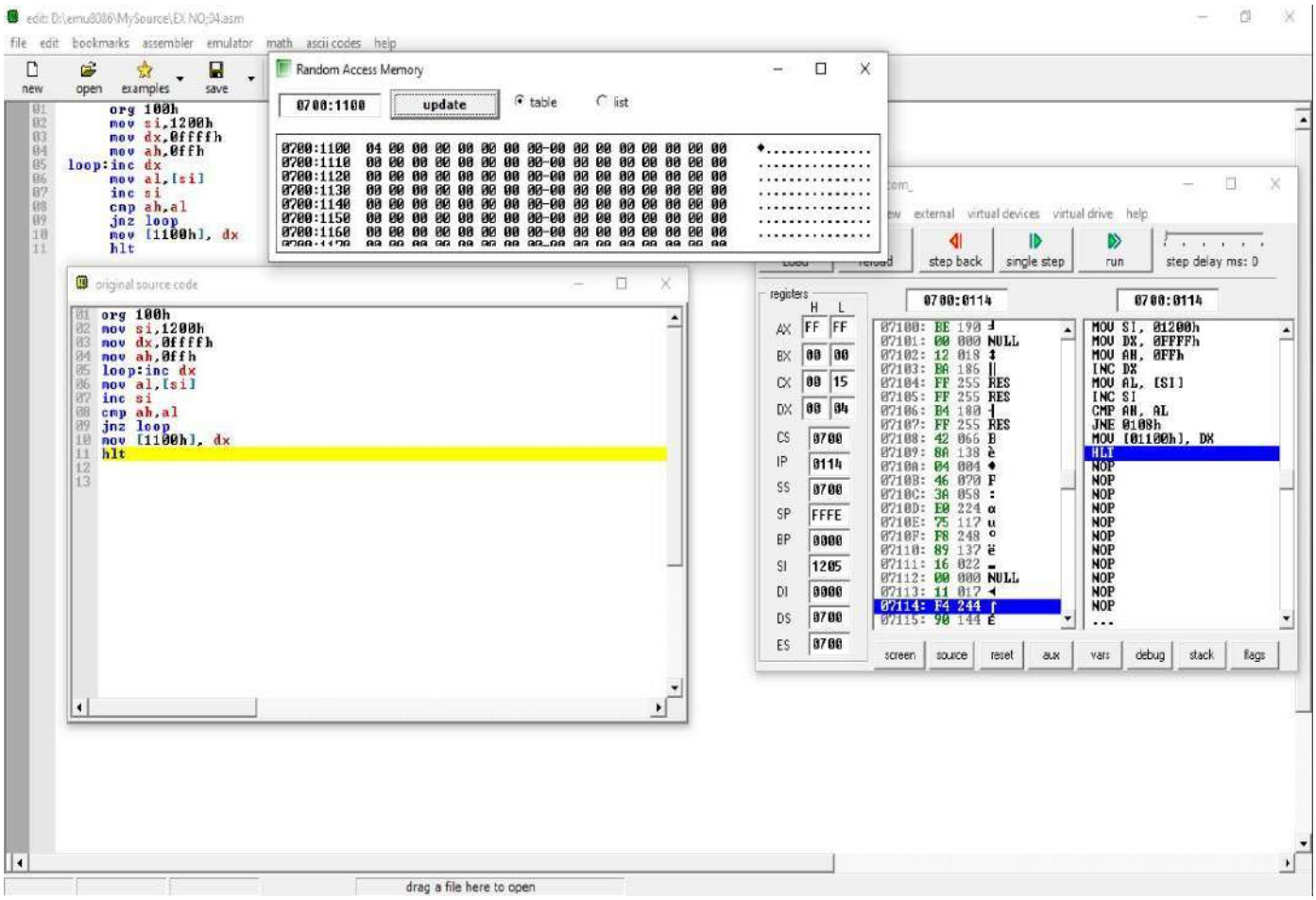


EMULATOR OUTPUT :-

INPUT : 23 01 05 38 FF


edit: D:\emu8086\MySource\EX NO;04.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
org 100h
mov si,1200h
mov dx,0ffffh
mov ah,0ffh
loop:inc dx
mov al,[si]
inc si
cmp ah,al
jnz loop
mov [1100h], dx
hlt
Random Access Memory
0700:1200
update
table
list
original source co...
drag a file here to open
emulator EX NO;04.com_
file math debug view external virtual devices virtual drive help
Load
reload
step back
single step
run
step delay ms: 0
registers
AX FF FF
BX 00 00
CX 00 15
DX 00 04
CS 0700
IP 0114
SS 0700
SP FFFE
BP 0000
SI 1205
DI 0000
DS 0700
ES 0700
0700:0114
0700:0114
message
the emulator is halted.
OK
NOP
screen source reset aux vars debug stack flags

OUTPUT : 04 00



4.9 Results :-

Thus, the instruction sets, addressing modes and to perform string operation by counting the number of characters stored in the memory was experimented.

Laboratory Report Cover Sheet

SRM Institute of Science and Technology
Faculty of Engineering and Technology
Department of Electronics and Communication Engineering

18ECC203J Microprocessor, Microcontroller & Interfacing Techniques Lab
Fifth Semester, 2020-21 (odd semester)

Name : **Pushpal Das**

Register No. : **RA1911004010565**

Day / Session : **1ST / 3&4**

Venue : **Online (G Mett)**

Title of Experiment : **Program To Demonstrate Rotate And Shift Operation Using 8086**

Date of Conduction : **17 AUG 21**

Date of Submission : **29 SEP 21**

Particulars	Max. Marks	Marks Obtained
Pre-lab	5	
Lab Performance	10	
Post-lab	15	
Total	30	

REPORT VERIFICATION

Date : 29 SEP 21
Staff Name : Mr.R.Prithiviraj

Signature :

Experiment 5.
Shift and Rotate operation

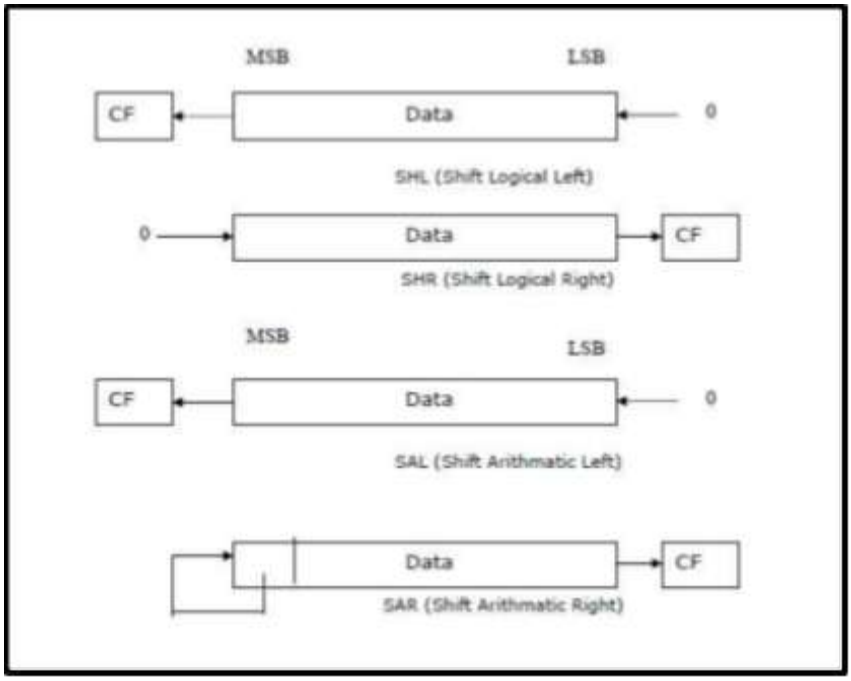
5.1 Aim(s) / Objective(s) / Purpose.

The purpose of this experiment is to learn Rotate and Shift commands in assembly language.

5.2 Introduction / Background

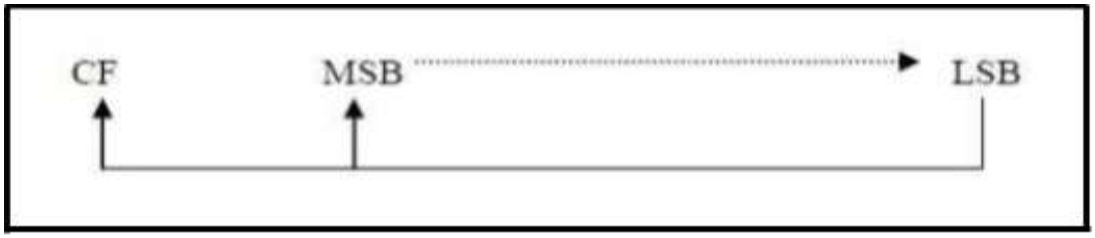
Shift and Rotate command: Shift and Rotate commands are used to convert a number to another form where some bits are shifted or rotated. Basic difference between “shift” and “rotate” is shift command makes “fall of” bits at the end of register whereas rotate command makes “Wrap around” at the end of the register. There are both arithmetic (SAL and SAR) and logical (SHL and SHR) Shift instructions. Graphical operations for these commands are shown below. SAL and SHL are two mnemonics for the same instruction. This instruction shifts each bit in the specified destination some number of bit positions to the left. As a bit is shifted out of the LSB operation, a 0 is put in the LSB position. The MSB will be shifted into CF. In the case of multi-bit shift, CF will contain the bit most recently shifted out from the MSB. Bits shifted into CF previously will be lost. The SAR instruction shifts each bit in the specified destination some number of bit positions to the right. As a bit is shifted out of the MSB position, a copy of the old MSB is put in the MSB position. In other words, the sign bit is copied into the MSB. The LSB will be shifted into CF. In the case of multiple bit shift, CF will contain the bit most recently shifted out from the LSB. Bits shifted into CF previously will be lost. The SHR instruction shifts each bit in the specified destination some number of bit positions to the right. As a bit is shifted out of the MSB position, a 0 is put in its place. The bit shifted out of the LSB position goes to

CF. In the case of multi- 4 bit shifts, CF will contain the bit most recently shifted out from the LSB. Bits shifted into CF previously will be lost.

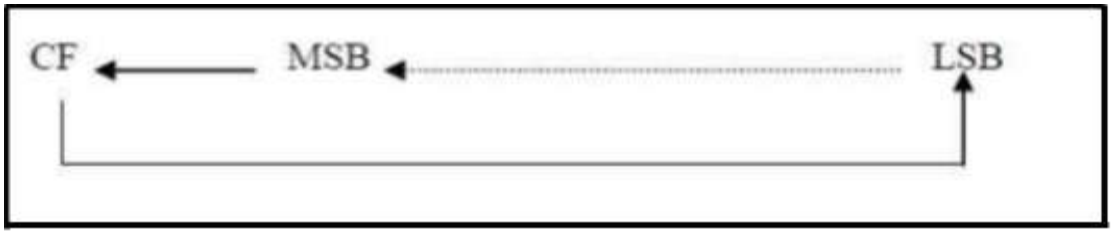


There are 4 types (ROL, ROR, RCL, RCR) of Rotate instructions. ROL = Rotate LEFT This instruction rotates all the bits in a specified word or byte to the left some number of bit positions. The data bit rotated out of MSB is circled back into the LSB. It is also copied into CF. In the case of multiple-bit rotate, CF will contain a copy of the bit most recently moved out of the MSB.

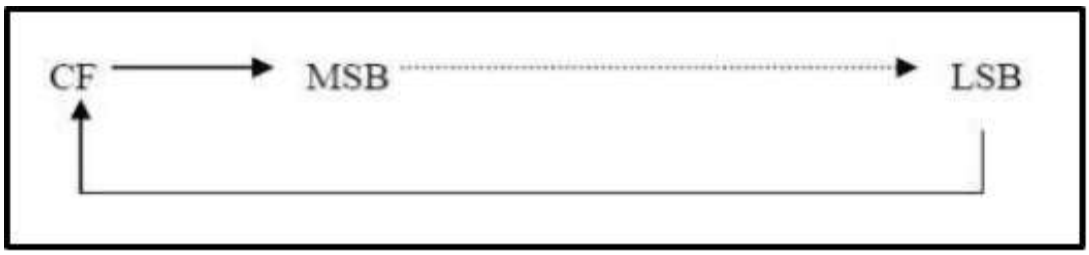
1. ROR = Rotate RIGHT, This instruction rotates all the bits in a specified word or byte some number of bit positions to right. The operation is desired as a rotate rather than shift, because the bit moved out of the LSB is rotated around into the MSB. The data bit moved out of the LSB is also copied into CF. In the case of multiple bit rotates, CF will contain a copy of the bit most recently moved out of the LSB.



2. RCL – Rotate with Carry LEFT; RCL Destination, Count This instruction rotates all the bits in a specified word or byte some number of bit positions to the left. The operation is circular because the MSB of the operand is rotated into the carry flag and the bit in the carry flag is rotated around into LSB of the operand. For multi-bit rotates, CF will contain the bit most recently rotated out of the MSB.



3. RCR – Rotate with Carry RIGHT; RCR Destination, Count This instruction rotates all the bits in a specified word or byte some number of bit positions to the right. The operation is circular because the LSB of the operand is rotated into the carry flag and the bit in the carry flag is rotate around into MSB of the operand. For multi-bit rotate, CF will contain the bit most recently rotated out of the LSB.



5.3 Software Requirement :

8086 Emulator (EMU8086)

5.4 Procedure:

a. Load the program in the emulator.
b. Load the input values to the corresponding address location or directly into the registers.
c. Save, Compile and Emulate the program.
d. Execute the program using Single run or Run until termination process.
e. Store the results of the process in the destination register
f. Terminate the program

5.5 Program Logic:

Addressing the string is done using the SI register, and the BL register is used to store the number of positive numbers, BH register is used to store the number of negative numbers. Hence each data is fetched from memory and is shifted left by one position. If the carry flag is set, then it denotes negative number, else positive number. The inputs from 00H to 7FH are positive numbers and 80H to FFH are negative numbers.

Program 5

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
1000		MOV BL,00H ;	B3, 00	Moves data from 00H memory location to BL.
1002		MOV BH,00H;	B7, 00	Moves data from 00H memory location to BH.
1004		MOV SI,1100H;	BE, 00, 11	Moves data from 1100H memory location to SI.
1007		MOV CL,01H;	B1, 01	Moves data from 01H memory location to CL.
1009		MOV CH,[SI];	8A, 2C	Data in SI moved to CH.
100B		INC SI;	46	Increment SI by 1.
100C	LOOP:	MOV AL,[SI];	8A, 04	Data in SI moved to AL.
100E		SHL AL,CL;	D2, E0	Shifts each bit in AL to the left CL times and 0 is stored at LSB position.
1010		JC NEGATIVE;	72, 04	Jump to negative if carry is 1.
1012		INC BL;	FE, C3	Increment BL by 1.
1014		JMP NEXT;	EB, 02	Go to Next.
1016	NEGATIVE:	INC BH;	FE, C7	Increment BH by 1.
1018	NEXT:	ADD SI,01H;	83, C6, 01	Add 01H and SI.
101B		DEC CH;	FE, CD	Decrement CH by 1.
101D		JNZ LOOP;	75, ED	Jump to Loop if the Zero flag is 0.
101F		MOV [1200],BL;	88, 1E, 00, 12	Contents of the BL register are moved to the 1200H memory location.

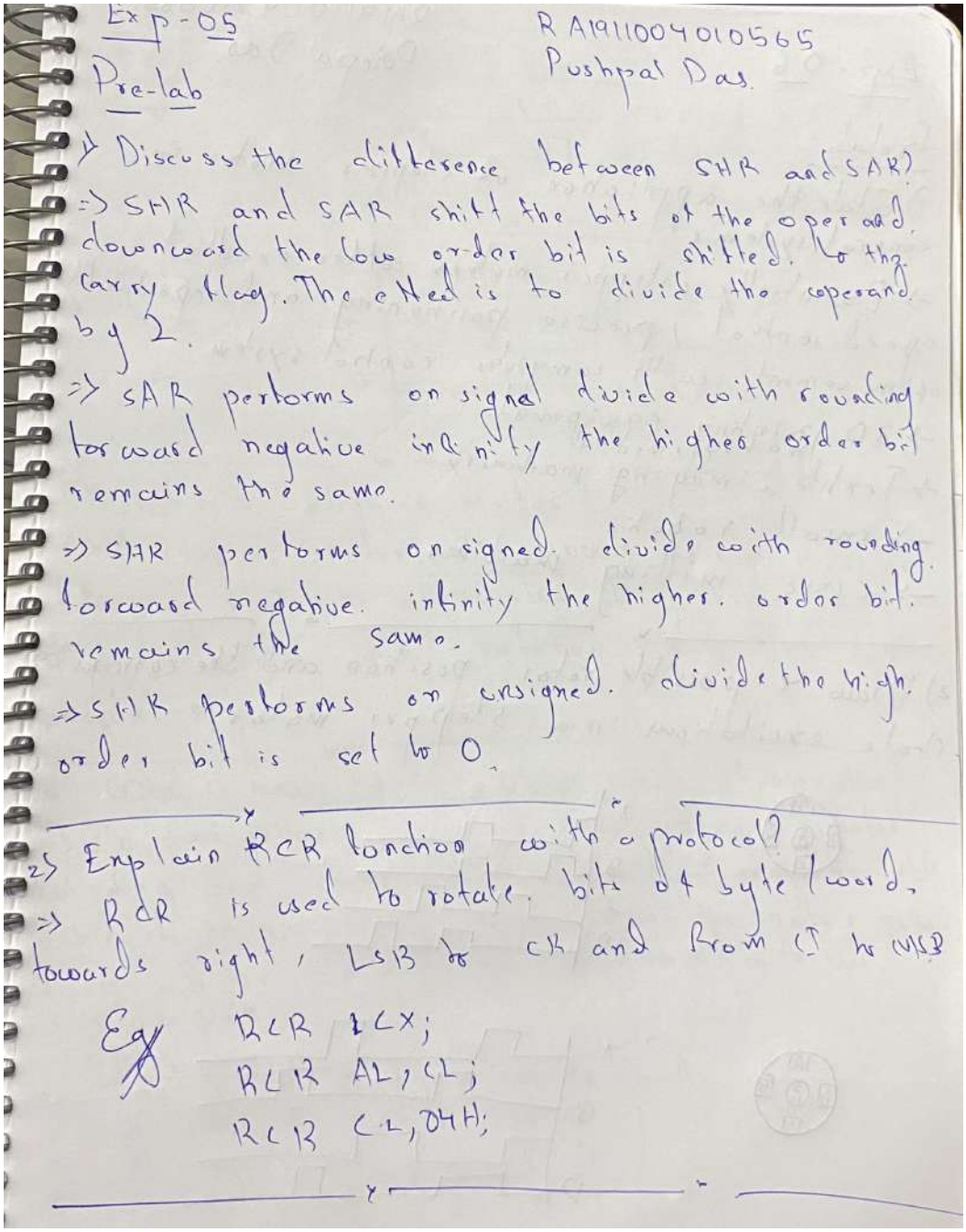
1023		MOV[1201],BH;	88, 3E, 01, 12	Contents of the BH register are moved to the 1201H memory location.
1027		HLT	F4	Program Terminated

Observation

INPUT ADDRESS	DATA
1100	05
1101	78
1102	78
1103	80
1104	81
1105	82
OUTPUT ADDRESS	DATA
1200	02
1201	03

5.7 Pre Lab Questions

1. Discuss the difference between SHR and SAR?
2. Explain RCR function with a protocol?



Exp - 05

RA1911004010565
Pushpal Das.

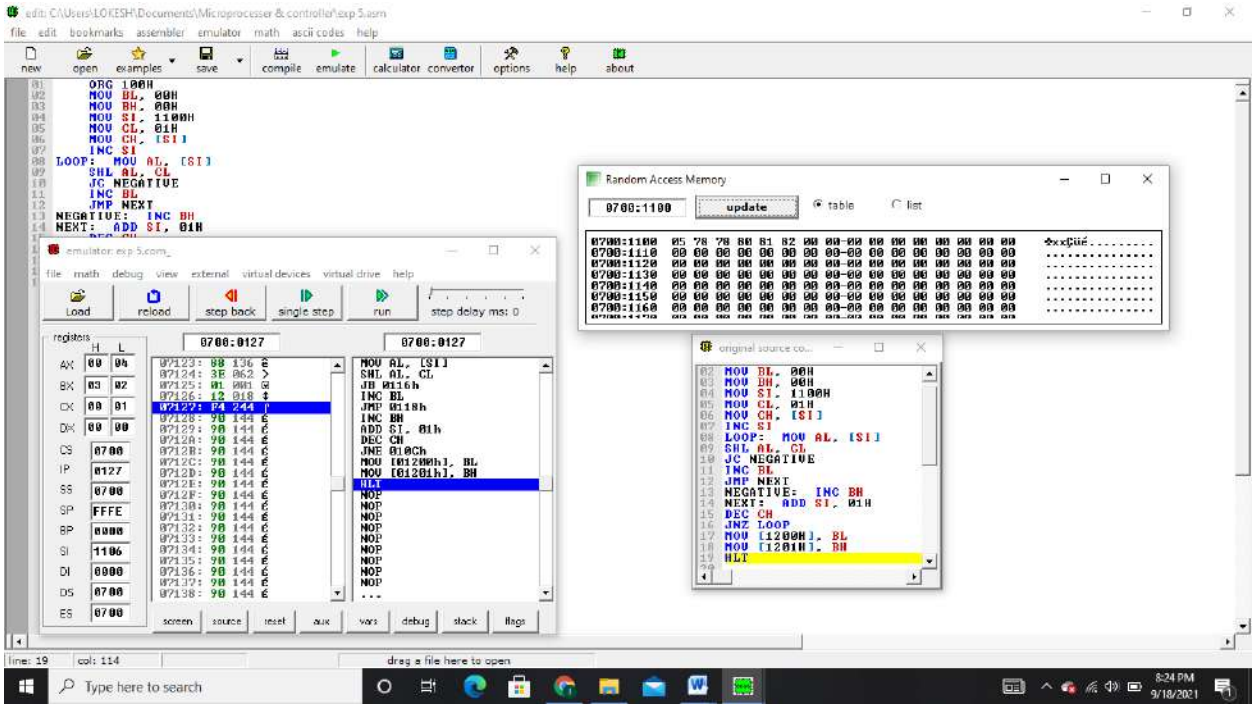
Pre-lab

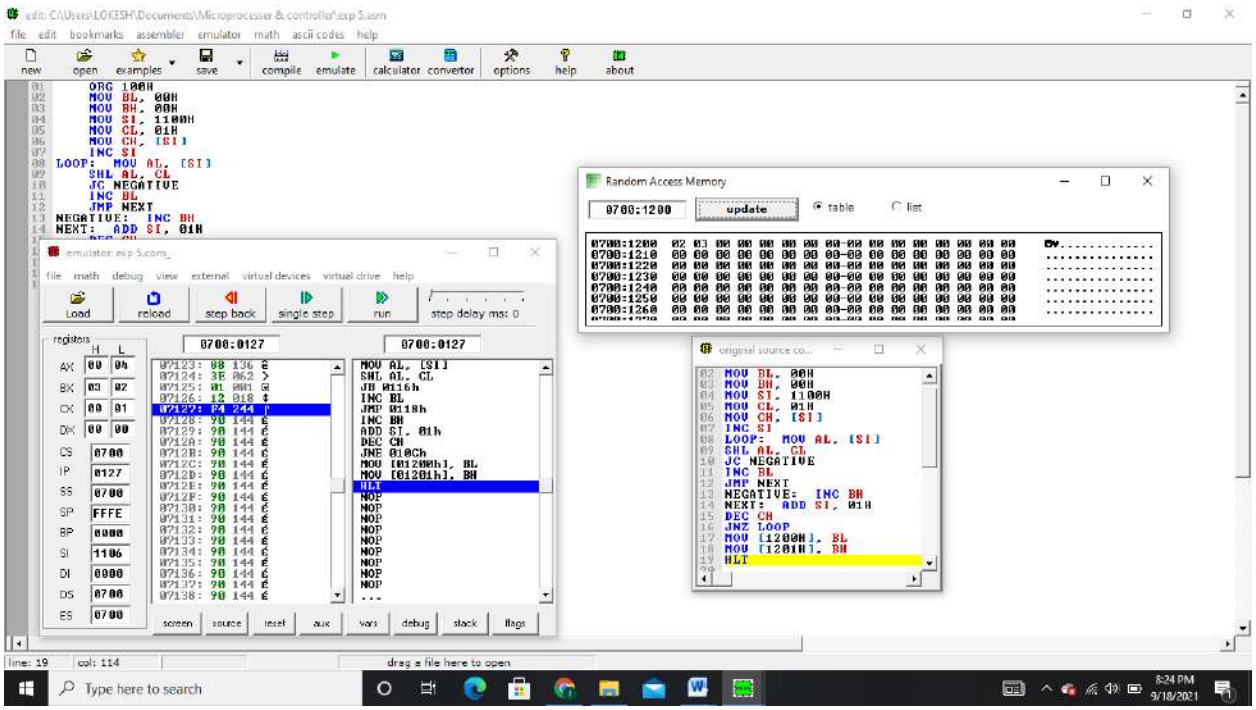
1) Discuss the difference between SHR and SAR?
=> SHR and SAR shift the bits of the operand downward the low order bit is shifted to the carry flag. The need is to divide the operand by 2.
=> SAR performs on signal divide with rounding toward negative infinity the highest order bit remains the same.
=> SHR performs on signed divide with rounding toward negative infinity the highest order bit remains the same.
=> SHR performs on unsigned divide the high order bit is set to 0.

2) Explain RCR function with a protocol?
=> RCR is used to rotate bits of byte/word towards right, LSB to CF and from CF to MSB

Eg
RCR [CX];
RCR AL, CL;
RCR CL, 04H;

Emulator Output:



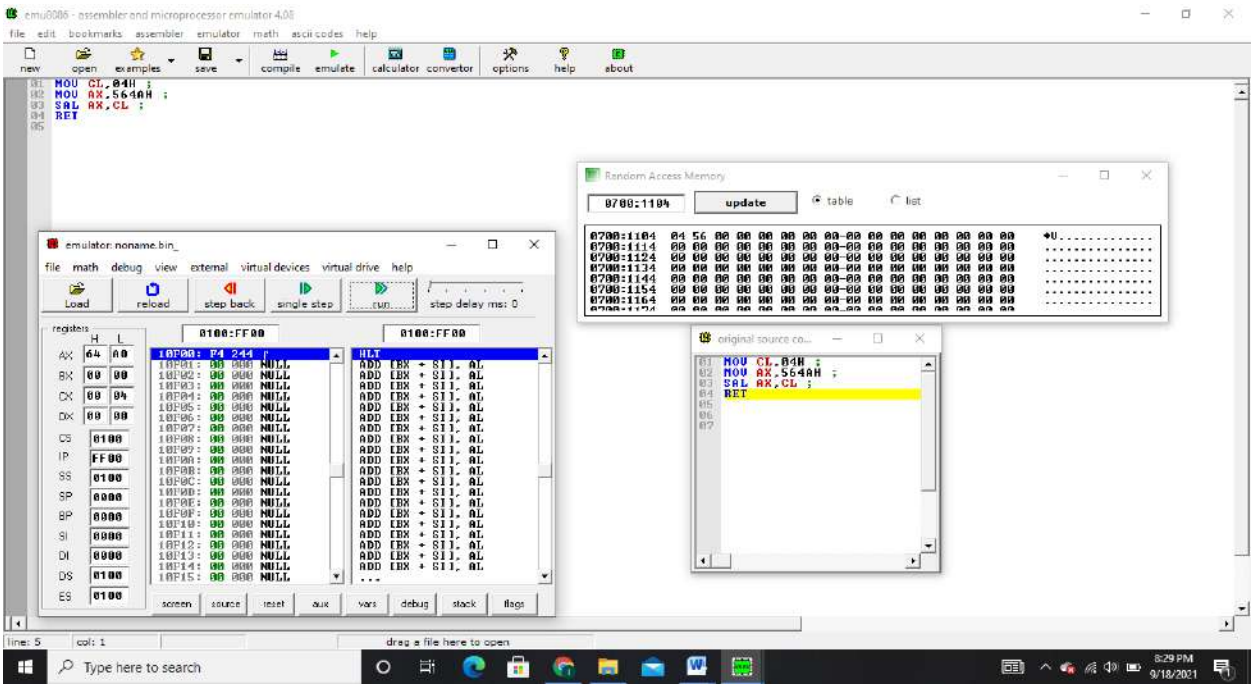


POST - LAB SOLUTIONS:

1. Find out the answer for a given program?

```
MOV CL,04H ;
MOV AX,564AH ;
SAL AX,CL ;
RET
```

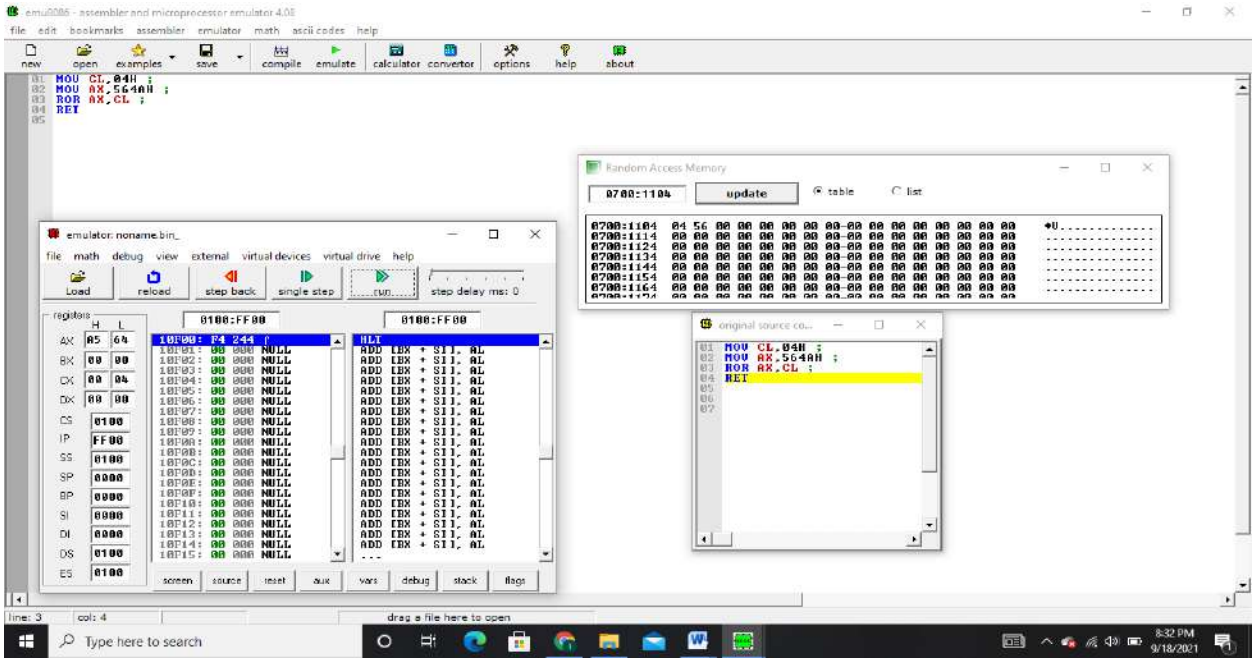
Sol. 1). The solution for the given program is as shown below:



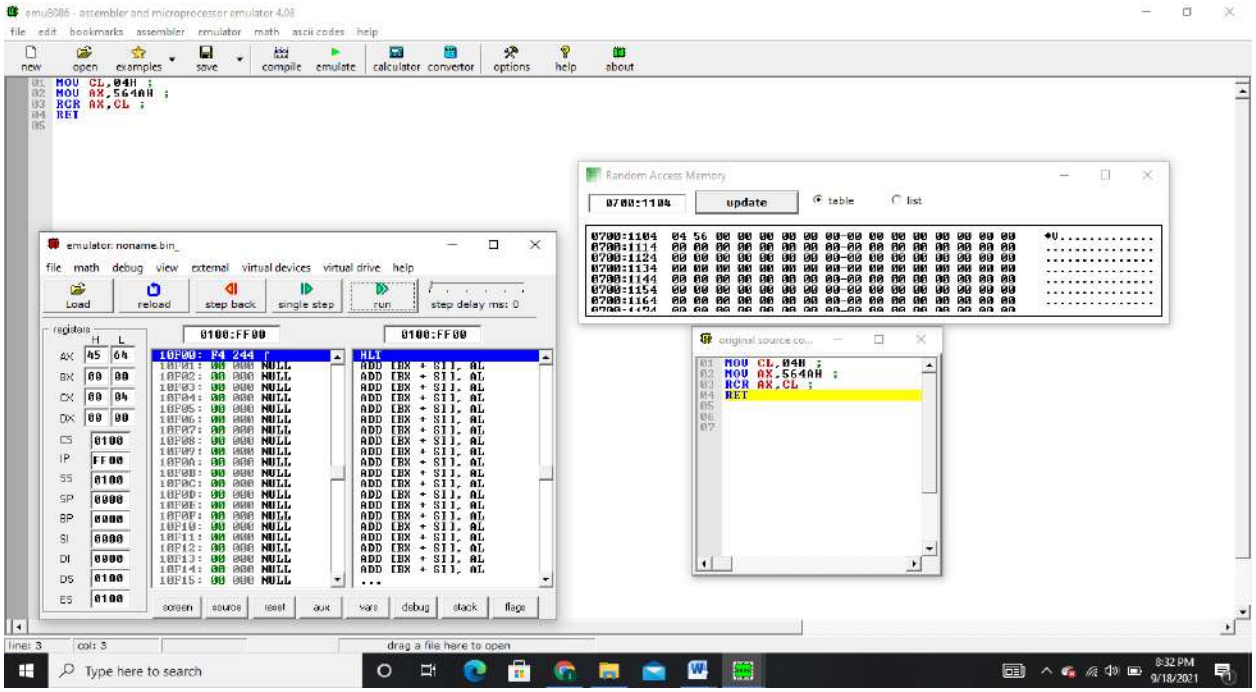
2.Find out the answer for the above program with ROR and RCR instead of SAL?

Sol. 2). The desired programs is shown as below:

à <u>ROR:</u>


emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
MOV CL,04H ;
MOV AX,564AH ;
ROR AX,CL ;
RET
Random Access Memory
0700:1104
update
table
list
emulator: noname.bin_
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms: 0
registers
0100:FF00
original source co...
MOV CL,04H ;
MOV AX,564AH ;
ROR AX,CL ;
RET
screen source reset aux vars debug stack flags
line: 3 col: 4
drag a file here to open
Type here to search
8:32 PM
9/18/2021

RCR:


emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
MOV CL,04H ;
MOV AX,564AH ;
RCR AX,CL ;
RET
Random Access Memory
0700:1104
update
table
list
emulator: noname.bin_
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms: 0
registers
0100:FF00
original source co...
MOV CL,04H ;
MOV AX,564AH ;
RCR AX,CL ;
RET
screen source reset aux vars debug stack flags
line: 3 col: 3
drag a file here to open
Type here to search
8:32 PM
9/18/2021

5.9 Results and Conclusion:

Thus, the instruction sets, addressing modes and to perform shift and rotate operation by given number of bits in the memory was experimented.

Laboratory Report Cover Sheet

SRM Institute of Science and Technology
Faculty of Engineering and Technology
Department of Electronics and Communication Engineering

18ECC203J Microprocessor, Microcontroller & Interfacing Techniques Lab

Fifth Semester, 2020-21 (odd semester)

Nam : Pushpal Das

Register No. : RA1911004010565

Day / Session : 1ST / 3&4

Venue : Online (G Meet)

Title of Experiment : Interfacing Of Stepper Motor Using 8086

Date of Conduction : 25 AUG 21

Date of Submission : 29 AUG 21

Particulars	Max. Marks	Marks Obtained
Pre-lab	5	
Lab Performance	10	
Post-lab	15	
Total	**30**	

REPORT VERIFICATION

Date : 29 SEP 21

Staff Name : Mr.R.Prithiviraj

Signature :

Experiment 6.
Interfacing of Stepper Motor with 8086

6.1 Aim(s) / Objective(s) / Purpose:

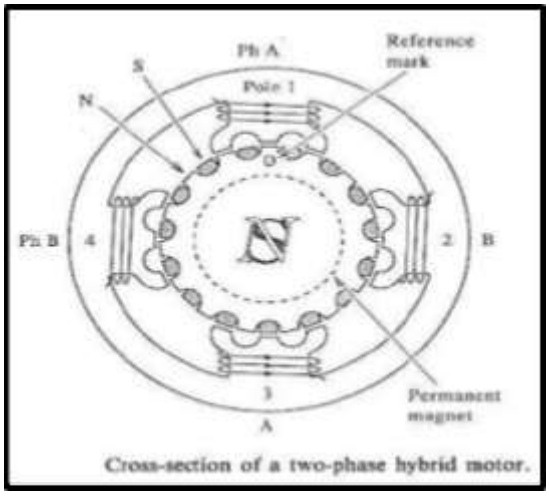
The purpose of this experiment to trigger a stepper motor with +5V, at an angle of 15 degree in each step in both clockwise and anticlockwise

6.2 Introduction / Background:

Stepper motor is a device used to obtain an accurate position control of rotating shafts. A stepper motor employs rotation of its shaft in terms of steps, rather than continuous rotation as in case of AC or DC motor. To rotate the shaft of the stepper motor, a sequence of pulses is needed to be applied to the windings of the stepper motor, in proper sequence. The numbers of pulses required for complete rotation of the shaft of the stepper motor are equal to the number of internal teeth on its rotor. The stator teeth and the rotor teeth lock with each other to fix a position of the shaft. With a pulse applied to the winding input, the rotor rotates by one teeth position or an angle x. The angle x may be calculated as.

x = 3600 / no. of rotor teeth

After the rotation of the shaft through angle x, the rotor locks itself with the next tooth in the sequence on the internal surface of the stator. The typical schematic of a typical stepper motor with four windings is as shown below.



Cross-section of a two-phase hybrid motor.

The stepper motors have been designed to work with digital circuits. Binary level pulses of 0-5V are required at its winding inputs to obtain the rotation of the shafts. The sequence of the pulses can be decided, depending upon the required motion of the shaft. By suitable sequence of the pulses the motor can be used in three modes of operation.

- One phase ON (medium torque)
- Two phase ON (high torque)
- Half stepping (low torque)

MOTION	STEPS	A	B	C	D	HEX VALUE
CLOCKWISE	1	0	0	1	1	03H
	2	0	1	1	0	06H
	3	1	1	0	0	0CH
	4	1	0	0	1	09H
ANTI CLOCKWISE	1	0	0	1	1	03H
	2	1	0	0	1	09H
	3	1	1	0	0	0CH
	4	0	1	1	0	06H

6.3.Materials / Equipment:

1. *K. M. Bhurchandi and A. K. Ray, "Advanced Microprocessors and Peripherals-with ARM and an Introduction to Microcontrollers and Interfacing ", Tata McGraw Hill, 3rd edition 2015*

2. *Doughlas.V.Hall, "Microprocessor and Interfacing : Programming and Hardware", 3rd edition, McGraw Hill, 2015*

6.4 Software Requirement:

Emulator 8086.

6.5 Procedure:

a. Load the program in the emulator.
b. Load the input values to the corresponding address location or directly into the registers.
c. Save, Compile and Emulate the program.
d. Execute the program using Single run or Run until termination process.
e. Store the results of the process in the destination register
f. Terminate the program

6.6 Program Logic:

There are three different stepping schemes for a stepper motor.

1. Wave scheme
2. 2- phase scheme
3. half stepping and mixed scheme.

6.6.1 Wave scheme:

The stepper motor windings A1, A2, B1, B2 can be cyclically excited with a DC current to run the motor in the clockwise direction. Consider the four rotor positions of the motor along with the stator excitations. The switching scheme for the wave mode excitation is given as follows.

Clockwise					Anti-Clockwise				
Step	A1	A2	B1	B2	Step	A1	A2	B1	B2
1	1	0	0	0	1	1	0	0	0
2	0	0	0	1	2	0	0	1	0
3	0	1	0	0	3	0	1	0	0
4	0	0	1	0	4	0	0	0	1

6.6.2 2- Phase scheme:

In this scheme the two adjacent stator windings are energized. There are two magnetic fields achieved in quadrature and none of the rotor pole faces can be in a direct alignment with the stator poles.The switching scheme for the 2- phase mode excitation is given as follows.

Clockwise					Anti -Clockwise				
Step	A1	A2	B1	B2	Step	A1	A2	B1	B2
1	1	0	0	1	1	1	0	1	0
2	0	1	0	1	2	0	1	1	0
3	0	1	1	0	3	0	1	0	1
4	1	0	1	0	4	1	0	0	1

6.6.3 Half stepping scheme:

The previously discussed two schemes have a step size of 30 degrees for the stepper motor under consideration. However, there is an offset of 15 degrees between these two schemes. By interleaving these two schemes, the step size can be reduced to 15 degrees there by improving the accuracy of the motor. This is called half stepping scheme.

The switching sequence is as follows:

1. A1 on
2. A1 and B1 on
3. B1 on
4. B1 and A2 on
5. A2 on

6. A2 and B2 on
7. B2 on 8. B2 and A1 on
9. A1 on etc.

Program 6:

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
		#start=stepper_motor.exe#		
7100	**START**	**MOV SI, 1100H**	BE, 00, 11	Contents of 1100H memory location are moved to SI Register
	NEXT_STEP			
7103	**WAIT**	**IN AL,07H**	E4, 07	Get Input data from device to accumulator
7105		**TEST AL,10000000b**	A8, 80	Test if the port is ready.
7107		**JMP WAIT**	74, FA	If the result of the TEST is zero then it jumps to wait.
7109		**MOV AL, [BX][SI]**	8A, 04	The accumulator gets updated.
710B		**OUT 7, AL**	E6, 07	The output is sent to port 7.
710D		**INC SI**	46	Incrementation of SI register.
710E		**CMP SI, 1104H**	81, FE, 04, 11	Comparison of values in SI register and the number of data values
7112		**JC NEXT_STEP**	72, EF	If carry is '0' jump to next step
7115		**MOV SI, 1100H**	BE, 00, 11	The counter gets updated.
7116		**JMP NEXT_STEP**	EB, EA	The program jumps to the next step

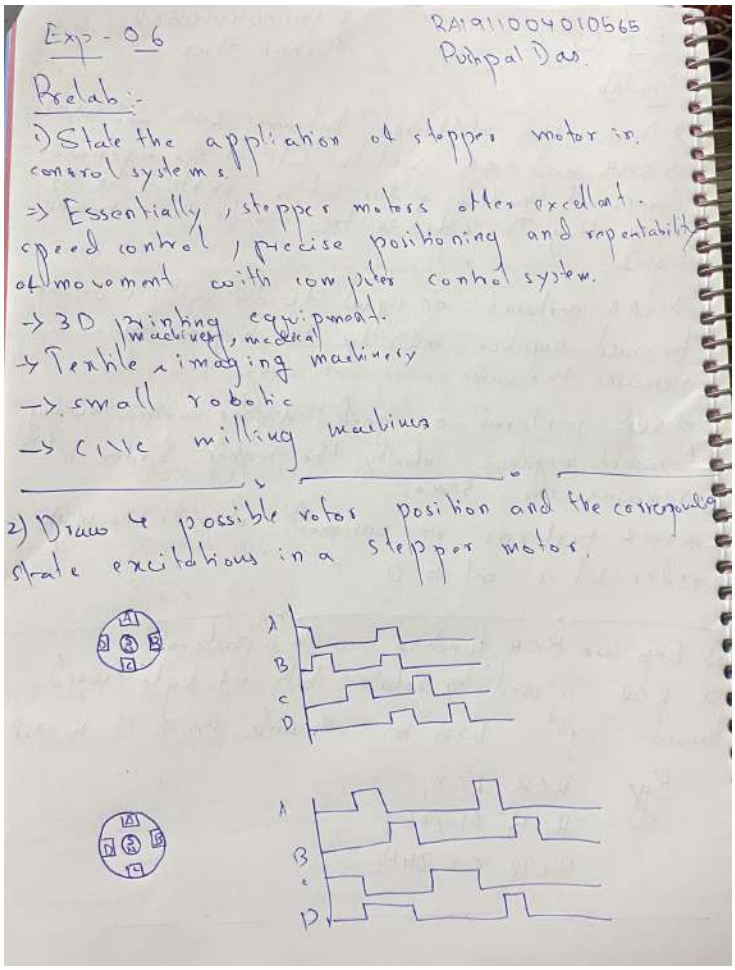
Observation:

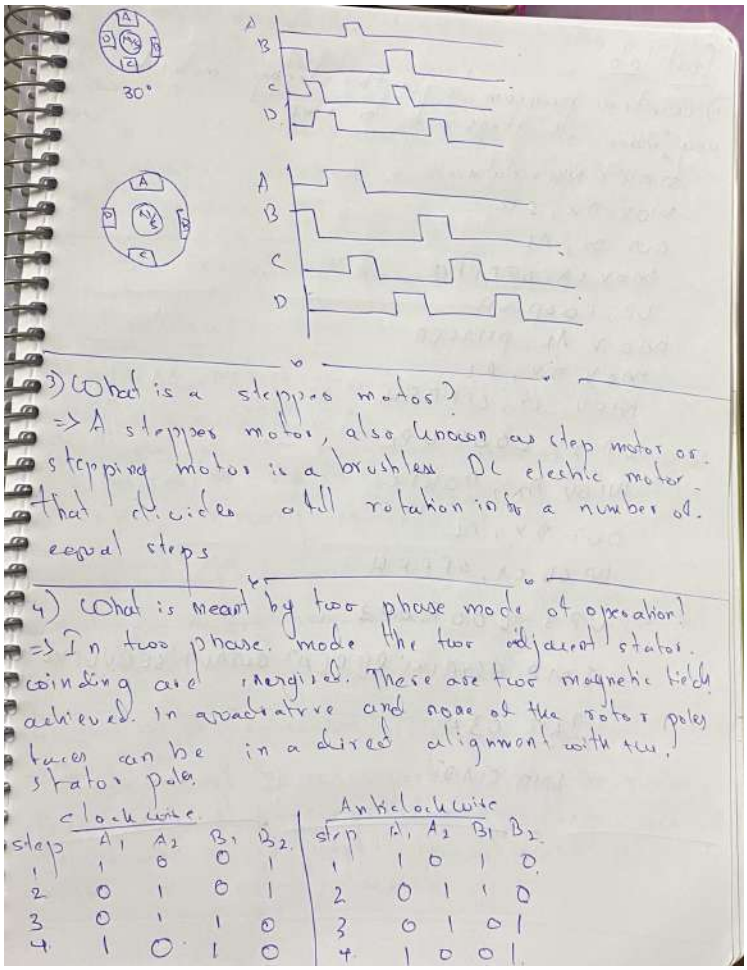
IN PUT ADDRESS	CLOCKWISE	ANTI-CLOCKWISE
1100	03	09
1101	06	0C
1102	0C	06
1103	09	03

6.7 Pre-Lab Questions

1. State applications of stepper motor in control systems
2. Draw the 4 possible rotor positions and the corresponding stator excitations in a stepper motor
3. What is stepper motor?
4. What is meant by two phase mode of operation?

PRE - LAB ANS:-





6.8 Post Lab Questions

1. Write a program to run the stepper motor for any number of steps and to stop it.
2. What is meant by step angle? How it is calculated.
3. State the advantages and disadvantages of stepper motor
4. What is synchronism in stepper motor?

POST LAB ANS:-

Post-lab

1) Write a program to run the stepper motor for any numbers of steps and to stop it.

```
START: Mov data.
MOV Dx, CTC.
OUT DX, AL
MOV CX, 0FFFFH.
UP: LOOP UP
MOV AL, PHASED
MOV DX, AL.
MOV CX, 0FFFFH.
UP 2: LOOP UP2.
MOV Dx, PORTC.
OUT DX, AL.
MOV CX, 0FFFH
UP3: LOOP OP3
JMP AGAIN: REPEAT OUTPUT SEQUENCE
INT 03H
END START
```

2) What is meant by step angle? How is calculated

=> Step angle of the stepper motor is defined as the angle traversed by the motor in one step

=> To calculate step angle, simply divide 360 by number of steps a motor takes to complete one revolution

3) State the advantages and disadvantages of stepper motor.

Advantages

=> Stepper motors provide ruggedness, high reliability, simplicity construction.

=> Allow for low maintenance effort excellent response to starting/stopping/reversing and will work in many environments

Disadvantages

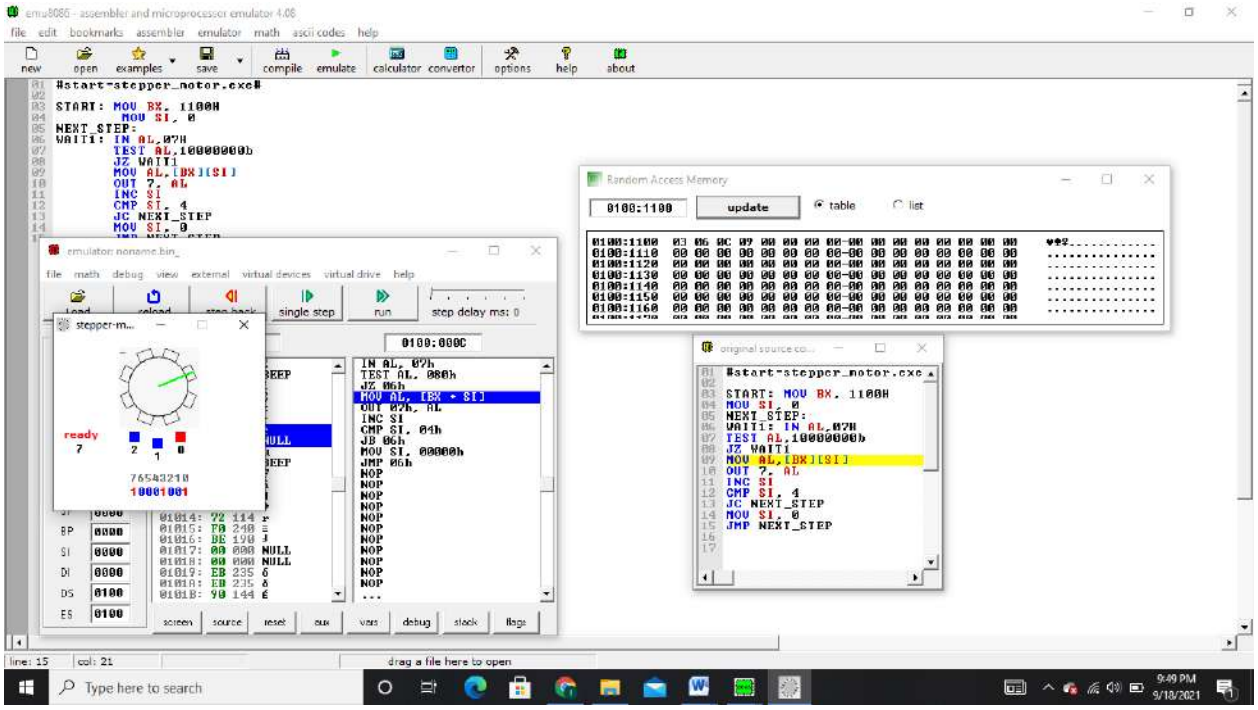
=> Stepper motors are in that the motor is provide a low frequency

4) What is synchronism in stepper motor?

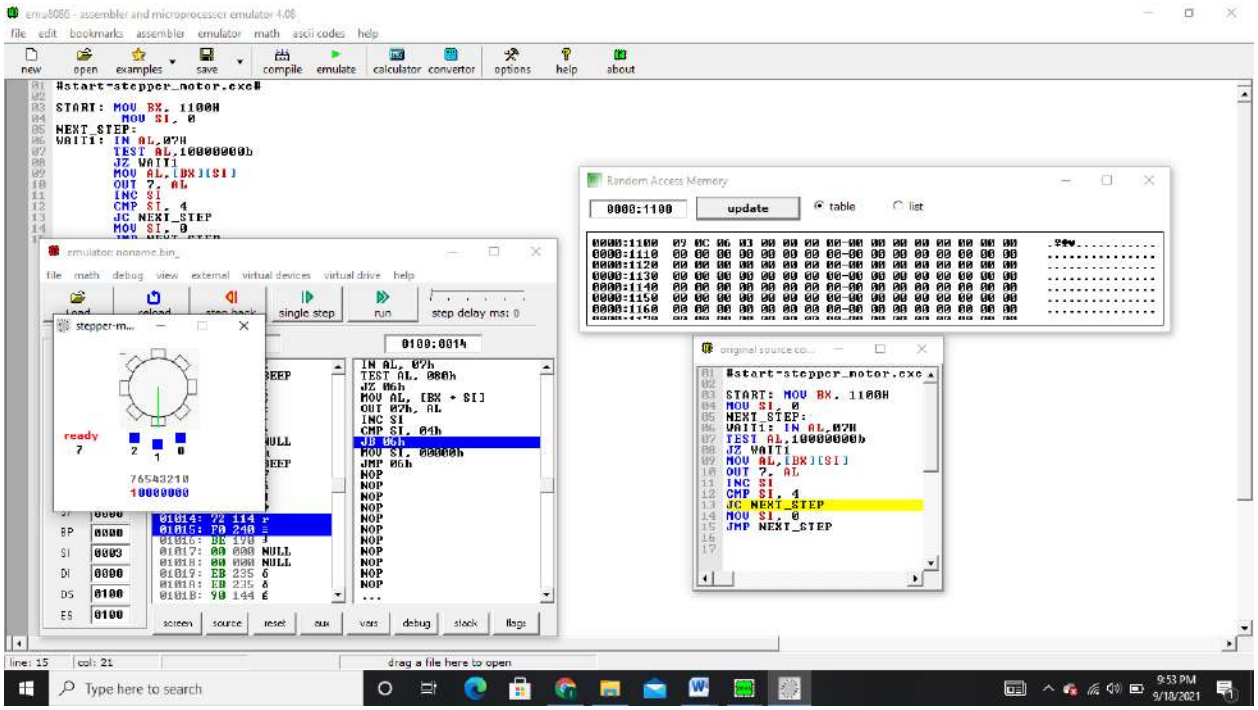
=> It is one of the correspondence btw the number of the pulse applied to the stepper motor and the number of steps through which the motor has actually moved.

Emulator Output:-

Clockwise Direction



Anti - Clockwise Direction:


emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
#start=stepper_motor.exe#
START: MOV BX, 1100H
MOV SI, 0
NEXT_STEP:
WAIT1: IN AL,07H
TEST AL,10000000b
JZ WAIT1
MOV AL,[BX][SI]
OUT 7, AL
INC SI
CMP SI, 4
JC NEXT_STEP
MOV SI, 0
Random Access Memory
0000:1100
update
table
list
emulator: noname.bin
file math debug view external virtual devices virtual drive help
load reload step back single step run step delay ms: 0
stepper-m...
ready
7 2 1 0
76543210
10000000
0100:0014
IN AL, 07h
TEST AL, 080h
JZ 06h
MOV AL, [BX + SI]
OUT 07h, AL
INC SI
CMP SI, 04h
JB 06h
MOV SI, 00000h
JMP 06h
NOP
AX 0000
BX 1100
CX 0004
DX 0000
SP 0000
BP 0000
SI 0003
DI 0000
DS 0100
ES 0100
screen source reset run vars debug stack flags
original source co...
#start=stepper_motor.exe
START: MOV BX, 1100H
MOV SI, 0
NEXT_STEP:
WAIT1: IN AL,07H
TEST AL,10000000b
JZ WAIT1
MOV AL,[BX][SI]
OUT 7, AL
INC SI
CMP SI, 4
JC NEXT_STEP
MOV SI, 0
JMP NEXT_STEP
line: 15 col: 21
drag a file here to open
Type here to search
9:53 PM
9/18/2021

Results and Conclusion:-

Thus, the stepper motor is interfaced with 8086 and the output is executed in port 07.

Laboratory Report Cover Sheet

SRM Institute of Science and Technology
Faculty of Engineering and Technology
Department of Electronics and Communication Engineering

18ECC203J Microprocessor, Microcontroller & Interfacing Techniques Lab

Fifth Semester, 2021-22 (odd semester)

Name : **Pushpal Das**

Register No. : **RA1911004010565**

Day / Session : **1ST / FN**

Venue : **ONLINE(G Meet)**

Title of Experiment : **7.INTERFACING OF LED DISPLAY WITH 8086**

Date of Conduction : **02 SEP 2021**

Date of Submission : **29 SEP 2021**

Particulars	Max. Marks	Marks Obtained
Pre-lab	5	
Lab Performance	10	
Post-lab	15	
Total	**30**	

REPORT VERIFICATION

Date : **29 SEP 2021**

Staff Name : Mr.R.Prithiviraj

Signature :

Experiment 7
Interfacing of LED Display with 8086

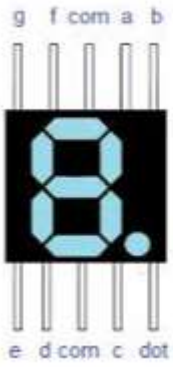
7.1 Aim(s) / Objective(s) / Purpose.

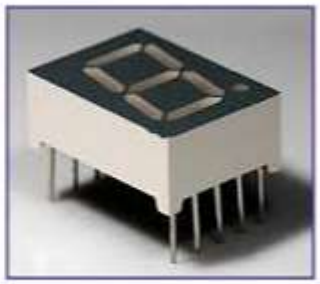
The purpose of this experiment to display a number in LED display with a small delay.

7.2 Introduction / Background

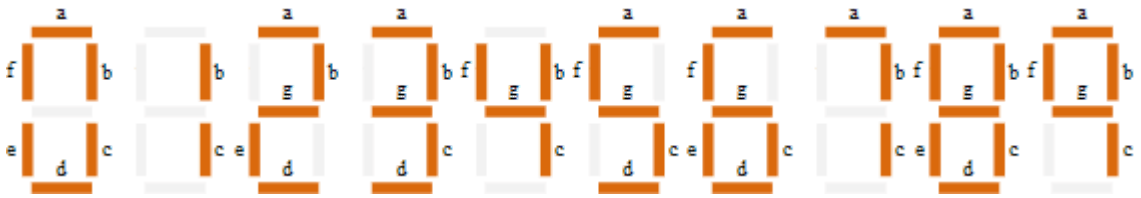
7-segment Display

The *7-segment display*, also written as "seven segment display", consists of seven LEDs (hence its name) arranged in a rectangular fashion as shown. Each of the seven LEDs is called a segment because when illuminated the segment forms part of a numerical digit (both Decimal and Hex) to be displayed.

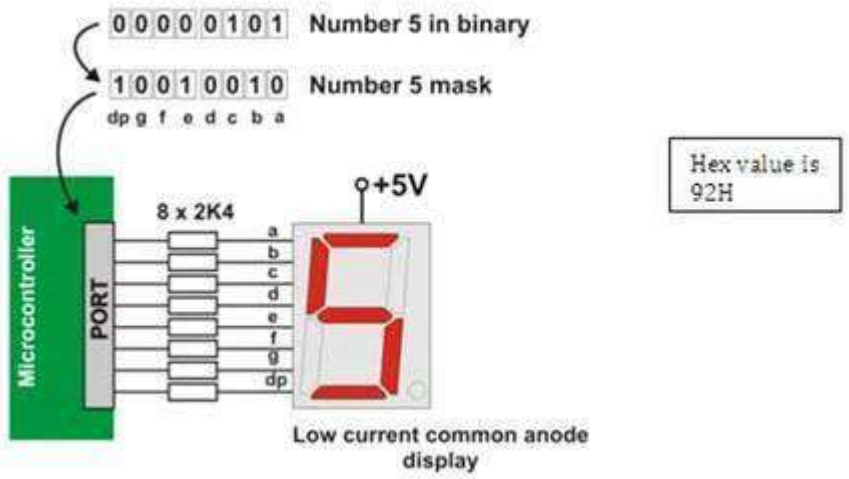




7-Segment Display Segments for all Numbers.



7-SEGMENT DISPLAY TYPES AND DISPLAY FORMAT

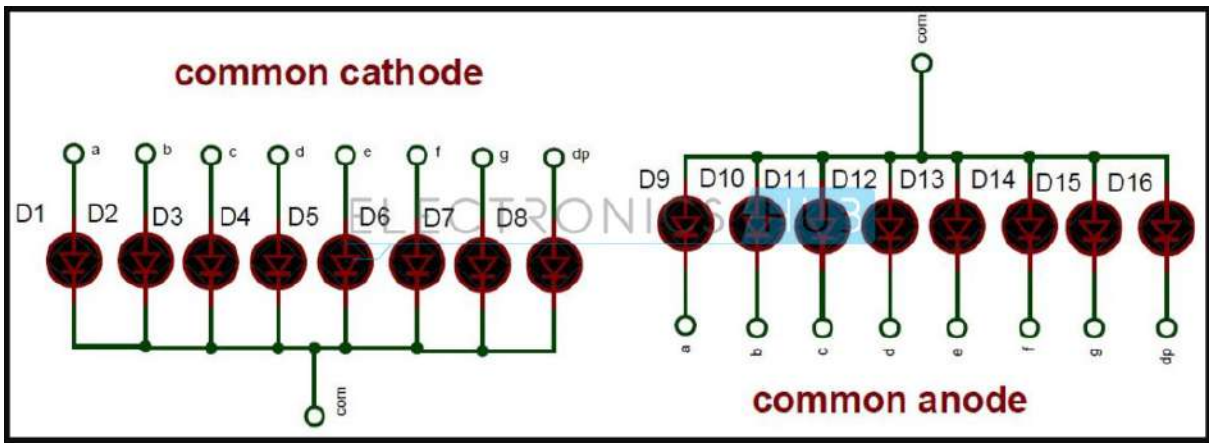


The hex decimal data corresponding to the segments which have to glow for displaying a character is output to port B

DIGIT	a	b	c	d	e	f	g	HEX Value
0	0	0	0	0	0	0	1	0x40
1	1	0	0	1	1	1	1	0xF9
2	0	0	1	0	0	1	0	0x24
3	0	0	0	0	1	1	0	0x30
4	1	0	0	1	1	0	0	0x19
5	0	1	0	0	1	0	0	0x12
6	0	1	0	0	0	0	0	0x02
7	0	0	0	1	1	1	1	0xF8
8	0	0	0	0	0	0	0	0x00
9	0	0	0	1	1	0	0	0x10

Circuit Principle

Seven segment displays internally consist of 8 LEDs. In these LEDs, 7 LEDs are used to indicate the digits 0 to 9 and single LED is used for indicating decimal point. Generally, seven segments are two types, one is common cathode and the other is common anode.
In common cathode, all the cathodes of LEDs are tied together and labelled as com. and the anode are left alone. In common anode, seven segments display all the anodes are tied together and cathodes are left freely. Below figure shows the internal connections of seven segments Display.



7.3 Materials / Equipment

1. *K. M. Bhurchandi and A. K. Ray, "Advanced Microprocessors and Peripherals-with ARM and an Introduction to Microcontrollers and Interfacing ", Tata McGraw Hill, 3rd edition 2015*

2. *Doughlas.V.Hall, "Microprocessor and Interfacing : Programming and Hardware", 3rd edition, McGraw Hill, 2015*

7.4 Software Requirement:

Emulator 8086.

7.5 Procedure

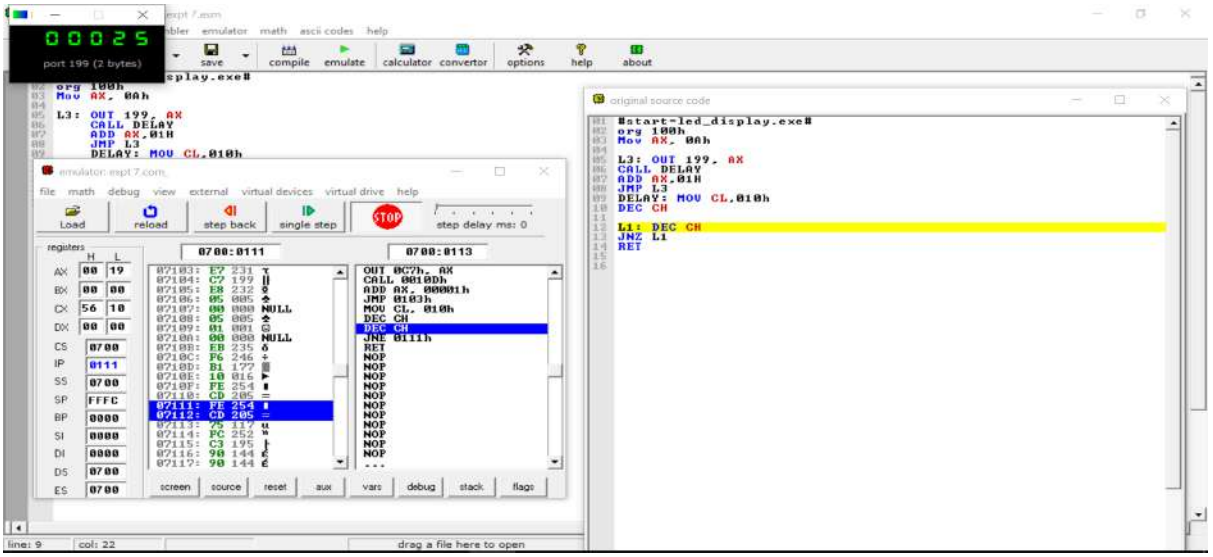
a. Load the program in the emulator.
b. Load the input values to the corresponding address location or directly into the registers.
c. Save, Compile and Emulate the program.
d. Execute the program using Single run or Run until termination process.
e. Store the results of the process in the destination register
f. Terminate the program

Program 7

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
		#start=led_display.exe#		
		Org 100h		
7100		**Mov AX, 00h**	**B8 00 00**	**Starting number is loaded as 00**
7103	**L3:**	**OUT 199, AX**	**E7 C7**	**199 is the LED display port address**
7105		**CALL DELAY**	**E8 05 00**	**00 will be displayed in the LED for a delay**

7108		ADD AX,01H	05 01 00	Input value is incremented by 1
710B		JMP L3	EB F6	Jump to label 3
701D	DELAY:	MOV CL,05H	B1 05	Contents in the memory location 05H are moved to CL register
710F		MOV CH,05H	B5 05	Contents in the memory location 05H are moved to CH register
7111	L1:	DEC CH	FE CD	The value in CH register is decremented by 1
7113		JNZ L1	75 FC	If carry is 0 then jump to Loop 1
7115	L2:	DEC CL	FE C9	The value of CL register is decremented by 1
7117		JNZ L2	75 FC	If carry is 0 then jump to Loop 2
7119		RET	C3	Return statement

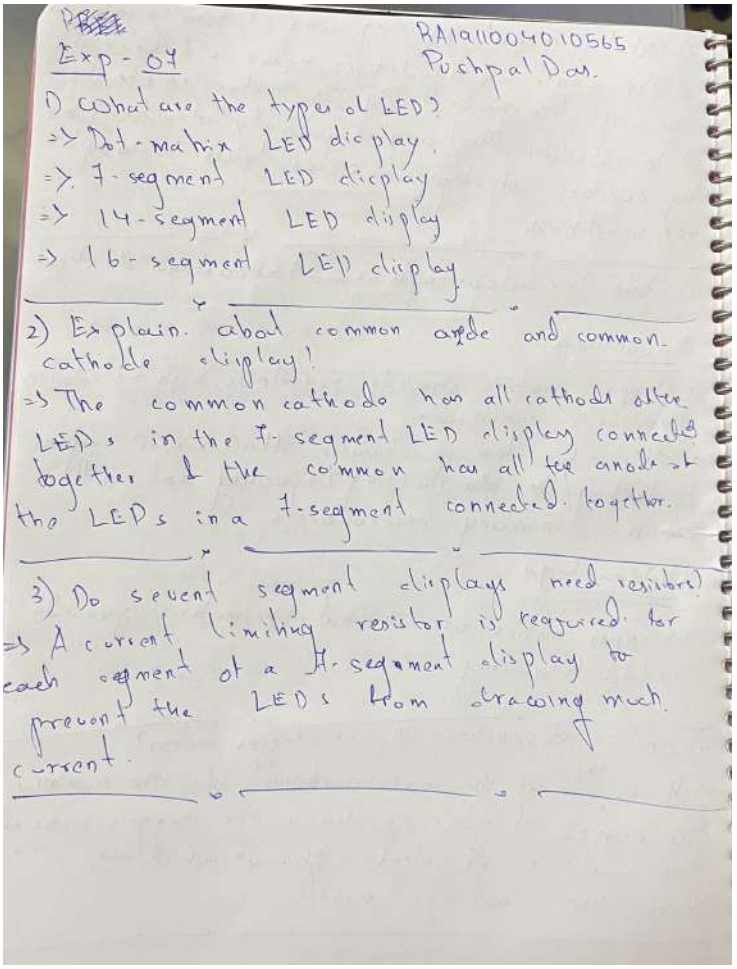
EMULATOR OUTPUT :-



7.7 Pre-Lab Questions

1. What are the types of LED?
2. Explain about common anode and common cathode display?
3. Do seven segment displays need resistors?

PRE LAB:-



Exp - 04

RA1911004010565
Pushpal Das.

1) What are the types of LED?
=> Dot-matrix LED display.
=> 7-segment LED display
=> 14-segment LED display
=> 16-segment LED display

2) Explain about common anode and common cathode display?
=> The common cathode has all cathods of the LEDs in the 7-segment LED display connected together & the common has all the anode of the LEDs in a 7-segment connected together.

3) Do seven segment displays need resistors?
=> A current limiting resistor is required for each segment of a 7-segment display to prevent the LEDs from drawing much current.

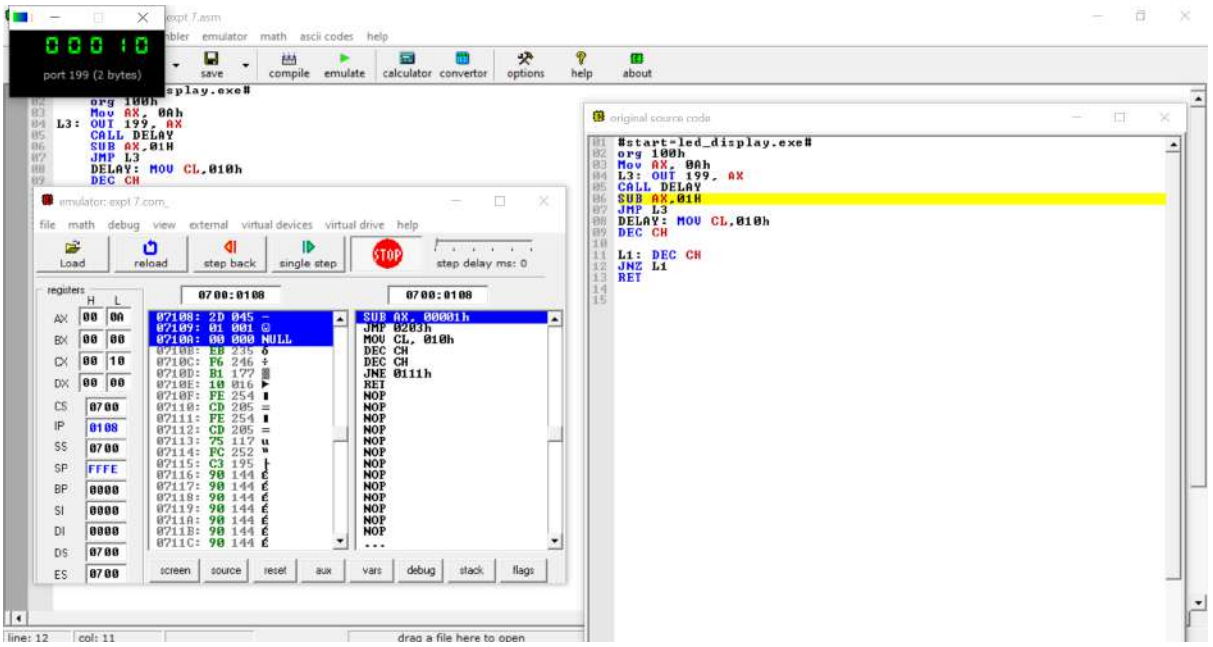
7.8 Post Lab Questions

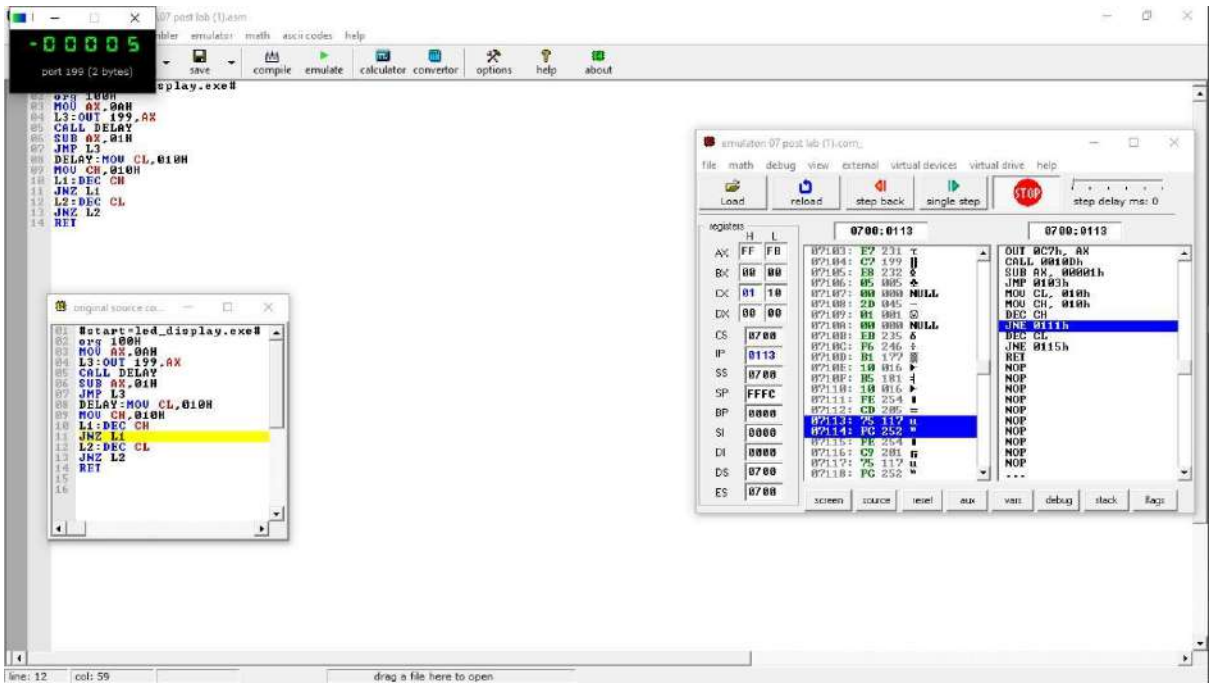
1. Write a program to display a number from 01 to 10 and decrements from 10 to 1 using emulator 8086.

2. Write a program to display a word with eight characters “JANA GANA” using emulator 8086.

POST LAB:-

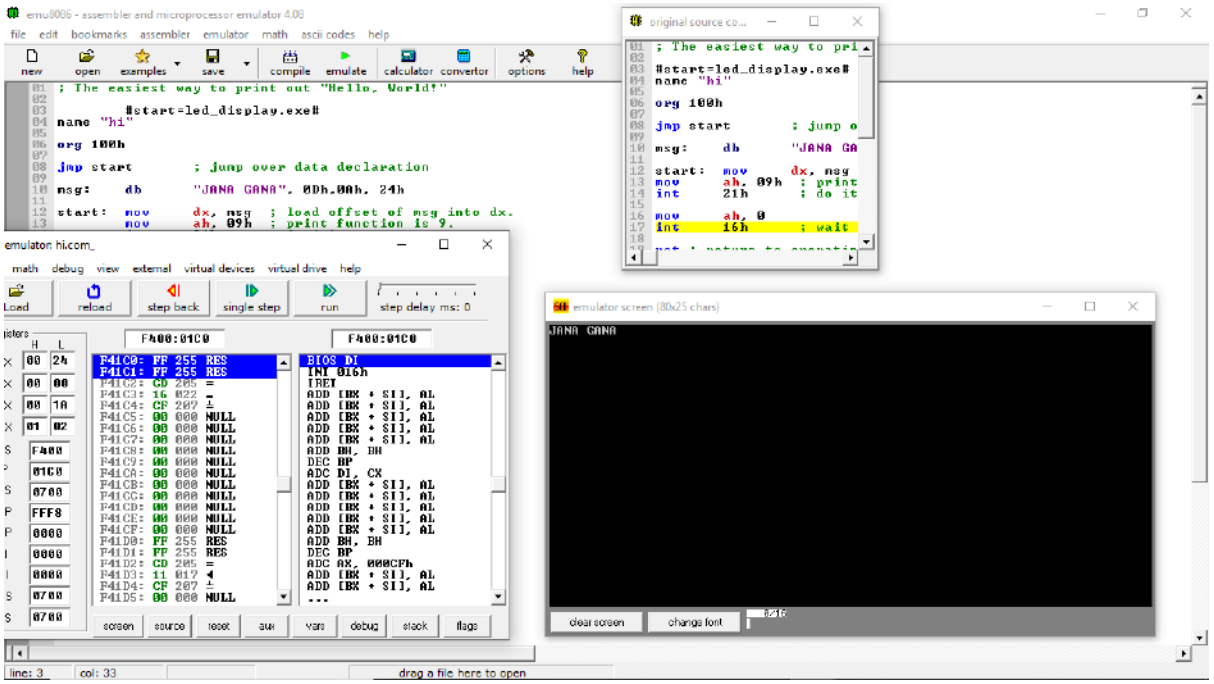
1. Write a program to display a number from 01 to 10 and decrements from 10 to 01 using emulator 8086





2. Write a program to display a word with eight characters “JANA GANA” using emulator 8086.

Using Emulator 8086:-



7.9 Results and Conclusion:-

Thus, the LED display is interfaced with 8086 and the output is executed in port address 199.

Laboratory Report Cover Sheet

SRM Institute of Science and Technology
Faculty of Engineering and Technology
Department of Electronics and Communication Engineering

18ECC203J Microprocessor, Microcontroller & Interfacing Techniques Lab

Fifth Semester, 2021-22 (Odd Semester)

Name : **Pushpal Das**

Register No. : **RA1911004010565**

Day / Session : **5 / 3&4**

Venue : **Online (Emu 8086)**

Title of Experiment : **Addition, Subtraction, Multiplication And Division Using 8051**

Date of Conduction : **08/09/21**

Date of Submission : **23/10/21**

Particulars	Max. Marks	Marks Obtained
Pre-lab	5	
Lab Performance	10	
Post-lab	15	
Total	**30**	

REPORT VERIFICATION

Date : **11/09/21**

Staff Name : Mr.R.Prithiviraj

Signature :

Experiment 8
Addition, Subtraction, Multiplication and Division using 8051

8.1. Aim(s) / Objective(s) / Purpose.

The purpose of this experiment is to add, subtract, multiply and divide the given two 8 bit numbers and store them in a memory location using 8051. The student should also be able to design the addition and subtraction with carry and borrow.

8.2 Introduction / Background

The purpose of this experiment is to learn about the registers, instruction sets, data transfer operation and logical operation of 8051 by using ADD, SUBB, MUL & DIV in the given two 8 bit numbers and store them in a memory location.

Tips: Any information copied directly or verbatim from Lab manuals or other references should be stated within quotes and referred, otherwise, it is considered plagiarism.

8.3 Materials / Equipment

1. *Muhammad Ali Mazidi and Janice GillispieMazidi, "The 8051 - Microcontroller and Embedded systems", 7th Edition, Pearson Education, 2011.*

2. *Subrataghoshal " 8051 Microcontroller Internals Instructions ,Programming And Interfacing",2nd edition Pearson 2010*

8.4 Hardware Requirement:

The 8051 Microprocessor kit, Power Supply.

Software Requirement :

8051 EdSim

8.5 Procedure

i) Enter the opcodes from memory location 4200
ii) Execute the program
iii) Check for the result at 4100 and 4101

Using the accumulator, subtraction is performed and the result is stored. Immediate addressing is employed. The SUBB instruction drives the result in the accumulator.

8.6 Program Logic:

To perform addition in 8051 one of the data should be in accumulator, another data can be in any of the general purpose register or in memory or immediate data. After addition the sum will be in accumulator. The sum of two 8-bit data can be either 8-bits(sum only) or 9-bits(sum and carry). The accumulator can accumulate only the sum and there is a carry the 8051 will indicate by setting carry flag. Hence one of the register is used to account for carry.

The 8051 has MUL instruction unlike many other 8-bit processors. MUL instruction multiplies the unsigned 8-bit integers in A and B. The lower order byte of the product is left in A and the higher order byte in B.

The 8051 has DIV instruction unlike many other 8-bit processors. DIV instruction divides the unsigned 8-bit integers in A and B. The accumulator receives the integer part of the quotient and the register B receives the remainder.

Program 8(a) : ADDITION

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
0000		Mov A, #04		Move the data (8 bit) to Acc.
0002		Mov B, #06		Move the data (8 bit) to B Register.
0005		ADD A, B		Add them, and store the result in A.
0007		Mov R0, A		Copy the result to a Register.

Observation

INPUT ADDRESS	DATA
0000	04
0002	06
OUTPUT ADDRESS	**DATA**
00	0A

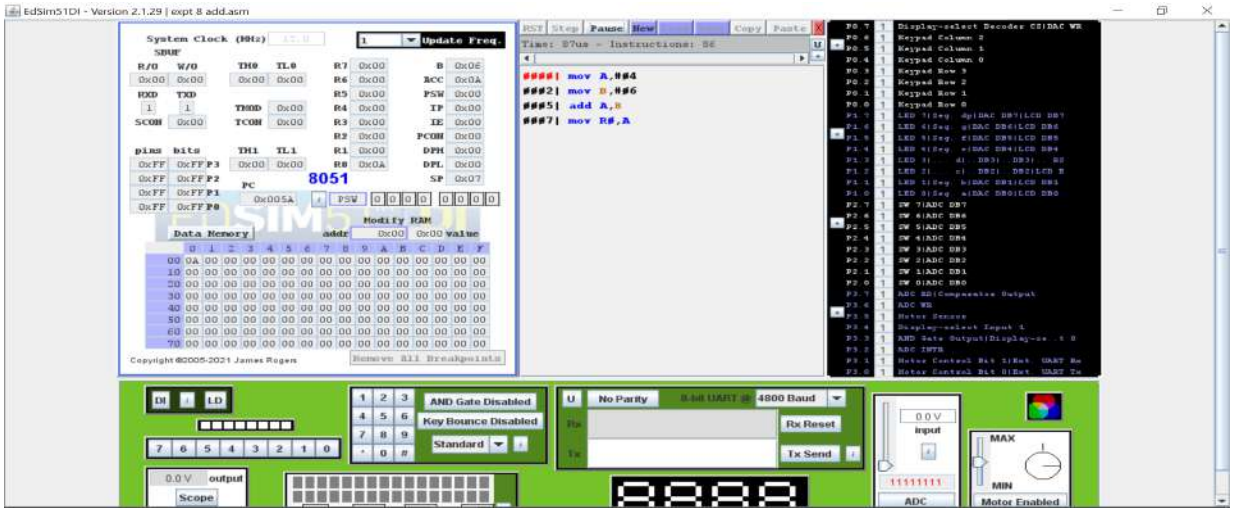
CALCULATIONS:

04 – 0000 0100

06 - 0000 0110

+

Result 0A -> 0000 1010



Program 8(b): SUBTRACTION

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
0000		Mov A, #06		Move a data (8 bit) to Acc
0002		Mov B, #04		Move a data (8 bit) to B Reg
0005		SUBB A, B;		Subtract them, and store result in A
0007		Mov R0, A;		Copy the result to a Reg

Observation

INPUT ADDRESS	**DATA**
0000	06
0002	04
OUTPUT ADDRESS	**DATA**
01	02

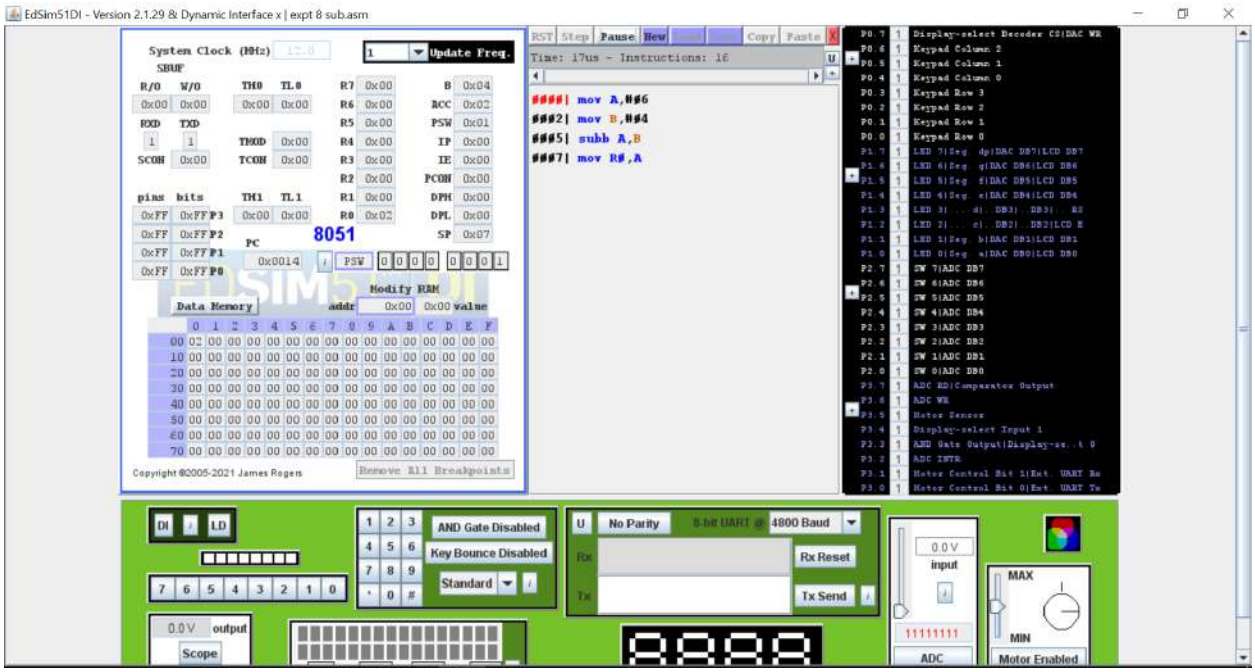
CALCULATIONS:

06 - 0000 0110

04 - 0000 0100

-

Result - 0000 0010 - 02



Program 8(c) : MULTIPLICATION

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
0000		Mov A, #04		Move a data (8 bit) to Acc
0002		Mov B, #02		Move a data (8 bit) to B Reg
0005		MUL AB;		Multiply them, and store result in A (and B- in case of 16 bit result)
0006		Mov R0, A;		Copy Lower half in A Reg
0007		Mov R1, B;		Copy Higher half in B Reg

Observation

OUTPUT ADDRESS	**DATA**
00	08
INPUT ADDRESS	**DATA**
0000	04
0002	02

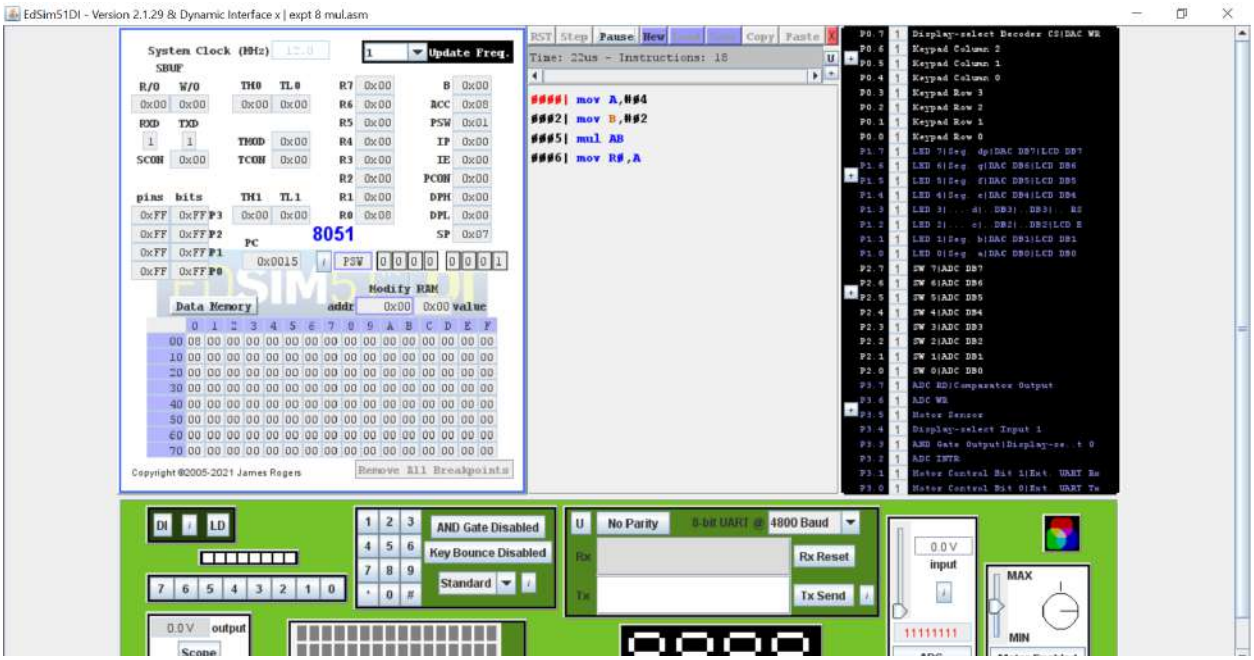
CALCULATIONS:

04 - 0000 0100

02 - 0000 0010

*

Result - 0000 1000 - 08



Program 8(d) : DIVISION

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
0000		Mov A, #05		Move a data (8 bit) to Acc
0002		Mov B, #02		Move a data (8 bit) to B Reg
0005		DIV AB;		Divide them, and store result in A (and B- in case of 16 bit result)
0006		Mov R0, A;		Copy Lower half in A Reg
0007		Mov R1,B;		Copy Higher half in B Reg

Observation

OUTPUT ADDRESS	DATA
00	02
01	01
INPUT ADDRESS	**DATA**
0000	05
0002	02

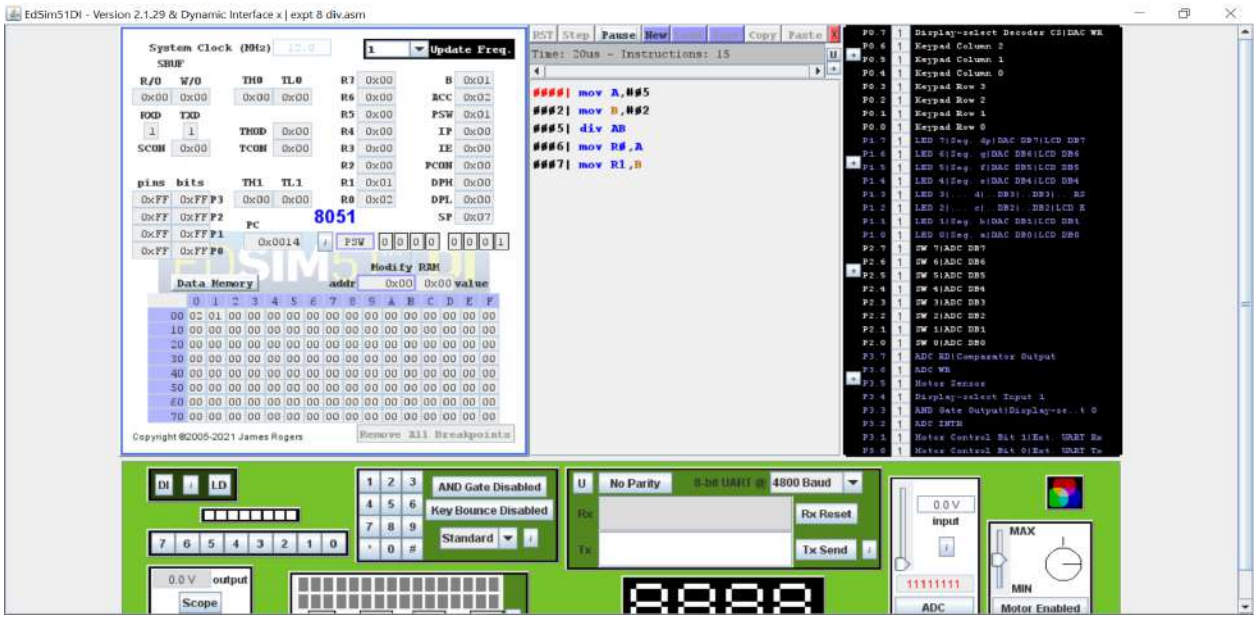
CALCULATIONS:

05 - 0000 0101

02 - 0000 0010

/

Result - 0000 0010 – 02



ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
0000		Mov A, #01		Move a data (8 bit) to Acc
0002		Mov B, #03		Move a data (8 bit) to B Reg
0005		ANL A, B		AND them, and store result in A
0007		Mov R0, A		Copy the result to a Reg

Observation

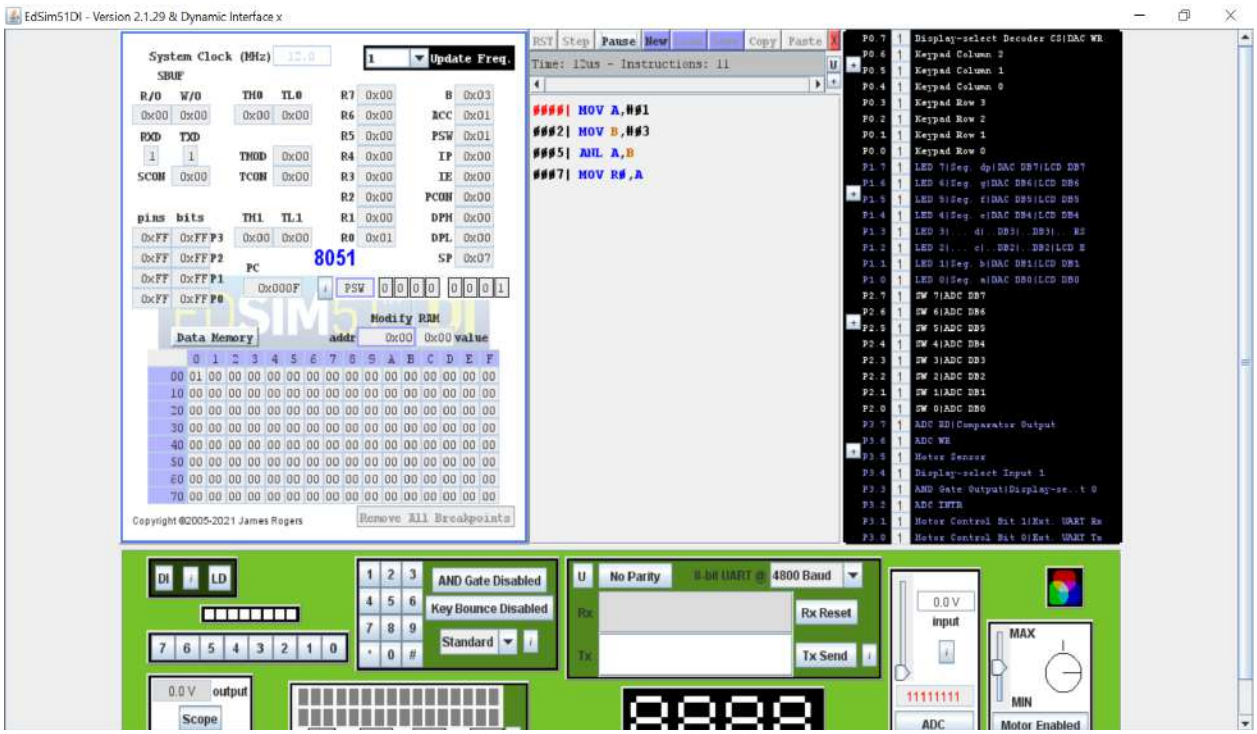
OUTPUT ADDRESS	DATA
00	01
INPUT ADDRESS	**DATA**
0000	01
0002	03

CALCULATIONS:

01 - 0000 0001

03 - 0000 0011

Result - 0000 0001 - 01



Program 8(f) : OR

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
0000		Mov A, #01		Move a data (8 bit) to Acc
0002		Mov B, #03		Move a data (8 bit) to B Reg
0005		ORL A, B		OR them, and store result in A
0007		Mov reg, a		Copy the result to a Reg

Observation

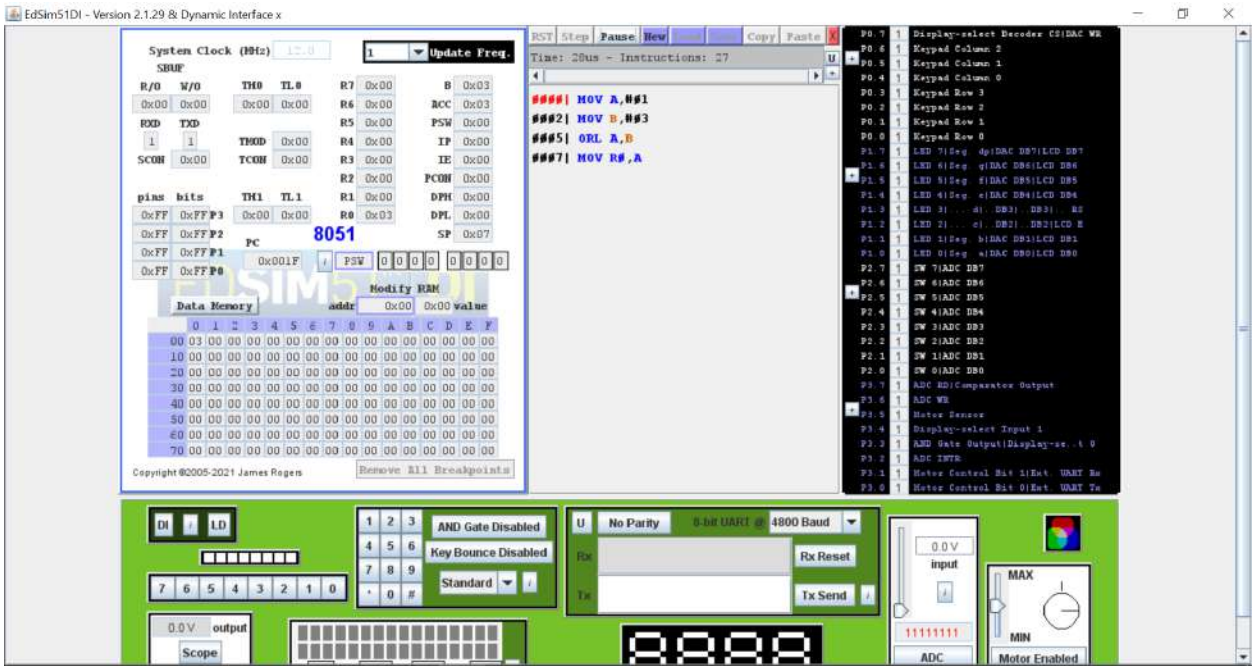
OUTPUT ADDRESS	**DATA**
00	03
INPUT ADDRESS	**DATA**
0000	01
0002	03

CALCULATIONS:

01 - 0000 0001

03 - 0000 0011

Result - 0000 0011 – 03



Program 8(g) : CPL

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
0000		Mov A, #02		Move a data (8 bit) to Acc
0002		CPL A		Performing not operation(complement)
0003		Mov reg, a		Mov Acc to reg

Program 8(g) :CPL Observation

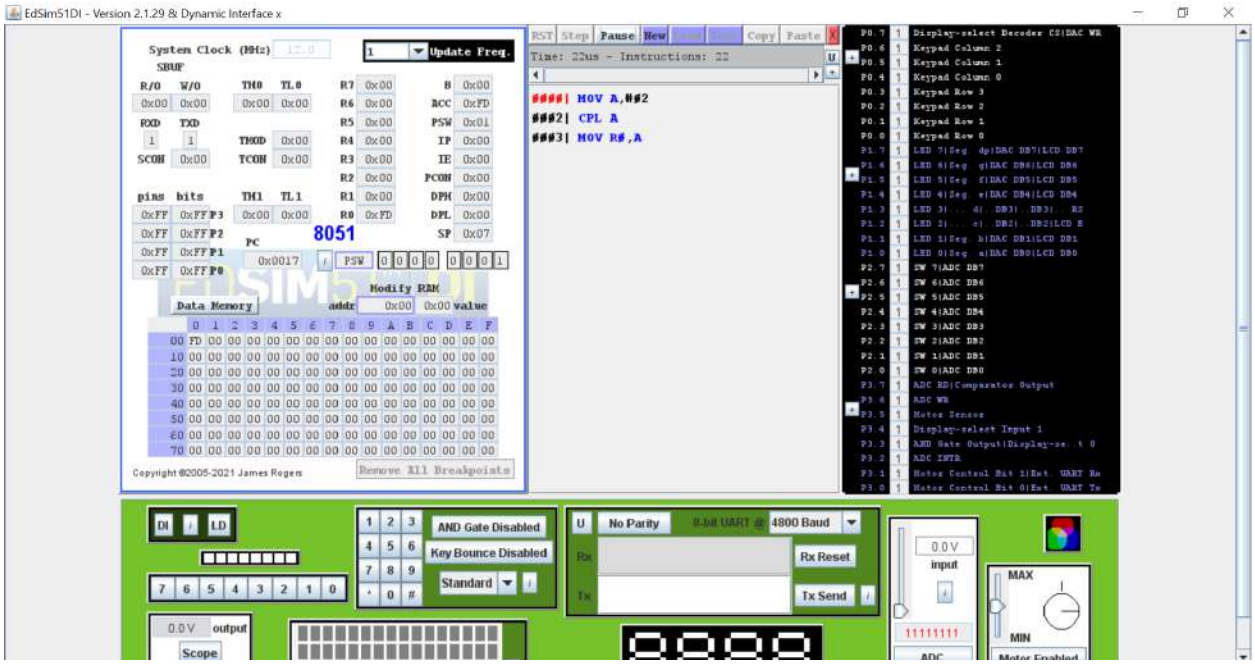
OUTPUT ADDRESS	DATA
00	FD

INPUT ADDRESS	DATA
0000	02

CALCULATIONS:

02 - 0000 0010

Result - NOT(02) - 1111 1101 -> FD


EdSim51DI - Version 2.1.29 & Dynamic Interface x
System Clock (MHz)
Update Freq.
8051
Data Memory
Modify RAM
Copyright ©2005-2021 James Rogers
Remove All Breakpoints
RST
Step
Pause
New
Copy
Paste
Time: 22us - Instructions: 22
MOV A,#2
##2| CPL A
##3| MOV R#,A
AND Gate Disabled
Key Bounce Disabled
Standard
No Parity
4800 Baud
Rx Reset
Tx Send
input
output
Scope
MAX
MIN

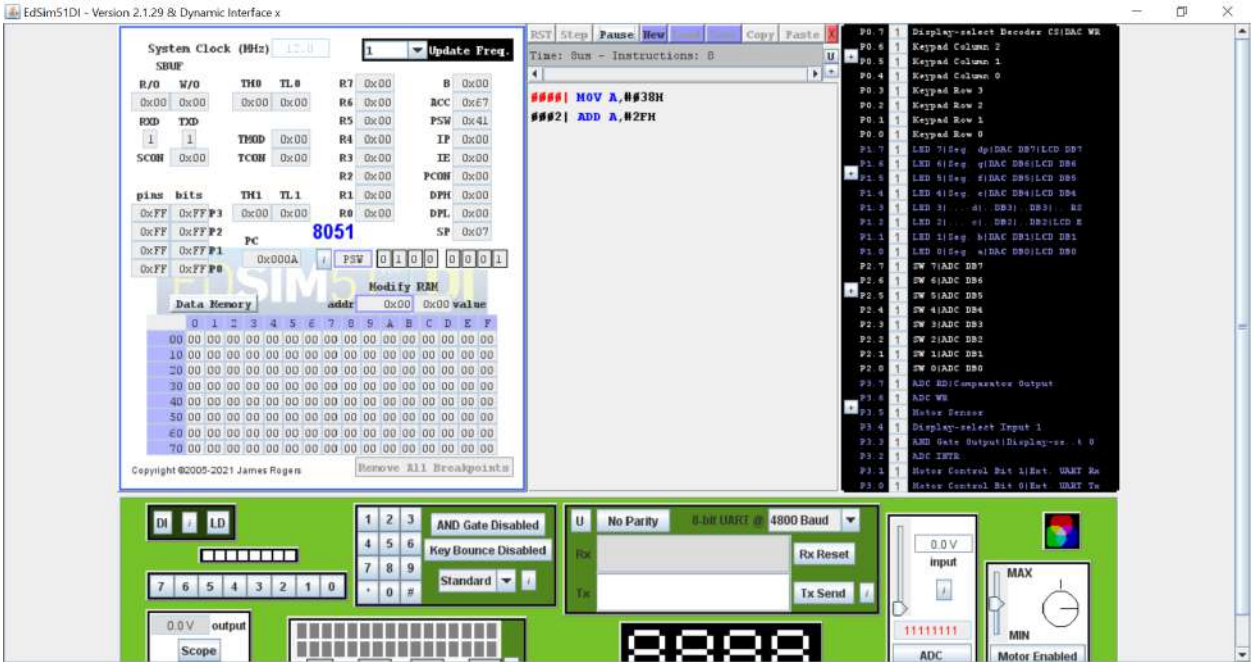
8.7 PRE - LAB SOLUTIONS:

1). Show the status of carry flag, auxiliary carry, and parity flags after the addition of 38H and 2FH in the following instructions

MOV A, #38H

ADD A, #2FH

The desired program is simulated as shown below:

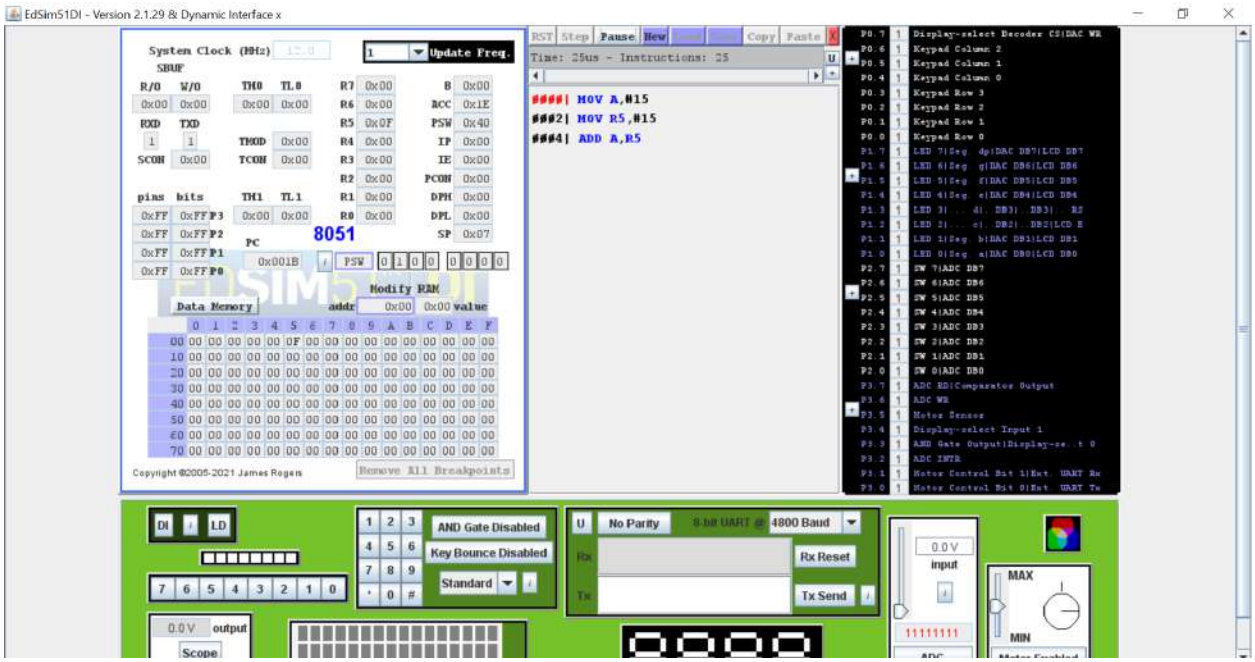


From the simulation, it is determined that the carry flag is set to 0, the auxiliary flag is set to 1, and lastly the parity flag is set to 1.

2). What is the result of the following code and where is it kept?

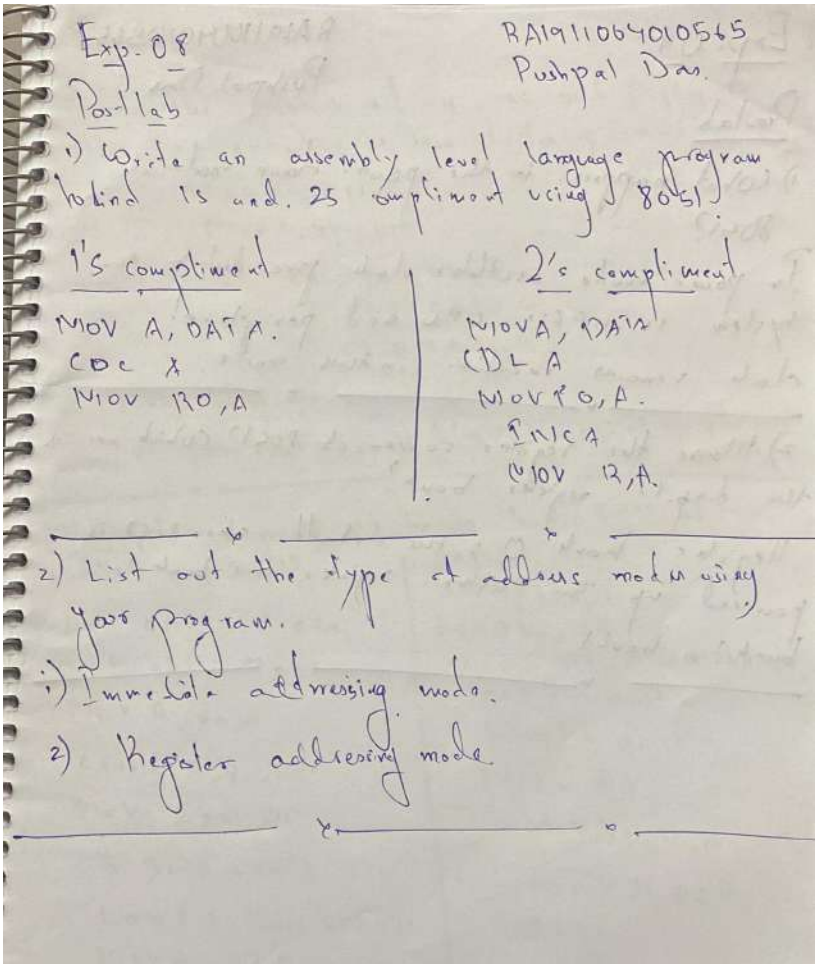
MOV A, #15
MOV R5, #15
ADD A, R5`

The simulation of the above program is as shown below:



From the simulation, the result is determined to be 1E, and it is stored in the Accumulator(ACC).

8.8 POST SOLUTIONS:



Exp. 08
Post Lab

RA1911064010565
Pushpal Das.

1) Write an assembly level language program to find 1s and 2s compliment using 8051.

1's compliment

```
MOV A, DATA.
CPL A
MOV R0, A
```

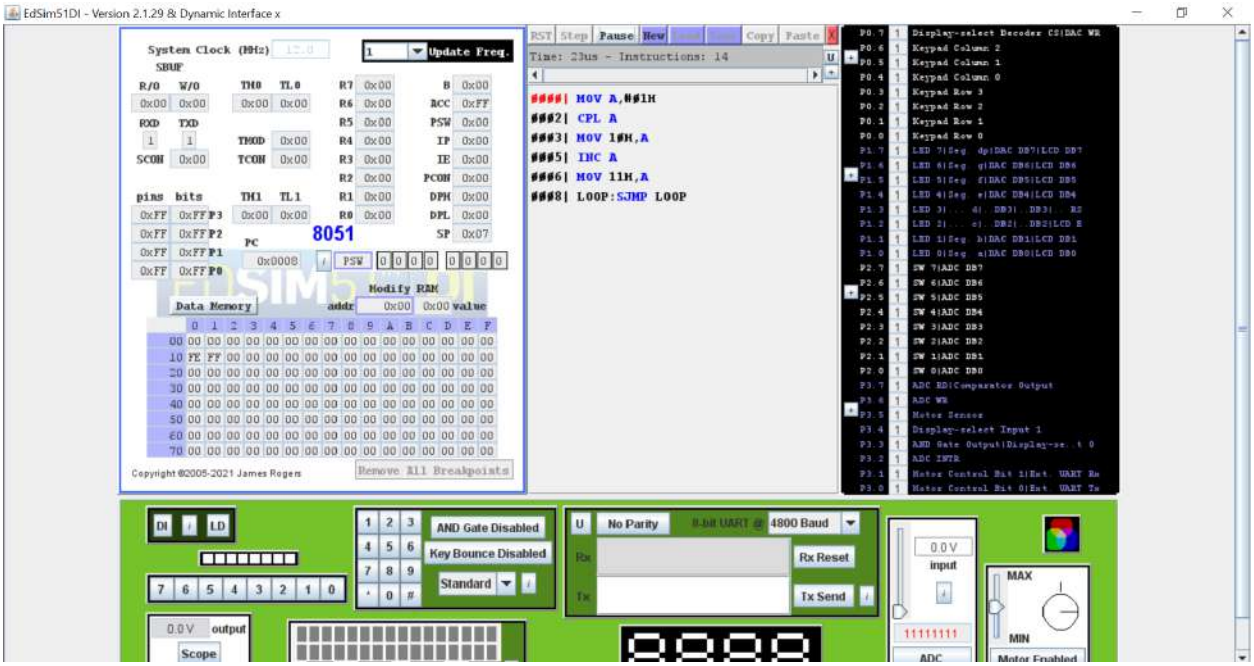
2's compliment

```
MOV A, DATA
CLR A
MOV R0, A.
INC A
MOV R3, A.
```

2) List out the type of address mode using your program.

i) Immediate addressing mode.

ii) Register addressing mode.



8.9 Result:

Thus, the registers, instruction sets, and arithmetic operators of 8051 by addition, subtraction, multiplication, and division in the given two 8 bit numbers and store them in a memory location was experimented.

SRM Institute of Science and Technology
Faculty of Engineering and Technology
Department of Electronics and Communication Engineering

18ECC203J Microprocessor, Microcontroller & Interfacing Techniques Lab

Fifth Semester, 2021-22 (Odd Semester)

Name : **Pushpal Das**

Register No. : **RA1911004010565**

Day / Session : **5 / 3&4**

Venue : **Online (EDSIM 51)**

Title of Experiment : **One's And Two's Complement, BCD To Hexadecimal Conversion And Hexadecimal To BCD Conversion using 8051**

Date of Conduction : **16/09/21**

Date of Submission : **23/10/21**

Particulars	Max. Marks	Marks Obtained
Pre-lab	5	
Lab Performance	10	
Post-lab	15	
Total	**30**	

REPORT VERIFICATION

Date : **22/09/21**

Staff Name : Mr.R.Prithiviraj

Signature :

Experiment 9
One's And Two's Complement, BCD To Hexadecimal Conversion And Hexadecimal To BCD Conversion using 8051

9.1. Aim(s) / Objective(s) / Purpose.

The purpose of this experiment is to find the 1's and 2's complement of a 8-bit number and to convert the BCD to HEXADECIMAL and vice versa of a given data using 8051 micro controller and store them in a memory location using 8051.

9.2 Introduction / Background

The purpose of this experiment is to learn about to find 1's and 2's complement and to learn number conversion from *BCD to HEXADECIMAL* and vice versa via *the registers, instruction sets, data transfer operation and logical operation of 8051 by using CPL, SWAP, ANL, ADD, MUL, DIV in the given two 8 bit numbers and store them in a memory location.*

Tips: Any information copied directly or verbatim from Lab manuals or other references should be stated within quotes and referred, otherwise, it is considered plagiarism.

9.3 Materials / Equipment

1. *Muhammad Ali Mazidi and Janice GillispieMazidi, "The 8051 - Microcontroller and Embedded systems", 7th Edition, Pearson Education, 2011.*
2. *Subrataghoshal " 8051 Microcontroller Internals Instructions ,Programming And Interfacing",2nd edition Pearson 2010*

9.4 Hardware Requirement:

The 8051 Microprocessor kit, Power Supply.

Software Requirement :

8051 EdSim

9.5 Procedure

i) Enter the opcodes from memory location
ii) Execute the program
iii) Check for the result at specified memory location in the program

9.6 Program Logic:

9.6.1 One's And Two's Complement

1. Move the data to Accumulator.
2. Complement the accumulator.
3. Move the one's complement output to the memory 4500H.
4. Add 01H with accumulator.
5. Move the two's complement output to the memory 4501H.

9.6.2. BCD To Hexadecimal Conversion And Hexadecimal To BCD Conversion

Binary coded decimal (BCD) is a system of writing numerals that assigns a four-digit binary code to each digit 0 through 9 in a decimal (base-10) numeral. The hexadecimal number system (also called base-16) is a number system that uses 16 unique symbols to represent a particular value. Those symbols are 0-9 and A-F.

BCD number to be converted is brought to the accumulator. Mask the lower order nibble using ANL instruction and swap their nibbles using SWAP instruction. Store the resultant value in any registors. Then multiply the result with 10 Decimal. Mask the higher order nibble and the result is added with the result obtained from above multiplication. Finally, the result is stored in memory .

The hexadecimal number system (also called base-16) is a number system that uses 16 unique symbols to represent a particular value. Those symbols are 0-9 and A-F. In this program, the hex number is converted to its equivalent BCD number. The hex number to be converted is brought to the accumulator and is divided by 100 D i.e 64 . DIV instruction of 8051 is used in this program. The remainder is now divided by 10 D. Swap the quotient and add the result with remainder obtained from above division. Finally, the result is stored in memory .

Program 9(a) : One's And Two's Complement

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
0000		MOV A,#02H		Copy Data To Register A
0002		CPL A		Find Complement Of Data In Accumulator
0003		MOV 30H,A		Store Result In 10H Memory Location,
0005		INC A		Increment Stored Result By 1.
0006		MOV 31H, A		Move The New Result to 11H Memory Location.
0008	**LOOP**	SJMP **LOOP**		Short To Jump Loop.

Observation

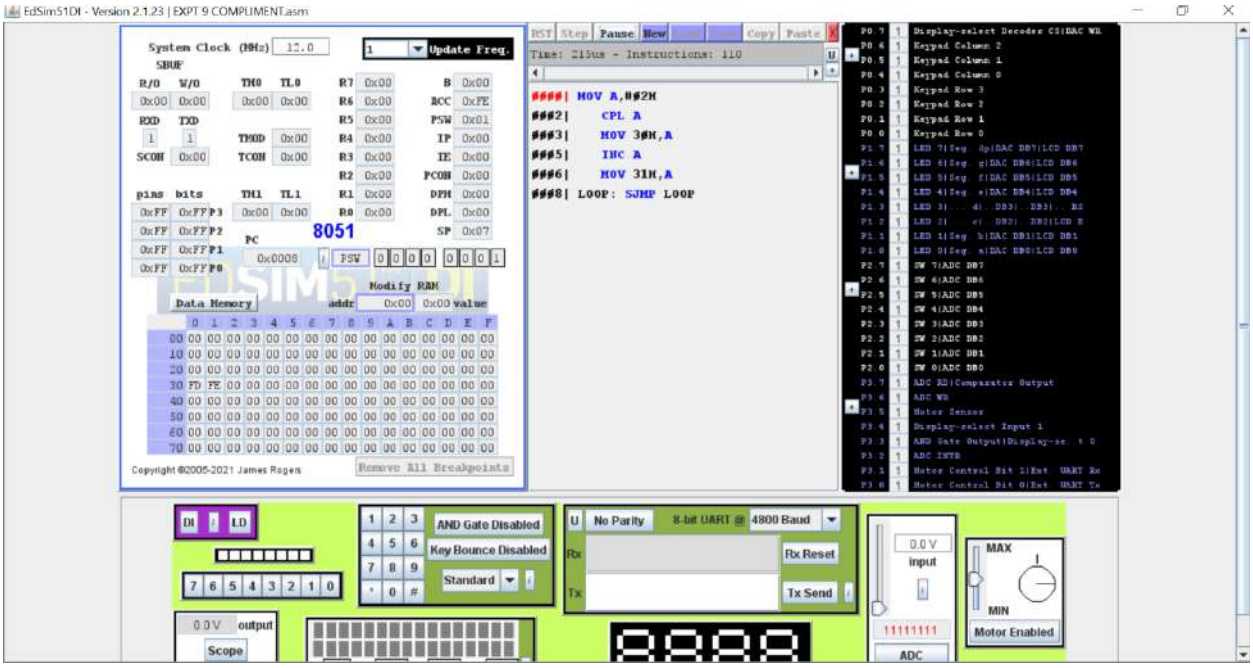
IN PUT ADDRESS	DATA
Register A	02

OUT PUT ADDRESS	DATA
30H	FD
31H	FE

CALCULATIONS: Input - 01H - 0000 0010

1's Complement Result - 1111 1101 - FD

2's Complement Result - 1111 1101 +1 - 1111 1110 – FE



Program 9(b): BCD to HEXADECIMAL

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
0000		MOV A,30H		Copy Data To Register A.
0002		MOV R5,A		Copy Data From Accumulator To Register R5.
0003		ANL A,#0F0H		Perform AND operation of data and F0 and store the result in A
0005		SWAP A		Swap The Digits Of Result In A.
0006		MOV R1,A		Copy the swapped data to register R1.
0007		MOV A,R5		Copy the original data from register R5 to A (accumulator)
0008		ANL A,#0FH		Perform AND operation of data and 0F and store the result in the accumulator.
000A		MOV R2,A		Copy The Result To R2
000B		MOV A,R1		Copy Data From R1 To A.
000C		MOV B,#0AH		Copy Data To Register B.
000F		MUL AB		Multiply The Data In A And B
0010		ADD A,R2		Add the data in A (lower half after multiplication) and R2
0011		MOV 31H,A		Store the result in the address 31H
0013	**HERE**	SJMP **HERE**		Halt

OUT PUT ADDRESS	DATA
31H	**3B**

Observation

IN PUT ADDRESS	DATA
30H	**59**

CALCULATIONS:

59 -> 0101 1001

F0 – 1111 0000

ANL – 0101 0000 – 50 – SWAP – 05 – R1

59 – 0101 1001

0F – 0000 1111

ANL – 0000 1001 – 09 – R2

05 – 0000 0101 – A

0A – 0000 1010 – B

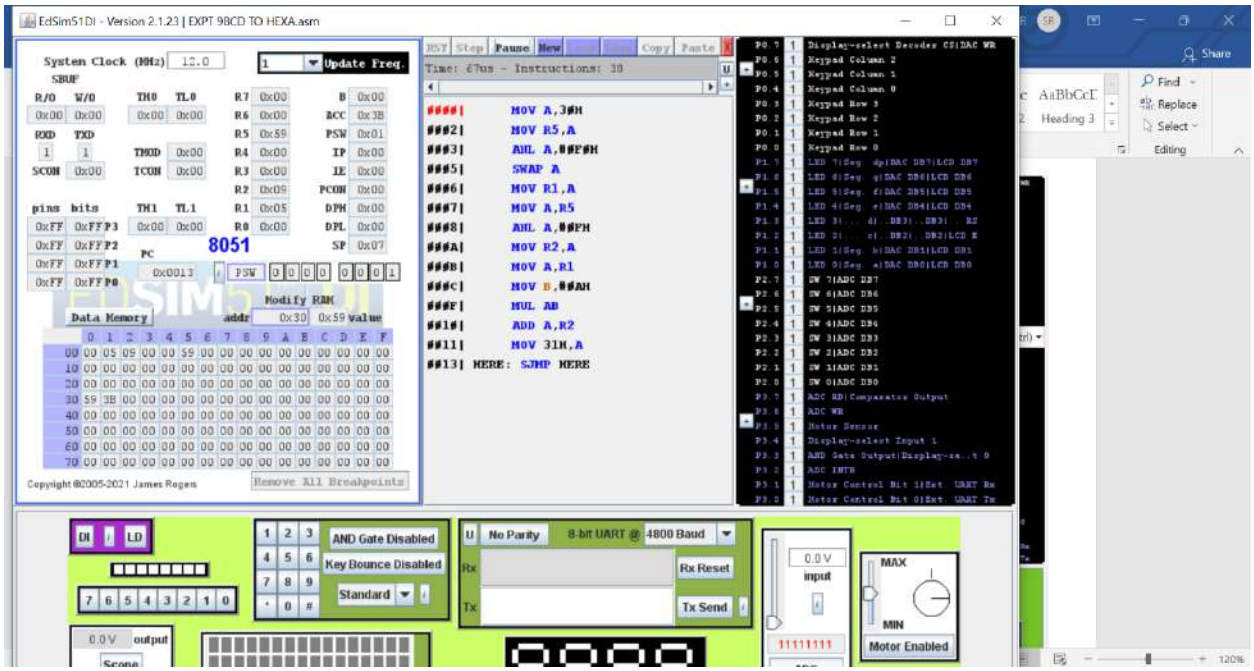
MUL AB – 0000 0000 0011 0010 – A: 0011 0010

ADD – 08 to A –

A- 0011 0010

09- 0000 1001

0011 1011 - #3B = 59



Program 9(c) : Hexadecimal to BCD

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
0000		MOV A,30H		Copy data to register A (Accumulator).

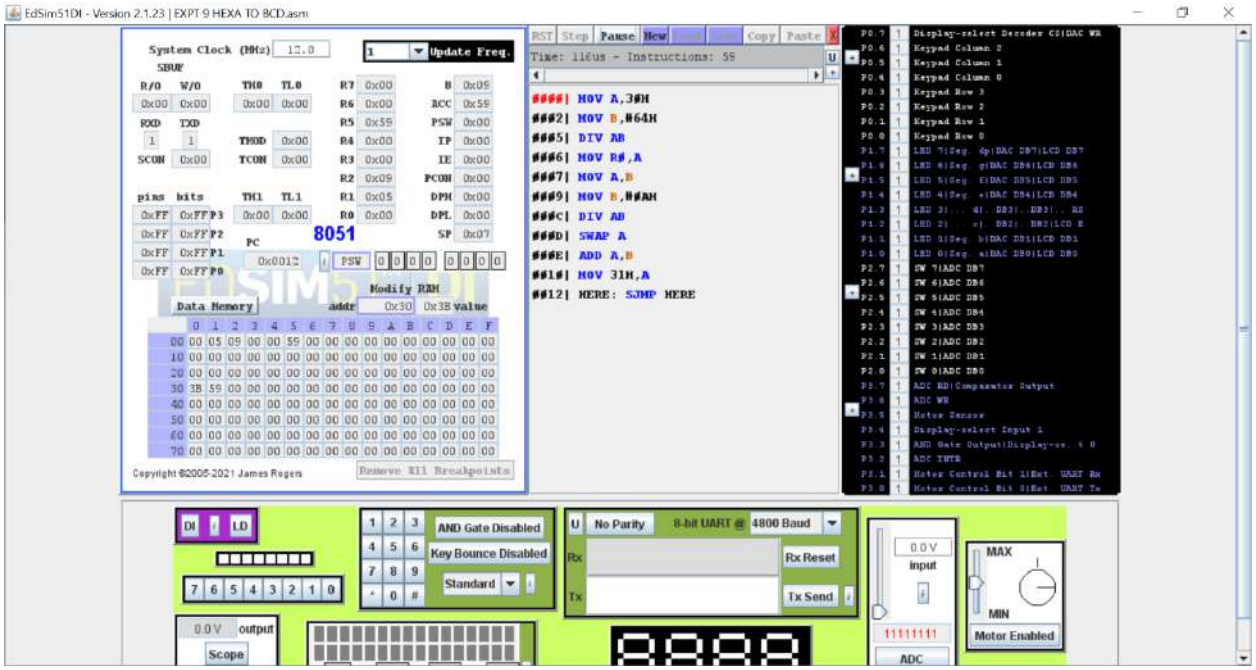
0002		MOV B,#64H		Copy data to register B
0005		DIV AB		Divide the data in A and B
0006		MOV R0,A		Copy quotient in A to register R0.
0007		MOV A,B		Copy data from register B to A.
0009		MOV B,#0AH		Copy Data To Register B.
000C		DIV AB		Divide the data in A and B
000D		SWAP A		Swap the Data in A(Quotient).
000E		ADD A,B		Add A and B (Remainder).
0010		MOV 31H,A		Store the result in 31h address
0012	**HERE**	SJMP **HERE**		Halt

Observation

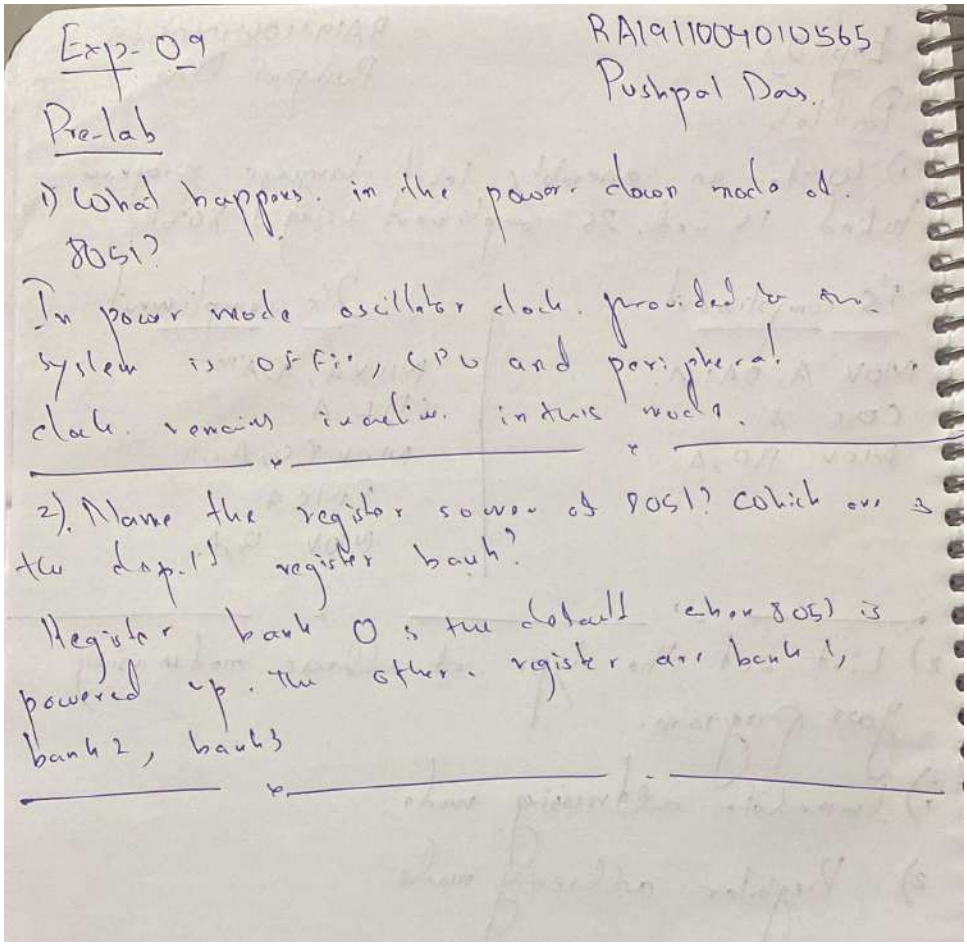
OUT PUT ADDRESS	**DATA**
31H	**59**

IN PUT ADDRESS	**DATA**
30H	**3B**

CALCULATIONS:
A - #3Bh – 0011 1011
B - #64h – 0110 0100
DIV AB–Q–#00h–R
– #3Bh B – #0Ah A – #3Ah
DIV AB – Q – #05h – R - #09h SWAP A - #50h ADD A, B - #59h



9.7 Pre Lab Solutions:


Exp- 09

RA1911004010565
Pushpal Das.

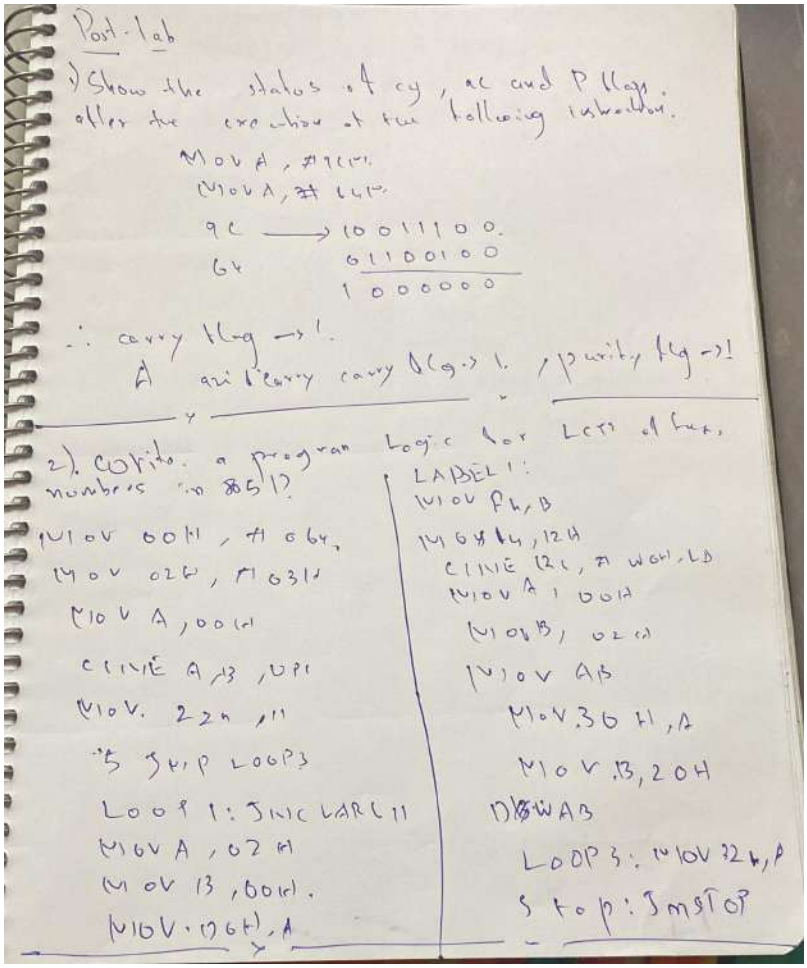
Pre-lab

1) What happens in the power down mode of 8051?

In power mode oscillator clock provided to the system is off i.e, CPU and peripheral clock remains inactive in this mode.

2). Name the register some of 8051 which are the default register bank?

Register bank 0 is the default when 8051 is powered up. The other register are bank 1, bank2, bank3



9.9 Result

Thus, the 1's and 2's complement of a 8-bit number and the number conversion from BCD to HEXADECIMAL and vice versa of a given data using 8051 micro controller were experimented.

SRM Institute of Science and Technology
Faculty of Engineering and Technology
Department of Electronics and Communication Engineering

18ECC203J Microprocessor, Microcontroller & Interfacing Techniques Lab

Fifth Semester, 2021-22 (Odd Semester)

Name : Pushpal Das

Register No. : RA1911004010565

Day / Session : 1ST / FN

Venue : Online (G Meet)

Title of Experiment : Interfacing Of DAC using 8051

Date of Conduction : 22/09/21

Date of Submission : 30/10/21

Particulars	Max. Marks	Marks Obtained
Pre-lab	5	
Lab Performance	10	
Post-lab	15	
Total	**30**	

REPORT VERIFICATION

Date : 30 SEP 21

Staff Name : Dr.R.Prithiviraj

Signature :

Experiment 10
Interfacing of DAC 8051

10.1 Aim(s) / Objective(s) / Purpose.

The purpose of this experiment is to learn about the interfacing of Digital to Analog converter with 8051 microcontroller using edsim51.

10.2 Introduction / Background

The purpose of this experiment is to learn about the registers, instruction sets, general-purpose registers, logical operators, indirect addressing, and loop instructions, compare instruction, exchange instruction, increment & decrement instruction and interfacing of 8051 with external device.

10.3 **Materials / Equipment**

1. *Muhammad Ali Mazidi and Janice GillispieMazidi, "The 8051 - Microcontroller and Embedded systems", 7th Edition, Pearson Education, 2011.*

2. *Subrataghoshal " 8051 Microcontroller Internals Instructions ,Programming And Interfacing",2nd edition Pearson 2010*

10.4 Hardware Requirement:

The 8051 Microcontroller kit, Power Supply.

Software Requirement:

8051 simulator (Edsim51)

10.5 Procedure

a. Load the program in the edsim51.

b. Load the input values to the corresponding address location or directly into the registers.

c. Save, Compile and Emulate the program.

d. Execute the program using Single run or Run until termination process.

e. Store the results of the process in the destination register

f. Terminate the program

10.6 Program Logic:

The physical quantities like temperature, pressure etc are required for the electronic circuit for data processing. The electrical equivalent of such parameters is obtained by the use of transducers. It is difficult to process or store the analog values so there is a conversion of analog signals to the digital domain for the ease of processing and again converting it into the analog domain for the real time output. This is of course achieved by the use of converters. The circuit which is used to convert a digital value into a analog value is a digital to analog converter.

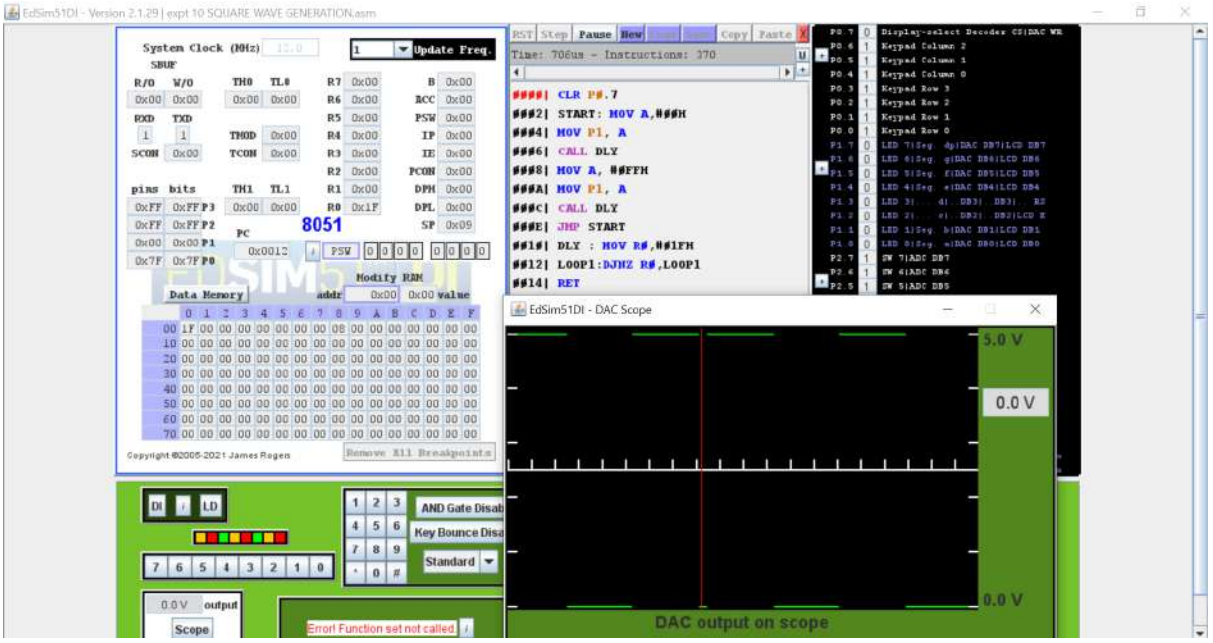
10.7 Square wave generation

The basic idea behind the generation of the waveform is the continuous generation of analog output at DAC. With 00(hex) as the input to the DAC2, the analog value is -5v. similarly , with FF (hex) as input, the output of the DAC is +5v. Outputting the digital data 00 and FF at regular intervals at DAC2, results in a square wave generation.

Program:

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
0000		CLR P0.7		Clearing the 7th bit of port 0.
0002	**START**	MOV A,#00H		Move 00 to Acc (Accumulator).
0004		MOV P1,A		Display connected to port 1 (Display).
0006		CALL DLY		Call Delay
0008		MOV A, #0FFH		Move data FF to Acc (Accumulator).
000A		MOV P1,A		Move data in Acc (Accumulator) to Port 1.
000C		CALL DLY		Call Delay
000E		JMP START		Unconditional Jump To Start.
0010	**DLY**	MOV R0, #01FH		Move 1F to R0.
0012	**LOOP1**	DJNZ R0,LOOP1		Decrement(R0), Jump if NonZero to loop1.
0014		RET		Return To Main Program.

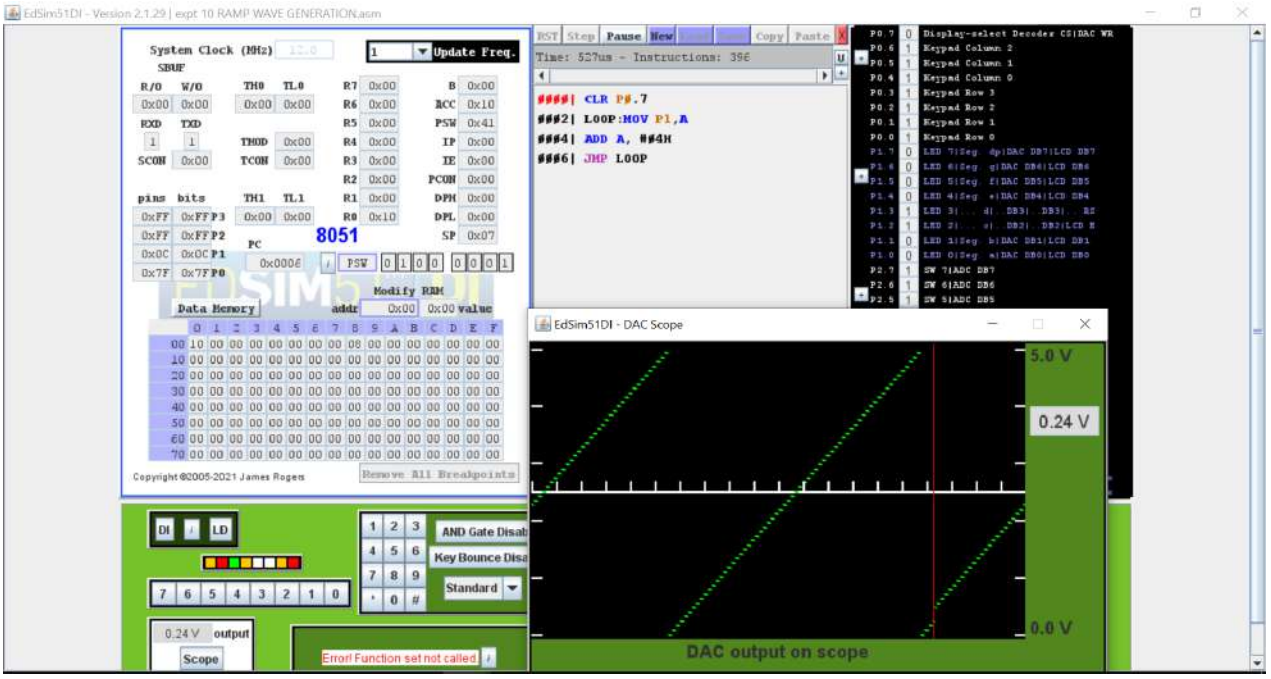
OUTPUT:



10.8 Ramp wave generation

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
0000		CLR P0.7		Clearing the 7th bit of port 0
0002	**LOOP**	MOV P1,A		Display Connected To Port 1(Display).
0004		ADD A,#04h		Increment Accumulator by 4.
0006		JMP LOOP		Unconditional Jump To Loop

OUTPUT:

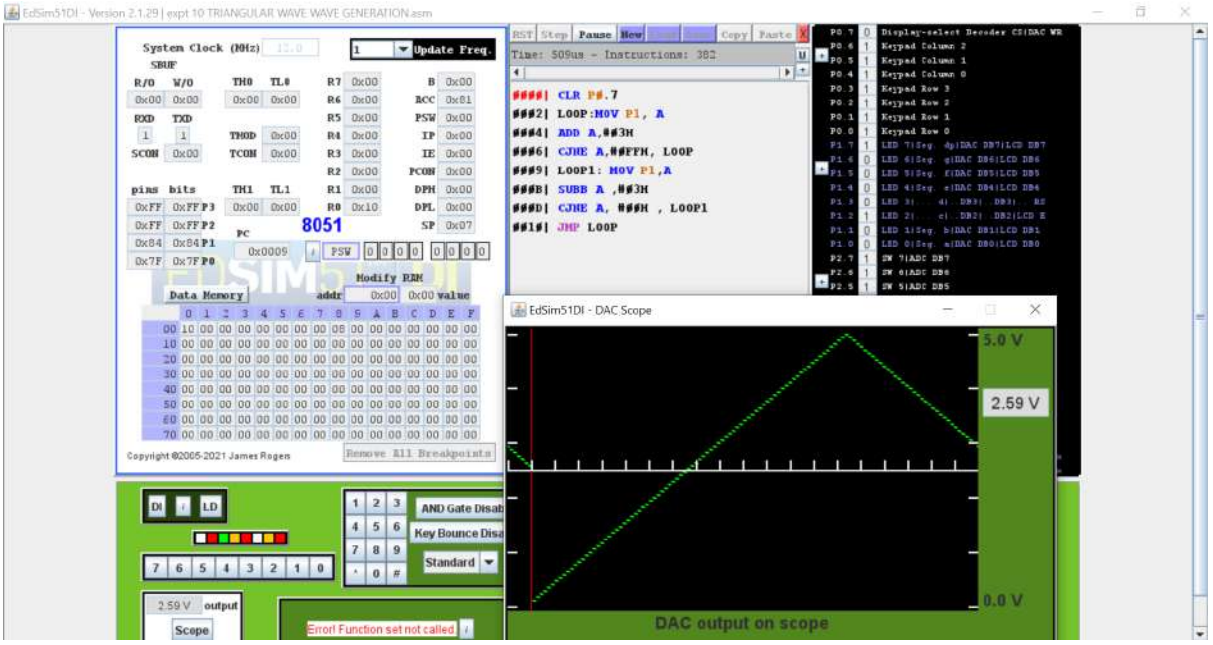


10.9 : Triangle wave generation

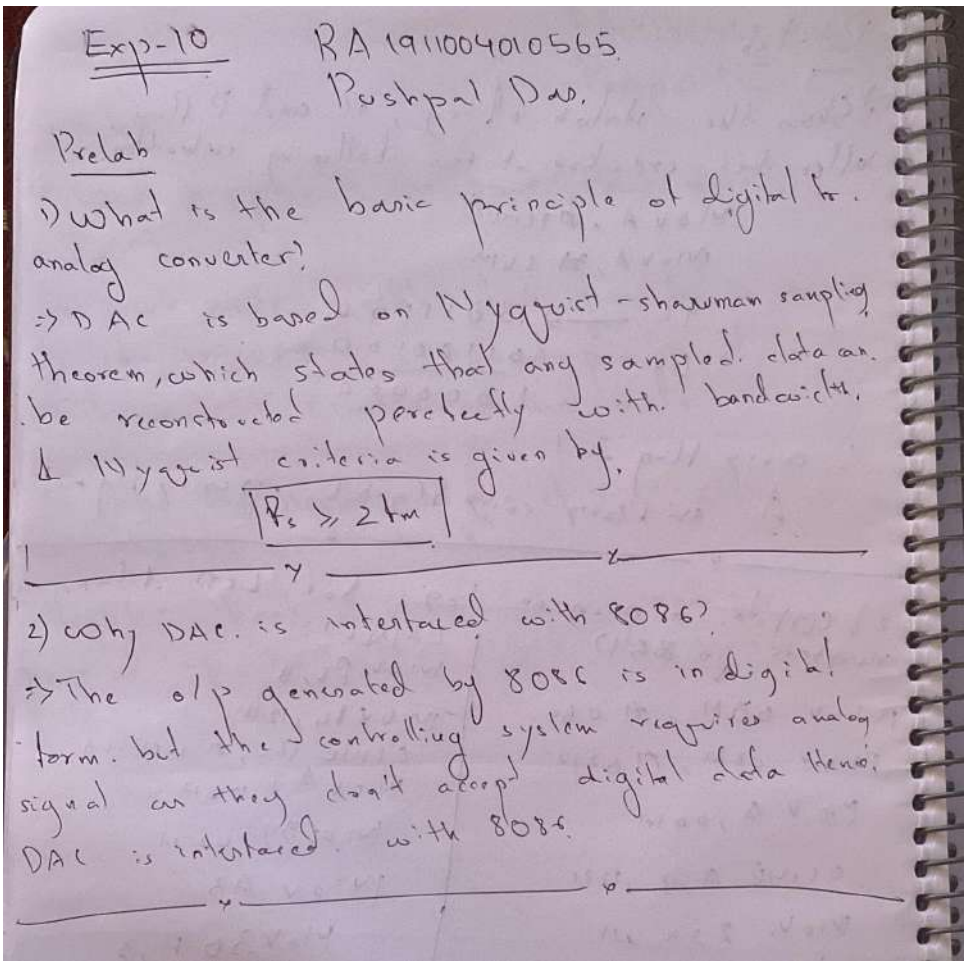
ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
0000		CLR P0.7		Clearing the 7th bit of port 0.
0002	LOOP	MOV P1,A		Move Acc content to P1(Display)
0004		ADD A,#03h		Add 3 to Acc.

0006		CJNE A,#0FFh, LOOP		Compare A, FF, Jump if not equal to LOOP
0009	LOOP1	MOV P1,A		Move Acc content to P1(Display)
000B		SUBB A,#03h		Subtract 3 from Acc (FF)
000D		CJNE A, #00h, LOOP1		Compare A,00, Jump if not equal to LOOP1.
0010		JMP LOOP		Unconditional Jump to Loop

OUTPUT:



10.10 Pre-Lab Solutions:



Exp-10 RA 1911004010565
Pushpal Das.

Prelab

1) What is the basic principle of digital to analog converter?

=> DAC is based on Nyquist-shannon sampling theorem, which states that any sampled data can be reconstructed perfectly with bandwidth & Nyquist criteria is given by.

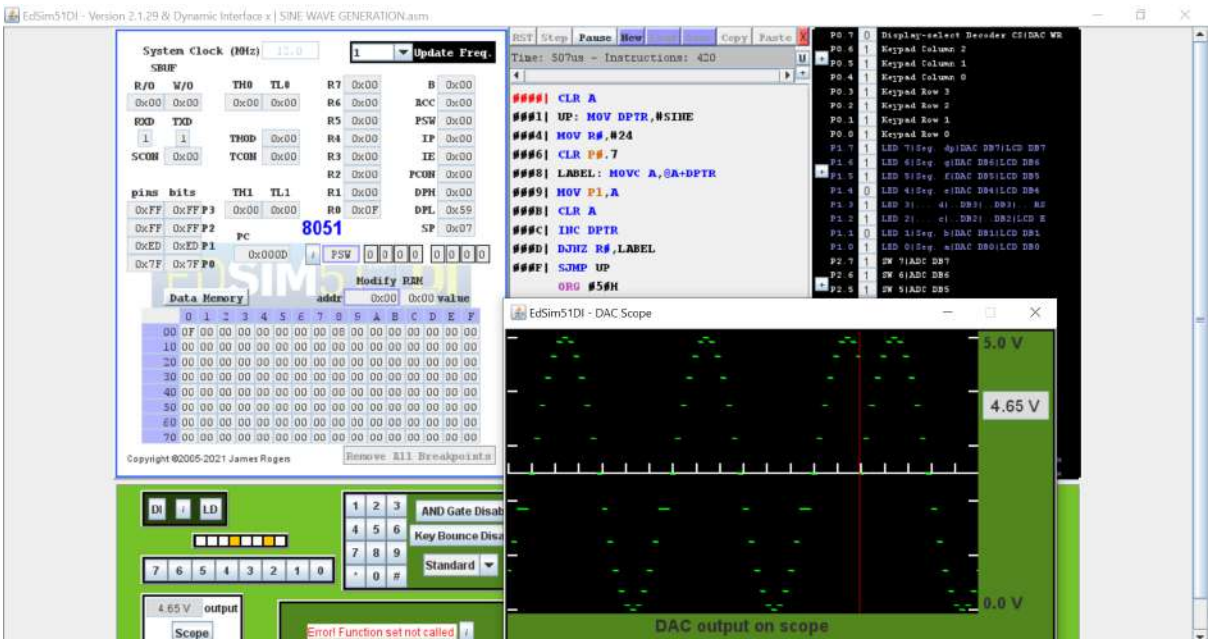
$$f_s \geq 2f_m$$

2) Why DAC is interfaced with 8086?

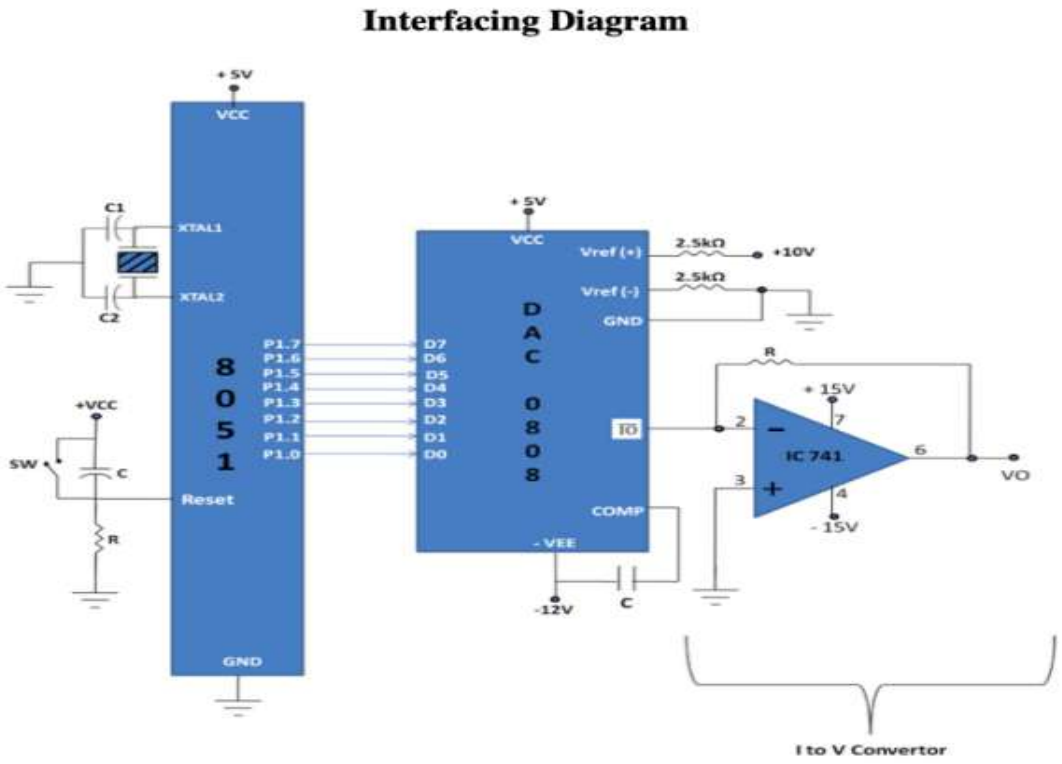
=> The o/p generated by 8086 is in digital form. but the controlling system requires analog signal as they don't accept digital data hence, DAC is interfaced with 8086.

10.11 Post-Lab Solutions:

1. Write a program to generate a SINE wave.



2. Draw interfacing diagram of DAC with 8051 microcontroller



10.12 Result:

Thus, the interfacing of 8051 with Digital to Analog Converter carried out through 8051 instructions.

SRM Institute of Science and Technology
Faculty of Engineering and Technology
Department of Electronics and Communication Engineering

18ECC203J Microprocessor, Microcontroller & Interfacing Techniques Lab

Fifth Semester, 2020-21 (odd semester)

Name : **Pushpal Das**

Register No. : **RA1911004010565**

Day / Session : **05/FN**

Venue : **Online**

Title of Experiment : **USART Programming in 8051**

Date of Conduction : **29/09/21**

Date of Submission : **30/10/21**

Particulars	Max. Marks	Marks Obtained
Pre-lab	5	
Lab Performance	10	
Post-lab	15	
Total	**30**	

REPORT VERIFICATION

Date :**10/10/21**

Staff Name : Dr.R.Prithiviraj

Signature :

Experiment 11
USART Programming in 8051

11.1. Aim(s) / Objective(s) / Purpose.

The purpose of this experiment is to learn about the serial port programming in 8051 microcontroller using edsim51.

11.2 Introduction / Background

. The purpose of this experiment is to learn about the serial ports registers, baud rate, timer registers, transmit and receive interrupt flags in serial port programming 8051 microcontroller.

11.3 Materials / Equipment

1. *Muhammad Ali Mazidi and Janice GillispieMazidi, "The 8051 - Microcontroller and Embedded systems", 7th Edition, Pearson Education, 2011.*

2. *Subrataghoshal " 8051 Microcontroller Internals Instructions ,Programming And Interfacing",2nd edition Pearson 2010*

11.4 Hardware Requirement:

The 8051 Microprocessor kit, Power Supply.

Software Requirement :

8051 EdSim

11.5 Procedure

a. Load the program in the edsim51.
b. Save, Compile and Emulate the program.
c. Execute the program using Single run or Run until termination process.
d. Store the results of the process in the destination register
e. Terminate the program

11.6 Program Logic:

Programming the 8051 to transfer data serially

1. The TMOD register is loaded with the value 20H, indicating the use of timer 1 in mode 2(8-bit auto reload to set the baud rate.

2. The TH1 is loaded with the value. To set the baud rate for serial data transfer.

3. The SCON register is loaded with the value 50H, indicating serial mode 1, where 8-bit data is framed with start and stop bits.

4. TR1 is set to start timer 1.
5. TI is cleared by the “CLR TI “ instruction
6. The character byte to be transferred serially is written into the SBUF register.
7. The TI flag is monitored with the use of the instruction “JNB TI, xx” to see if the character has been transferred completely.
8. To transfer the next character, go to step 5.

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
0000		**CLR SM0**		
0002		**SETB SM1**		Put serial port in 8-bit UART mode
0004		**MOV A, PCON**		Copy PCON to the accumulator
0006		**SETB ACC.7**		Set the accumulator MSB
0008		**MOV PCON, A**		Copy the accumulator back to PCON
000A		**MOVTMOD, #20H**		Put timer 1 in 8-bit auto-reload interval timing mode
000D		**MOV TL1, #243**		Put -13 in timer 1 lower byte (timer will overflow every 13us)
0010		**MOV TH1, #243**		Put same value in low byte so when timer is first started it will overflow after 1 us
0013		**SETB TR1**		Start timer 1
0015		**MOV 30H,#'s'**		Put data to be sent in RAM, start address 30H
0018		**MOV 31H,#’u’**		Put data to be sent in RAM, start address 31H
001B		**MOV 32H,#’k’**		Put data to be sent in RAM, start address 32H
001E		**MOV 33H,#’e’**		Put data to be sent in RAM, start address 33H
0021		**MOV 34H,#’s’**		Put data to be sent in RAM, start address 34H
0024		**MOV 35H,#’h’**		Put data to be sent in RAM, start address 35H
0027		**MOV 36H,#0**		Null-terminate the data
002A	**AGAIN**	**MOV A, @R0**		Move from location pointed to by R0 to the accumulator
002C		**MOV R0,#30H**		Put the data start address in R0.
002D		**JZ Finish**		If the accumulator contains 0, no more data to be sent, jump to finish

002F		**MOV SBUF, A**		Move data to be sent to the serial port.
0031		**INC R0**		Increment R0 to point at next byte of data to be sent
0032		**JNB TI, $**		Wait for TI to be set, indicating serial port has finished sending byte
0035		**CLR TI**		Clear TI
0037		**JMP again**		Send next byte
0039	**Finish**	**JMP $**		Do nothing

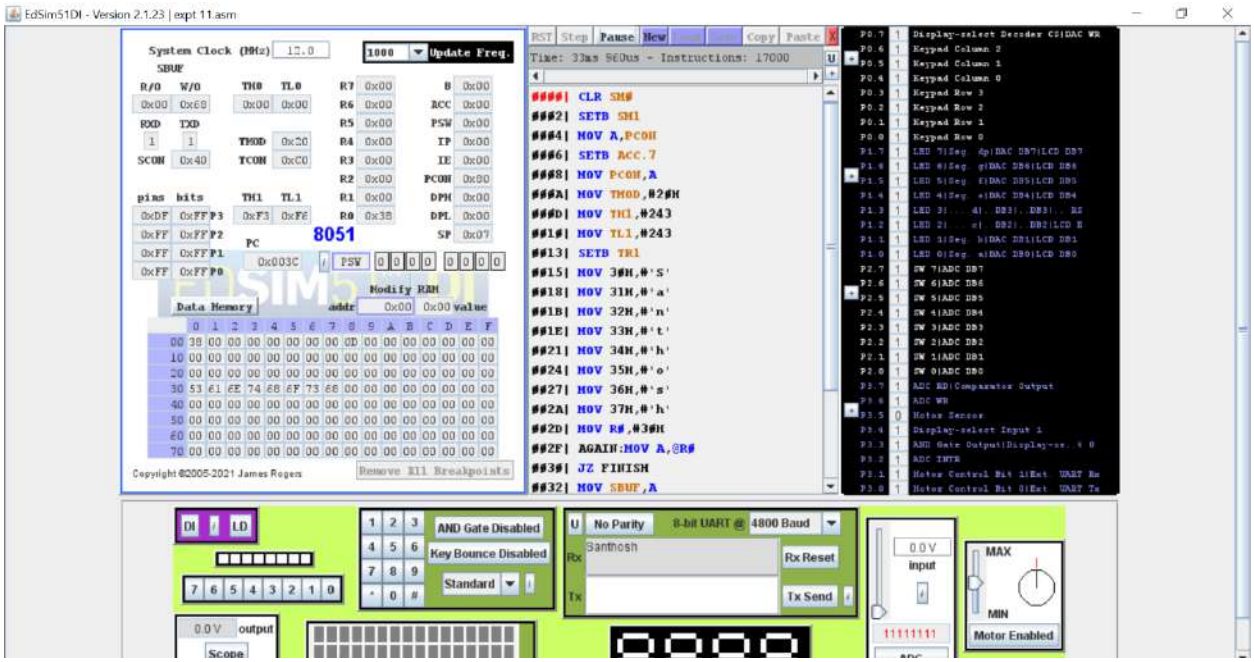
Program 11 : apL

Observation

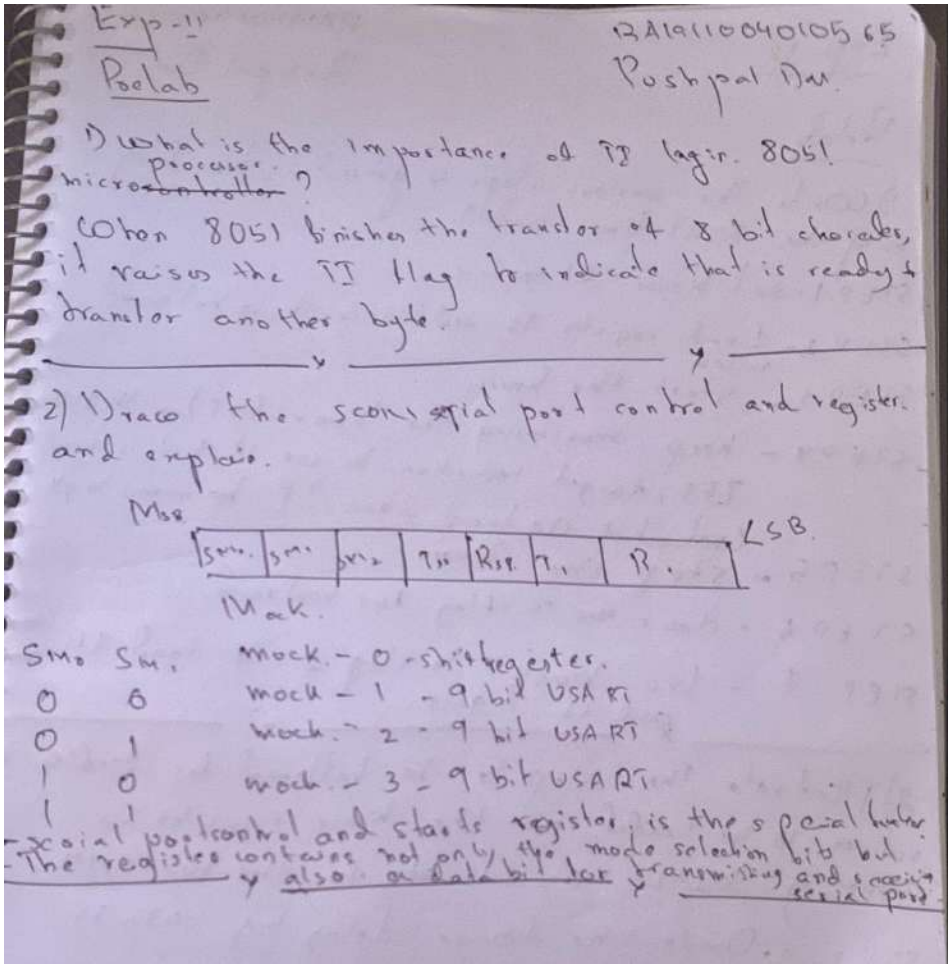
IN PUT ADDRESS	**DATA**
30H	**S**
31H	**a**
32H	**n**
33H	**t**
34H	**h**
35H	**o**
36H	**s**
37H	**h**

OUT PUT ADDRESS	**DATA**
Rx	**Santhosh**

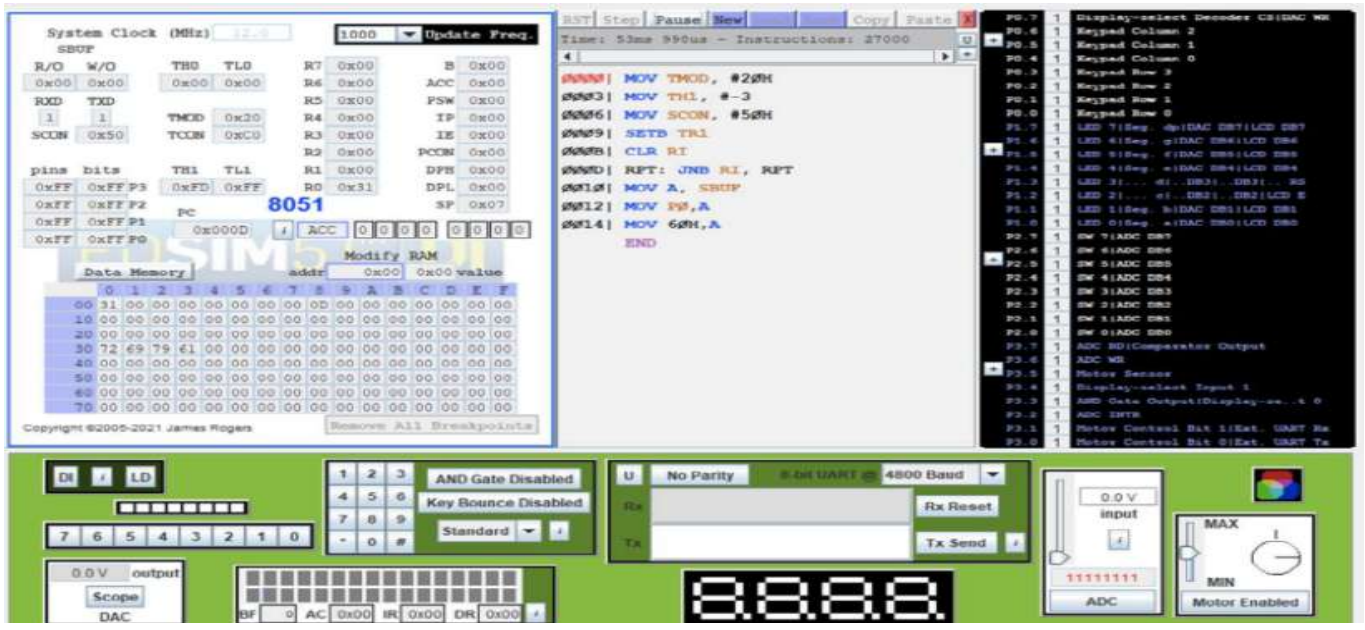
SIMULATION RESULT:



11.7 Pre-Lab Solutions:



11.8 Post-Lab Solutions:



11.9 Result.

Thus, the serial port programming in 8051 microcontroller is carried out through 8051 instructions.

SRM Institute of Science and Technology
Faculty of Engineering and Technology
Department of Electronics and Communication Engineering

18ECC203J Microprocessor, Microcontroller & Interfacing Techniques Lab

Fifth Semester, 2020-21 (odd semester)

Name : **Pushpal Das**

Register No. : **RA1911004010565**

Day / Session : **1ST /FN**

Venue : **Online(G MEET)**

Title of Experiment : **Timer Programming in 8051**

Date of Conduction : **22 OCT 2021**

Date of Submission : **14 NOV 2021**

Particulars	Max. Marks	Marks Obtained
Pre-lab	5	
Lab Performance	10	
Post-lab	15	
Total	**30**	

REPORT VERIFICATION

Date :

Staff Name : Dr.R.Prithiviraj

Signature :

Experiment 12
Timer Programming in 8051

12.1. Aim(s) / Objective(s) / Purpose.

The purpose of this experiment is to learn about the timer programming in 8051 microcontroller using edsim51.

12.2 Introduction / Background

. *The purpose of this experiment is to learn about the timer registers, delay generation, timer mode programming, and timer interrupt flags of timer programming 8051 microcontroller.*

12.3 Materials / Equipment

1. *Muhammad Ali Mazidi and Janice GillispieMazidi, "The 8051 - Microcontroller and Embedded systems", 7th Edition, Pearson Education, 2011.*

2. *Subrataghoshal " 8051 Microcontroller Internals Instructions ,Programming And Interfacing",2nd edition Pearson 2010*

12.4 Hardware Requirement:

The 8051 Microprocessor kit, Power Supply. For Experimenting in EdSim - Online (Nil)

Software Requirement :

8051 EdSim.

12.5 Procedure

a. Load the program in the edsim51.
b. Save, Compile and Emulate the program.
c. Execute the program using Single run or Run until termination process.
d. Store the results of the process in the destination register
e. Terminate the program

12.6 Program Logic:

8051 MODE1 TIMER PROGRAMMING: (16 bit timer mode)

Steps to generate a time delay:

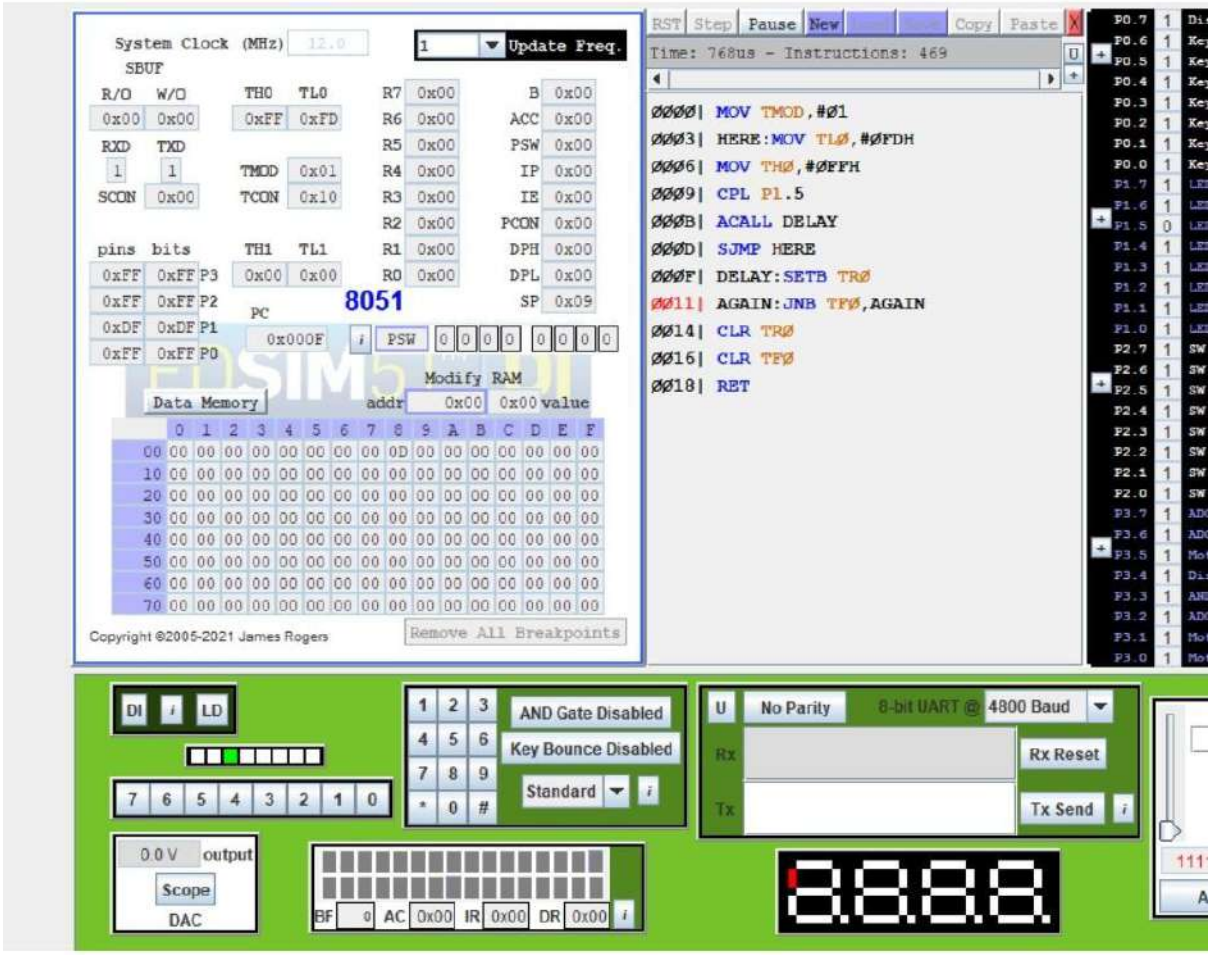
1. Load the TMOD value register indicating which timer (timer 0 or timer 1) is to be used and which timer mode (0 or 1) is selected
2. Load registers TL and TH with initial count value
3. Start the timer

4. Keep monitoring the timer flag (TF) with the JNB TFx, target instruction to see if it is raised Get out of the loop when TF becomes high
5. Stop the timer
6. Clear the TF flag for the next round
7. Go back to Step 2 to load TH and TL AGAIN

Program 1:

LABEL	MNEMONICS	COMMENTS
	MOV TMOD,#01	**Timer 0, mode 1(16-bit mode)**
HERE	**MOV TL0,#0F2H**	**TL0=F2H, the low byte**
	MOV TH0,#0FFH	**TH0=FFH, the high byte**
	CPL P1.5	**Complement/Toggle P1.5**
	ACALL DELAY	
	SJMP HERE	
DELAY	**SETB TR0**	**Start the timer 0**
AGAIN	**JNB TF0,AGAIN**	**Monitor timer flag 0 until it rolls over**
	CLR TR0	**Stop timer 0**
	CLR TF0	**Clear timer 0 flag**
	RET	

CODE & SIMULATION :

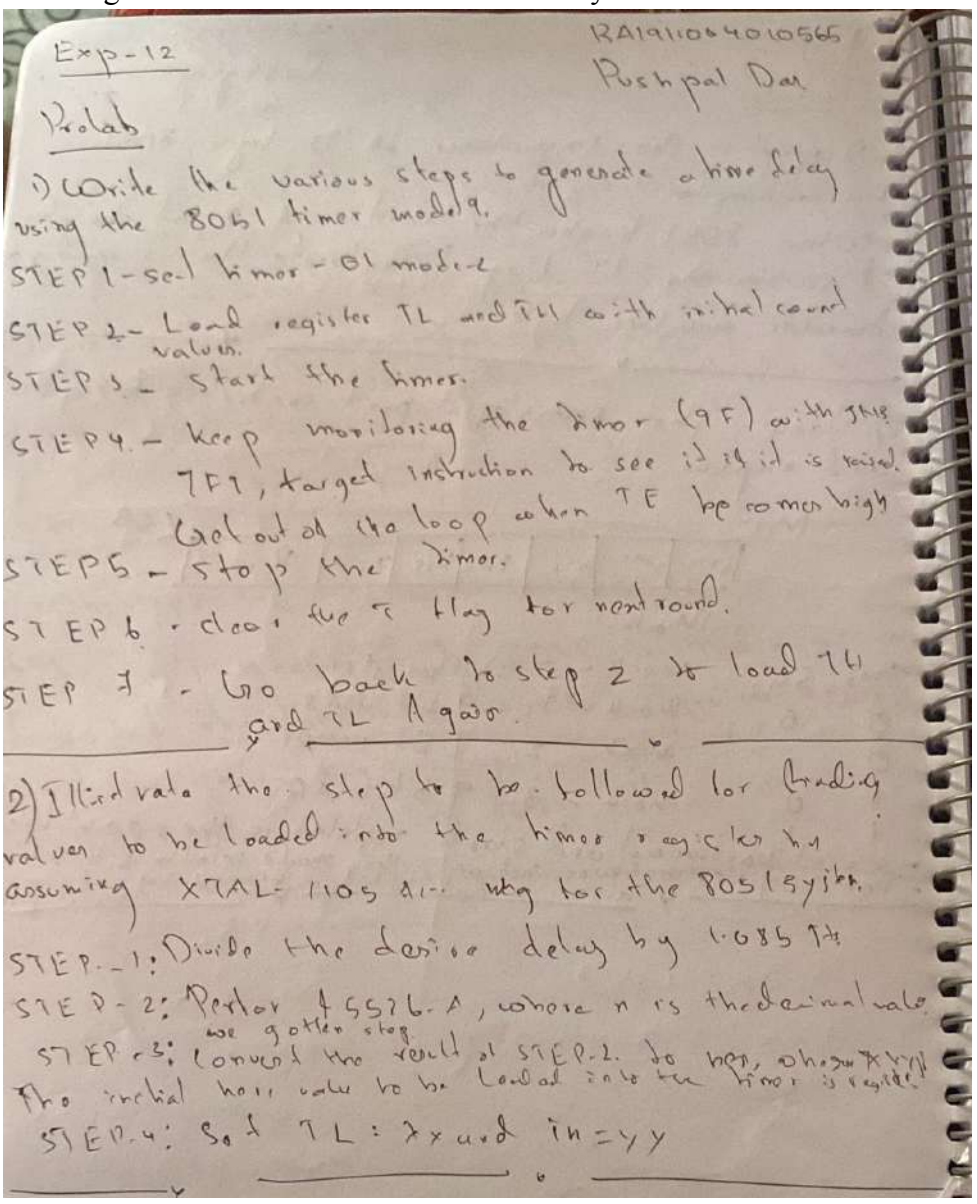


Result Observation:

The timer programming in 8051 microcontroller using edsim51 is experimented for different timer values and observed.

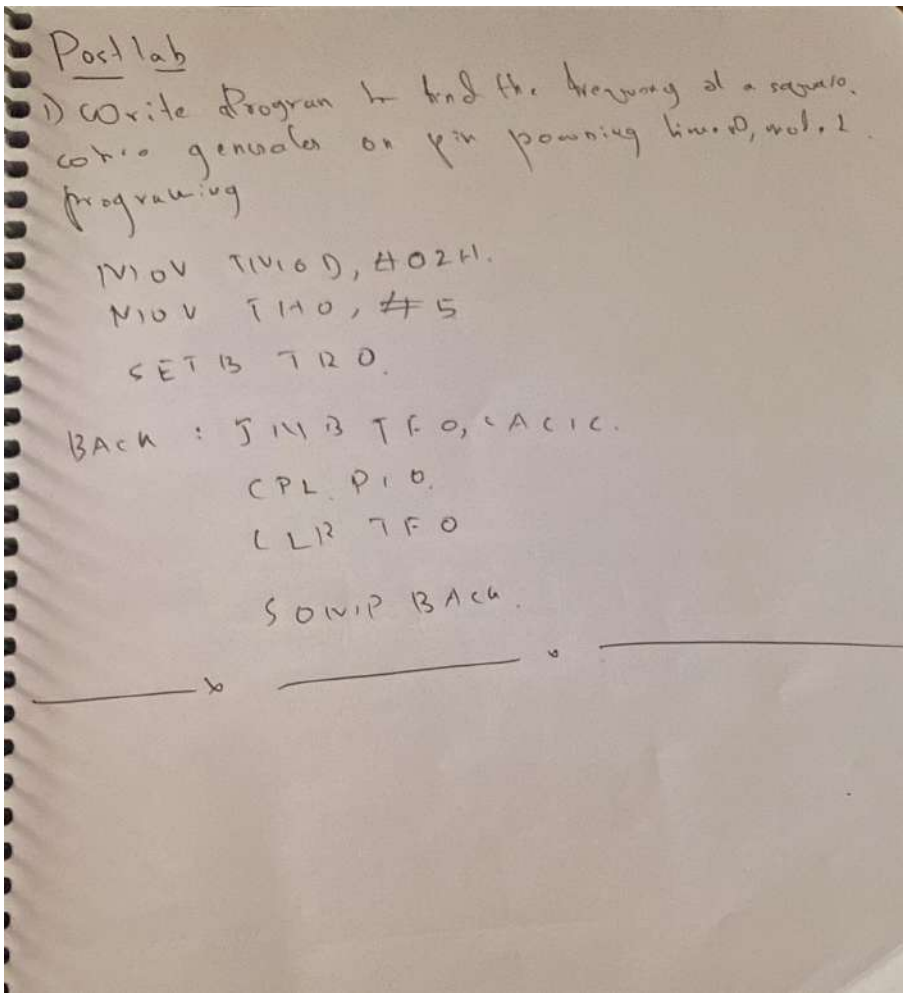
12.7 Pre-Lab Questions:

1. Write the various steps to generate a time delay using the 8051 timer's Mode 2.
2. Illustrate the steps to be followed for finding values to be loaded into the timer register by assuming XTAL=11.0592 MHz for the 8051 system.



12.8 Post-Lab Questions:

1. Write a program to find the frequency of a square wave generated on pin P1.0 using Timer 0, Mode 2 (8-bit auto-reload) programming.



12.9 Result.

Thus, the Timer programming in 8051 microcontroller is carried out through 8051 instructions.

8SRM Institute of Science and Technology Faculty of Engineering and Technology Department of Electronics and Communication Engineering
18ECC203J **MICROPROCESSOR AND MICROCONTROLLER** **Fifth Semester, 2021-22 (odd semester)**

Mini Project Report

Name :Pushpal Das

Register No. :RA1911004010565

Day / Session :

Venue : Online

Project Title :Potentiometer reading in LCD. Lab

Supervisor :

Team Members :PUSHPAL DAS(RA1911004010565) HRIVU DAS MUNSHI(RA1911004010566) KAPIL DEV KUMAR TIWARY.(RA1911004010568)

Particulars	**Max. Marks**	**Marks Obtained**
Objective & Description	05	
Result Analysis	10	
Presentation	10	

Report	05	
Total	**30**	

REPORT VERIFICATION

Date :17/10/2021

Staff Name : Mr.R.Prithiviraj

Potentiometer reading in LCD.

College Name

→ SRM Institute of Science and Technology.

Team member details

TEAM MEMBER 1

- NAME Pushpal Das.
- EMAIL ID pushpaldas2001@gmail.com.
- DISCIPLINE Electronics and Communication Engineering.
- YEAR 3rd year.
- MOBILE 8910497557.

TEAM MEMBER 2

- NAME Hrivu Das Munshi.
- EMAIL ID hd9713@srmist.edu.in
- DISCIPLINE Electronics and Communication Engineering.
- YEAR 3rd year.
- MOBILE 8436155632.

TEAM MEMBER 3

- NAME Kapil Dev Kumar Tiwary.
- EMAIL ID kt6757@srmist.edu.in

• DISCIPLINE Electronics and Communication Engineering. •
YEAR 3rd year.
• MOBILE 77799 24719

Introduction

The main objective of our team is to make a reading through the potentiometer and successfully displaying it in the LED. Here, it is done on a software called proteus, ADC interfacing with 8051 and LCD display with using proteus.

As we all know 8051 do not have inbuilt ADC so we are using external IC of 8bit and 1 channel so that we can send Analog Signal to 8051.

As IC0804 is 8-bit ADC

1. Max decimal value of 8 bit -255

2. Min decimal value of 8 bit - 000

As analog signal changes Decimal count on LCD display changes: - Analog Signal Decimal Value on LCD 100% - 255 50% - 128 0% - 000

Embedded Programming is done by using Microvision Keil 5 software.

Components required

Software Components Required

1.Proteus professional

The Proteus Design Suite combines ease of use with a powerful feature set to enable the rapid design, test and layout of professional printed circuit boards

2. µVision IDE

The µVision IDE combines project management, run-time environment, build facilities, source code editing, and program debugging in a single powerful environment. µVision is easy-to-use and accelerates your embedded software development. µVision supports multiple screens and allows you to create individual window layouts anywhere on the visual surface.

Hardware Components Required

1. **8051 Microcontroller (AT89C51)**

The **AT89C51** is an age old 8-bit microcontroller from the Atmel family. It works with the popular 8051 architecture and hence is used by most beginners till date. It is a 40 pin IC package with 4Kb flash memory. It has four ports and all together provide 32 Programmable GPIO pins. It does not have in-built ADC module and supports only USART communication. Although it can be interfaced with external **ADC IC** like the ADC084 or the ADC0808.

The **AT89C51** is no longer in production and Atmel does not support new design. Instead, the new AT89S51 is recommended for new applications. But, since the AT89C51 has a strong community support if your motive is to learn embedded then AT89C51 can still be a good choice.

2. **Liquid Crystal Display (16*2 LCD Display)**

An LCD (Liquid Crystal Display) screen is an electronic display module and has a wide range of applications. A 16x2 LCD display is very basic module and is very commonly used in various devices and circuits. A 16x2 LCD means it can display 16 characters per line and there are 2 such lines. In this LCD each character is displayed in 5x7 pixel matrix. The 16 x 2 intelligent alphanumeric dot matrix display is capable of displaying 224 different characters and symbols. This LCD has two registers, namely, Command and Data.
Command register stores various commands given to the display. Data register stores data to be displayed. The process of controlling the display involves

putting the data that form the image of what you want to display into the data registers, then putting instructions in the instruction register. In your arduino project Liquid Crystal Library simplifies this for you so you don't need to know the low-level instructions. Contrast of the display can be adjusted by adjusting the potentiometer to be connected across VEE pin.

3. **ADC IC0804 (1 channel- Only one analog signal can connect) (8bit ADC- Analog Signal converted into 8bit digital signal)**

A converter that is used to change the analog signal to digital is known as an analog to digital converter or ADC converter. This converter is one kind of integrated circuit or IC that converts the signal directly from continuous form to discrete form. This converter can be expressed in A/D, ADC, A to D. The inverse function of DAC is nothing but ADC. The analog to digital converter symbol is shown below.

The process of converting an analog signal to digital can be done in several ways. There are different types of ADC chips available in the market from different manufacturers like the ADC08xx series. So, a simple ADC can be designed with the help of discrete components.

The main features of ADC are sample rate and bit resolution.

- → The sample rate of an ADC is nothing but how fast an ADC can convert the signal from analog to digital.
- → Bit resolution is nothing but how much accuracy can an analog to digital converter can convert the signal from analog to digital.

One of the major benefits of ADC converter is the high data acquisition rate even at multiplexed inputs. With the invention of a wide variety of ADC integrated circuits(IC's), data acquisition from various sensors becomes more accurate and faster. Dynamic characteristics of the high-performance ADCs are improved measurement repeatability, low power consumption, precise throughput, high linearity, excellent Signal-to-Noise Ratio (SNR), and so on.

A variety of applications of the ADCs are measurement and control systems, industrial instrumentation, communication systems, and all other sensory-based systems. Classification of ADCs based on factors like performance, bit rates, power, cost, etc.

4. **Potentiometer**

A **potentiometer** is a three-terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider.[1] If only two terminals are used, one end and the wiper, it acts as a **variable resistor** or **rheostat**.

The measuring instrument called a potentiometer is essentially a voltage divider used for measuring electric potential (voltage); the component is an implementation of the same principle, hence its name.

Potentiometers are commonly used to control electrical devices such as volume controls on audio equipment. Potentiometers operated by a mechanism can be used as position transducers, for example, in a joystick. Potentiometers are rarely used to directly control significant power (more than a watt), since the power dissipated in the potentiometer would be comparable to the power in the controlled load

5. **Resistor (10K ohm)**

A 10k ohm resistor has **4 colour bands: brown, black, orange, and gold for 5% tolerance**, respectively. A 1k ohm resistor has 4 colour bands: brown, black, red, and gold for 5% tolerance, respectively. Commonly used in **breadboards and perf boards**, these 10K resistors make excellent pull-ups, pull-downs, and current limiters. To determine the value of a given resistor look for the gold or silver tolerance band and rotate the resistor as in the photo on the left.

6. **Capacitor (150uF)**

A **capacitor** is a device that stores electrical energy in an electric field. It is a passive electronic component with two terminals.

The effect of a capacitor is known as capacitance. While some capacitance exists between any two electrical conductors in proximity in a circuit, a capacitor is a component designed to add capacitance to a circuit. The capacitor was originally known as a **condenser** or **condensator**.[1]This name and its cognates are still widely used in many languages, but rarely in English, one notable exception being condenser microphones, also called capacitor microphones.

7.Respack8

Respack is a device just similar to resistance box used for the variation of the resistances as per use of the circuit but there is subtle difference in the respack that is the resistance present in it are of same value and here the respack used RESPACK-8 which consists of 8 resistances of equal value i.e. 1 K ohm.

CODE

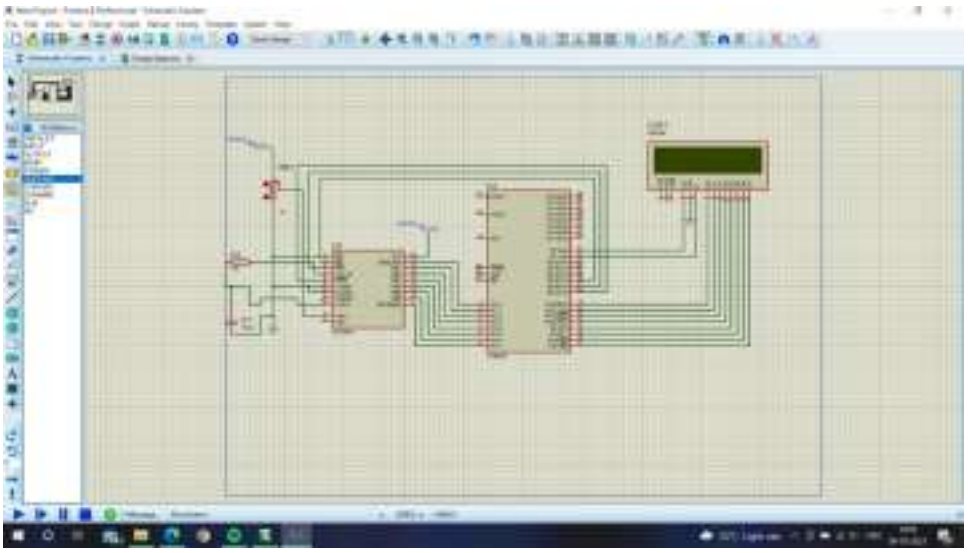
```
#include<reg51.h>

sfr mydata = 0x90;

sbit rd= P2^5;

sbit wr= P2^6;

sbit intr= P2^7;

sbit RS = P2^0;

sbit EN = P2^1;

//(0x90)is address of port1, else you can write as : #defintre
mydata P1


void delay(int n)

{

int i,j;

for(i=0;i<n;i++)

for(j=0;j<255;j++);

}

void lcd_cmd(char a)

{

P3 = a;

RS = 0;
EN = 1;
```

```
delay(10);
EN = 0;
}

void lcd_data(char
a) {
P3 = a;
RS = 1;
EN = 1;
delay(10);
EN = 0;
}

void display(char
*ptr) {
while(*ptr != '\0')
{
lcd_data(*ptr);
ptr++;
}
}
void main()
{
unsigned char
value; char temp[4];
int i=0;
```

```
lcd_cmd(0x01); //clear screen
lcd_cmd(0x0E); //Display On, Cursor
Blinking lcd_cmd(0x38); //2 lines and 5*7
matrix
lcd_cmd(0x80); //Force Cursor to beginning of
first display("ADC Value=");
lcd_cmd(0xC0);
//mydata = 0xEF;
intr = 1;
rd = 1;
wr = 1;
while(1)
{
lcd_cmd(0xC0);
wr = 0;
wr = 1;
while(intr==1);
 rd=0;
value = mydata;

while(i<3)
{
temp[i] = (value%10) + '0'; // For converting into
ASCII value = value / 10;
i++;
}
```

```
for(i=2;i>=0;i--)
{
lcd_cmd(0x06);
lcd_data(temp[i]);
}
i=0;
delay(1000);
rd = 1;
} //Program_for_all
```

Output.



DRIVE LINK(video simulation)

https://drive.google.com/file/d/1J4LxQgVJVo9M_aEcPL1wPFIpmVUnJk3S/view?usp=sharing

DRIVE LINK(files)

https://drive.google.com/drive/folders/1nfXYJJkRycaGNtQe5DfhouKXekj5EE

cq?usp=sharing

Conclusion

Analog to digital conversion is a very important task in embedded electronics, as most of the sensors provide output as analog values and to feed them into microcontroller which only understand binary values, we have to convert them into Digital values. So, to be able to process the analog data, microcontrollers need **Analog to Digital Converter**.

Some microcontroller has inbuilt ADC like Arduino, MSP430, PIC16F877A but some microcontrollers don't have it like 8051, Raspberry Pi etc and we have to use some external Analog to digital converter ICs like ADC0804, ADC0808. Below you can find various examples of ADC with different microcontrollers.