DEPSTAR

Devang Patel Institute of Advance Technology and Research

(A Constitute Institute of CHARUSAT)

Certificate

This is to certify that

Mr./Mrs. Achyut Gopalbhai buddhe

of 3CSE-1-A *Class,*

ID. No. 23DCS005 *has satisfactorily completed*

his/ her term work in MPCO - CSE202 *for*

the ending in NOV *20*24*/20*25

Date : 15/10/24

Sign. of Faculty

Head of Department

CSE202 : Microprocessor and Computer Organization A.Y. : 2024-25

Charotar University of Science and Technology [CHARUSAT]

Devang Patel Institute of Advance Technology and Research [DEPSTAR]

Department of Computer Science & Engineering

Practical List

Subject code	:	CSE202	Semester	:	3	Academic Year	:	2024-25
Subject name	:	Microprocessor and Computer Organization						

Sr. No.	Aim	Hrs.	CO
1.	Implement a circuit in Logisim to display a given binary number in decimal on a seven segment display.	2	1
2.	Implement a circuit in Logisim which performs Addition and Subtraction of sign number.	2	1
3.	Implement a 4-bit common bus system to interface four 4-bit registers with a common bus using i. Multiplexer and ii. Decoder and tri-state buffers.	2	2
4.	Implement a circuit in Logisim which performs Arithmetic and Logic units.	4	2
5.	Implement a common bus system with ALU, 8 registers and 1 memory unit with necessary control signals.	2	2
6.	(Basics of assembly level programming) Perform following operations on 8-bit data Addition, and, Logical left shift, Rotate left with carry Subtraction, or, Logical right shift, Rotate left without carry Multiplication, xor, Arithmetic left shift, Rotate right with carry Division, not, Arithmetic right shift, Rotate right without carry	4	3
7.	(Array handling in Assembly level programming) Create an array. Perform addition of all even numbers from array and save answer in one variable.	2	3
8.	(String Handling in Assembly level language) Find out whether the given string is palindrome or not and print the appropriate message. Don't use procedure.	2	3
9.	(Procedure in Assembly Level Language) Write an assembly code to evaluate the answer of below given series and store the answer in the ANS variable. Program should have only one procedure to compute the factorial of a number. Series: 1! -2+3!-4+5!-6+7!-8	2	3
10.	Write a program that performs multiplication using the booth algorithm.	2	2
11.	Write a program to convert a given number system to another number system.	4	1
12.	Write an assembly level code for a given C program.	2	3

Prepared By:	Gaurang Patel	**Date:**	24/06/24

Date: 1/7/2024

EXPERIMENT NO. 1

AIM: Implement a circuit in logisim to display given decimal number in binary on to seven segment display.

TASK: i. display 0 to 9 on a seven segment display with binary number as input
ii. display 0 to 99 using two seven segment displays with binary number as input

TABLES OF CALCULATIONS:

1) 0 to 9:

A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	1	1	0	1	1	1	1
1	0	1	1	1	1	1	1	0	1	1
1	1	0	0	1	1	1	1	0	1	1
1	1	0	1	1	0	1	1	0	1	1
1	1	1	0	1	0	1	1	1	1	1
1	1	1	1	1	1	1	1	0	1	1

2) 0 to 99

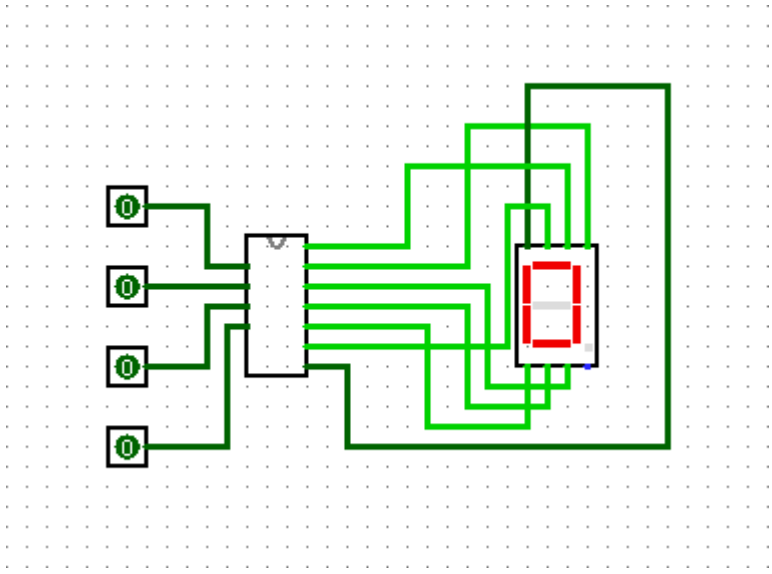
A	B	C	D	E	F	G	a	b	c	d	e	f	g	h
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	1	0	0	0	0	0	0	0	1	1
0	0	0	0	0	1	1	0	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
0	0	0	0	1	0	1	0	0	0	0	0	1	0	1
0	0	0	0	1	1	0	0	0	0	0	0	1	1	0
0	0	0	0	1	1	1	0	0	0	0	0	1	1	1
0	0	0	1	0	0	0	0	0	0	0	1	0	0	0
0	0	0	1	0	0	1	0	0	0	0	1	0	0	1
0	0	0	1	0	1	0	0	0	0	1	0	0	0	0
0	0	0	1	0	1	1	0	0	0	1	0	0	0	1
0	0	0	1	1	0	0	0	0	0	1	0	0	1	1
0	0	0	1	1	0	1	0	0	0	1	0	0	1	1
0	0	0	1	1	1	0	0	0	0	1	0	1	0	0
0	0	0	1	1	1	1	0	0	0	1	0	1	0	1
0	0	1	0	0	0	0	0	0	0	1	0	1	1	0
0	0	1	0	0	0	1	0	0	0	1	0	1	1	1
0	0	1	0	0	1	0	0	0	0	1	1	0	0	0
0	0	1	0	0	1	1	0	0	0	1	1	0	0	1
0	0	1	0	1	0	0	0	0	1	0	0	0	0	0
0	0	1	0	1	0	1	0	0	1	0	0	0	0	1
0	0	1	0	1	1	0	0	0	1	0	0	0	1	1
0	0	1	0	1	1	1	0	0	1	0	0	0	1	1
0	0	1	1	0	0	0	0	0	1	0	0	1	0	0
0	0	1	1	0	0	1	0	0	1	0	0	1	0	1
0	0	1	1	0	1	0	0	0	1	0	0	1	1	0
0	0	1	1	0	1	1	0	0	1	0	0	1	1	1
0	0	1	1	1	0	0	0	0	1	0	1	0	0	0
0	0	1	1	1	0	1	0	0	1	0	1	0	0	1
0	0	1	1	1	1	0	0	0	1	1	0	0	0	0
0	0	1	1	1	1	1	0	0	1	1	0	0	0	1

0	1	0	0	0	0	0	0	0	1	1	0	0	1	1
0	1	0	0	0	0	1	0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0	0	1	1	0	1	0	0
0	1	0	0	0	1	1	0	0	1	1	0	1	0	1
0	1	0	0	1	0	0	0	0	1	1	0	1	1	0
0	1	0	0	1	0	1	0	0	1	1	0	1	1	1
0	1	0	0	1	1	0	0	0	1	1	1	0	0	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1
0	1	0	1	0	0	0	0	1	0	0	0	0	0	0
0	1	0	1	0	0	1	0	1	0	0	0	0	0	1
0	1	0	1	0	1	0	0	1	0	0	0	0	1	1
0	1	0	1	0	1	1	0	1	0	0	0	0	1	1
0	1	0	1	1	0	0	0	1	0	0	0	1	0	0
0	1	0	1	1	0	1	0	1	0	0	0	1	0	1
0	1	0	1	1	1	0	0	1	0	0	0	1	1	0
0	1	0	1	1	1	1	0	1	0	0	0	1	1	1
0	1	1	0	0	0	0	0	1	0	0	1	0	0	0
0	1	1	0	0	0	1	0	1	0	0	1	0	0	1
0	1	1	0	0	1	0	0	1	0	1	0	0	0	0
0	1	1	0	0	1	1	0	1	0	1	0	0	0	1
0	1	1	0	1	0	0	0	1	0	1	0	0	1	1
0	1	1	0	1	0	1	0	1	0	1	0	0	1	1
0	1	1	0	1	1	0	0	1	0	1	0	1	0	0
0	1	1	0	1	1	1	0	1	0	1	0	1	0	1
0	1	1	1	0	0	0	0	1	0	1	0	1	1	0
0	1	1	1	0	0	1	0	1	0	1	0	1	1	1
0	1	1	1	0	1	0	0	1	0	1	1	0	0	0
0	1	1	1	0	1	1	0	1	0	1	1	0	0	1
0	1	1	1	1	0	0	0	1	1	0	0	0	0	0
0	1	1	1	1	0	1	0	1	1	0	0	0	0	1
0	1	1	1	1	1	0	0	1	1	0	0	0	1	1
0	1	1	1	1	1	1	0	1	1	0	0	0	1	1
1	0	0	0	0	0	0	0	1	1	0	0	1	0	0

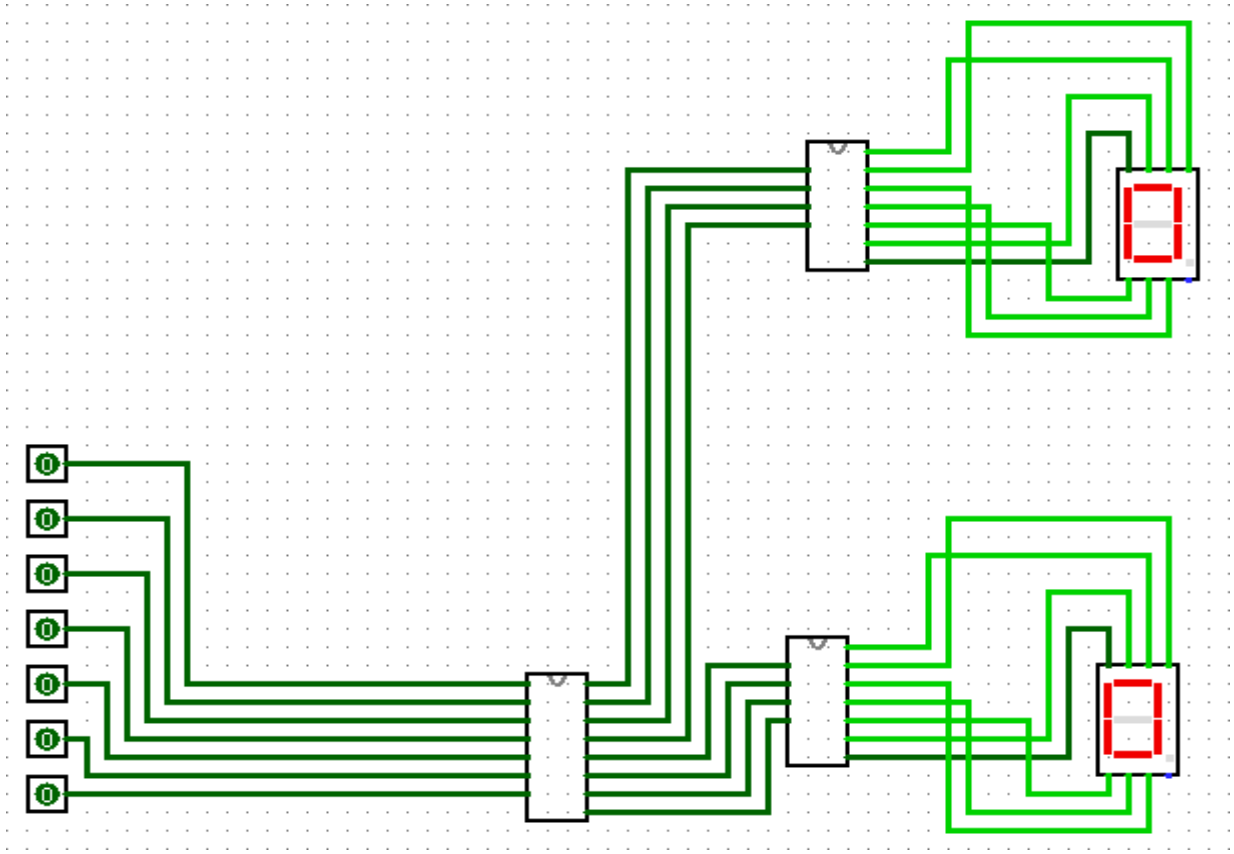
1	0	0	0	0	0	1	0	1	1	0	0	1	0	1
1	0	0	0	0	1	0	0	1	1	0	0	1	1	0
1	0	0	0	0	1	1	0	1	1	0	0	1	1	1
1	0	0	0	1	0	0	0	1	1	0	1	0	0	0
1	0	0	0	1	0	1	0	1	1	0	1	0	0	1
1	0	0	0	1	1	0	0	1	1	1	0	0	0	0
1	0	0	0	1	1	1	0	1	1	1	0	0	0	1
1	0	0	1	0	0	0	0	1	1	1	0	0	1	1
1	0	0	1	0	0	1	0	1	1	1	0	0	1	1
1	0	0	1	0	1	0	0	1	1	1	0	1	0	0
1	0	0	1	0	1	1	0	1	1	1	0	1	0	1
1	0	0	1	1	0	0	0	1	1	1	0	1	1	0
1	0	0	1	1	0	1	0	1	1	1	0	1	1	1
1	0	0	1	1	1	0	0	1	1	1	1	0	0	0
1	0	0	1	1	1	1	0	1	1	1	1	0	0	1
1	0	1	0	0	0	0	1	0	0	0	0	0	0	0
1	0	1	0	0	0	1	1	0	0	0	0	0	0	1
1	0	1	0	0	1	0	1	0	0	0	0	0	1	1
1	0	1	0	0	1	1	1	0	0	0	0	0	1	1
1	0	1	0	1	0	0	1	0	0	0	0	1	0	0
1	0	1	0	1	0	1	1	0	0	0	0	1	0	1
1	0	1	0	1	1	0	1	0	0	0	0	1	1	0
1	0	1	0	1	1	1	1	0	0	0	0	1	1	1
1	0	1	1	0	0	0	1	0	0	0	1	0	0	0
1	0	1	1	0	0	1	1	0	0	0	1	0	0	1
1	0	1	1	0	1	0	1	0	0	1	0	0	0	0
1	0	1	1	0	1	1	1	0	0	1	0	0	0	1
1	0	1	1	1	0	0	1	0	0	1	0	0	1	1
1	0	1	1	1	0	1	1	0	0	1	0	0	1	1
1	0	1	1	1	1	0	1	0	0	1	0	1	0	0
1	0	1	1	1	1	1	1	0	0	1	0	1	0	1
1	1	0	0	0	0	0	1	0	0	1	0	1	1	0
1	1	0	0	0	0	1	1	0	0	1	0	1	1	1
1	1	0	0	0	1	0	1	0	0	1	1	0	0	0
1	1	0	0	0	1	1	1	0	0	1	1	0	0	1

CIRCUITS:

0 to 9:

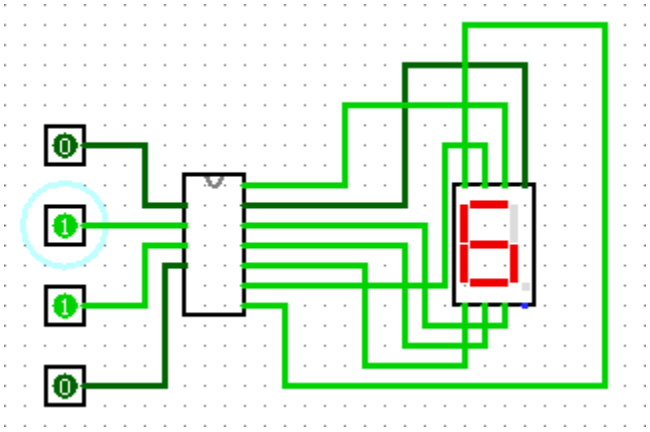


0 to 99:

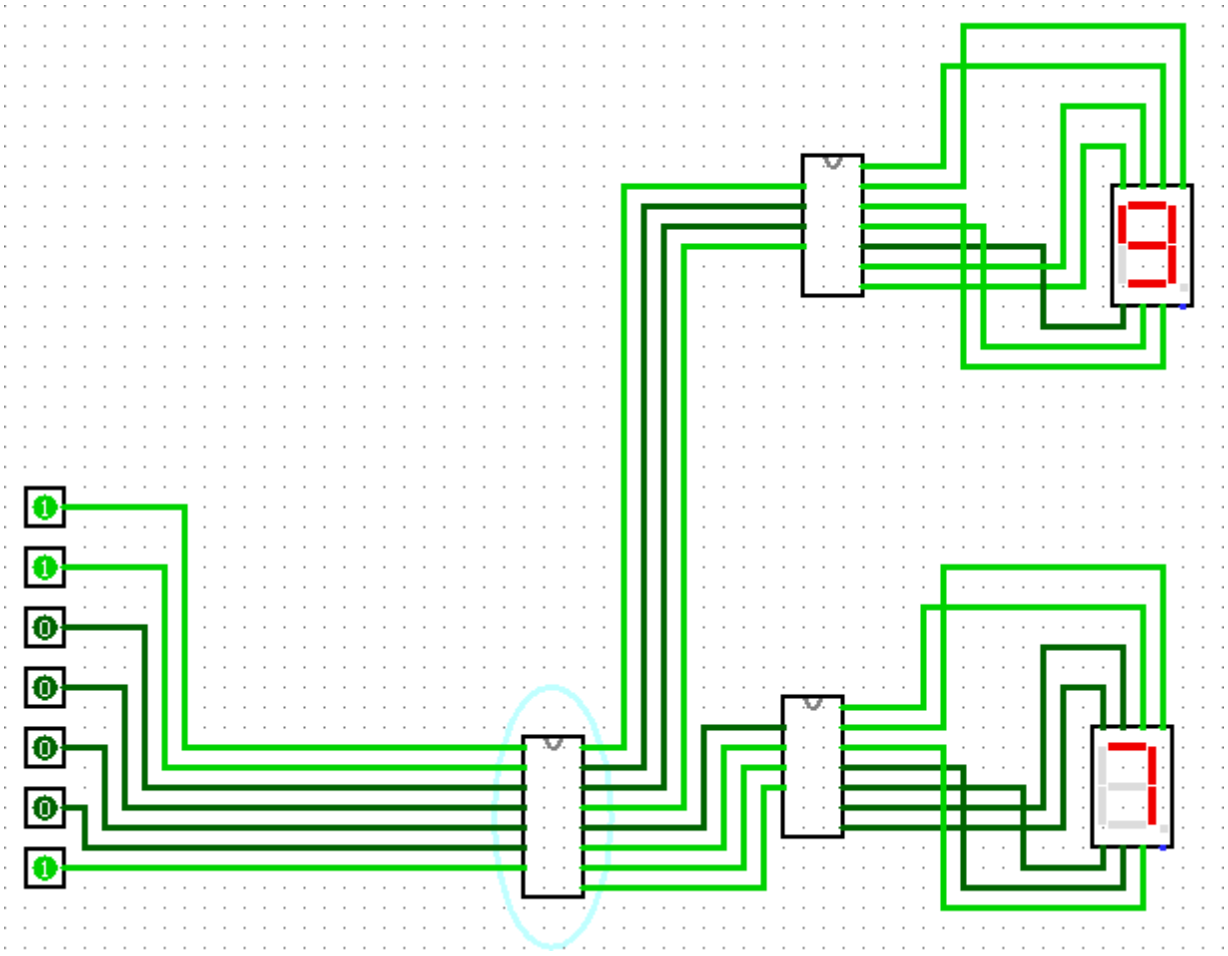


OUTPUTS:

0 to 9:



0 to 99:



CONCLUSION:

From this experiment we can conclude that:

i. We can design a seven segment display with binary number as input.

ii. We can also design two seven segment displays with binary number as input.

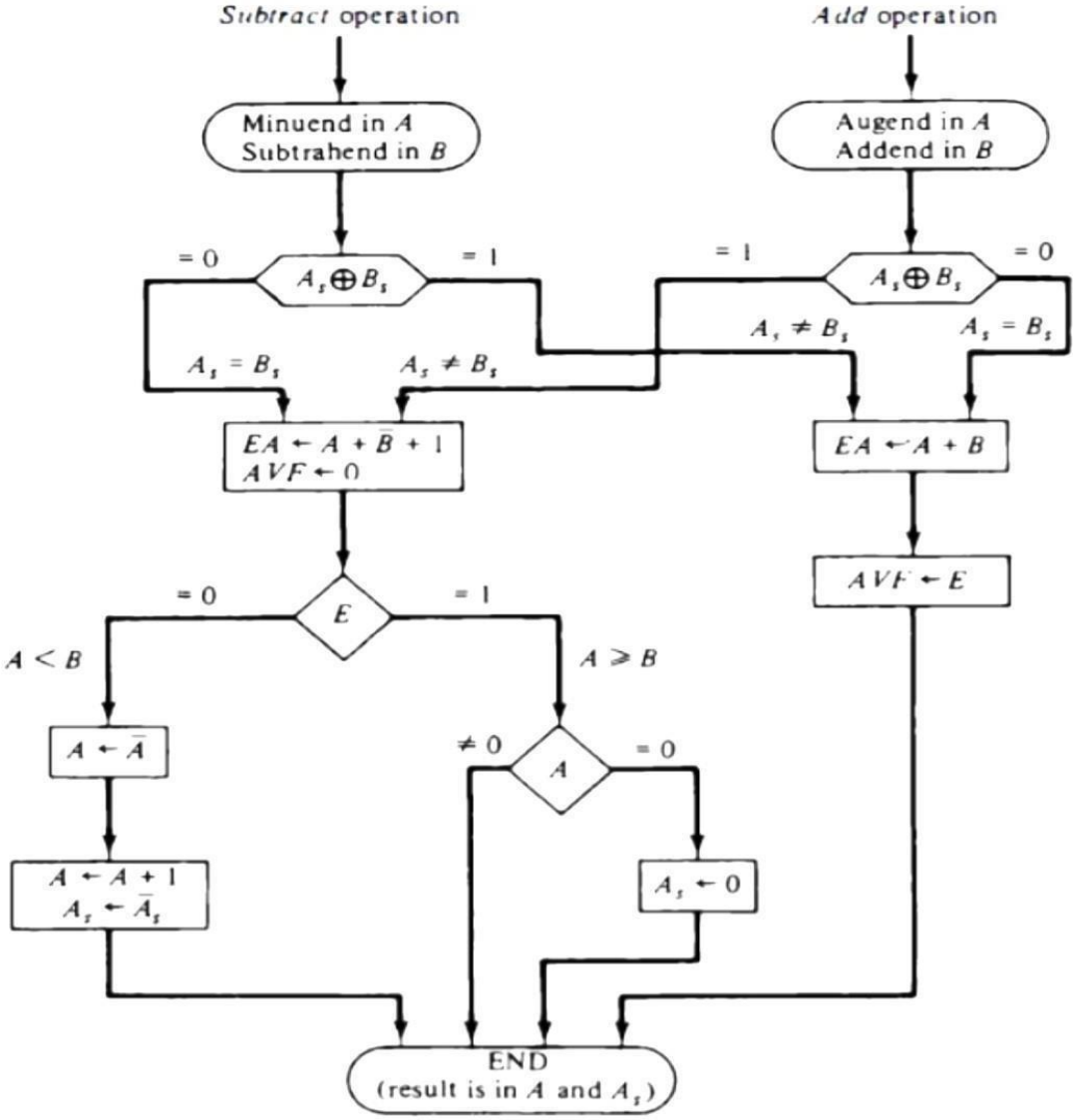
iii. And we can create 0 to 9 and 0 to 99 counter based on seven segment application.

Date:8/7/2024

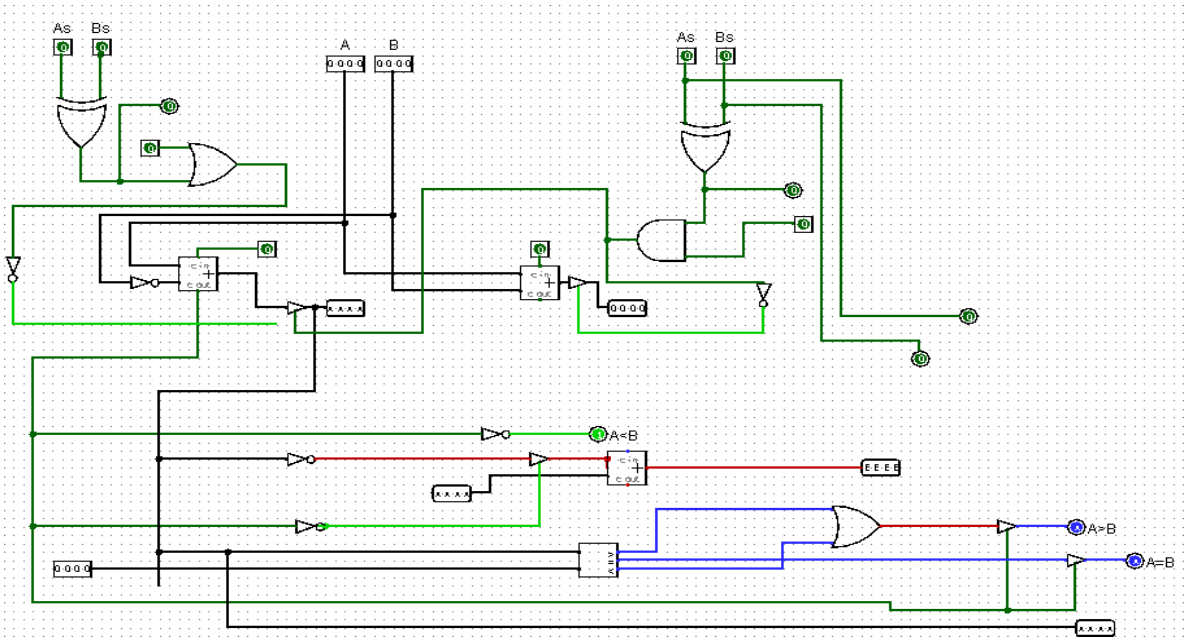
EXPERIMENT NO. 2

AIM: Implement a circuit in Logisim which perform Addition and Subtraction of sign number.

ALGORITHM:

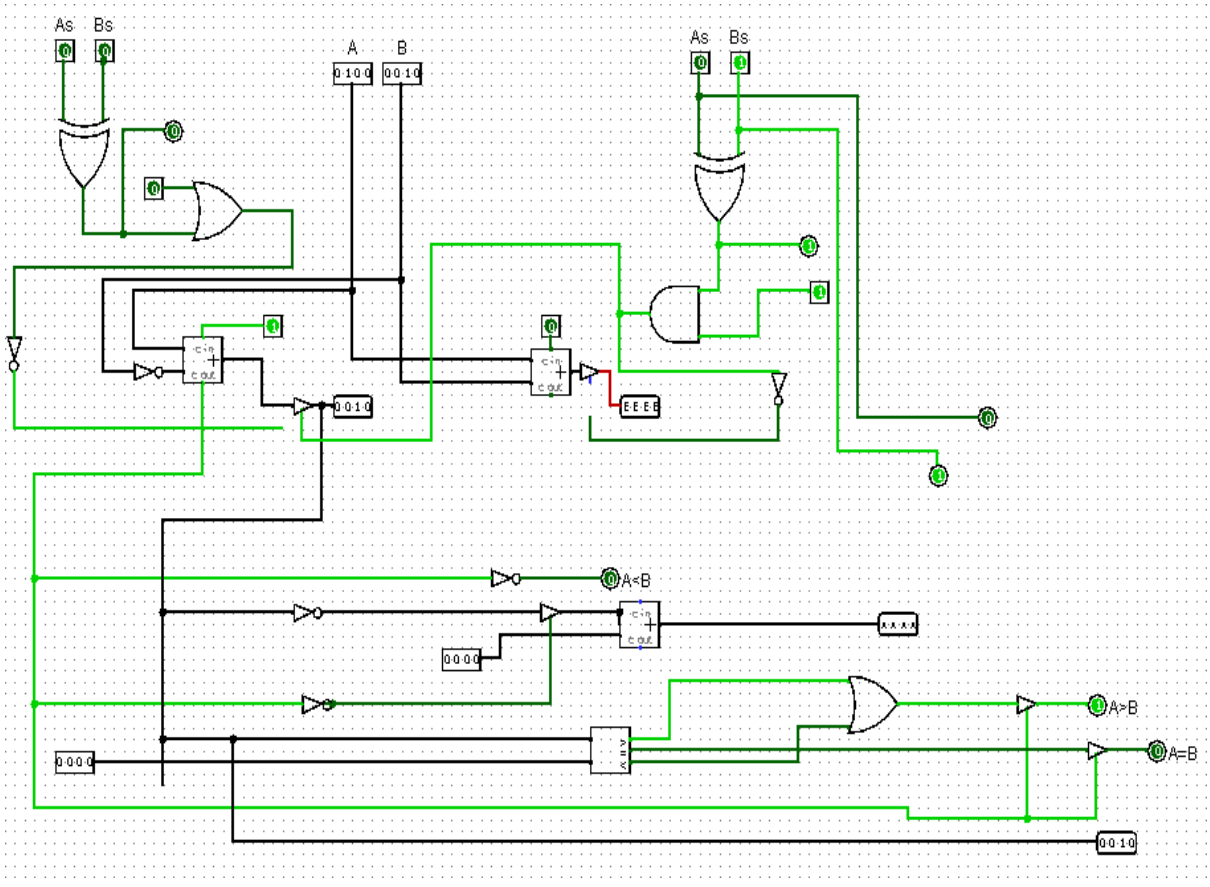


CIRCUITS:

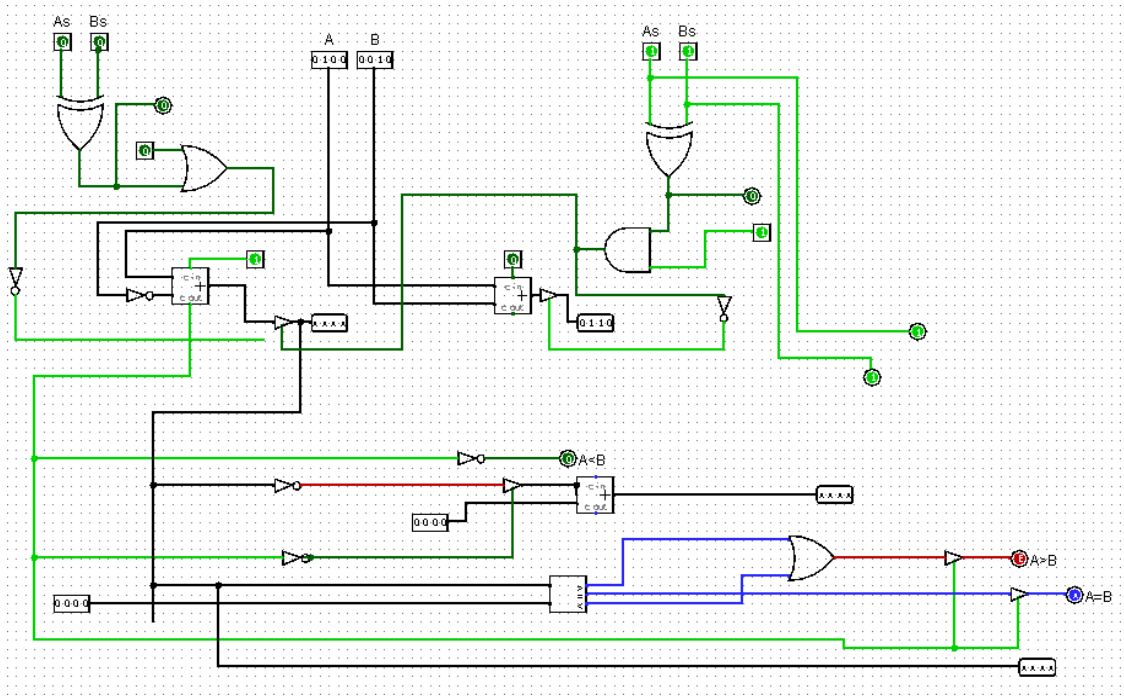


OUTPUTS:

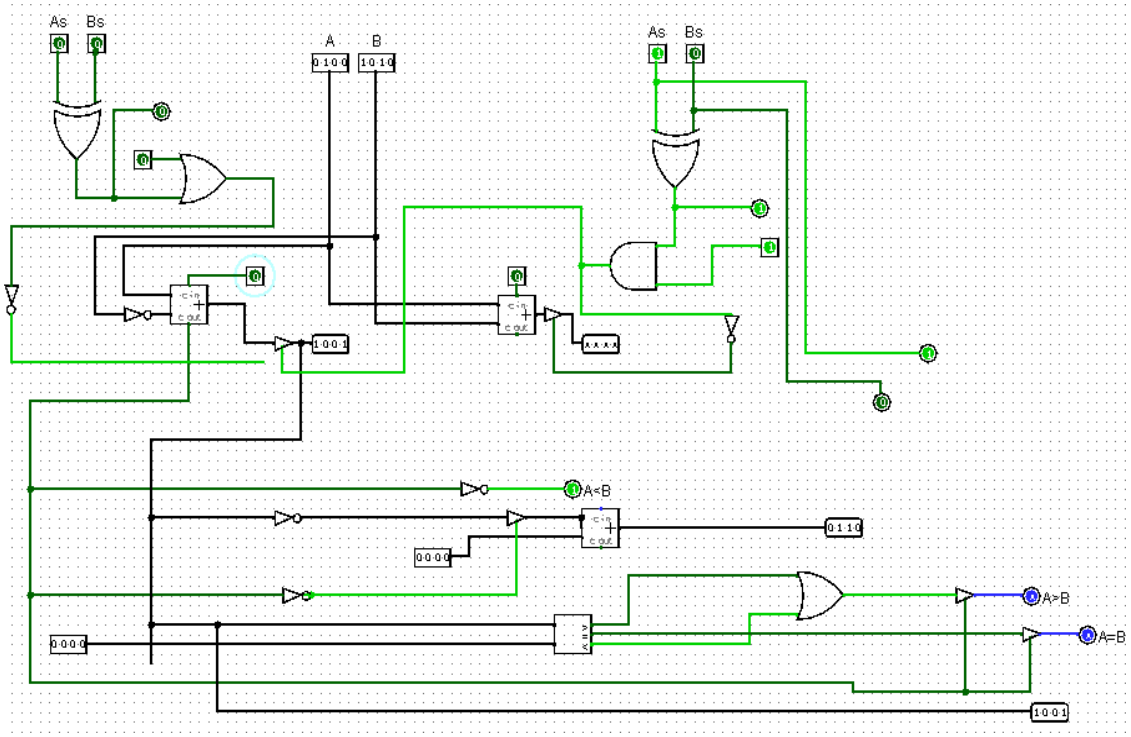
1) -(A+B)



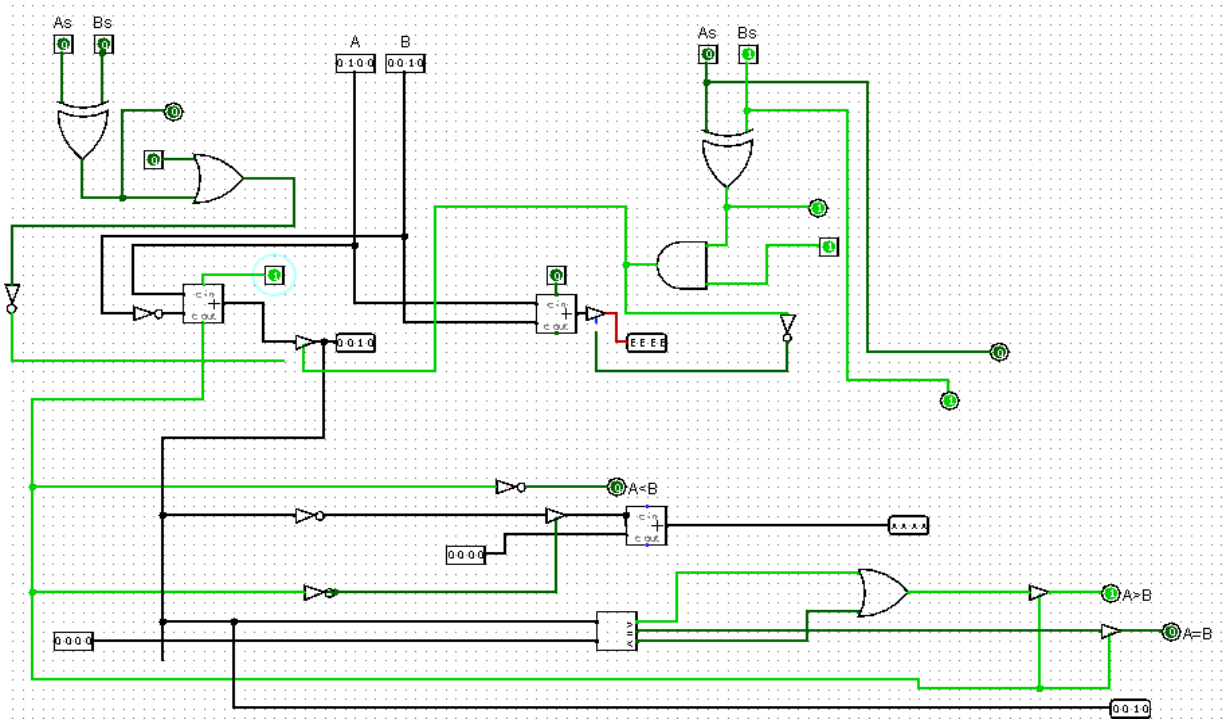
2)(-A+(-B))= -A-B


As
Bs
A
B
0100
0010
As
Bs
0110
A<B
0000
A>B
A=B
0000
XXXX

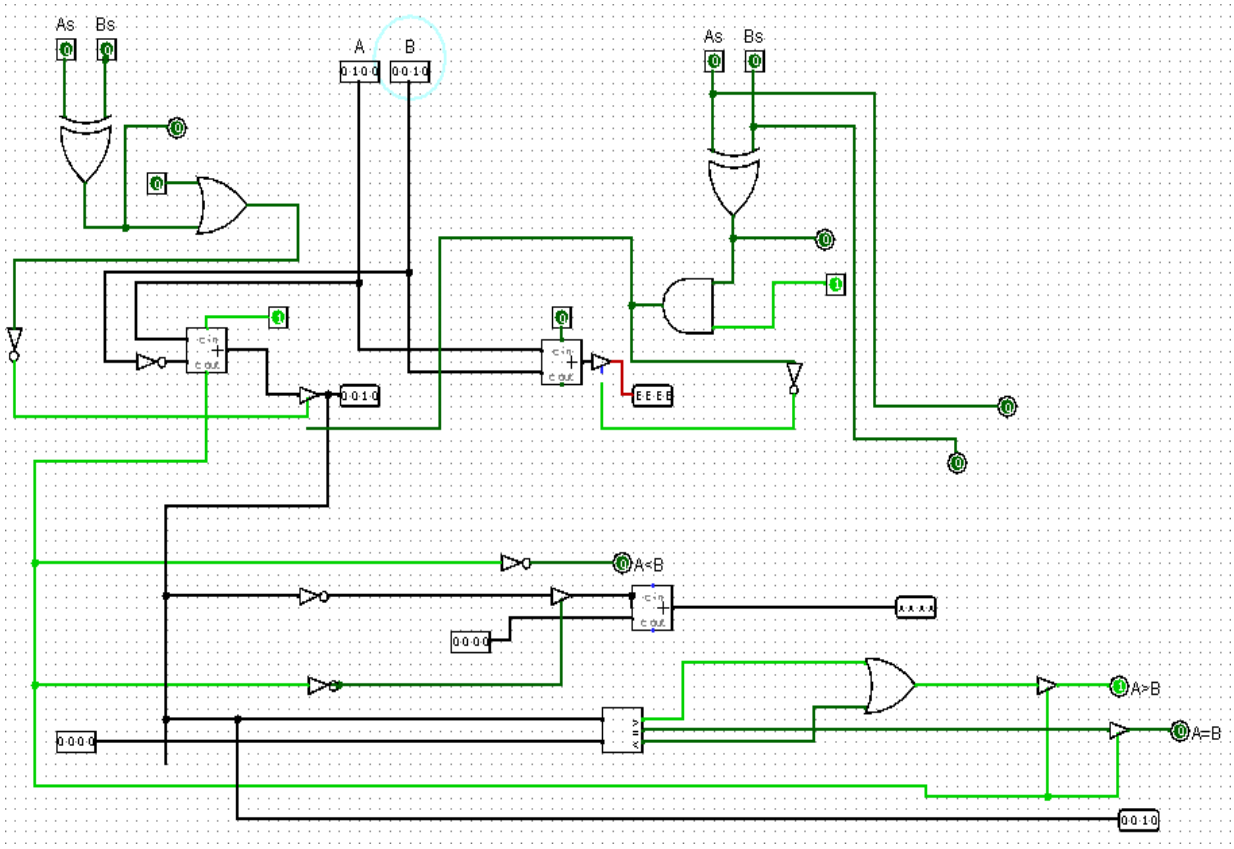
3)(-A+B)


As
Bs
A
B
0100
1010
As
Bs
1001
A<B
0000
0110
A>B
A=B
0000
1001

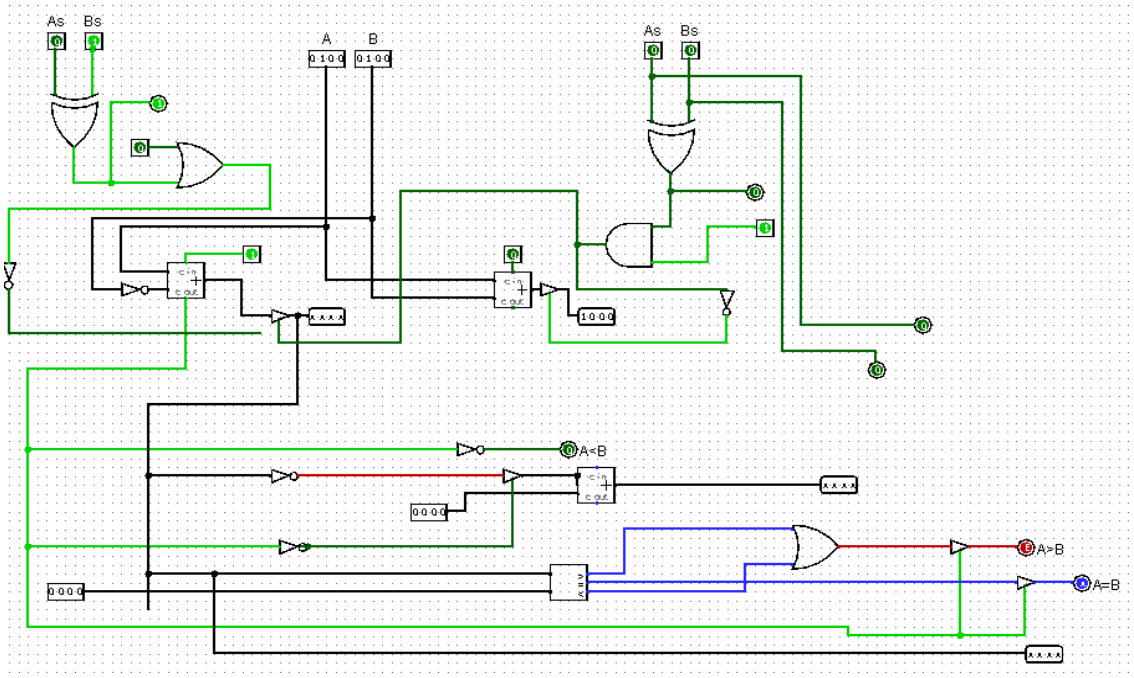
4)(A-(-B))=A+B


As
Bs
A
B
0·1·0·0
0·0·1·0
0·0·1·0
E·E·E·E
A<B
0·0·0·0
A>B
A=B
0·0·1·0

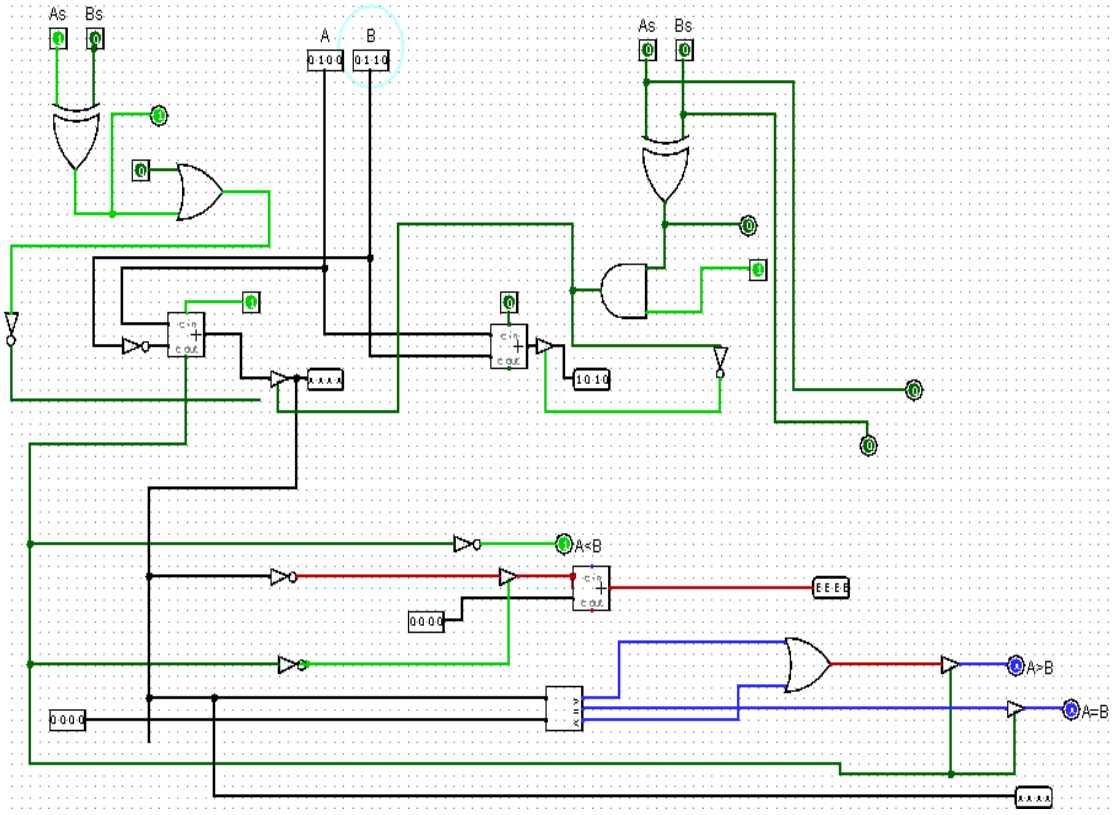
5) (A-B)


As
Bs
A
B
0·1·0·0
0·0·1·0
0·0·1·0
E·E·E·E
A<B
0·0·0·0
A>B
A=B
0·0·1·0

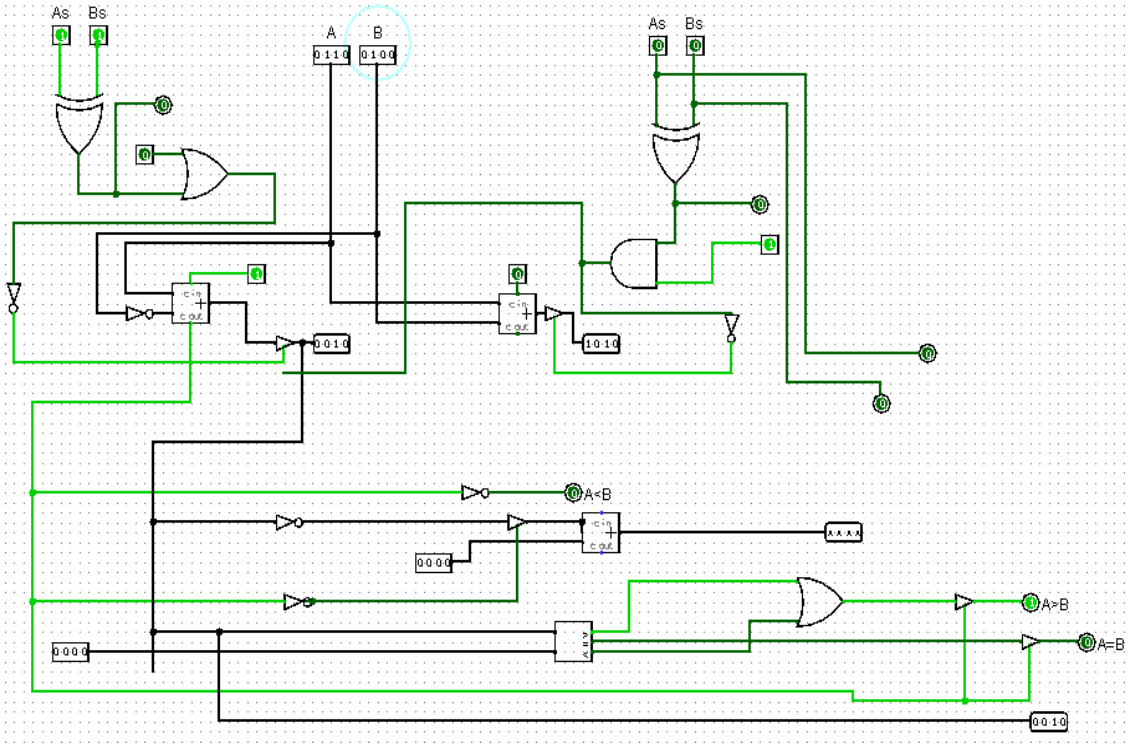
6) (A-(-B))


As
Bs
A
B
0100
0100
As
Bs
1000
A<B
0000
A>B
A=B
0000

7)(-A-B)


As
Bs
A
B
0100
0110
As
Bs
1010
A<B
EEEE
0000
A>B
A=B
0000

8)(-A-(-B))



C

CONCLUSION:

In Logisim, implementing a circuit for the addition and subtraction of signed numbers demonstrates fundamental concepts of digital arithmetic and signed number representation. The circuit effectively handles two's complement arithmetic, ensuring correct results for both positive and negative operands. This practical exercise enhances understanding of binary operations, logic gates, and circuit design principles, reinforcing the core concepts of digital electronics and computer architecture.

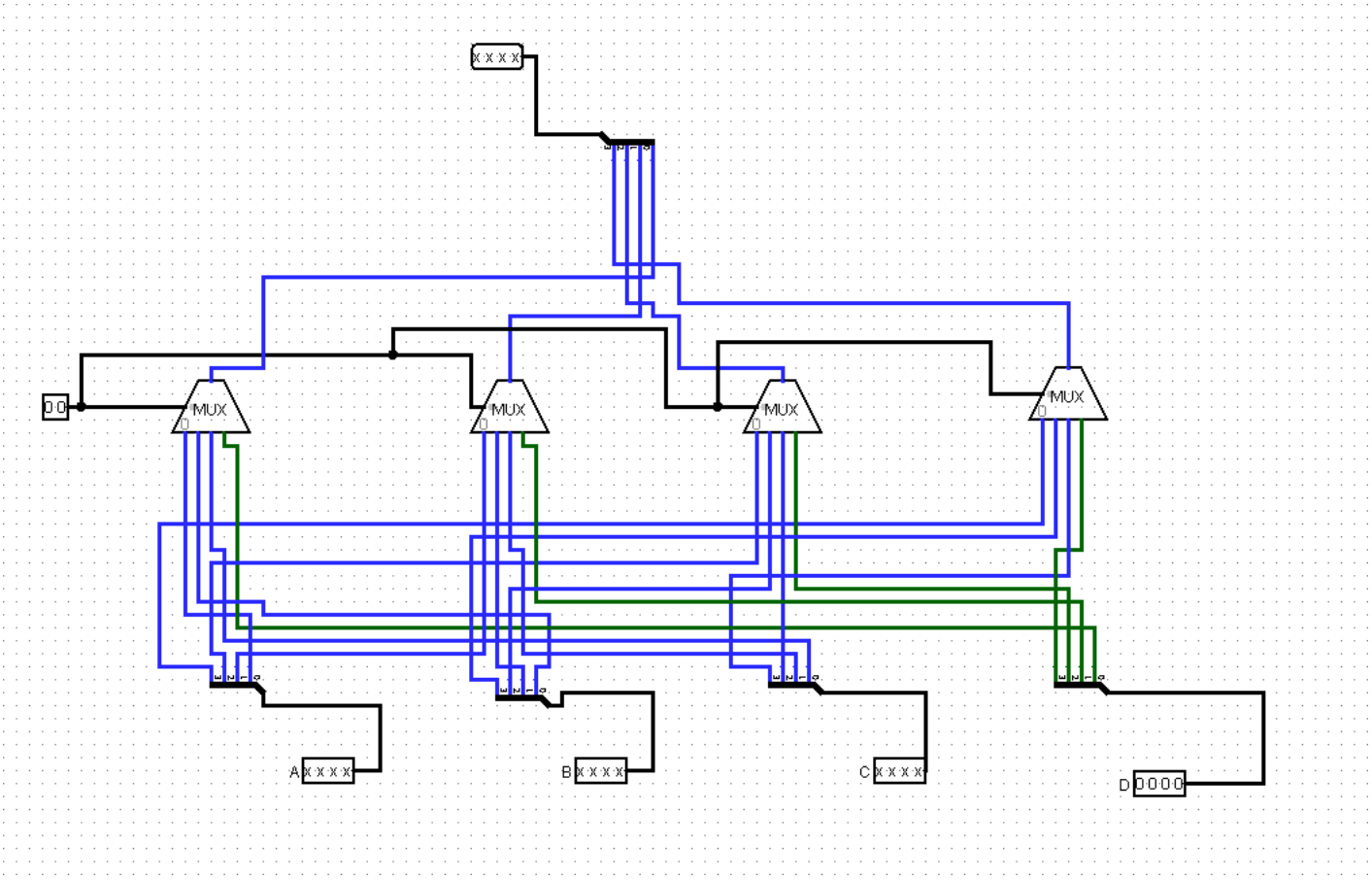
Date: 15/7/2024

EXPERIMENT NO. 3

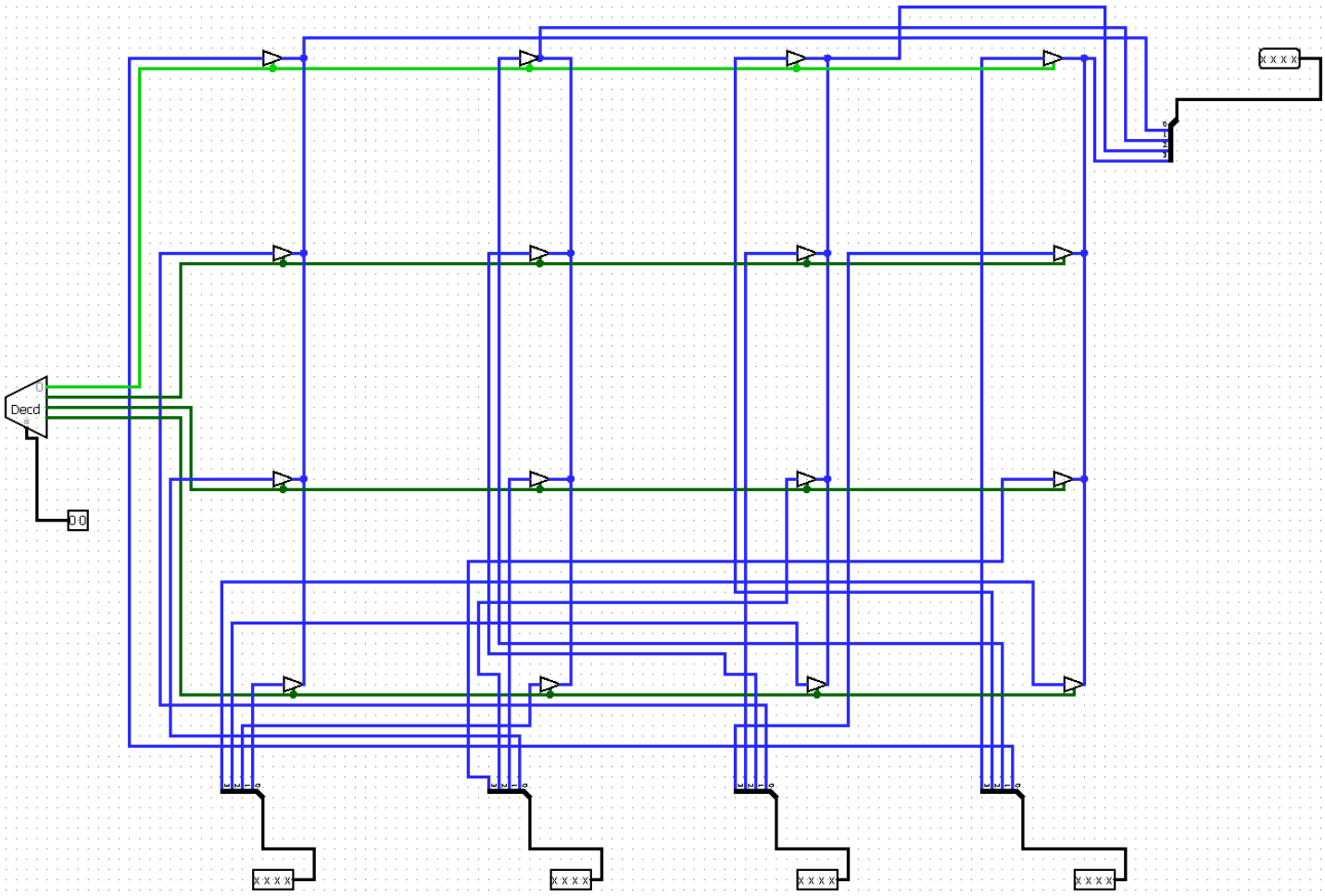
AIM: Implement a 4-bit common bus system to interface four 4-bit registers with a common bus using i. Multiplexer and ii. Decoder and tristate buffers.

CIRCUITS:

i. USING MULTIPLEXER

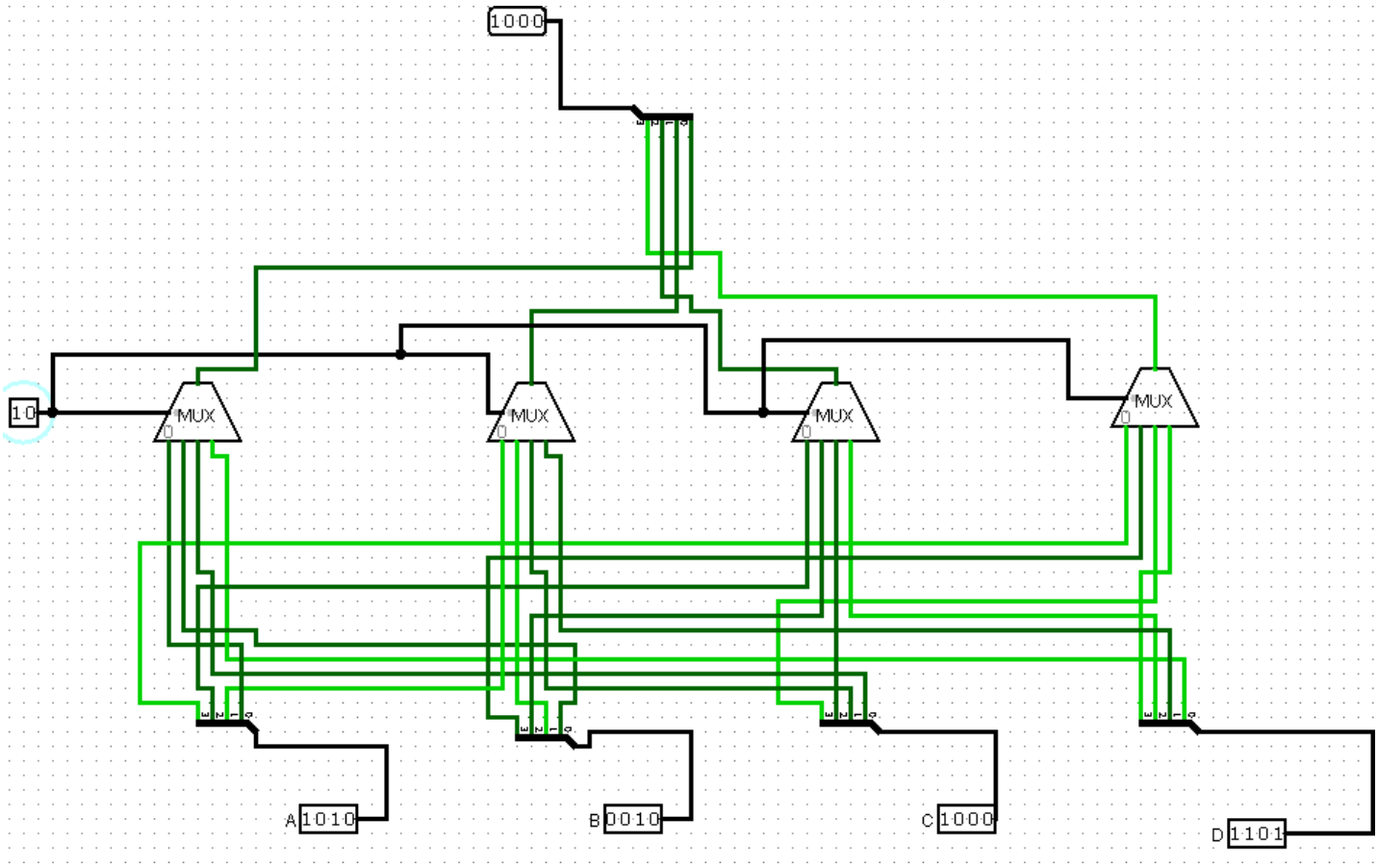


ii. USING DECODER AND TRISTATE BUFFERS.

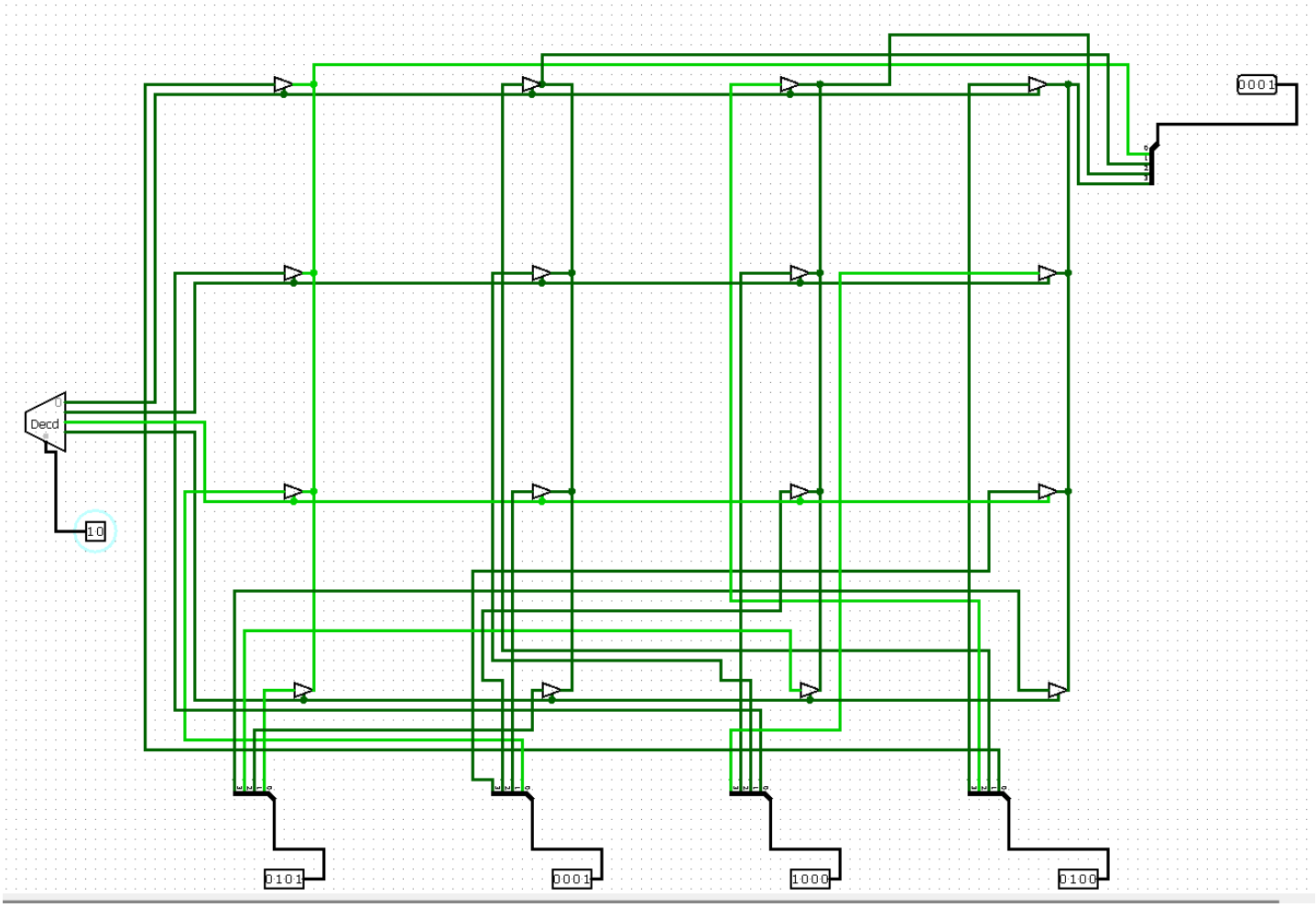


OUTPUTS:

i. USING MULTIPLEXER



ii. USING DECODER AND TRISTATE BUFFERS.



CONCLUSION:

Implementing a 4-bit common bus system using multiplexers and decoder with tri-state buffers proved effective in interfacing four 4-bit registers. The multiplexer method efficiently selects and routes data between registers and the bus, offering simplicity in control logic. Conversely, the decoder and tri-state buffers method provided straightforward enable/disable functionality for each register's connection to the bus, ensuring data integrity and minimal bus contention. Both approaches demonstrate practical solutions for interfacing multiple registers with a common bus in digital systems design.

Date: 02/08/2024

EXPERIMENT NO. 4

AIM: Implement arithmetic and logic unit circuits in Logisim.

OBJECTIVES:

i. Implement 1-bit, 2-bit, 4-bit and 8-bit arithmetic unit circuits

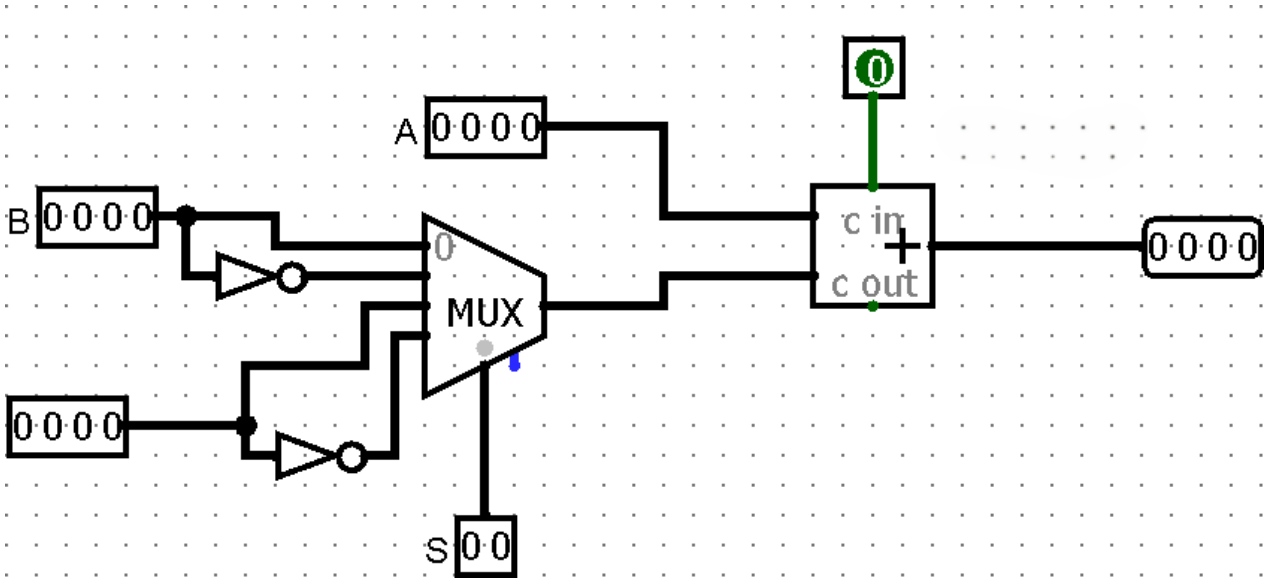
ii. Implement 1-bit, 2-bit, 4-bit and 8-bit logical unit circuits for four logical functions

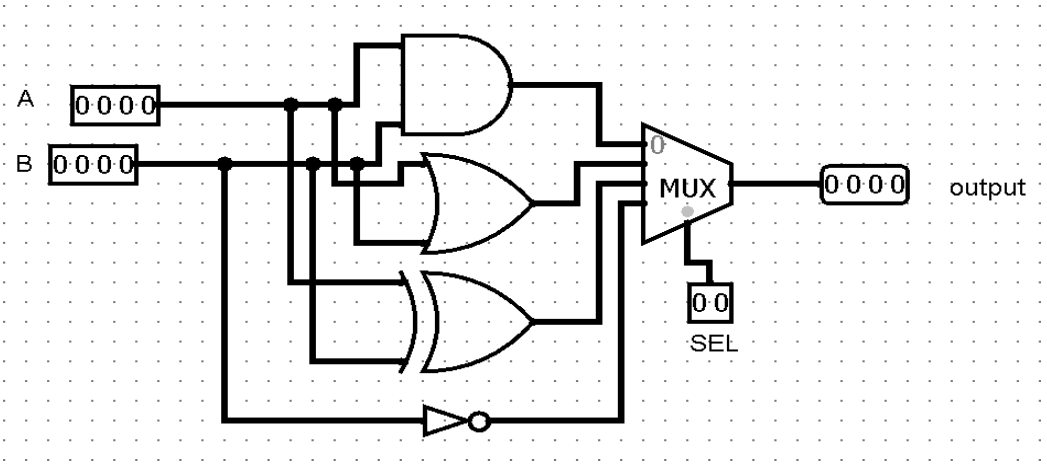
iii.Implement 1-bit and 2-bit logical unit circuits for sixteen logical functions

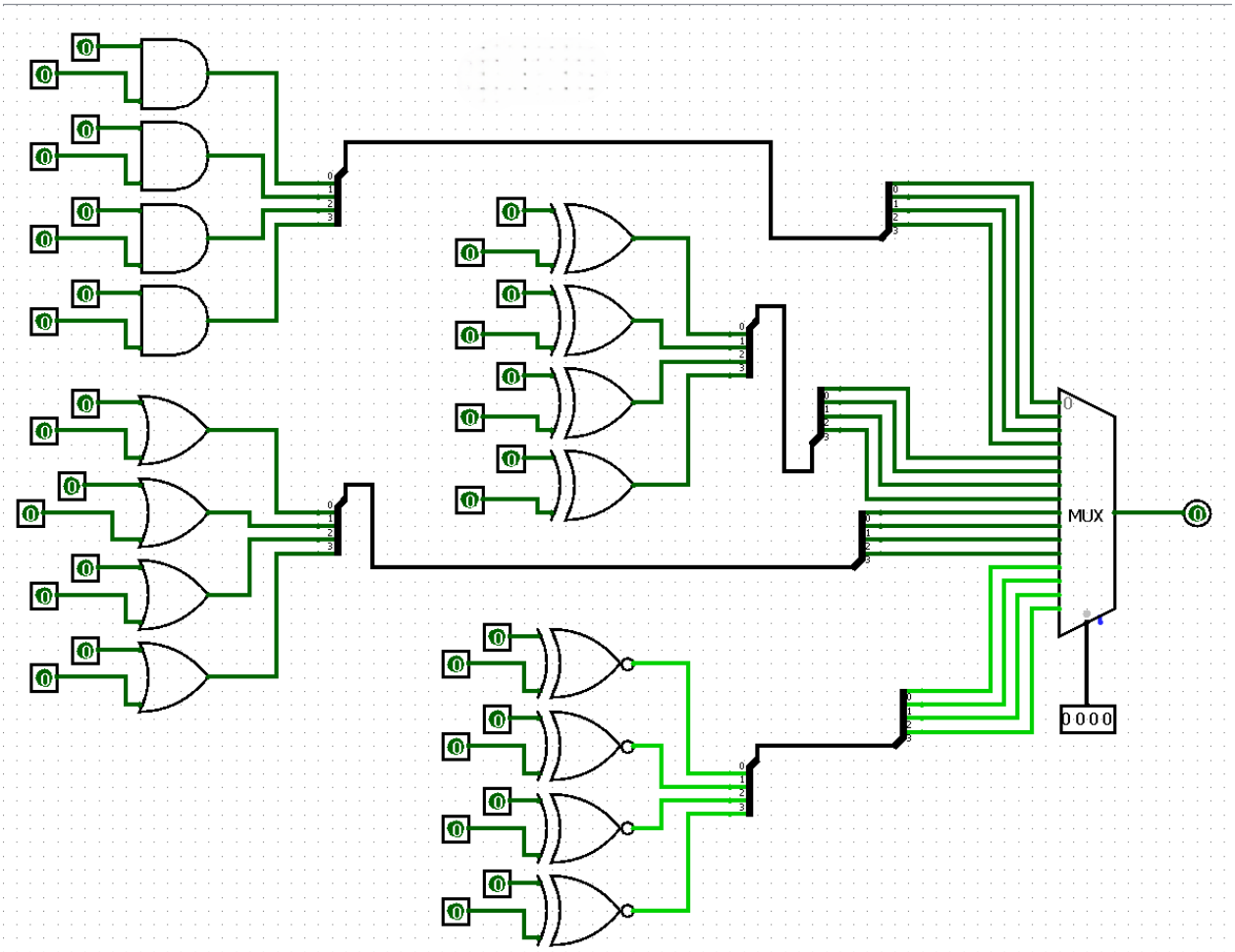
iv. Implement 2-bit, 4-bit and 8-bit bidirectional shifter

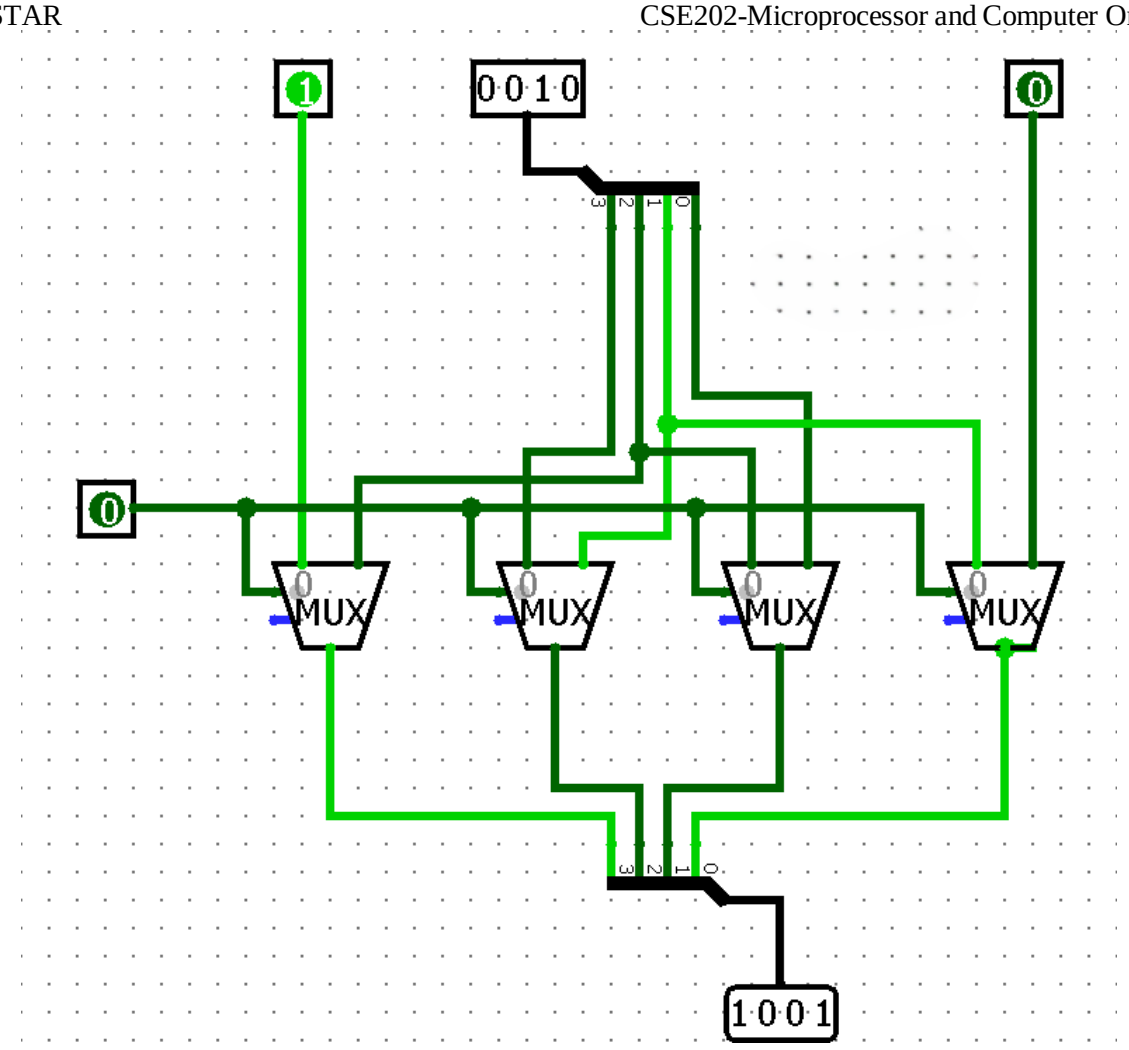
v. Implement 1-bit, 2-bit, 4-bit and 8-bit ALU

CIRCUITS:



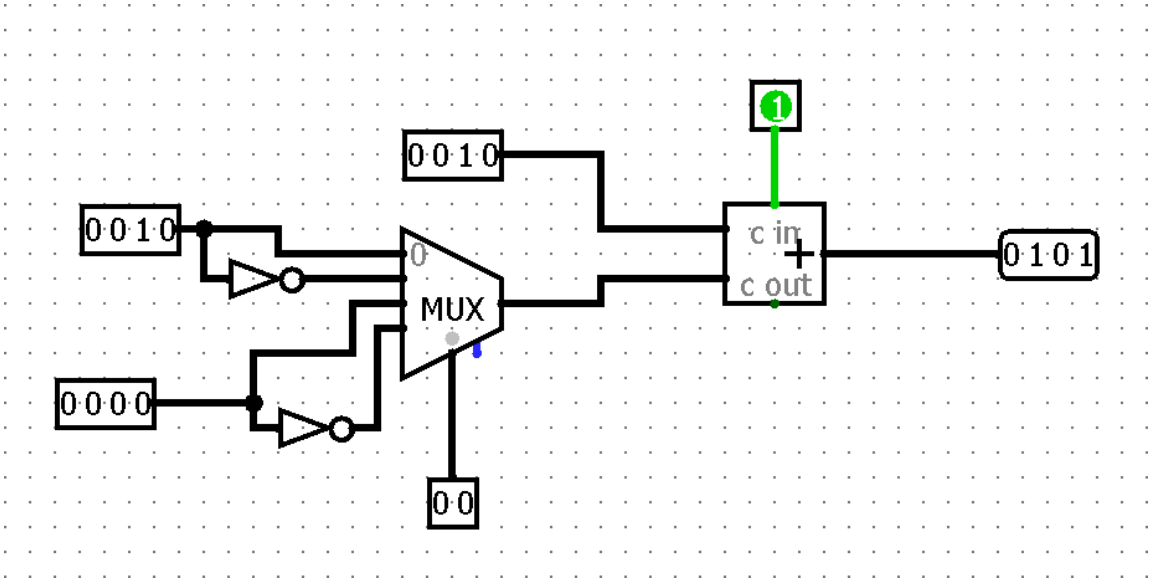

A
0 0 0 0
B
0 0 0 0
MUX
0 0 0 0
output
0 0
SEL


MUX
0 0 0 0

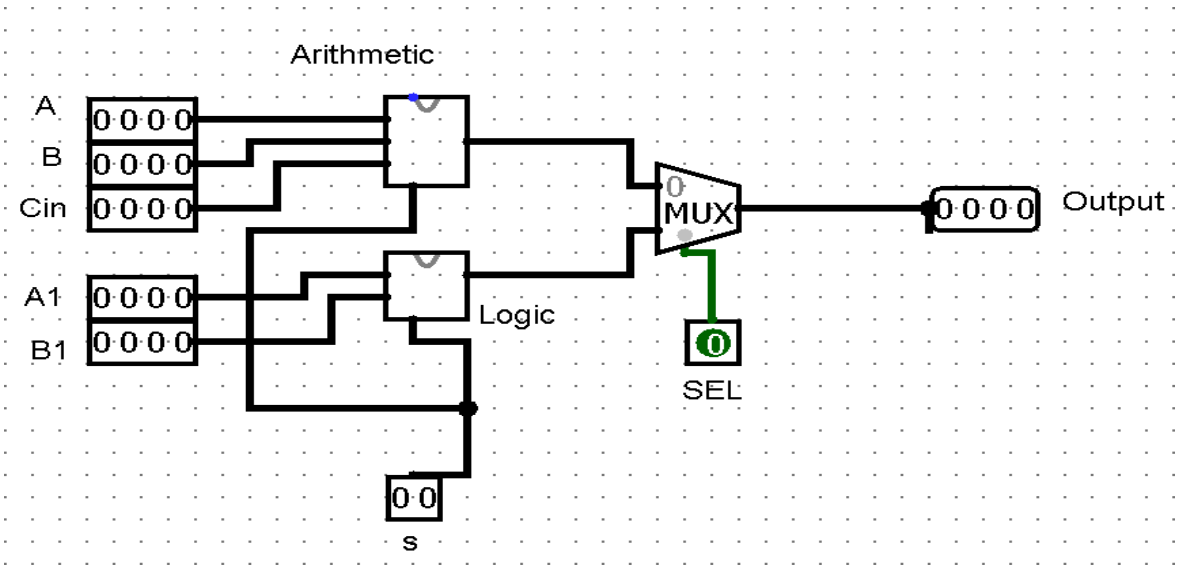

1
0010
0
3 2 1 0
0
MUX
MUX
MUX
MUX
3 2 1 0
1001

OUTPUTS:

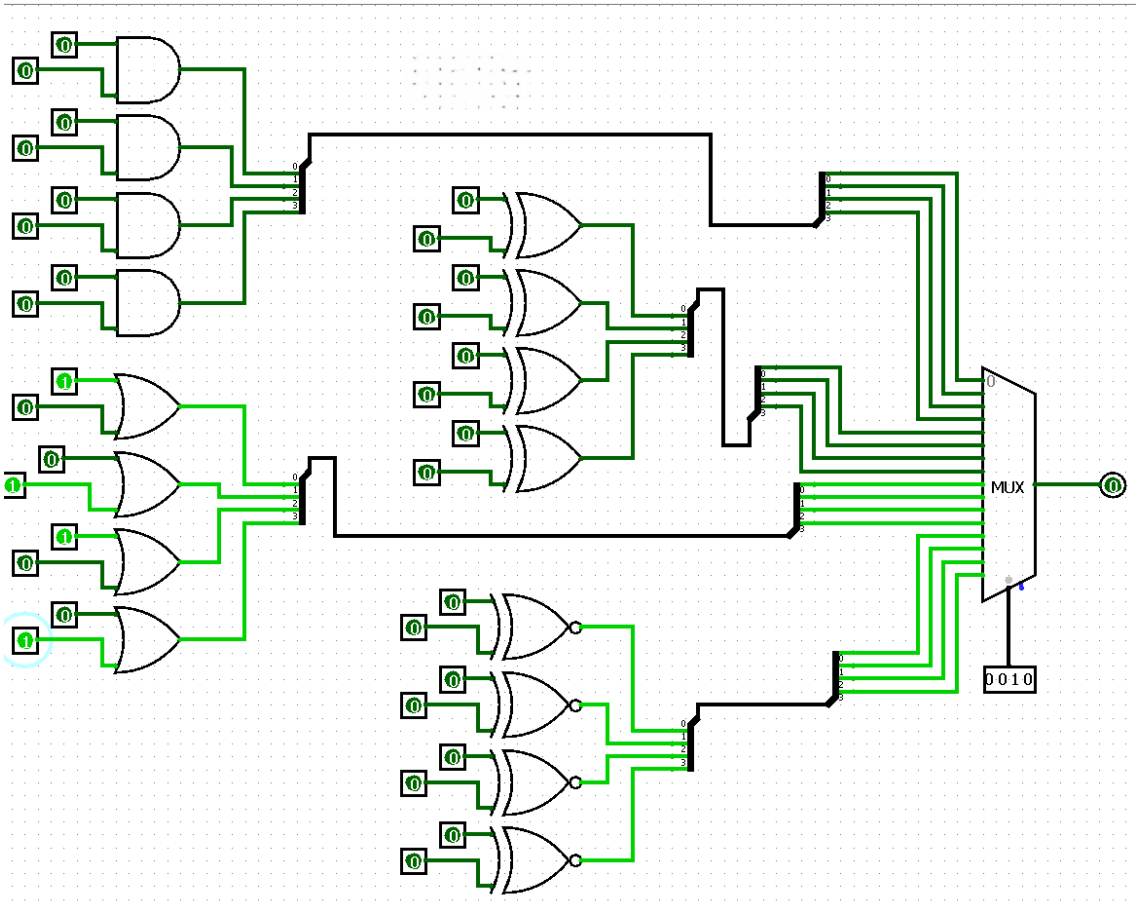
(1)


1
0010
0010
0
MUX
c in
c out
0101
0000
00

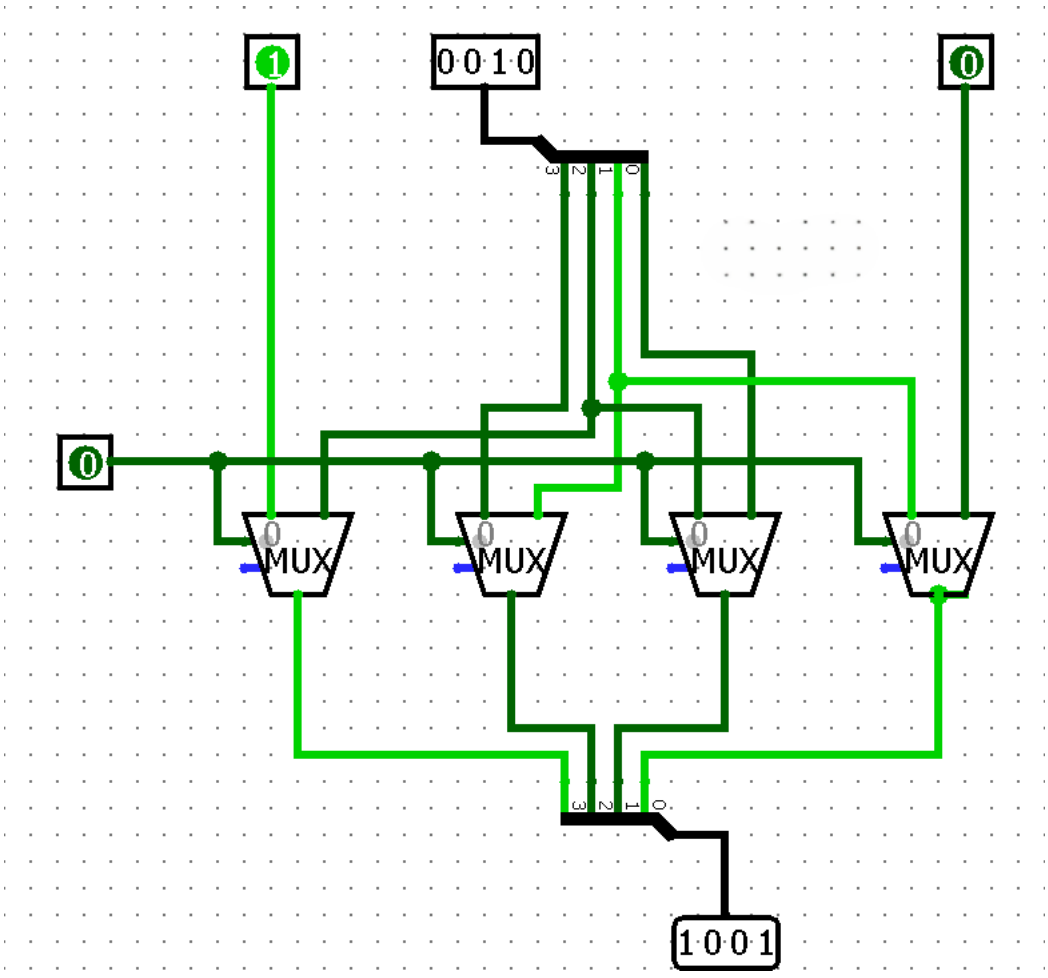
(2)


Arithmetic
A
B
Cin
0 0 0 0
0 0 0 0
0 0 0 0
A1
B1
0 0 0 0
0 0 0 0
Logic
0
MUX
0 0 0 0
Output
0
SEL
0 0
s

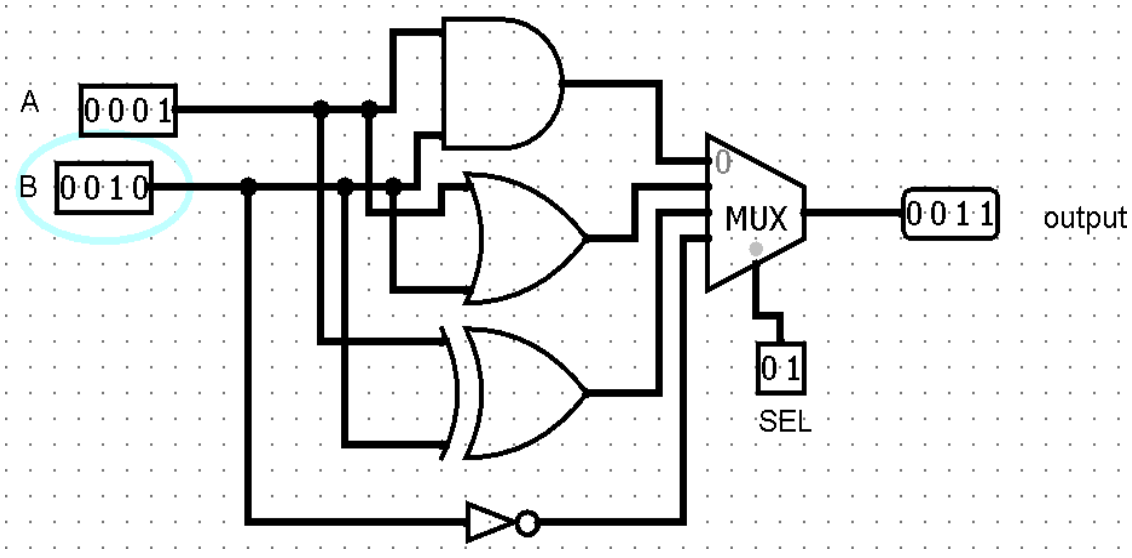
(3)


MUX
0 0 1 0

(4)


1
0010
0
0
MUX
MUX
MUX
MUX
1001

(5)


A
0001
B
0010
MUX
0011
output
01
SEL

CONCLUSION:

Implementing arithmetic and logic unit circuits in Logisim involves designing and simulating digital circuits to perform arithmetic and logical operations, enhancingunderstanding of fundamental computer architecture principles.

Date: 13/08/23

EXPERIMENT NO. 5

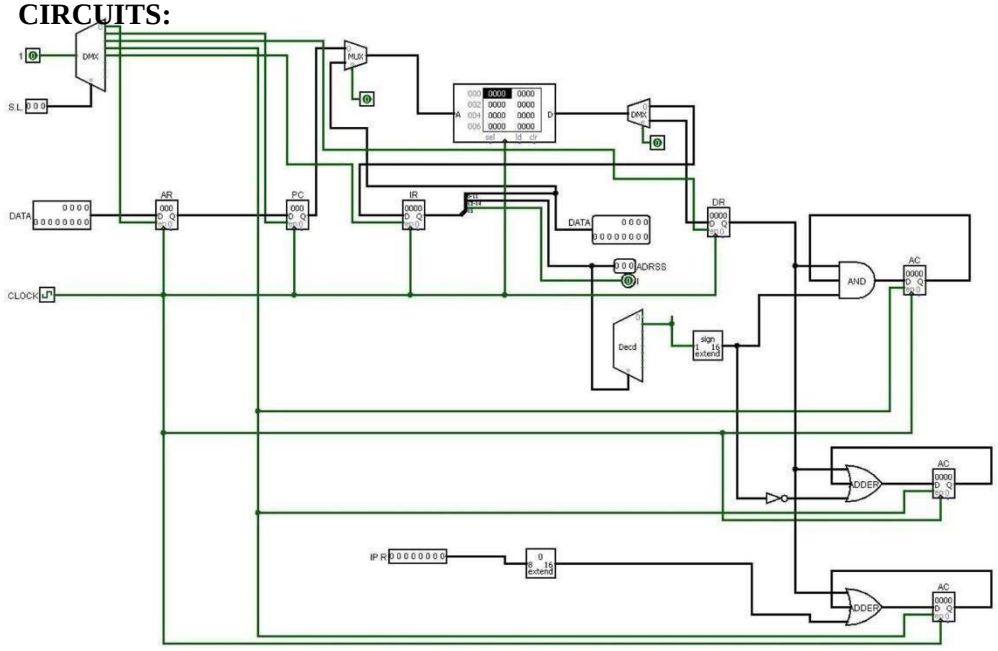
AIM: Implement a common bus system with ALU, 8 registers and 1 memory unit with necessary control signals.

OBJECTIVES:

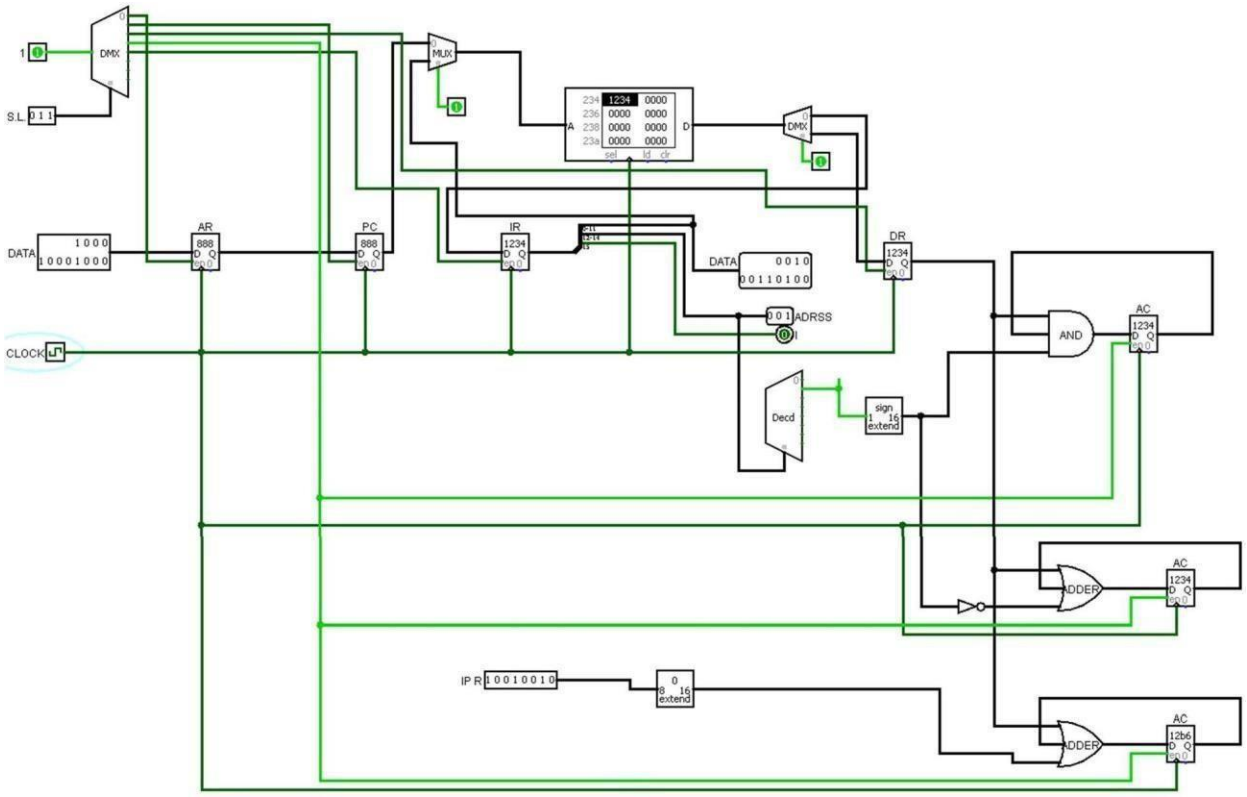
i. Implement 16-bit ALU that is suitable for common bus system

ii. Implement 16-bit common bus system with ALU, 8 registers and 1 memory unit

CIRCUITS:



OUTPUTS:



CONCLUSION :

By this practical I learnt about implementing a common bus system that interates an ALU Arithmetic logic units multiple registers and memory Unit. I designed a logical a 16 Bit ALU that can Interact with the common bus system, allowing data transfer between ALU.

Date:

EXPERIMENT NO. 6

AIM: Perform following operations with basic assembly language programming:

addition	**and**	**logical left shift**	**rotate left with carry**
subtraction	**or**	**logical right shift**	**rotate left without carry**
multiplication	**xor**	**arithmetic left shift**	**rotate right with carry**
division	**not**	**arithmetic right shift**	**rotate right without carry**

Objective 1:

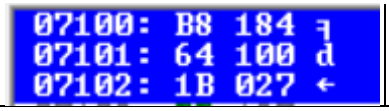
Perform addition of two 32-bit numbers i. 1xxx1B64h and ii. 9135F13Ah and store result at memory location 30020h. (consider xxx is last three digits of your enrolment number)

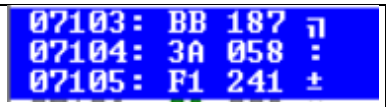
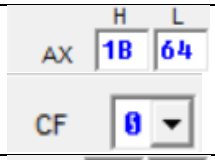
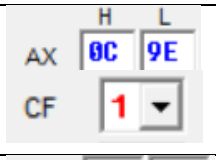
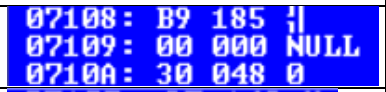
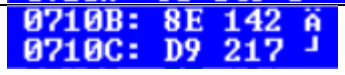
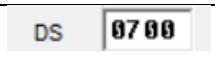
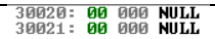
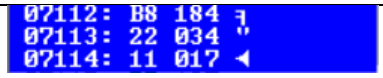
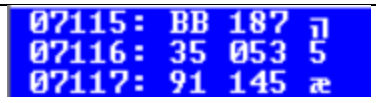
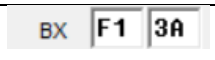
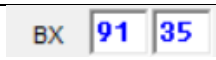
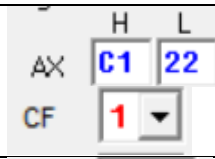
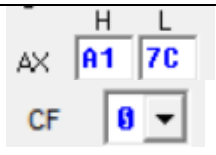
Code:

```
org 100h
mov ax,1B64h
mov bx,0F13Ah
add ax,bx
mov cx,3000h
mov ds,cx
mov di,0020h
mov [di],ax
mov ax,1005h
mov bx,9135h
adc ax,bx
inc di
inc di
mov [di],ax
ret
```

STEP-BY-STEP EXECUTION:

Instruction in Assembly Language	**Screenshot of the Instruction in program memory**	**Value in Instruction Pointer**	**Operation executed by instruction**	**(Before execution of instruction) Content of to be affected Registers/Memory locations and flags**	**(After execution of instruction) Content of affected Registers/Memory locations and flags**
mov ax,1B64h		**0100**	**Store lower bit of number A**		

	 07100: B8 184 ¸ 07101: 64 100 d 07102: 1B 027 ←		**in Ax**	 H L AX 00 00	 H L AX 1B 64

mov bx,0F13Ah		**0103**	**Store lower bit of number B in Bx**		
add ax,bx		**0106**	**Add content of Ax and Bx**		
mov cx,3000h		**0108**	**Store segment address in Cx**		
mov ds,cx		**010B**	**copy content of cx into ds**		
mov di,0020h		**010D**	**Store offset address in di**		
mov [di],ax		**0110**	**Store content of Ax At 30020h**		
mov ax,1122h		**0112**	**Store higher bit of number A in Ax**		
mov bx,9135h		**0115**	**Store higher bit of number B in Bx**		
adc ax,bx		**0118**	**Add Content Of Ax and Bx With Carry**		
inc di		**011A**	**Increment di**		
inc di		**011B**	**Increment di**		
mov [di],ax		**011C**	**Store content of Ax At 30022h**		

Objective 2:

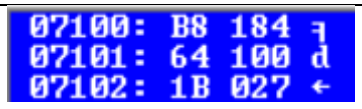
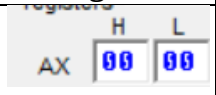
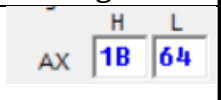
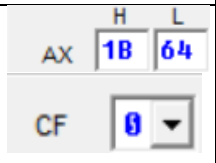
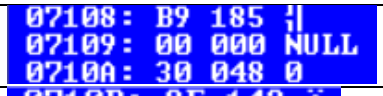
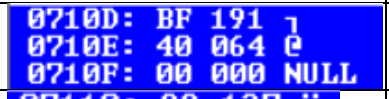
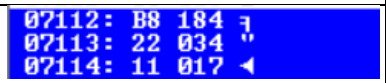
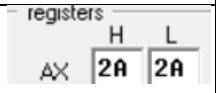
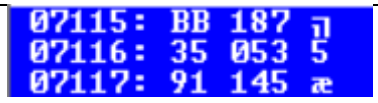
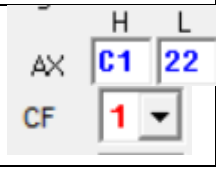
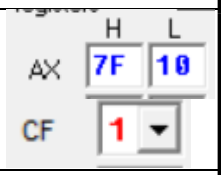
Perform Subtraction of two 32-bit numbers i. 1xxx1B64h and ii. 9135F13Ah and store result at memory location 30040h. (consider xxx is last three digits of your enrolment number)

Code:

```
org 100h
mov ax,1B64h
mov bx,0F13Ah
sub ax,bx
mov cx,3000h
mov ds,cx
mov di,0040h
```

mov [di],ax
mov ax,1005h
mov bx,9135h
sbb ax,bx
inc di
inc di
mov [di],ax
ret

STEP-BY-STEP EXECUTION:

Instruction in Assembly Language	Screenshot of the Instruction in program memory	Value in Instruction Pointer	Operation executed by instruction	(Before execution of instruction) Content of to be affected Registers/Memory locations and flags	(After execution of instruction) Content of affected Registers/Memory locations and flags
mov ax,1B64h		0100	Store lower bit of number Ain Ax		
mov bx,0F13Ah		0103	Store lower bit of number B in Bx		
sub ax,bx		0106	Sub content of Ax and Bx		
mov cx,3000h		0108	Store segment address in Cx		
mov ds,cx		010B	copy content of cx into ds		
mov di,0040h		010D	Store offset address in di		
mov [di],ax		0110	Store content of Ax At 30040h		
mov ax,1122h		0112	Store higher bit of number A in Ax		
mov bx,9135h		0115	Store higher bit of number B in Bx		
sbb ax,bx		0118	Sub Content Of Ax and Bx With Barrow		

inc di	0711A: 47 071 G	**011A**	**Incriment di**	DI 0040	DI 0041
inc di	0711B: 47 071 G	**011B**	**incriment di**	DI 0041	DI 0042
mov [di],ax	0711C: 89 137 ë 0711D: 05 005 ♣	**011C**	**Store content of Ax At 30122h**	30042: 00 000 NULL 30043: 00 000 NULL	30042: 0C 012 ♀ 30043: 7F 127 ⌂

Objective 3:

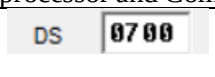
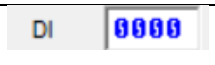
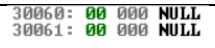
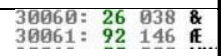
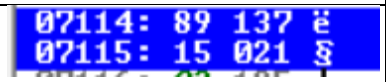
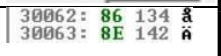
Perform Multiplication of two 16-bit numbers i. Exxxh and ii. A2B3h and store result at memory location 30060h. (consider xxx is last three digits of your enrolment number)

Code:

```
org 100h
mov  ax,0E005h
mov  dx,0A2B3h
mul dx
mov cx,3000h
mov ds,cx
mov di,0060h
mov [di],ax
inc di
inc di
mov [di],dx
ret
```

STEP-BY-STEP EXECUTION:

Instruction in Assembly Language	**Screenshot of the Instruction in program memory**	**Value in Instruction Pointer**	**Operation executed by instruction**	**(Before execution of instruction) Content of to be affected Registers/Memory locations and flags**	**(After execution of instruction) Content of affected Registers/Memory locations and flags**
mov ax,0E122h	07100: B8 184 ╕ 07101: 22 034 " 07102: E1 225 ß	**0100**	**Store lower bit of number A in Ax**	AX 00 00	AX E1 22
Mov dx,0A2B3h	07103: BA 186 ‖ 07104: B3 179 │ 07105: A2 162 ó	**0103**	**Store lower bit of number B in Dx**	DX 00 00	DX A2 B3
mul dx	07106: F7 247 ≈ 07107: E2 226 Γ	**0106**	**Mul content of Ax and Bx Store lower Bit in Ax andhigher bit in Dx**	AX C1 22 DX A2 B3 CF 0	AX 1C F2 DX 8E 89 CF 1
mov cx,3000h	07108: B9 185 ╣ 07109: 00 000 NULL 0710A: 30 048 0	**0108**	**Store segment address in Cx**	CX 00 17	CX 30 00

mov ds,cx		**010B**	**copy content of cx into ds**		
mov di,0060h		**010D**	**Store offset address in di**		
mov [di],ax		**0110**	**Store content of Ax At 30060h**		
inc di		**0112**	**Incriment di**		
inc di		**0113**	**incriment di**		
mov [di],ax		**0114**	**Store content of Dx At 30062h**		

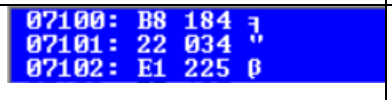
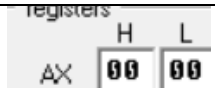
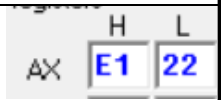
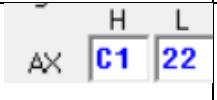
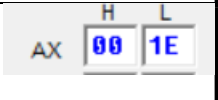
Objective 4:

Perform Division on Exxxh by 0777h and store result at memory location 30070h. (consider xxx is last three digits of your enrolment number)

Code:

```
org 100h
mov ax,0E005h
mov bx,0777h
div bx
mov cx,3000h
mov ds,cx
mov di,0070h
mov [di],ax
ret
```

STEP-BY-STEP EXECUTION:

Instruction in Assembly Language	Screenshot of the Instruction in program memory	Value in Instruction Pointer	Operation executed by instruction	(Before execution of instruction) Content of to be affected Registers/Memory locations and flags	(After execution of instruction) Content of affected Registers/Memory locations and flags
mov ax,0E122h		**0100**	**Store number A in Ax**		
Mov dx,0777h		**0103**	**Store lower bit of number B in Bx**		
div bx		**0106**	**Div content of Ax and Bx Store in Ax**		

mov cx,3000h	07108: B9 185 ╣ 07109: 00 000 NULL 0710A: 30 048 0	**0108**	**Store segment address in Cx**	CX 00 13	CX 30 00
mov ds,cx	0710B: 8E 142 Ä 0710C: D9 217 ┘	**010B**	**copy content of cx into ds**	DS 0700	DS 3000
mov di,0070h	0710D: BF 191 ┐ 0710E: 70 112 p 0710F: 00 000 NULL	**010D**	**Store offset address in di**	DI 0000	DI 0070
mov [di],ax	07110: 89 137 ë 07111: 05 005 ♣	**0110**	**Store content of Ax At 30070h**	30070: 00 000 NULL 30071: 00 000 NULL	30070: 1E 030 ▲ 30071: 00 000 NULL

Objective 5:

Perform ANDing on Cxxxh by 00FFh and store result at memory location 30080h. (consider xxx is last three digits of your enrolment number)

Code:

```
org 100h
mov ax,0C005h
mov bx,00FFh
and ax,bx
mov cx,3000h
mov ds,cx
mov di,0080h
mov [di],ax
ret
```

STEP-BY-STEP EXECUTION:

Instruction in Assembly Language	**Screenshot of the Instruction in program memory**	**Value in Instruction Pointer**	**Operation executed by instruction**	**(Before execution of instruction) Content of to be affected Registers/Memory locations and flags**	**(After execution of instruction) Content of affected Registers/Memory locations and flags**
mov ax,0C122h	07100: B8 184 ╕ 07101: 22 034 " 07102: C1 193 ┴	**0100**	**Store number A in Ax**	registers H L AX 00 00	H L AX C1 22
Mov dx,00FFh	07103: BB 187 ╗ 07104: FF 255 RES 07105: 00 000 NULL	**0103**	**Store number B in Bx**	BX 00 00	BX 00 FF
and ax,bx	07106: 23 035 # 07107: C3 195 ├	**0106**	**and content of Ax and Bx Store in Ax**	H L AX C1 22	H L AX C1 22
mov cx,3000h	07108: B9 185 ╣ 07109: 00 000 NULL 0710A: 30 048 0	**0108**	**Store segment address in Cx**	CX 00 13	CX 30 00
mov ds,cx	0710B: 8E 142 Ä 0710C: D9 217 ┘	**010B**	**copy content of cx into ds**	DS 0700	DS 3000

mov di,0080h	0710D: BF 191 ┐ 0710E: 80 128 Ç 0710F: 00 000 NULL	**010D**	**Store offset address in di**	DI 0000	DI 0080
mov [di],ax	07110: 89 137 ë 07111: 05 005 ♣	**0110**	**Store contentof Ax At 30080h**	30080: 00 000 NULL 30081: 00 000 NULL	30080: 42 066 B 30081: 00 000 NULL

Objective 6:

Perform ORing on Cxxxh by 00FFh and store result at memory location 300A0h. (consider xxx is last three digits of your enrolment number)

Code:

```
org 100h
mov ax,0C005h
mov bx,00FFh
or ax,bx
mov cx,3000h
mov ds,cx
mov di,00A0h
mov [di],ax
ret
```

STEP-BY-STEP EXECUTION:

Instruction in Assembly Language	Screenshot of the Instruction in program memory	Value in Instruction Pointer	Operation executed by instruction	(Before execution of instruction) Content of to be affected Registers/Memory locations and flags	(After execution of instruction) Content of affected Registers/Memory locations and flags
mov ax,0C122h	07100: B8 184 ╕ 07101: 22 034 " 07102: C1 193 ┴	**0100**	**Store number A in Ax**	AX H 00 L 00	AX H C1 L 22
Mov dx,00FFh	07103: BB 187 ╗ 07104: FF 255 RES 07105: 00 000 NULL	**0103**	**Store number B in Bx**	BX 00 00	BX 00 FF
or ax,bx	07106: 0B 011 ♂ 07107: C3 195 ├	**0106**	**or content ofAx and Bx Store in Ax**	AX H C1 L 22	AX H C0 L FF
mov cx,3000h	07108: B9 185 ╣ 07109: 00 000 NULL 0710A: 30 048 0	**0108**	**Store segment address in Cx**	CX 00 13	CX 30 00
mov ds,cx	0710B: 8E 142 Ä 0710C: D9 217 ┘	**010B**	**copy content of cx into ds**	DS 0700	DS 3000
mov di,00A0h	0710D: BF 191 ┐ 0710E: 80 128 Ç 0710F: 00 000 NULL	**010D**	**Store offset address in di**	DI 0000	DI 00A0
mov [di],ax	07110: 89 137 ë 07111: 05 005 ♣	**0110**	**Store contentof Ax At 300A0h**	300A0: 00 000 NULL 300A1: 00 000 NULL	300A0: FF 255 RES 300A1: C0 192 └

Objective 7:

Perform XORing on Cxxxh by 00FFh and store result at memory location 300B0h. (consider xxx is last three digits of your enrolment number)

Code:

```
org 100h
mov ax,0C005h
mov bx,00FFh
xor ax,bx
mov cx,3000h
mov ds,cx
mov di,00B0h
mov [di],ax
ret
```

STEP-BY-STEP EXECUTION:

Instruction in Assembly Language	Screenshot of the Instruction in program memory	Value in Instruction Pointer	Operation executed by instruction	(Before execution of instruction) Content of to be affected Registers/Memory locations and flags	(After execution of instruction) Content of affected Registers/Memory locations and flags
mov ax,0C122h	07100: B8 184 07101: 22 034 " 07102: C1 193 ┴	0100	Store number A in Ax	AX H:00 L:00	AX H:C1 L:22
Mov dx,00FFh	07103: BB 187 ╗ 07104: FF 255 RES 07105: 00 000 NULL	0103	Store number B in Bx	BX 00 00	BX 00 FF
xor ax,bx	07106: 0B 011 ♂ 07107: C3 195 ├	0106	xor content of Ax and Bx Store in Ax	AX H:C1 L:22	AX H:C0 L:B9
mov cx,3000h	07108: B9 185 ╣ 07109: 00 000 NULL 0710A: 30 048 0	0108	Store segment address in Cx	CX 00 13	CX 30 00
mov ds,cx	0710B: 8E 142 Ä 0710C: D9 217 ┘	010B	copy content of cx into ds	DS 0700	DS 3000
mov di,00B0h	0710D: BF 191 ┐ 0710E: B0 128 Ç 0710F: 00 000 NULL	010D	Store offset address in di	DI 0000	DI 00B0
mov [di],ax	07110: 89 137 ë 07111: 05 005 ♣	0110	Store content of Ax At 300B0h	300B0: 00 000 NULL 300B1: 00 000 NULL	300B0: BD 189 ╜ 300B1: C0 192 └

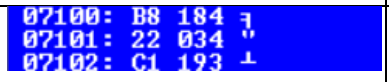
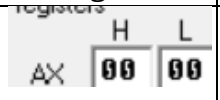
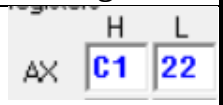
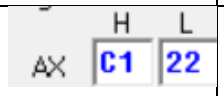
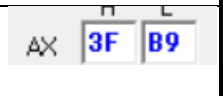
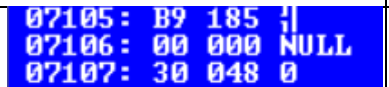
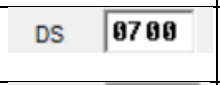
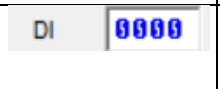
Objective 8:

Perform NOT operation on Cxxxh and store result at memory location 300C0h. (consider xxx is last three digits of your enrolment number)

Code:

org 100h
mov ax,0C005h
not ax
mov cx,3000h
mov ds,cx
mov di,00C0h
mov [di],ax
ret

STEP-BY-STEP EXECUTION:

Instruction in Assembly Language	Screenshot of the Instruction in program memory	Value in Instruction Pointer	Operation executed by instruction	(Before execution of instruction) Content of to be affected Registers/Memory locations and flags	(After execution of instruction) Content of affected Registers/Memory locations and flags
mov ax,0C122h		0100	Store number A in Ax		
not ax		0103	not content of Ax Store in Ax		
mov cx,3000h		0105	Store segment address in Cx		
mov ds,cx		0108	copy content of cx into ds		
mov di,00C0h		010A	Store offset address in di		
mov [di],ax		0110	Store contentof Ax At 300C0h		

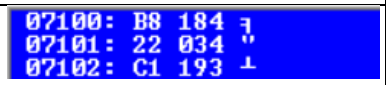
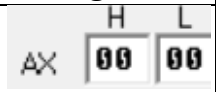
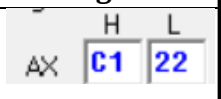
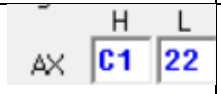
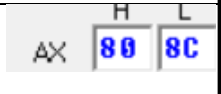
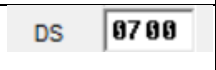
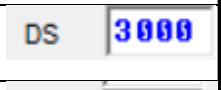
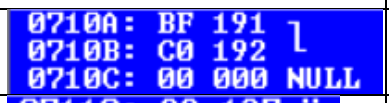
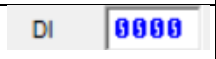
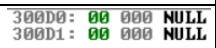
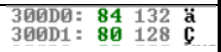
Objective 9:

Perform logical left shift on Cxxxh and store result at memory location 300D0h. (consider xxx is last three digits of your enrolment number)

Code:

org 100h
mov ax,0C005h
shl ax ,1
mov cx,3000h
mov ds,cx
mov di,00D0h
mov [di],ax
ret

STEP-BY-STEP EXECUTION:

Instruction in Assembly Language	Screenshot of the Instruction in program memory	Value in Instruction Pointer	Operation executed by instruction	(Before execution of instruction) Content of to be affected Registers/Memory locations and flags	(After execution of instruction) Content of affected Registers/Memory locations and flags
mov ax,0C122h		0100	Store number A in Ax		
shl ax,1		0103	Shift Left content of Ax Store in Ax		
mov cx,3000h		0105	Store segment address in Cx		
mov ds,cx		0108	copy content of cx into ds		
mov di,00D0h		010A	Store offset address in di		
mov [di],ax		0110	Store content of Ax At 300D0h		

Objective 10:

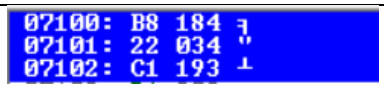
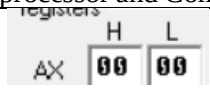
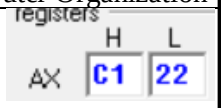
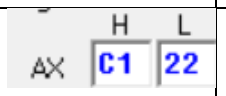
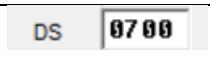
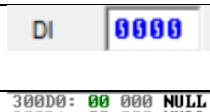
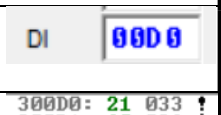
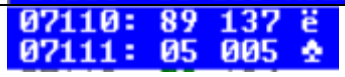
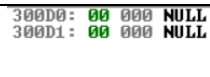
Perform logical right shift on Cxxxh and store result at memory location 300D0h. (consider xxx is last three digits of your enrolment number)

Code:

```
org 100h
mov ax,0C005h
shr ax,1
mov cx,3000h
mov ds,cx
mov di,00D0h
mov [di],ax
ret
```

STEP-BY-STEP EXECUTION:

Instruction in Assembly Language	Screenshot of the Instruction in program memory	Value in Instruction Pointer	Operation executed by instruction	(Before execution of instruction) Content of to be affected Registers/Memory locations and flags	(After execution of instruction) Content of affected Registers/Memory locations and flags

mov ax,0C122h		**0100**	**Store number A in Ax**		
shr ax,1		**0103**	**Shift Right content of Ax Store in Ax**		
mov cx,3000h		**0105**	**Store segment address in Cx**		
mov ds,cx		**0108**	**copy content of cx into ds**		
mov di,00D0h		**010A**	**Store offset address in di**		
mov [di],ax		**0110**	**Store content of Ax At 300D0h**		

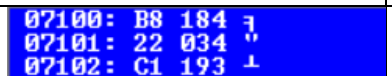
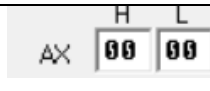
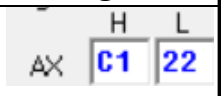
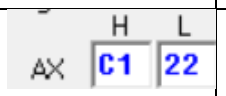
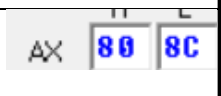
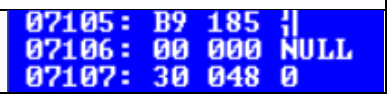
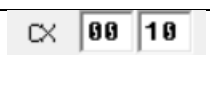
Objective 11:

Perform arithmetic left shift on Cxxxh and store result at memory location 300E0h. (consider xxx is last three digits of your enrolment number)

Code:

```
org 100h
mov ax,0C005h
sal ax,1
mov cx,3000h
mov ds,cx
mov di,00E0h
mov [di],ax
ret
```

STEP-BY-STEP EXECUTION:

Instruction in Assembly Language	**Screenshot of the Instruction in program memory**	**Value in Instruction Pointer**	**Operation executed by instruction**	**(Before execution of instruction) Content of to be affected Registers/Memory locations and flags**	**(After execution of instruction) Content of affected Registers/Memory locations and flags**
mov ax,0C122h		**0100**	**Store number A in Ax**		
sal ax,1		**0103**	**Arithmetic Shift Left content of Ax Store in Ax**		
mov cx,3000h		**0105**	**Store segment address in Cx**		

mov ds,cx	07108: 8E 142 Ä 07109: D9 217 ┘	**0108**	**copy content of cx into ds**	DS 0700	DS 3000
mov di,00E0h	0710A: BF 191 ┐ 0710B: C0 192 └ 0710C: 00 000 NULL	**010A**	**Store offset address in di**	DI 0000	DI 00E0
mov [di],ax	07110: 89 137 ë 07111: 05 005 ♣	**0110**	**Store contentof Ax At 300E0h**	300E0: 00 000 NULL 300E1: 00 000 NULL	300E0: 84 132 ä 300E1: 80 128 Ç

Objective 12:

Perform arithmetic right shift on Cxxxh and store result at memory location 300F0h. (consider xxx is last three digits of your enrolment number)

Code:

```
org 100h
mov ax,0C005h
sar ax,1
mov cx,3000h
mov ds,cx
mov di,00F0h
mov [di],ax
ret
```

STEP-BY-STEP EXECUTION:

Instruction in Assembly Language	**Screenshot of the Instruction in program memory**	**Value in Instruction Pointer**	**Operation executed by instruction**	**(Before execution of instruction) Content of to be affected Registers/Memory locations and flags**	**(After execution of instruction) Content of affected Registers/Memory locations and flags**
mov ax,0C122h	07100: B8 184 ╕ 07101: 46 070 F 07102: C0 192 └	**0100**	**Store number A in Ax**	AX H 00 L 00	AX H C1 L 22
sar ax,1	07103: D1 209 ╤ 07104: F8 248 °	**0103**	**Arithmetic Shift Right content of Ax Store in Ax**	AX H C1 L 22	AX H E0 L 23
mov cx,3000h	07105: B9 185 ╣ 07106: 00 000 NULL 07107: 30 048 0	**0105**	**Store segment address in Cx**	CX 00 10	CX 30 00
mov ds,cx	07108: 8E 142 Ä 07109: D9 217 ┘	**0108**	**copy content of cx into ds**	DS 0700	DS 3000
mov di,00F0h	0710A: BF 191 ┐ 0710B: C0 192 └ 0710C: 00 000 NULL	**010A**	**Store offset address in di**	DI 0000	DI 00F0
mov [di],ax	07110: 89 137 ë 07111: 05 005 ♣	**011D**	**Store content of Ax At 300F0h**	300F0: 00 000 NULL 300F1: 00 000 NULL	300F0: 21 033 ! 300F1: E0 224 α

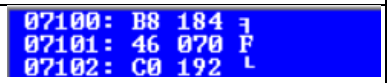
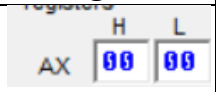
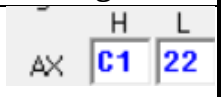
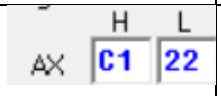
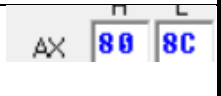
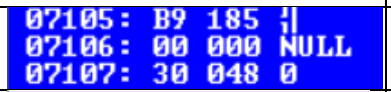
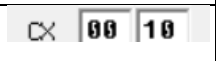
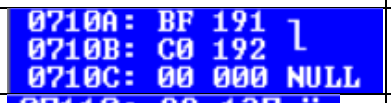
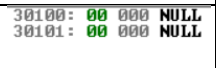
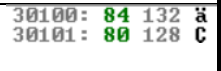
Objective 13:

Perform rotate left shift with carry on Cxxxh and store result at memory location 30100h. (consider xxx is last three digits of your enrolment number)

Code:

```
org 100h
mov ax,0C005h
rcl ax,1
mov cx,3000h
mov ds,cx
mov di,0100h
mov [di],ax
ret
```

STEP-BY-STEP EXECUTION:

Instruction in Assembly Language	Screenshot of the Instruction in program memory	Value in Instruction Pointer	Operation executed by instruction	(Before execution of instruction) Content of to be affected Registers/Memory locations and flags	(After execution of instruction) Content of affected Registers/Memory locations and flags
mov ax,0C122h		0100	Store number A in Ax		
rcl ax,1		0103	Rotate Left Shift with carry content of Ax Store in Ax		
mov cx,3000h		0105	Store segment address in Cx		
mov ds,cx		0108	copy content of cx into ds		
mov di,0100h		010A	Store offset address in di		
mov [di],ax		0110	Store content of Ax At 30100h		

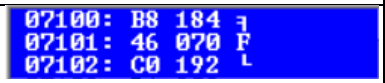
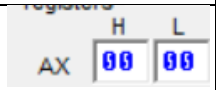
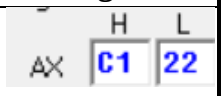
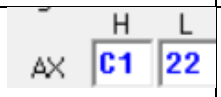
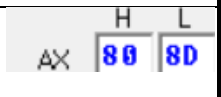
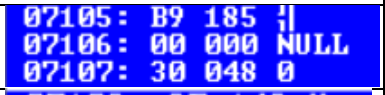
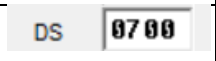
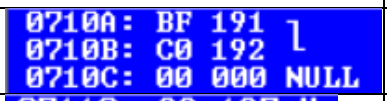
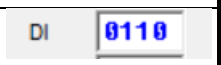
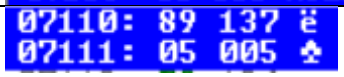
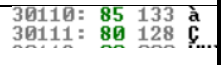
Objective 14:

Perform rotate left shift without carry on Cxxxh and store result at memory location 30110h. (consider xxx is last three digits of your enrolment number)

Code:

```
org 100h
mov ax,0C005h
rol ax,1
mov cx,3000h
```

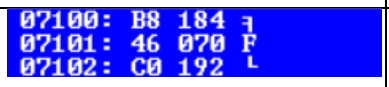
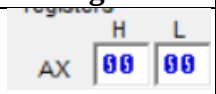
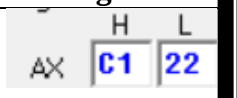
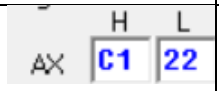
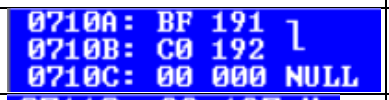
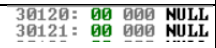
mov ds,cx
mov di,0110h
mov [di],ax
ret
STEP-BY-STEP EXECUTION:

Instruction in Assembly Language	Screenshot of the Instruction in program memory	Value in Instruction Pointer	Operation executed by instruction	(Before execution of instruction) Content of to be affected Registers/Memory locations and flags	(After execution of instruction) Content of affected Registers/Memory locations and flags
mov ax,0C122h		0100	Store number A in Ax		
rol ax,1		0103	Rotate Left Shift without carry content of Ax Store in Ax		
mov cx,3000h		0105	Store segment address in Cx		
mov ds,cx		0108	copy content of cx into ds		
mov di,0110h		010A	Store offset address in di		
mov [di],ax		0110	Store content of Ax At 30110h		

Objective 15:

Perform rotate right shift with carry on Cxxxh and store result at memory location 30120h. (consider xxx is last three digits of your enrolment number)

Code:
org 100h
mov ax,0C005h
rcr ax,1
mov cx,3000h
mov ds,cx
mov di,0120h
mov [di],ax
ret
STEP-BY-STEP EXECUTION:

Instruction in Assembly Language	Screenshot of the Instruction in program memory	Value in Instruction Pointer	Operation executed by instruction	(Before execution of instruction) Content of to be affected Registers/Memory locations and flags	(After execution of instruction) Content of affected Registers/Memory locations and flags
mov ax,0C122h		0100	Store number A in Ax		
rcr ax,1		0103	Rotate Right Shift with carry content of Ax Store in Ax		
mov cx,3000h		0105	Store segment address in Cx		
mov ds,cx		0108	copy content of cx into ds		
mov di,0120h		010A	Store offset address in di		
mov [di],ax		0110	Store content of Ax At 30120h		

Objective 16:

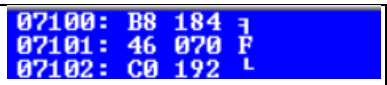
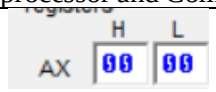
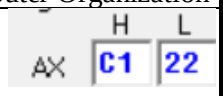
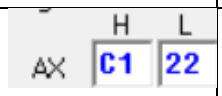
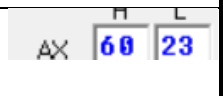
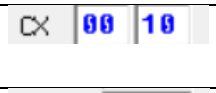
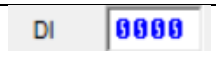
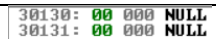
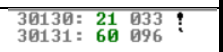
Perform rotate right shift without carry on Cxxxh and store result at memory location 30130h. (consider xxx is last three digits of your enrolment number)

Code:

```
org 100h
mov ax,0C005h
ror ax,1
mov cx,3000h
mov ds,cx
mov di,0130h
mov [di],ax
ret
```

STEP-BY-STEP EXECUTION:

Instruction in Assembly Language	Screenshot of the Instruction in program memory	Value in Instruction Pointer	Operation executed by instruction	(Before execution of instruction) Content of to be affected Registers/Memory locations and flags	(After execution of instruction) Content of affected Registers/Memory locations and flags

mov ax,0C122h		**0100**	**Store number A in Ax**		
ror ax,1		**0103**	**Rotate Right Shift without carry content of Ax Store in Ax**		
mov cx,3000h		**0105**	**Store segment address in Cx**		
mov ds,cx		**0108**	**copy content of cx into ds**		
mov di,0130h		**010A**	**Store offset address in di**		
mov [di],ax		**0110**	**Store contentof Ax At 30130h**		

CONCLUSION:

In conclusion, this experiment successfully demonstrated various assembly language operations, including arithmetic, logical, shift, and rotate functions on 32 -bit numbers. Each operation's results were accurately stored in specific memory locations, showcasing efficient handling of data and manipulation of registers using instructions like ADD, SUB, MUL, DIV, and logical shifts.

Date:19-09-24

EXPERIMENT NO. 7

AIM: Create an array. Perform addition of all even numbers from array and save answer in one variable.

ALGORITHM:

Step:1 Set Data Segment:

Load the value 2000h into the AX register.

Set the DS register to AX, initializing the data segment to 2000h.

Step:2 Initialize Index and Counter:

Set SI to 50h, pointing to the starting address of the array in memory (DS:50h).

Set CL to 5h, initializing the loop counter to 5, which represents the number of elements in the

Step:3 Initialize Sum:

Set AL to 0, initializing the accumulator to store the sum of even numbers.

Step:4 Loop through the Array:

Repeat the following steps 5 times (controlled by CL).

Step:5 Inside the Loop:

Load the byte at the address [SI] (the current element in the array) into BL.

Use the test bl, 01h instruction to check if the least significant bit (LSB) is 0 (indicating an even number).

If the number is odd (LSB is 1), skip the addition step by jumping to L1.

If the number is even, add it to the accumulator (AL) using add al, bl.

Increment SI to point to the next element in the array.

Use the loop instruction to decrease CL and repeat the loop until CL becomes 0.

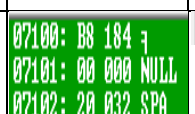
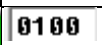
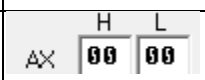
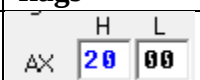
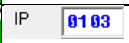
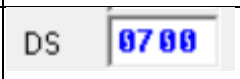
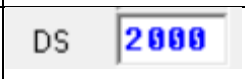
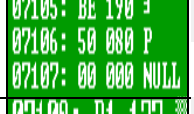
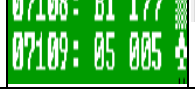
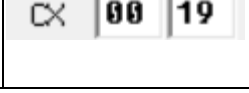
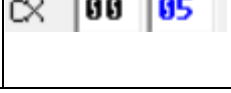
Step:6 End the Program:

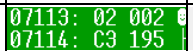
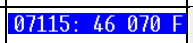
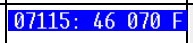
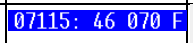
When all 5 elements have been processed, return control to the caller using the ret instruction.

CODE:

```
org 100h

mov ax,2000h
mov ds,ax
mov si,50h
mov cl,5h

mov al,0h
L2:
 mov  bl,[si]
 test   bl,01h
 JNZ L1

 add al,bl
L1:
inc si
loop L2

ret
```

STEP-BY-STEP EXECUTION:

Instruction in Assembly Language	**Screenshot of the Instruction in program memory**	**Value in Instruction Pointer**	**Operation executed by instruction**	**(Before execution of instruction) Content of to be affected Registers/Memory locations and flags**	**(After execution of instruction) Content of affected Registers/Memory locations and flags**
mov ax,2000h			**mov ax,2000h loads the value in ax**		
mov ds,ax			**Mov da,ax loads the value from ax,ds**		
mov si,50h			**Mov si,50h set the si to 50h**		
mov cl,5h			**Mov cl,5h set the cl to the 5h**		

mov al,0h		IP 010A	**Mov al,0h set the al to the 0h**	AX H 20 L 00	AX H 20 L 00
mov bl,[si]		IP 010C	**Mov bl,[si] set the bl to the si**	BX 00 00	BX 00 04
test bl,01h		IP 010E	**The instruction `test bl, 01h` checks if the least significant bit of the `BL` register is 1 or 0 .**	BX 00 04	BX 00 04
JNZ L1		IP 0111	**The instruction `JNZ L1` means "Jump to the label `L1` if the result of the previous operation is not zero**		
add al,bl		IP 0113	**Add al,bl means that al + bl**		
inc si		IP 0115	**Increase Si by 1**	SI 0050	SI 0051
LOOP L2		IP 0116	LOOP traversal for the cl times here 5		
inc si		IP 0115	2nd traversal of the loop	SI 0051	SI 0052
add al,bl		IP 0113		AX H 20 L 04	AX 20 06
Inc si		IP 0115	3rd traversal of the loop	SI 0052	SI 0053
Inc si		IP 0115	4th traversal of the loop	SI 0052	SI 0054
Inc si		IP 0115	5th traversal of the loop	SI 0054	SI 0055

CONCLUSION:

The given assembly code sums all the even numbers from a memory array of 5 elements. It checks each number using a bitwise test to determine if it's even, and adds even values to the `AL` register. The loop iterates over each element in the array, and the sum of all even numbers is stored in the `AL` register when the program completes. This demonstrates basic use of looping, conditional checks, and arithmetic in 8086 assembly language.

Date:

EXPERIMENT NO. 8

AIM: Find out whether the given string is palindrome or not and print appropriate message. Don't use procedure.

ALGORITHM:

Input the String:

- **Load the string into memory.**

Initialize Pointers:

- **Set two pointers: one at the beginning of the string and the other at the end of the string.**

Compare Characters:

- **Compare the character at the start with the character at the end.**
- **If the characters match, move the start pointer forward and the end pointer backward.**

Repeat Until Middle:

- **Repeat the comparison until the two pointers meet in the middle of the string (or cross each other).**

Palindrome Check:

- **If all characters match, the string is a palindrome.**
- **If any character does not match, the string is not a palindrome.**

Output the Result:

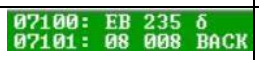
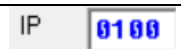
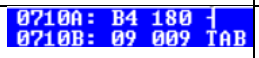
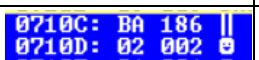
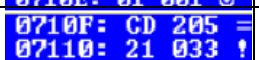
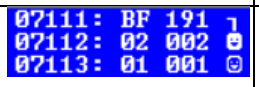
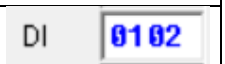
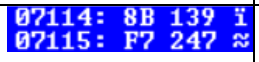
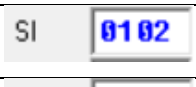
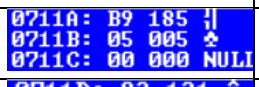
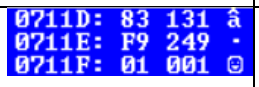
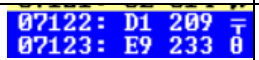
- **Print whether the string is a palindrome or not.**

1. Palindrome:

CODE:

```
  no e prac
org Iøøh
n1 :
s db abcba'
s size
dB øDh.øAh.'$'
start:
; printin actual s tring...
offset s ;
int 21b
lea d i. s
add s i. s —size
dec si ; point to
nou cx.s_size
Je IS —palindrome
shr c x. 1 ;divide
nov
nou b 1. t sil
cnp a 1. bl
last char'
•sin le char
is palindrome!
Sne not_palindeone
dec si
loop next—char
s _pa indi•one
nov
nou dx.offset
int 21b
.jnp stop
no t —pa 1 in drone :
nov ah.9
nou dx offset
int 21iI
; wait for any
ret
nsgl  db  "this
nsg2  db  "this
nsgl
nsg2
key preg:
is palidrone!$"
is not pal i drone' $ "
```

STEP-BY-STEP EXECUTION:

Instruction in Assembly Language	Screenshot of the Instruction in program memory	Value in Instruction Pointer	Operation executed by instruction	(Before execution of instruction) Content of to be affected Registers/Memory locations and flags	(After execution of instruction) Content of affected Registers/Memory locations and flags
jmp start			Direct jump at start	-	-
mov ah,9			Store the value in ah		
mov dx,offset s;			Offset address of string s store in dx.		
Int 21h			Int 21h use for see the output on screen	-	
lea di,s			Load effective Address of string s store in di.		
mov si,di			Copy the data of di and store in si.		
add si,s_size			add the size of a string in si.		
dec si			Decrement in si.		
mov cx,s_size			The size of string is store in cx.		
cmp cx,1			Compare the contents of general purpose register with 1.		
je is_palindrome			Jump if equals the previous comparison	-	-
shr cx,1			Right shift of the value of cx by 1.		
mov al,[di]			The data of [di] is stored in al.		
mov bl,[si]			The data of [si] is stored in bl.		
cmp al,bl			Compare the contents of al and bl.		

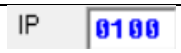
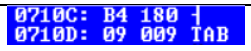
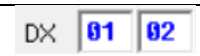
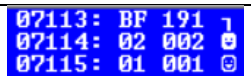
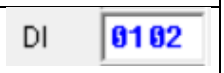
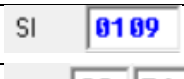
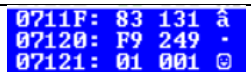
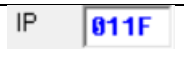
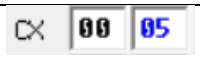
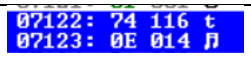
jne not_palindrome	0712A: 75 117 u 0712B: 0D 013 CREI	IP 012A	Jump if not equals to each other.	-	-
inc di	0712C: 47 071 G	IP 012C	Increment the data of di.	DI 0102	DI 0103
dec si	0712D: 4E 078 N	IP 012D	Decrement the data of si.	SI 0106	SI 0105
loop next_char	0712E: E2 226 Γ 0712F: F4 244 ⌠	IP 012E	Execute the next_char loop.	-	-
mov al,[di]	07124: 8A 138 è 07125: 05 005 ♣	IP 0124	The data of [di] is stored in al.	AX 09 61	AX 09 62
mov bl,[si]	07126: 8A 138 è 07127: 1C 028 ∟	IP 0126	The data of [si] is stored in bl.	BX 00 61	BX 00 62
cmp al,bl	07128: 3A 058 : 07129: C3 195 ├	IP 0128	Compare the contents of al and bl.	AX 09 62 BX 00 62	AX 09 62 BX 00 62
jne not_palindrome	0712A: 75 117 u 0712B: 0D 013 CREI	IP 012A	Jump if not equals to each other.	-	-
inc di	0712C: 47 071 G	IP 012C	Increment the data of di.	DI 0103	DI 0104
dec si	0712D: 4E 078 N	IP 012D	Decrement the data of si.	SI 0105	SI 0104
loop next_char	0712E: E2 226 Γ 0712F: F4 244 ⌠	IP 012E	Execute the next_char loop.	-	-
mov ah,9	07130: B4 180 ┤ 07131: 09 009 TAB	IP 0130	Value store in ah.	AX 09 62	AX 09 62
mov dx,offset msg1	07132: BA 186 ‖ 07133: 45 069 E 07134: 01 001 ☺	IP 0132	Offset address of msg1 is store in dx	DX 01 02	DX 01 45
int 21h	07135: CD 205 = 07136: 21 033 !	IP 0135	Output show on screen.	-	abcba this is palidrome!
jmp stop	07137: EB 235 δ 07138: 07 007 BEEP	IP 0137	Direct jump at stop	-	-
mov ah,0	07140: B4 180 ┤ 07141: 00 000 NULL	IP 0140	0 store in ah register.	AX 09 24	AX 00 24
int 16h	F41C0: FF 255 RES F41C1: FF 255 RES	IP 01C0	Output show on screen.	-	abcba this is palidrome!

2. Not Palindrome:

CODE:

```
e prac
1ØØh
J n Start
01
s db '23CSØ24'
s _s ize
db ØDh.ØAh.'$'
start :
; printin actual string...
nou dx.offset s;
ine 2ih
lea dies
nov s i.di
add s i. s —size
dec si ; point to last char!
nov cx.s_size
cnp c x. 1
; sin Char
shr c x. 1 ; divide by 2 V
next—char:
nou al. Idi)
ne not _palindrone
nC di
dec si
loop next _ehae
is —pa 1 ind rone
nou ax. offset nagi
int 2ih
n ot —pa indrone :
nou dx.offset
ine 2ih
stop:
;uait for any
nov ah.Ø
int
ret
nsg2 db
"this
"this
g2
key preg:
is not pal idronet$••
```

STEP-BY-STEP EXECUTION:

Instruction in Assembly Language	Screenshot of the Instruction in program memory	Value in Instruction Pointer	Operation executed by instruction	(Before execution of instruction) Content of to be affected Registers/Memory locations and flags	(After execution of instruction) Content of affected Registers/Memory locations and flags
jmp start			Direct jump at start	-	-
mov ah,9			Store the value in ah		
mov dx,offset s;			Offset address of string s store in dx.		
Int 21h			Int 21h use for see the output on screen	-	
lea di,s			Load effective Address of string s store in di.		
mov si,di			Copy the data of di and store in si.		
add si,s_size			add the size of a string in si.		
dec si			Decrement in si.		
mov cx,s_size			The size of string is store in cx.		
cmp cx,1			Compare the contents of general purpose register with 1.		
je is_palindrome			Jump if equals the previous comparison	-	-
shr cx,1			Right shift of the value of cx by 1.		
mov al,[di]			The data of [di] is stored in al.		
mov bl,[si]			The data of [si] is stored in bl.		
cmp al,bl			Compare the contents of al and bl.		
jne not_palindrome			Jump if not equals to each other.	-	-

mov ah,9	0713B: B4 180 ┤ 0713C: 09 009 TAB	IP 013B	Value store in ah.	AX 09 32	AX 09 32
mov dx,offset msg2	0713D: BA 186 ‖ 0713E: 5A 090 Z 0713F: 01 001 ☺	IP 013D	Offset address of msg1 is store in dx	DX 01 02	DX 01 5A
int 21h	07140: CD 205 = 07141: 21 033 !	IP 0140	Output show on screen.	-	23CS024 this is not palidrome!
jmp stop	07137: EB 235 δ 07138: 07 007 BEEP	IP 0137	Direct jump at stop	-	-
mov ah,0	07142: B4 180 ┤ 07143: 00 000 NULL	IP 0142	0 store in ah register.	AX 09 24	AX 00 24
int 16h	F41C0: FF 255 RES F41C1: FF 255 RES	IP 01C0	Output show on screen.	-	23CS024 this is not palidrome!

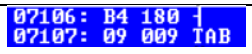
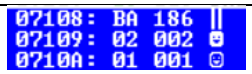
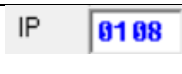
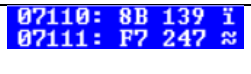
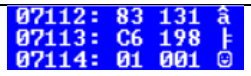
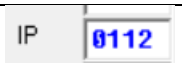
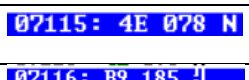
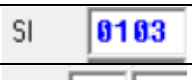
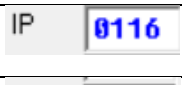
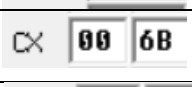
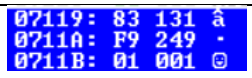
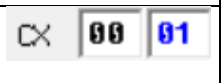
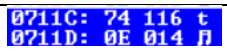
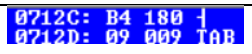
3. Single Character:

CODE:

o Pg I guh
startt
; print in S
int 21b
lea d i.'
add
dee ; po int to char'
je is_palindrone •sin le char
shr cx.l :divide Ky 2!
next _char :
nov
nov b I.
cop
dec
loop next—char
is pal in drone'
nou ax. offset nsgt
int 21 h
*top
not _pa 1 indrone :
ax. of f set
ine 21b
top :
; "ait for any
n sg2
key pres:
int 16b
nsg2 db
"t his
Is
pal

not pal idrone'$••

STEP-BY-STEP EXECUTION:

Instruction in Assembly Language	Screenshot of the Instruction in program memory	Value in Instruction Pointer	Operation executed by instruction	(Before execution of instruction) Content of to be affected Registers/Memory locations and flags	(After execution of instruction) Content of affected Registers/Memory locations and flags
jmp start			Direct jump at start	-	-
mov ah,9			Store the value in ah		
mov dx,offset s;			Offset address of string s store in dx.		
Int 21h			Int 21h use for see the output on screen	-	
lea di,s			Load effective Address of string s store in di.		
mov si,di			Copy the data of di and store in si.		
add si,s_size			add the size of a string in si.		
dec si			Decrement in si.		
mov cx,s_size			The size of string is store in cx.		
cmp cx,1			Compare the contents of general purpose register with 1.		
je is_palindrome			Jump if equals the previous comparison	-	-
mov ah,9			Value store in ah.		
mov dx,offset msg2			Offset address of msg1 is store in dx		
int 21h			Output show on screen.	-	
jmp stop			Direct jump at stop	-	-
mov ah,0			0 store in ah register.		
int 16h			Output show on screen.	-	

CONCLUSION:

The code checks if the string "abcba" is a palindrome by comparing characters from both ends inward. If all characters match, it prints a message stating that the string is a palindrome. If a mismatch is found, it prints a message indicating the string is not a palindrome.

Date:

EXPERIMENT NO. 9

AIM: Write an assembly code to evaluate the answer of below given series and store the answer in the ANS variable. Program should have only one procedure to compute the factorial of a number. Series: 1! -2+3!-4+5!-6+7!-8

ALGORITHM:

Setup Data and Initialize Variables:

- **Initialize data segment (ds) using the mov instruction.**
- **Set up cx to start at 0001h (11 in decimal).**

Loop through Numbers:

- **Start a loop (next) that iterates over numbers from cx = 11 down to 1.**

Check if Even or Odd:

- **Use the division instruction (div) to divide the current number by 2.**
- **If the remainder (ah) is 1, the number is odd, so call the factorial_fun function.**
- **If the number is even, skip the factorial calculation.**

Factorial Function:

- **If the number is odd, the factorial_fun procedure calculates the factorial of the number stored in cx:**
  - **Start with 1 and multiply it by cx, decrementing cx by 1 each time, until cx reaches 1.**
  - **Add the result to the answer (ans).**

Update the Result:

- **For even numbers, subtract the number from the ans.**
- **For odd numbers, the factorial result is added to the ans.**

Repeat the Process:

- **After each iteration, decrement cx and repeat the loop until cx becomes zero.**

End Program:

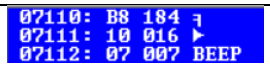
- **Once the loop is complete, the program halts (hlt).**

CODE:

```
mode pract ica
, 23CS024
. data
I dup O
ans dw
1 dup 2
two db
tcx  dw
I dup 2
code
```

```
@dat a
mov ax,
mov ds,ax
0008h
mov  cx,
OOOlh
mov  ax,
next :
mov dx,cx
mov ax, dx
div two
cmp ah,01h
j ne for_even
mov tcx, cx
call factorial_fun
mov cx,tcx
f or_even:
sub ans, dx
I oop next
hlt
factorial fun proc
mov  ax,OOj1h
lb:
mul cx
I oop lb
add ans, ax
ret
factorial_fun  endp
```

STEP-BY-STEP EXECUTION:

Instruction in Assembly Language	**Screenshot of the Instruction in program memory**	**Value in Instruction Pointer**	**Operation executed by instruction**	**(Before execution of instruction) Content of to be affected Registers/Memory locations and flags**	**(After execution of instruction) Content of affected Registers/Memory locations and flags**
mov ax,@data			Segment address of data is move into the ax.		

mov ds,ax	07113: 8E 142 Ä 07114: D8 216 ╪	IP 0003	The value of ax is store into data segment.	DS 0700	DS 0710
mov cx,0008h	07115: B9 185 ╣ 07116: 08 008 BACK 07117: 00 000 NULL	IP 0005	0008h value store in cx.	CX 00 46	CX 00 08
mov ax,0001h	07118: B8 184 ╕ 07119: 01 001 ☺ 0711A: 00 000 NULL	IP 0008	0001h value store in ax.	AX 07 10	AX 00 01
mov dx,cx	0711B: 8B 139 ï 0711C: D1 209 ╤	IP 000B	The value of cx is store in dx.	DX 00 00	DX 00 08
mov ax,dx	0711D: 8B 139 ï 0711E: C2 194 ┬	IP 000D	The value of dx is store in ax.	AX 00 01	AX 00 08
div two	0711F: F6 246 ÷ 07120: 36 054 6 07121: 02 002 ☻ 07122: 00 000 NULL	IP 000F	Division of ax and two(0002h).	AX 00 08	AX 00 04
cmp ah,01h	07123: 80 128 Ç 07124: FC 252 ⁿ 07125: 01 001 ☺	IP 0013	Higher bytes of ax compare with 01h.	AX 00 04	It is not same data.
jne for_even	07126: 75 117 u 07127: 0B 011 ♂	IP 0016	Jump if previous statement is false.	-	Direct jump at for_even
sub ans,dx	07133: 29 041) 07134: 16 022 ▬ 07135: 00 000 NULL 07136: 00 000 NULL	IP 0023	Substraction of ans and dx and it's store in ans.	07100: 00 000 NULL 07101: 00 000 NULL	07100: F8 248 ° 07101: FF 255 RES
loop next	07137: E2 226 Γ 07138: E2 226 Γ	IP 0027	Repeat next loop.	-	-
mov dx,cx	0711B: 8B 139 ï 0711C: D1 209 ╤	IP 000B	The value of cx is store in dx.	DX 00 02	DX 00 01
mov ax,dx	0711D: 8B 139 ï 0711E: C2 194 ┬	IP 000D	The value of dx is store in ax.	AX 00 01	AX 00 01
div two	0711F: F6 246 ÷ 07120: 36 054 6 07121: 02 002 ☻ 07122: 00 000 NULL	IP 000F	Division of ax and two(0002h).	AX 00 01	AX 01 00
cmp ah,01h	07123: 80 128 Ç 07124: FC 252 ⁿ 07125: 01 001 ☺	IP 0013	Higher bytes of ax compare with 01h.	AX 01 00	It is same data.
jne for_even	07126: 75 117 u 07127: 0B 011 ♂	IP 0016	Jump if previous statement is false.	-	-
mov tcx,cx	07128: 89 137 ë 07129: 0E 014 ♫ 0712A: 03 003 ♥ 0712B: 00 000 NULL	IP 0018	The value of cx store in tcx.	07103: 03 003 ♥ 07104: 00 000 NULL	07103: 01 001 ☺ 07104: 00 000 NULL
call factorial_fun	0712C: E8 232 Φ 0712D: 0B 011 ♂ 0712E: 00 000 NULL	IP 001C	Call factorial function.	-	-
mov ax,0001h	0713A: B8 184 ╕ 0713B: 01 001 ☺ 0713C: 00 000 NULL	IP 002A	0001h store in ax.	AX 01 00	AX 00 01
mul cx	0713D: F7 247 ≈ 0713E: E1 225 ß	IP 002D	Multiplication of ax and cx. it's store in ax.	AX 00 01	AX 00 01
loop lb	0713F: E2 226 Γ 07140: FC 252 ⁿ	IP 002F	Decrease the value of cx and Repeat loop lb if	CX 00 01	CX 00 00

			cx is non-zero.		
add ans,ax	07141: 01 001 ☺ 07142: 06 006 ♠ 07143: 00 000 NULL 07144: 00 000 NULL	IP 0031	Addition of ans and ax.value is store in ans.	07100: 1A 026 → 07101: 14 020 ¶	07100: 1B 027 ← 07101: 14 020 ¶
ret	-	IP 0035	-	-	0710:0000 141B
mov cx,tcx	0712F: 8B 139 ï 07130: 0E 014 ♫ 07131: 03 003 ♥ 07132: 00 000 NULL	IP 001F	The value of tcx is store in cx.	CX 00 00	CX 00 01
sub ans,dx	07133: 29 041) 07134: 16 022 ▬ 07135: 00 000 NULL 07136: 00 000 NULL	IP 0023	Substraction of ans and dx and it's store in ans.	07100: 1B 027 ← 07101: 14 020 ¶	07100: 1B 027 ← 07101: 14 020 ¶
loop next	07137: E2 226 Γ 07138: E2 226 Γ	IP 0027	Repeat next loop.if cx is non-zero.	-	-
hlt	07139: F4 244 ⌠	IP 0029	Stop the code	-	the emulator is halted.

CONCLUSION:

This algorithm loops through numbers from 11 down to 1, checking if each number is even or odd. For odd numbers, it calculates the factorial and adds it to the answer (ans). For even numbers, it subtracts the number from ans. The result is stored in ans, which is updated after each iteration.

Date:

EXPERIMENT NO. 10

AIM: Write a program which perform multiplication using booth algorithm

ALGORITHM:

Input Multiplicand and Multiplier:

- **Take input for the number of bits (bits), the multiplicand (m), and the multiplier (r).**
- **Check if the input numbers can be represented within the given number of bits.**

Convert to Binary:

- **Convert the multiplicand (m) and multiplier (r) to their binary representations using the decToBin() function.**

Calculate Two's Complement:

- **Compute the two's complement of the multiplicand to use for subtraction by inverting the bits and adding 1.**

Initialize Registers:

- **Initialize A (accumulator) to all zeros.**
- **Initialize Q1 to zero (used in Booth's algorithm).**
- **Initialize Q with the binary representation of the multiplier.**

Perform Booth's Multiplication:

- **Loop for bits iterations:**
  - **Check the last bit of Q and Q1.**
  - **If Q0 is 1 and Q1 is 0, subtract M (i.e., A = A - M).**
  - **If Q0 is 0 and Q1 is 1, add M (i.e., A = A + M).**
  - **Perform an arithmetic right shift on A, Q, and Q1.**

Output Binary Result:

- **After all iterations, concatenate A and Q to form the final binary result.**

Convert Binary to Decimal:

- **If the result is negative, convert the two's complement to a positive number and add a negative sign.**
- **Print the decimal result.**

End Program:

- **Output the result in both binary and decimal formats and exit the program.**

CODE:

```
#include <iostream>
#include <string>
#include <cmath>

using namespace std;

// Function to convert decimal to binary
string decToBin(int num, int bits) {
string bin = "";
for (int i = bits - 1; i >= 0; i--) {
bin += (num & (1 << i)) ? '1' : '0';  // Check each bit from left to right
}
return bin;
}

// Function to calculate 2's complement of a binary string
string twosComp(string bin) {
// Invert the bits (1's complement)
for (int i = 0; i < bin.size(); i++) {
bin[i] = (bin[i] == '1') ? '0' : '1';  // Flip bits (1 -> 0, 0 -> 1)
}

// Add 1 to get 2's complement
int carry = 1;
for (int i = bin.size() - 1; i >= 0; i--) {
if (bin[i] == '0' && carry == 1) {
bin[i] = '1';
carry = 0;  // No more carry, stop here
} else if (bin[i] == '1' && carry == 1) {
bin[i] = '0';  // Still a carry, continue
}
}

return bin;
}

// Function to add two binary numbers
string binAdd(string a, string b) {
string result = "";
int carry = 0;

for (int i = a.size() - 1; i >= 0; i--) {
int bitA = a[i] - '0'; // Convert char to int
int bitB = b[i] - '0';
int sum = bitA + bitB + carry;

result = char((sum % 2) + '0') + result; // Add the sum bit to result
carry = sum / 2;  // Calculate carry for next iteration
}
```

```
return result;
}

// Function to perform arithmetic right shift on A and Q
void rightShift(string &A, string &Q, char &Q1) {
Q1 = Q.back();  // Store the last bit of Q (Q0)
Q = A.back() + Q.substr(0, Q.size() - 1);  // Shift Q by 1 bit
A = A[0] + A.substr(0, A.size() - 1);       // Shift A by 1 bit (preserve sign bit)
}

// Booth's Algorithm implementation
string boothMult(int m, int r, int bits) {
// Convert multiplicand (M) and multiplier (Q) to binary
string M = decToBin(m, bits);
string Q = decToBin(r, bits);
string A(bits, '0'); // Initialize A to 0 (same size as M and Q)
char Q1 = '0';  // Initialize Q-1 to 0

// Compute -M using 2's complement
string negM = twosComp(M);

// Print initial values
cout << "Initial values:\n";
cout << "M  = " << M << "\n";
cout << "-M = " << negM << "\n";
cout << "Q  = " << Q << "\n";
cout << "A  = " << A << "\n";

// Perform the algorithm for 'bits' number of steps
for (int i = 0; i < bits; i++) {
// Check the last bit of Q (Q0) and Q-1
if (Q.back() == '1' && Q1 == '0') {
A = binAdd(A, negM);  // A = A - M
} else if (Q.back() == '0' && Q1 == '1') {
A = binAdd(A, M);  // A = A + M
}

// Right shift A, Q, and Q1
rightShift(A, Q, Q1);

// Print step-by-step results
cout << "\nStep " << i + 1 << ":\n";
cout << "A = " << A << " Q = " << Q << " Q1 = " << Q1 << "\n";
}

return A + Q;  // Return the concatenated result of A and Q
}

// Function to check if a number can be represented in 'bits' number of bits
bool isInRange(int num, int bits) {
int maxVal = pow(2, bits - 1) - 1; // Max value for signed integer
int minVal = -pow(2, bits - 1);      // Min value for signed integer
return (num >= minVal && num <= maxVal);
```

```
}

int main() {
int bits, multiplicand, multiplier;

// Input number of bits and numbers
cout << "Enter size of binary number (number of bits): ";
cin >> bits;
cout << "Enter multiplicand: ";
cin >> multiplicand;
cout << "Enter multiplier: ";
cin >> multiplier;

// Check if the input numbers are within range for the given bit size
if (!isInRange(multiplicand, bits) || !isInRange(multiplier, bits)) {
cout << "Error: Numbers are out of range for " << bits << " bits." << endl;
return 1;  // Exit if inputs are invalid
}

// Perform multiplication using Booth's Algorithm
string result = boothMult(multiplicand, multiplier, bits);

// Print binary result
cout << "\nBinary result: " << result << endl;

// Convert binary result to decimal
int decimalResult = (result[0] == '1') ? -stoi(twosComp(result), nullptr, 2) : stoi(result, nullptr, 2);
cout << "Decimal result: " << decimalResult << endl;

cout << "Program is created by DHRUV_LOKADIYA_23CS041" << endl;

return 0;
}
```

OUTPUTS:

```
Enter size of binary number (number of bits): 8
Enter multiplicand: 25
Enter multiplier: 30
Initial values:
M  = 00011001
-M = 11100111
Q  = 00011110
A  = 00000000

Step 1:
A = 00000000 Q = 00001111 Q1 = 0

Step 2:
A = 11110011 Q = 10000111 Q1 = 1

Step 3:
A = 11111001 Q = 11000011 Q1 = 1

Step 4:
A = 11111100 Q = 11100001 Q1 = 1

Step 5:
A = 11111110 Q = 01110000 Q1 = 1

Step 6:
A = 00001011 Q = 10111000 Q1 = 0

Step 7:
A = 00000101 Q = 11011100 Q1 = 0

Step 8:
A = 00000010 Q = 11101110 Q1 = 0

Binary result: 0000001011101110
Decimal result: 750
Program is created by DHRUV LOKADIYA-23CS041
```

```
Enter size of binary number (number of bits): 4
Enter multiplicand: 40
Enter multiplier: 41
Error: Numbers are out of range for 4 bits.
```

```
Enter size of binary number (number of bits): 6
Enter multiplicand: 15
Enter multiplier: 4
Initial values:
M  = 001111
-M = 110001
Q  = 000100
A  = 000000

Step 1:
A = 000000 Q = 000010 Q1 = 0

Step 2:
A = 000000 Q = 000001 Q1 = 0

Step 3:
A = 111000 Q = 100000 Q1 = 1

Step 4:
A = 000011 Q = 110000 Q1 = 0

Step 5:
A = 000001 Q = 111000 Q1 = 0

Step 6:
A = 000000 Q = 111100 Q1 = 0

Binary result: 000000111100
Decimal result: 60
Program is created by DHRUV LOKADIYA-23CS041
```

CONCLUSION:

This code implements Booth's Algorithm for multiplying two integers using binary arithmetic. It efficiently handles signed numbers, converting them to binary, performing bitwise operations, and using two's complement for subtraction. The result is displayed in both binary and decimal formats. It also ensures the inputs fit within the specified bit size.

Date:

EXPERIMENT NO. 11

AIM: Write a program to convert a given number system to other number system.

TASK:
Ask user to choose one form given number system: Any radix
Ask user to enter a number (number can be float) for that number system.
Check that number entered for given number system is correct.
If not give the error
If correct then convert entered number in required radix number system
If you can show the conversion, then it's well and good.

NOTE:
Program can be implemented using any programming language.
You have to submit the program in assignment section of team after wards.
If you turn in Late Marks will be deducted.

SAMPLE CALCULATIONS OF NUMBER CONVERSIONS:

ALGORITHM:

Input Base and Number:

- **Prompt the user to input the base of the number (fromBase) and the number in that base (numStr).**
- **Validate that the base is between 2 and 36.**

Validate Number:

- **For each character in the string numStr:**
  - **If the character is a digit (0-9), check if its value is within the given base.**
  - **If the character is a letter (A-Z or a-z), ensure it represents a valid digit for the base (e.g., for base 16, letters should only range from A to F).**
  - **If the character is invalid, print an error and exit.**

Convert to Decimal:

- **Initialize decimalValue to 0.**
- **For each digit in numStr, convert it to its decimal equivalent using its base and the appropriate power of the base (taking into account both integer and fractional parts if present).**
- **Update decimalValue as the sum of all digit values times their respective powers of the base.**

Input Target Base:

- **Prompt the user to input the target base (toBase) to which they want to convert the number.**
- **Validate that the base is between 2 and 36.**

Convert Decimal to Target Base:

- **Separate the integer and fractional parts of decimalValue.**
- **Convert the integer part by repeatedly dividing by toBase and storing the**

remainders (digits), which are later reversed to form the result.

- **Convert the fractional part by multiplying it by toBase, extracting the integer part, and repeating this process for a fixed number of digits or until the fractional part becomes zero.**

Output the Result:

- **Print the converted number in the target base.**

Program End:

- **End the program after printing the result and exit gracefully.**

CODE:

```
#include   <stdio.h>
#include  <stdlib.h>
#include <string.h>
#include <math.h>

// Function prototypes
int isValidNumber(const char *numStr, int base);
double convertToDecimal(const char *numStr, int base);
void convertFromDecimal(double num, int base);

// Main function
int main() {
int fromBase, toBase;
char numStr[50];

printf("Enter the base of the number system (2-36): ");
scanf("%d", &fromBase);
if (fromBase < 2 || fromBase > 36) {
printf("Invalid base!\n");
return 1;
}

printf("Enter the number: ");
scanf("%s", numStr);

if (!isValidNumber(numStr, fromBase)) {
printf("Invalid number for the given base!\n");
return 1;
}

double decimalValue = convertToDecimal(numStr, fromBase);

printf("Enter the base to convert to (2-36): ");
scanf("%d", &toBase);
if (toBase < 2 || toBase > 36) {
printf("Invalid base!\n");
return 1;
}
```

```
printf("The number %s in base %d is ", numStr, fromBase);
convertFromDecimal(decimalValue, toBase);
printf(" in base %d.\n", toBase);
printf("\n This file prepared by 23CS024-Hardik Hadiya.");

return 0;
}

// Function to check if the number is valid for the given base
int isValidNumber(const char *numStr, int base) {
for (int i = 0; i < strlen(numStr); i++) {
char c = numStr[i];
if (c >= '0' && c <= '9') {
if (c - '0' >= base) return 0;
} else if (c >= 'A' && c <= 'Z') {
if (c - 'A' + 10 >= base) return 0;
} else if (c >= 'a' && c <= 'z') {
if (c - 'a' + 10 >= base) return 0;
} else if (c == '.') {
// Allow decimal point in floating numbers
continue;
} else {
return 0;
}
}
return 1;
}

// Function to convert a number in a given base to decimal
double convertToDecimal(const char *numStr, int base) {
double decimalValue = 0;
int len = strlen(numStr);
int point = -1;

for (int i = 0; i < len; i++) {
if (numStr[i] == '.') {
point = i;
break;
}
}

int power = (point == -1) ? len - 1 : point - 1;

for (int i = 0; i < len; i++) {
char c = numStr[i];
if (c == '.') continue;
int digitValue = (c >= '0' && c <= '9') ? c - '0' :
(c >= 'A' && c <= 'Z') ? c - 'A' + 10 : c - 'a' + 10;
decimalValue += digitValue * pow(base, power--);
}

return decimalValue;
```

```
}

// Function to convert a decimal number to a given base
void convertFromDecimal(double num, int base) {
char result[50];
int index = 0;
long intPart = (long)num;
double fracPart = num - intPart;

// Convert integer part
while (intPart > 0) {
int digitValue = intPart % base;
result[index++] = (digitValue < 10) ? digitValue + '0' : digitValue + 'A' - 10;
intPart /= base;
}
result[index] = '\0';

// Reverse the integer part of the result
for (int i = 0, j = index - 1; i < j; i++, j--) {
char temp = result[i];
result[i] = result[j];
result[j] = temp;
}

// Add fractional part
if (fracPart > 0) {
strcat(result, ".");
for (int i = 0; i < 10 && fracPart > 0; i++) {
fracPart *= base;
int digitValue = (int)fracPart;
fracPart -= digitValue;
char digitChar = (digitValue < 10) ? digitValue + '0' : digitValue + 'A' - 10;
result[strlen(result) + 1] = '\0';
result[strlen(result)] = digitChar;
}
}

printf("%s", result);
}
```

OUTPUTS:

1. Decimal to Binary:

```
Enter the base of the number system (2-36): 10
Enter the number: 24
Enter the base to convert to (2-36): 2
The number 24 in base 10 is 11000 in base 2.

 This file prepared by 23CS024-Hardik Hadiya.
```

2. Decimal to Octal:

```
Enter the base of the number system (2-36): 10
Enter the number: 24
Enter the base to convert to (2-36): 8
The number 24 in base 10 is 30 in base 8.

 This file prepared by 23CS024-Hardik Hadiya.
```

3. Decimal to Hexadecimal:

```
Enter the base of the number system (2-36): 10
Enter the number: 24
Enter the base to convert to (2-36): 16
The number 24 in base 10 is 18 in base 16.

 This file prepared by 23CS024-Hardik Hadiya.
```

4. Binary to Decimal:

```
Enter the base of the number system (2-36): 2
Enter the number: 11000
Enter the base to convert to (2-36): 10
The number 11000 in base 2 is 24 in base 10.

 This file prepared by 23CS024-Hardik Hadiya.
```

5. Base-5 to Base-3:

```
Enter the base of the number system (2-36): 5
Enter the number: 24
Enter the base to convert to (2-36): 3
The number 24 in base 5 is 112 in base 3.

 This file prepared by 23CS024-Hardik Hadiya.
```

6. If ,we select 24 numeric value of Base-3,than give invalid number.

```
Enter the base of the number system (2-36): 3
Enter the number: 24
Invalid number for the given base!
```

CONCLUSION:

This C program converts numbers between different bases (from base 2 to base 36). It validates the input number, converts it to a decimal, and then re-converts the decimal to the desired target base. The program can handle both integer and fractional parts of numbers, ensuring precise conversions across a wide range of bases.