Lab File

Digital Electronics and Computer Organisation

(CSE 207)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Submitted to:

Ms Neha Arora

Ast. Professor

ECE Department, ASET

Submitted by:

Shaina Mehta

A2305219268

B.tech. C.S.E.

3CSE-4Y

AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY

AMITY UNIVERSITY UTTAR PRADESH

NOIDA -201301

Exp No	Assignment Category	Code	Name of Experiment	Date of Allotment	Date of Evaluation	Max Marks	Marks Obtained	Faculty Sign
1	Mandatory Experiment		To verify and interpret the logic and truth table for AND, OR, NOT, NAND, NOR, Ex-OR, Ex-NOR gates.	17-07-2020	10-08-2020			
2			To design the half adder using both the gates and verify its operation.	24-07-2020	10-08-2020			
3			To design full adder by using XOR and NAND gates and verify its truth table.	31-07-2020	10-08-2020			
4			To design the half subtractor and full subtractor circuit by using the XOR, NOT, AND and NAND gates and verify their truth table.	7-08-2020	13-08-2020			
5			To design the logical unit of A.L.U.	14-08-2020	20-08-2020			
6			To design a binary adder and subtractor using full adder circuit.	21-08-2020	27-08-2020			
7			To design and simulate 4-bit BCD adder.	4-09-2020	24-09-2020			
8			To design 4-bit bus using three-state buffer gate and decoder.	18-09-2020	24-09-2020			
9			To design a 4-bit bus using multiplexers.	25-09-2020	16-10-2020			
10			To design and simulate 4-to2-Priority	2-10-2020	16-10-2020			

			Encoder using logic gates.					
11			To study the architecture and instruction set of 8085 Microprocessor.	16-10-2020	22-10-2020			
	Viva	Viva						

Experiment 1

Aim: To verify and interpret the logic and truth table for AND, OR, NOT, NAND, NOR, Ex-OR, Ex-NOR gates.

Tools Used: Circuit Verse and Virtual Labs.

Theory: Logic gates are the basic building blocks of any digital system. Logic gates are electronic circuits having one or more than one input and only one output. The relationship between the input and the output is based on a certain logic. Based on this, logic gates are named as

- AND gate
- OR gate
- NOT gate
- NAND gate
- NOR gate
- Ex-OR gate
- Ex-NOR gate

AND gate: The AND gate is an electronic circuit that gives a high output (1) only if all its inputs are high and low output (0) when all or any one input is low. A dot (.) is used to show the AND operation i.e. A.B or can be written as AB.

Y=A.B

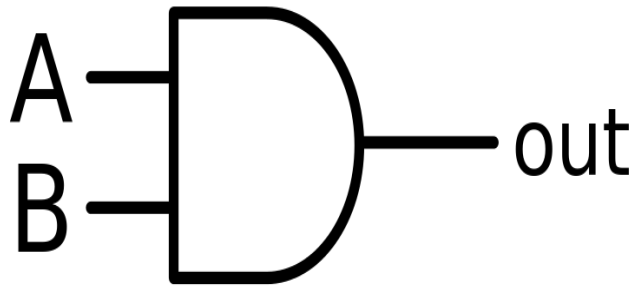


Fig 1: Symbol of AND Gate

A	B	Y(Output)
0	0	0
0	1	0
1	0	0
1	1	1

Table 1: Truth Table of AND Gate

OR gate: The OR gate is an electronic circuit that gives a high output (1) if one or more of its inputs are high and gives lower output (0) when if all the inputs are low. A plus (+) is used to show the OR operation.

Y= A+B

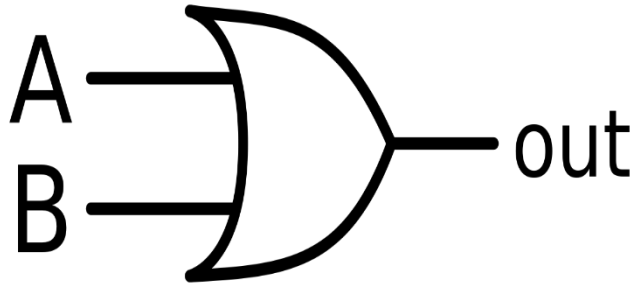


Fig 2: Symbol of OR Gate

A	B	Y(Output)
0	0	0
0	1	1
1	0	1
1	1	1

Table 2: Truth Table of OR Gate

NOT gate: The NOT gate is an electronic circuit that produces an inverted version of the input at its output i.e. it gives low output (0) when high input (1) is passed and it gives high output (1) when low input (0) is passed. It is also known as an inverter. If the input variable is A, the inverted output is known as NOT A. This is also shown as A' or A with a bar over the top, as shown at the outputs.

Y= A'

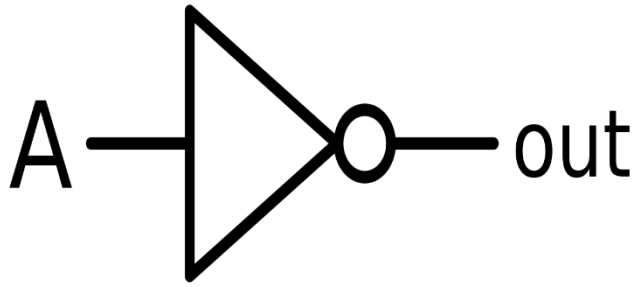


Fig 3: Symbol of NOT Gate

A	Y(Output)
0	1
1	0

Table3: Truth Table of NOT Gate

NAND gate: This is a NOT-AND gate which is equal to an AND gate followed by a NOT gate. The outputs of all NAND gates are high if any of the inputs are low. The symbol is an AND gate with a small circle on the output. The small circle represents inversion.

Y= (A.B)'

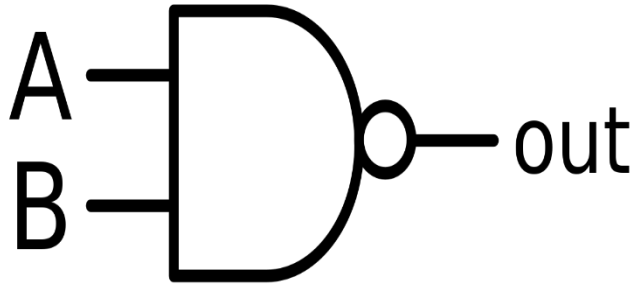


Fig 4: Symbol of NAND Gate

A	B	Y(Output)
0	0	1
0	1	1
1	0	1
1	1	0

Table 4: Truth Table of NAND Gate

NOR gate: This is a NOT-OR gate which is equal to an OR gate followed by a NOT gate. The outputs of all NOR gates are low if any of the inputs are high. The symbol is an OR gate with a small circle on the output. The small circle represents inversion.

Y= (A+B)'

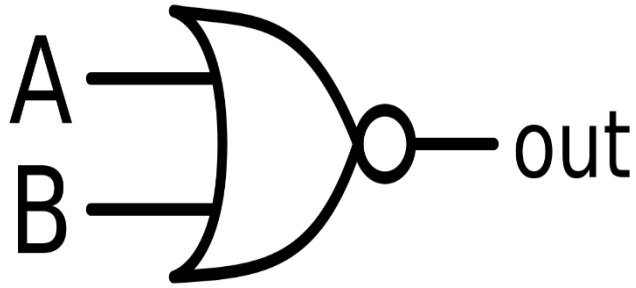


Fig 5: Symbol of NOR Gate

A	B	Y(Output)
0	0	1
0	1	1
1	0	1
1	1	1

Table 5: Truth Table of NOR Gate

Note: NAND and NOR Gate are Universal Gates. Universal Gate is a gate which can implement any Boolean function without need to use any other gate type.

Ex-OR gate: The 'Exclusive-OR' gate is a circuit which will give a high output if either, but not both of its two inputs are high. An encircled plus sign (⊕) is used to show the Ex-OR operation.

Ex-OR gate is created from AND, NAND and OR gates. The output is high only when both the inputs are different.

Y= A⊕B

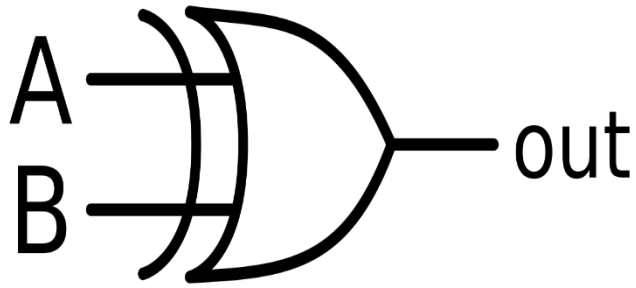


Fig 6: Symbol of Ex-OR Gate

A	B	Y(Output)
0	0	0
0	1	1
1	0	1
1	1	0

Table 6: Truth Table of XOR Gate

Ex-NOR gate: The 'Exclusive-NOR' gate circuit does the opposite to the EX-OR gate. It will give a low output if either, but not both of its two inputs are high. The symbol is an EX-OR gate with a small circle on the output. The small circle represents inversion. Ex-NOR gate is created from AND, NOT and OR gates. The output is high only when both the inputs are same.

Y= (A⊕B)'

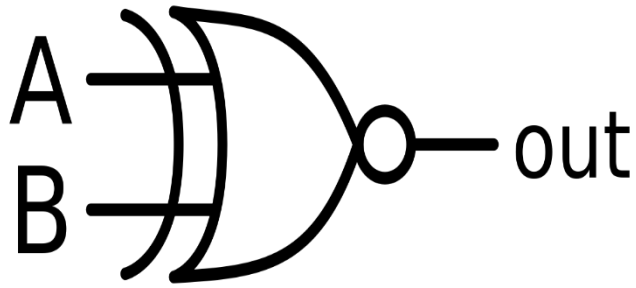


Fig 7: Symbol of Ex-NOR Gate

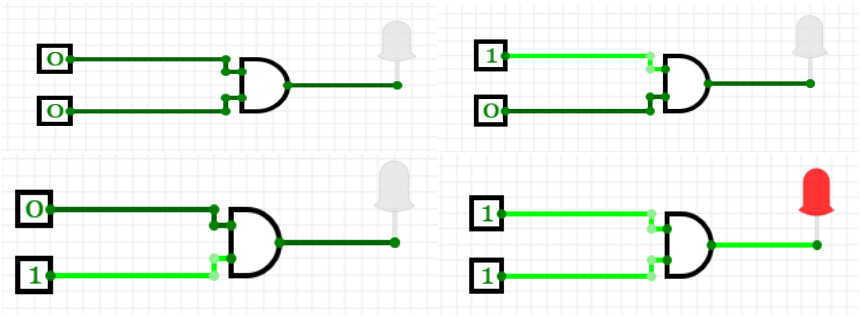
A	B	Y(Output)
0	0	1
0	1	0
1	0	0
1	1	1

Table 7: Truth Table of XNOR Gates

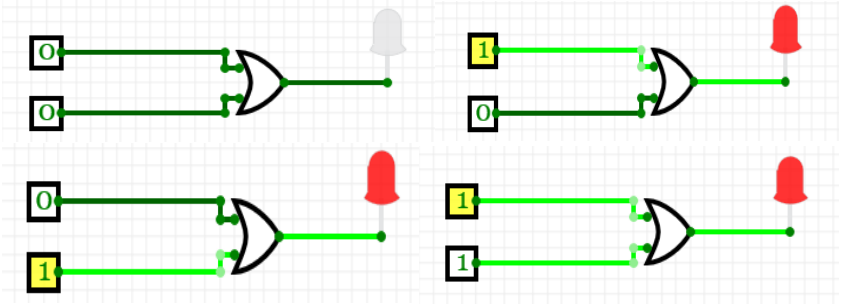
Observations:

Circiut Represenation of:

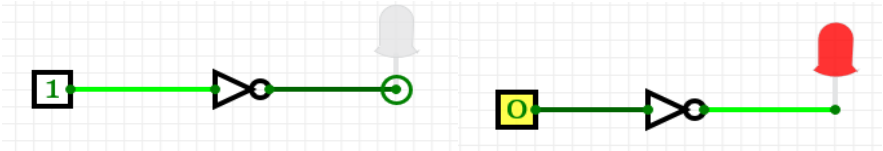
(A) AND Gate:



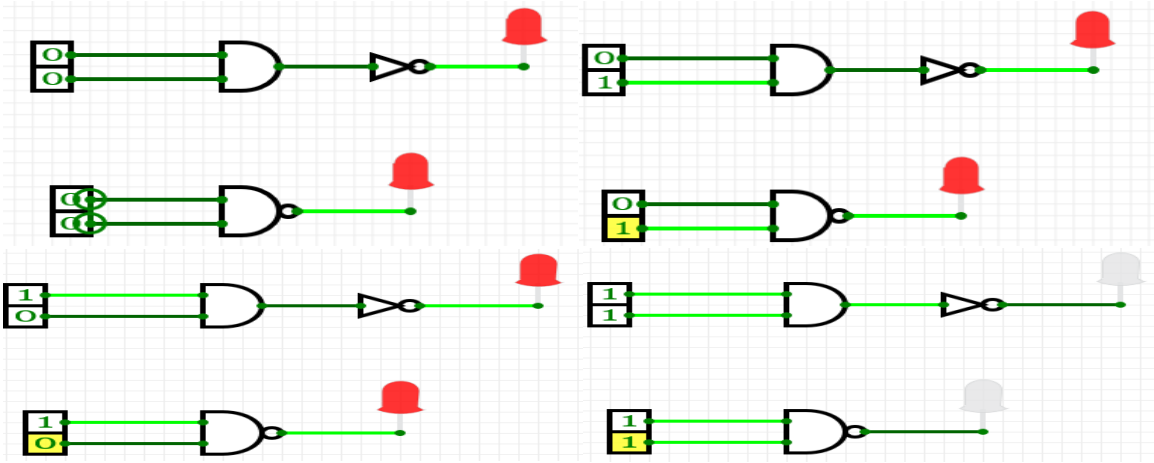
(B) OR Gate:



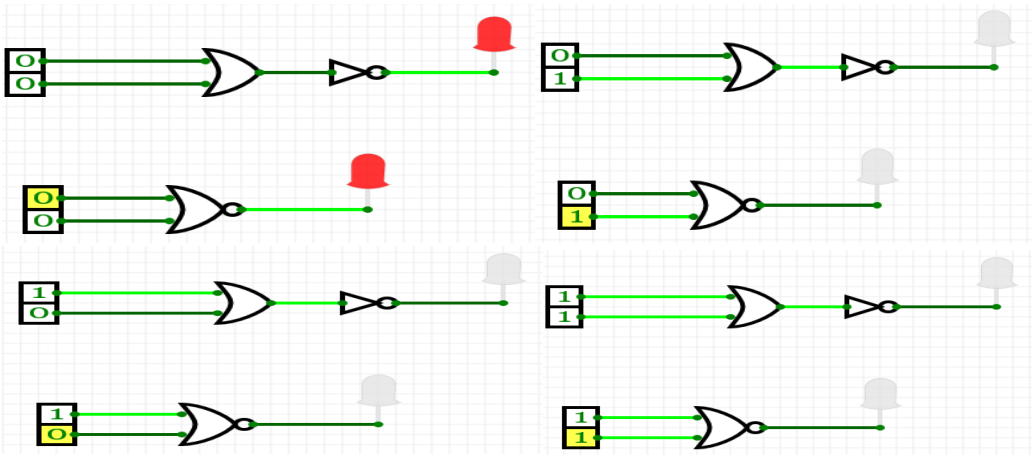
(C) NOT Gate:



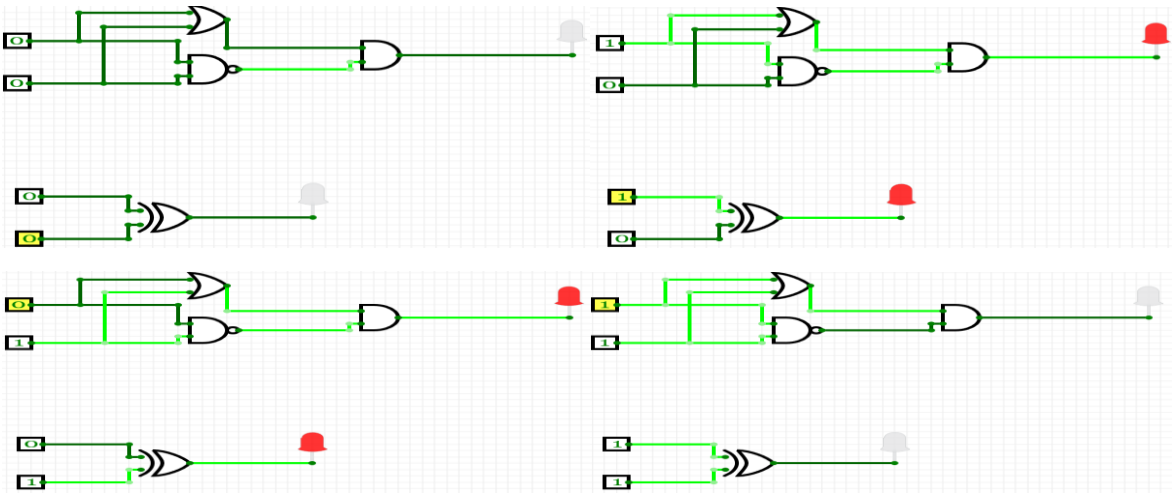
(D) NAND Gate:



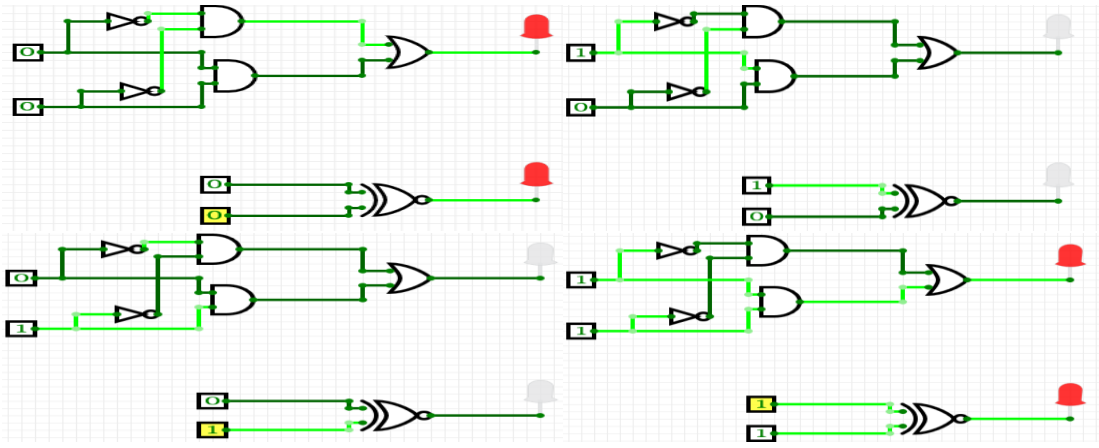
(E) NOR Gate:



(F) Ex-OR:



(G) Ex-NOR:



Result: The truth table of AND, OR, NOT, NAND, NOR, Ex-OR, and Ex-NOR gates have been verified.

CRITERIA	TOTAL MARKS	MARKS OBTAINED	COMMENTS
(A) CONCEPT	2		
(B) IMPLEMENTATION	2		
(C) PERFORMANCE	2		
TOTAL	6		

Experiment 2

Aim: To design the half adder using both the gates and verify its operation.

Tools Used: Virtual Labs and Circuit Verse.

Theory: Adders digital circuits that carry out addition of numbers. Adders are a key component of arithmetic logic unit. Adders can be constructed for most of the numerical representations like Binary Coded Decimal (BDC), are Excess – 3, Gray code, Binary etc. out of these, binary addition is the most frequently performed task by most common adders. Apart from addition, adders are also used in certain digital applications like table index calculation, address decoding etc. Binary addition is similar to that of decimal addition. Some basic binary additions are shown below.

0 0 1 1

+0 +1 +0 +1

0 1 1 (carry) 10

Fig 1: Schematic Representation of Half Adder

Half Adder Circuits: Half adder is a combinational circuit that performs simple addition of two binary numbers. The block diagram of a half adder is shown below.

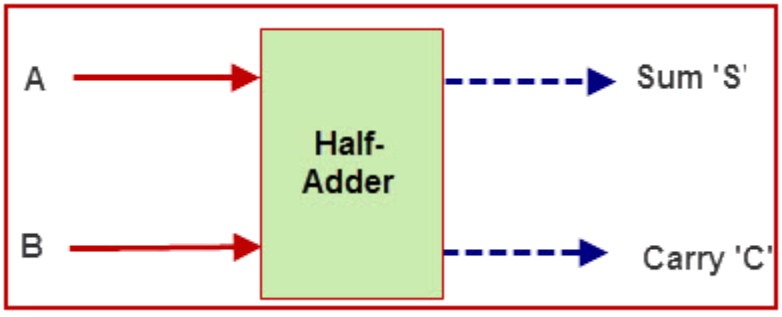
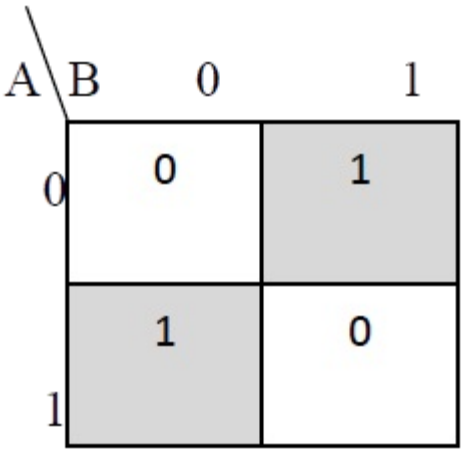


Fig 2: Block Diagram of Half Adder Circuit

If we assume A and B as the two bits whose addition is to be performed, then the sum output of the binary addition carried out above is similar to that of an Ex-OR operation while the carry output is similar to that of an AND operation. The same can be verified with help of Karnaugh Map.

A	B	Sum
0	0	0
0	1	1
1	0	1
1	1	0

Table1: Truth Table of Sum Output Line of the Half Adder Circuit



Sum=AB'+A'B.

Fig 3: K-map Representation of the Sum Output Line of the Half Adder Circuit

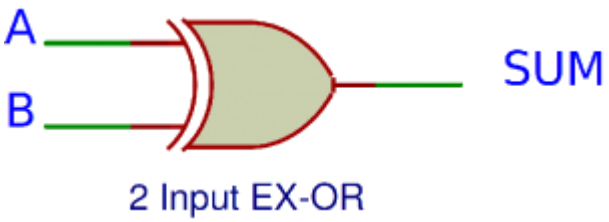
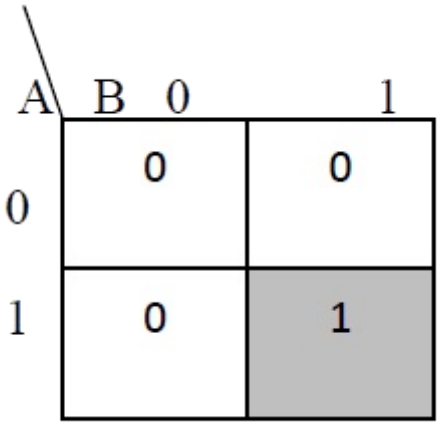


Fig 4: Logic Gate Representation of the Sum Output of the Half Adder Circuit

A	B	Carry
0	0	0
0	1	0
1	0	0
1	1	1

Table 2: Truth Table of Carry Output Line of the Half Adder Circuit



Carry=A.B.

Fig 5: K-map Representation of the Carry Output Line of the Half Adder Circuit

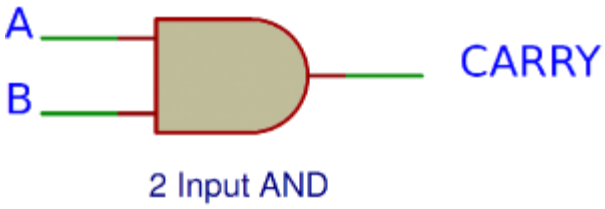


Fig 6: Logic Gate Representation of the Sum Output of the Half Adder Circuit

Half Adder Circuit Using AND and XOR Gates: If A and B are binary inputs to the half adder, then the logic function to calculate sum S is Ex – OR of A and B and logic function to calculate carry C is AND of A and B. Combining these two, the logical circuit to implement the combinational circuit of half adder.

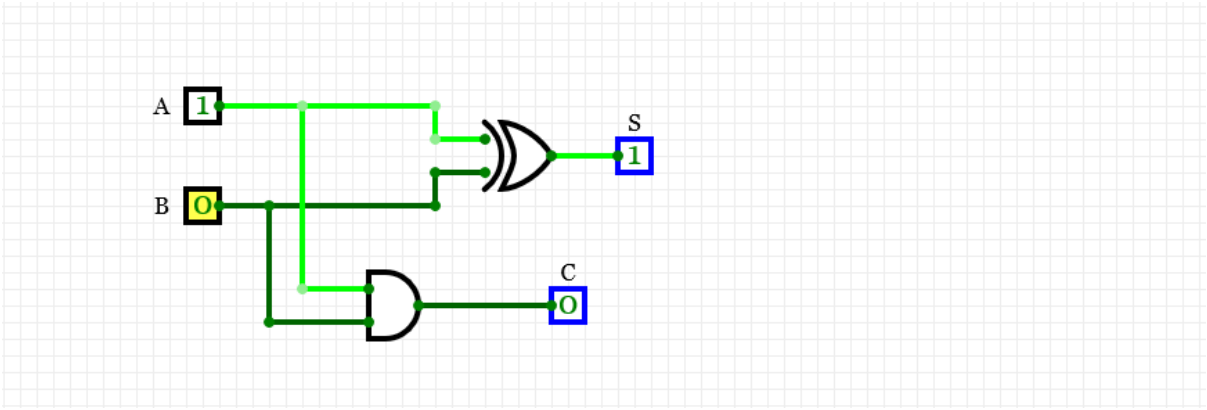
As we know that NAND and NOR are called universal gates as any logic system can be implemented using these two, the half adder circuit can also be implemented using them. We know that a half adder circuit has one Ex – OR gate and one AND gate. Here, we show the designing and implementation of Half Adder Circuit using NAND gate only.

Half Adder Circuit Using NAND Gate: Five to seven NAND gates are required in order to design a half adder. The circuit to realize half adder using NAND gates.

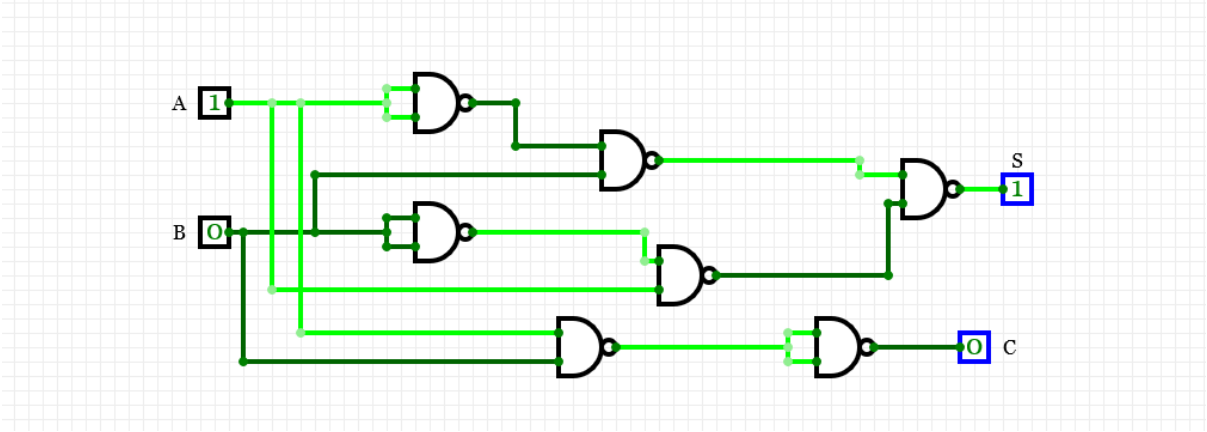
Observations:

Circuit representation of Half Adder Circuit using:

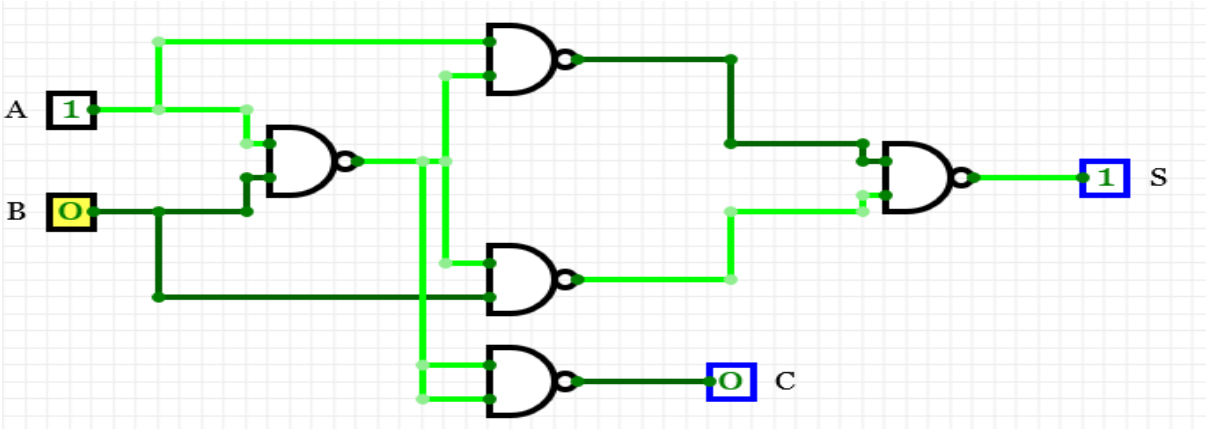
(A) XOR and AND Gates:



(B) NAND Gate:



OR



Result: The operation of Half Adder Circuit has been verified successfully.

CRITERIA	TOTAL MARKS	MARKS OBTAINED	COMMENTS
(A) CONCEPT	2		
(B) IMPLEMENTATION	2		
(C) PERFORMANCE	2		
TOTAL	6		

Experiment 3

Aim: To design full adder by using XOR and NAND gates and verify its truth table.

Tools Used: Virtual Labs and Circuit Verse.

Theory: Adders digital circuits that carry out addition of numbers. Adders are a key component of arithmetic logic unit. Adders can be constructed for most of the numerical representations like Binary Coded Decimal (BDC), are Excess – 3, Gray code, Binary etc. out of these, binary addition is the most frequently performed task by most common adders. Apart from addition, adders are also used in certain digital applications like table index calculation, address decoding etc. Binary addition is similar to that of decimal addition. Some basic binary additions are shown below.

0 0 1 1

+0 +1 +0 +1

0 1 1 (carry) 10

Fig 1: Schematic Representation of Half Adder

Full adder Circuits: Full adder is a digital circuit used to calculate the sum of three binary bits which is the main difference between full adder and half adder. Full adders are complex and difficult to implement when compared to half adders. Two of the three bits are same as before which are A, the augend bit and B, the addend bit. The additional third bit is carry bit from the previous stage and is called 'Carry' – in generally represented by CIN. It calculates the sum of three bits along with the carry. The output carry is called Carry – out and is represented by COUT. The block diagram of a full adder with A, B and CIN as inputs and S, COUT as outputs is shown below.

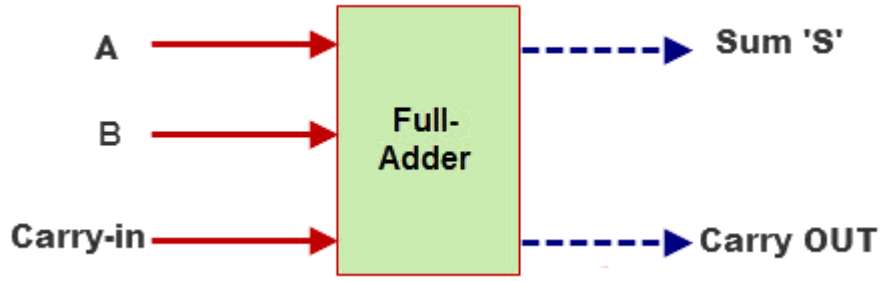
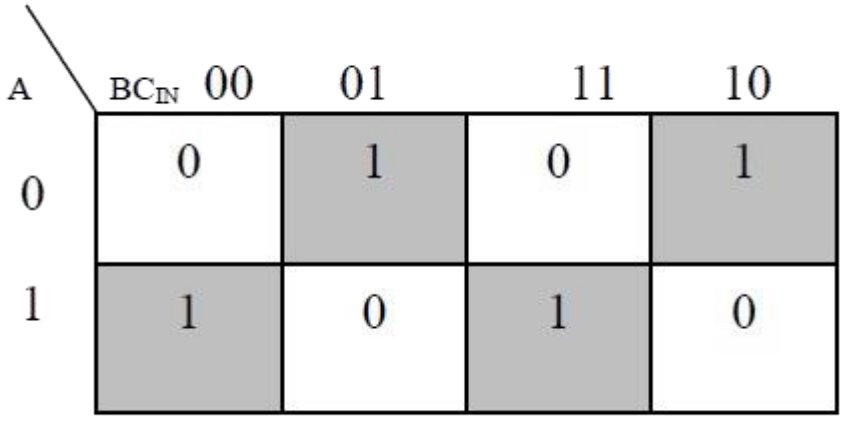


Fig 2: Block Diagram of Full Adder Circuit

Input			Output	
A	B	Cin	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

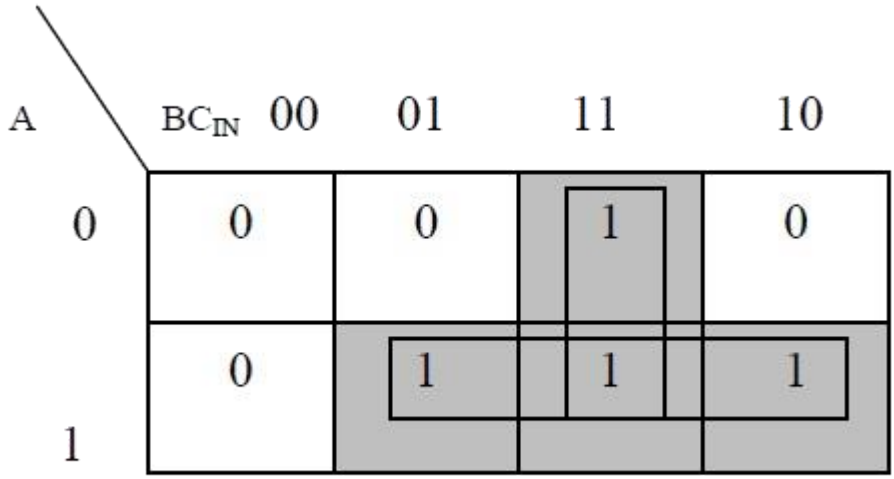
Table 1: Truth Table of Full Adder Circuit

Based on the truth table, the Boolean functions for Sum (S) and Carry – out (COUT) can be derived using K – Map.



The simplified equation for sum is $S = A'B'C_{IN} + A'BC_{IN}' + ABC_{IN} + AB'C_{IN}$

Fig 2: K-map Representation of the Sum Output Line of the Full Adder Circuit



The simplified equation for COUT is COUT = AB + AC_{IN} + BC_{IN}

Fig 3: K-map Representation of the Carry Output Line of the Full Adder Circuit

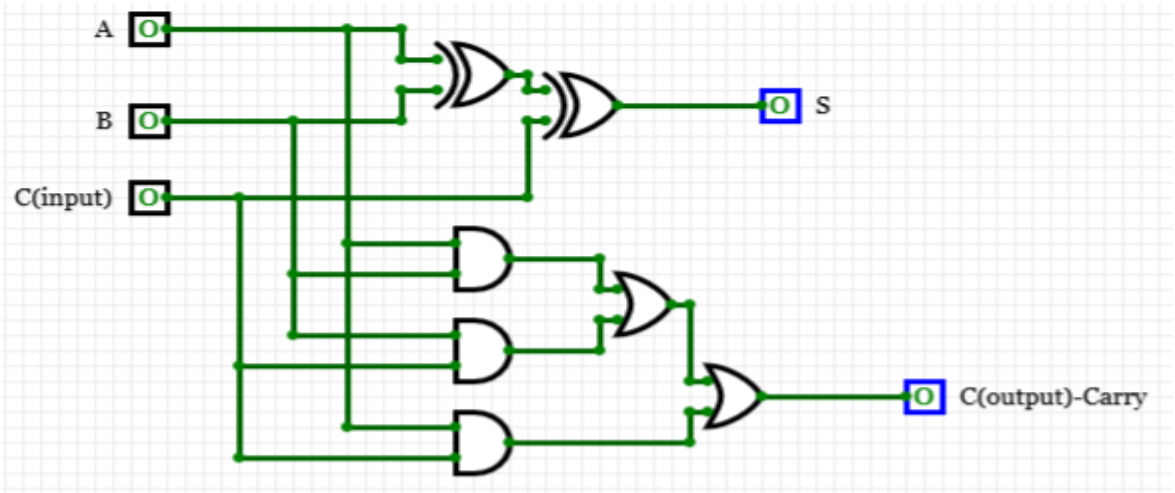
Full Adder Circuit Using XOR and AND Gate: In order to implement a combinational circuit for full adder, it is clear from the equations derived above, that we need four 3-input AND gates and one 4-input OR gates for Sum and three 2-input AND gates and one 3-input OR gate for Carry – out.

Full Adder Circuit Using NAND Gate: As mentioned earlier, a NAND gate is one of the universal gates and can be used to implement any logic design.

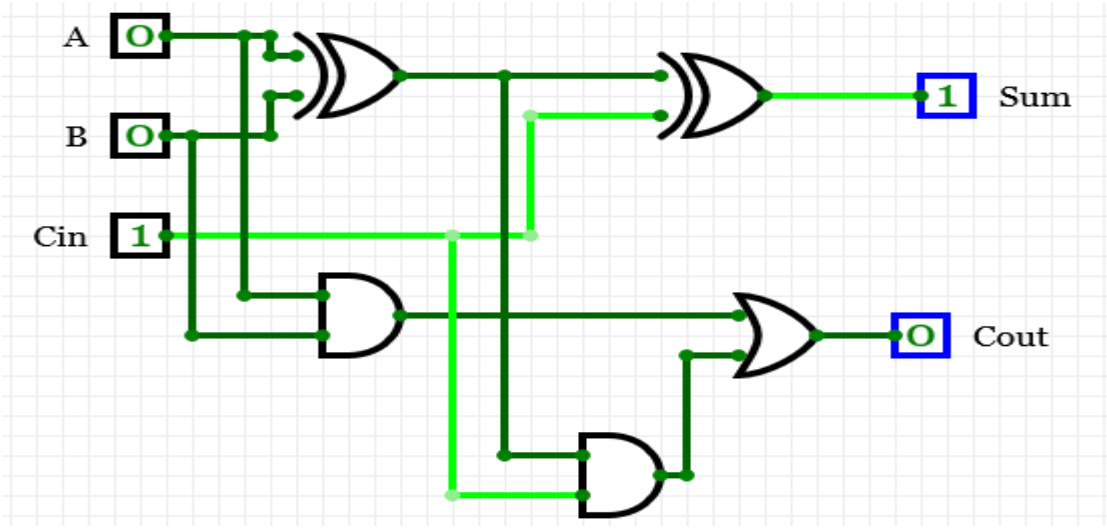
Circuit Representation:

Circuit representation of full adder circuit using:

(A) XOR and AND Gates:

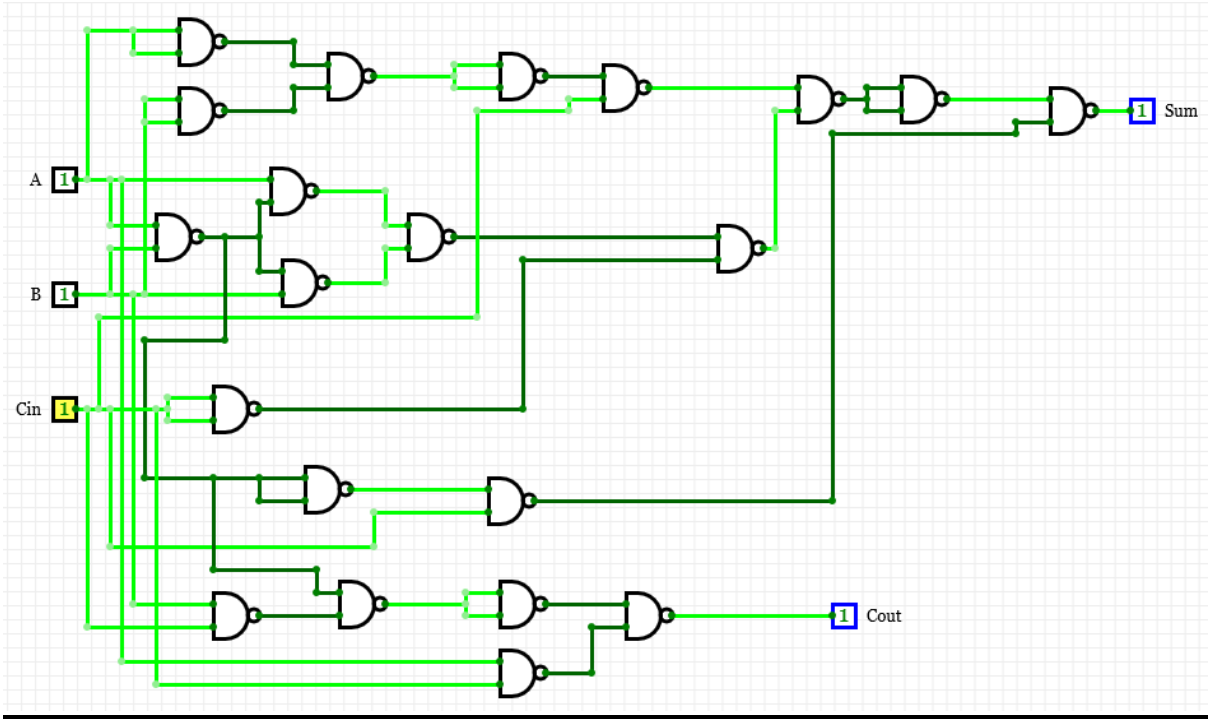


OR



(By Using Half Adder Circuit)

(B) NAND Gate:



Result: The operation of Full Adder Circuit has been verified successfully.

CRITERIA	TOTAL MARKS	MARKS OBTAINED	COMMENTS
(A) CONCEPT	2		
(B) IMPLEMENTATION	2		
(C) PERFORMANCE	2		
TOTAL	6		

Experiment 4

Aim: To design the half subtractor and full subtractor circuit by using the XOR, NOT, AND and NAND gates and verify their truth table.

Theory: Subtractor circuits take two binary numbers as input and subtract one binary number input from the other binary number input. Similar to adders, it gives out two outputs, difference and borrow (carry-in the case of Adder). There are two types of subtractors.

- Half Subtractor
- Full Subtractor

Half Subtractor Circuit: The half-subtractor is a combinational circuit which is used to perform subtraction of two bits. It has two inputs, X (minuend) and Y (subtrahend) and two outputs D (difference) and B (borrow). The logic symbol and truth table are shown below.

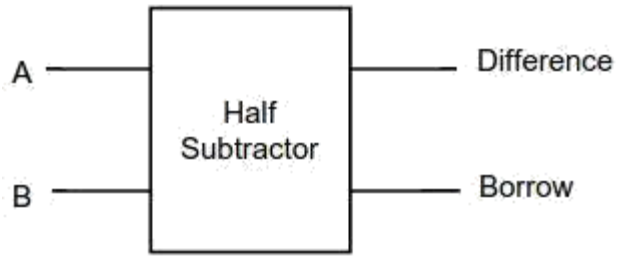


Fig 1: Block Diagram of Half Subtractor Circuit

Input		Output	
A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Table 1: Truth Table of Half Subtractor Circuit

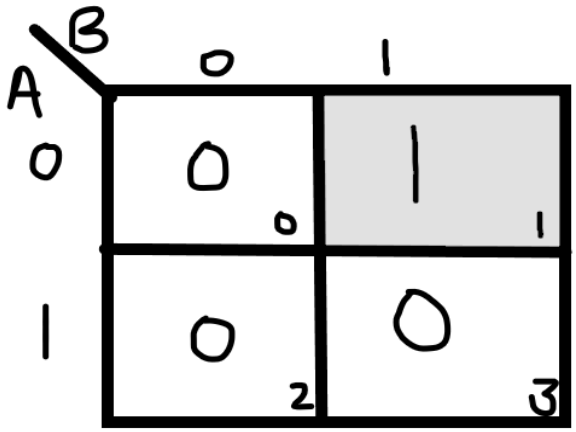


Fig: K-map for Difference Output Line for Half Subtractor Circuit

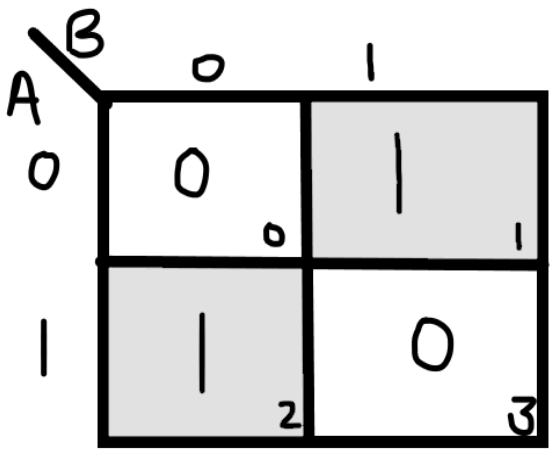


Fig: K-map of Borrow Output Line for the Half Subtractor Circuit

From the above truth table and K- map we can find the boolean expression as:

$$\mathbf{D = A \oplus B}$$

$$\mathbf{Bout = A' Y}$$

From the equation we can draw the half-subtractor circuit using XOR, OR, AND and NAND gates.

Full Subtractor Circuit: A full subtractor is a combinational circuit that performs subtraction involving three bits, namely minuend, subtrahend, and borrow-in. It accepts three inputs: minuend, subtrahend and a borrow bit and it produces two outputs: difference and borrow. The logic symbol and truth table are shown below.

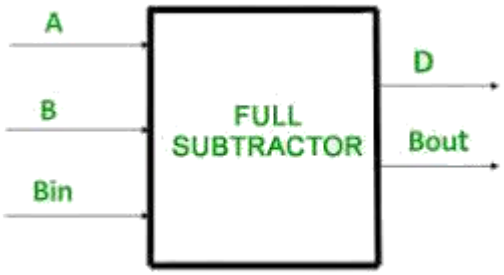


Fig 2: Block Diagram of Full Subtractor Circuit

Input			Output	
A	B	Borrow Input (Bin)	Difference (D)	Borrow Output (Bout)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Table 2: Truth Table of Full Subtractor Circuit

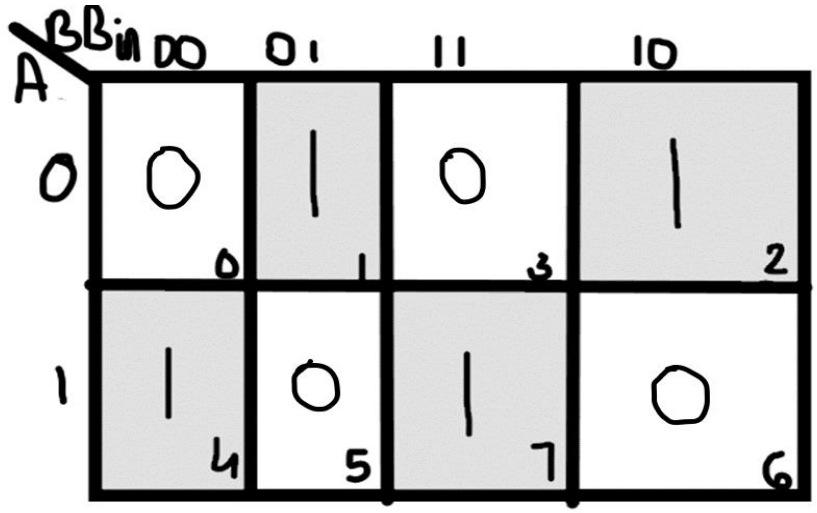


Fig: K-map for Difference Output Line for Full Subtractor Circuit

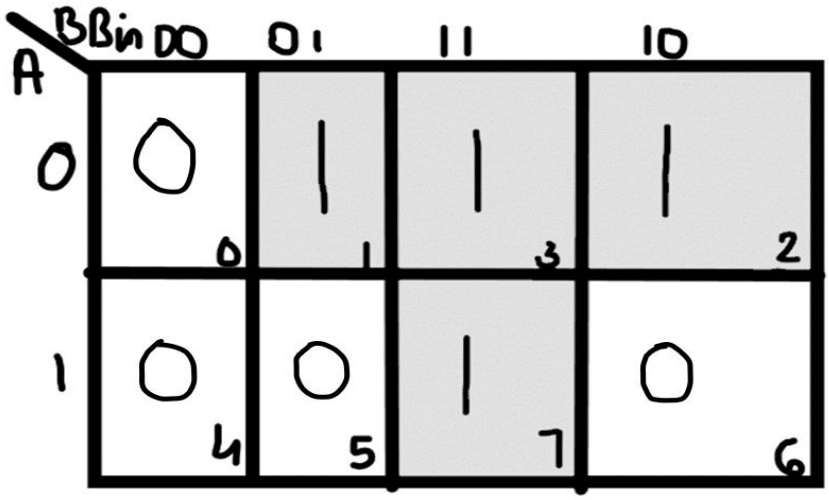


Fig: K-map for Borrow Output Line for Full Subtractor Circuit

From the above truth table we can find the boolean expression as:

$$D = A \oplus B \oplus Bin$$
$$B = A' Bin + A' B + B Bin$$

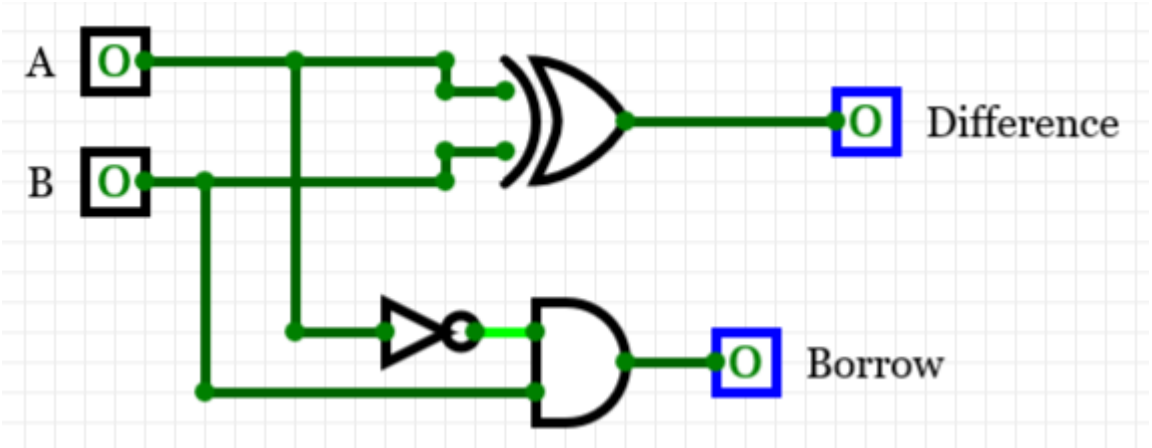
From the equation we can draw the Full-subtractor circuit using XOR, OR, AND and NAND gates.

Observations:

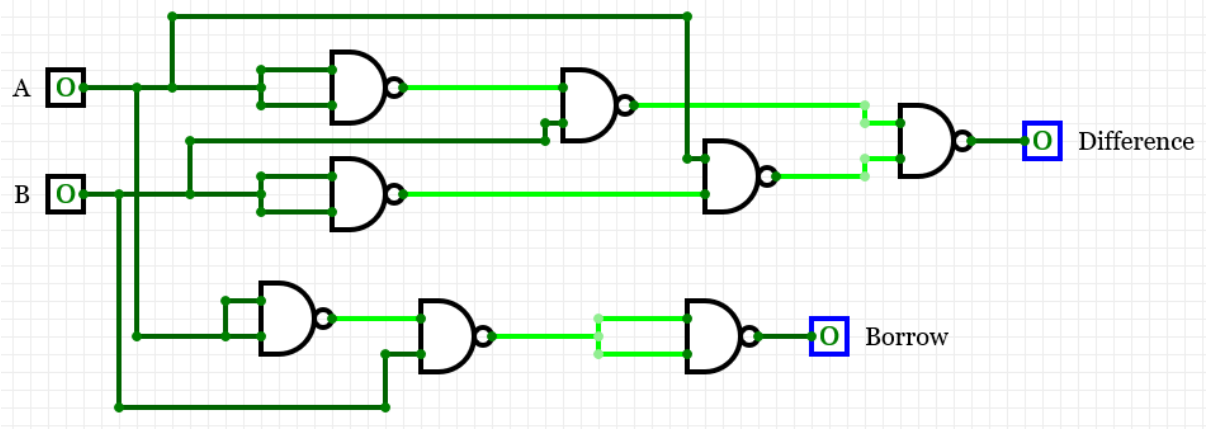
(A) Half Subtractor:

The circuit representation of Half Subtractor Circuit using:

(1) XOR, OR and AND Gates:



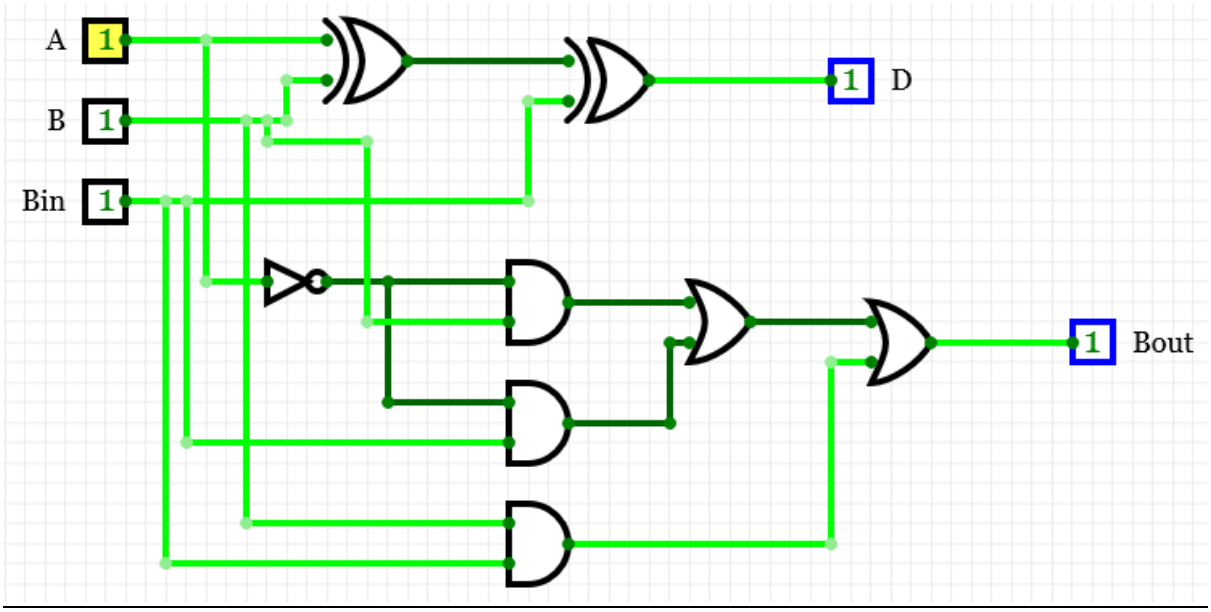
(2) NAND Gate:



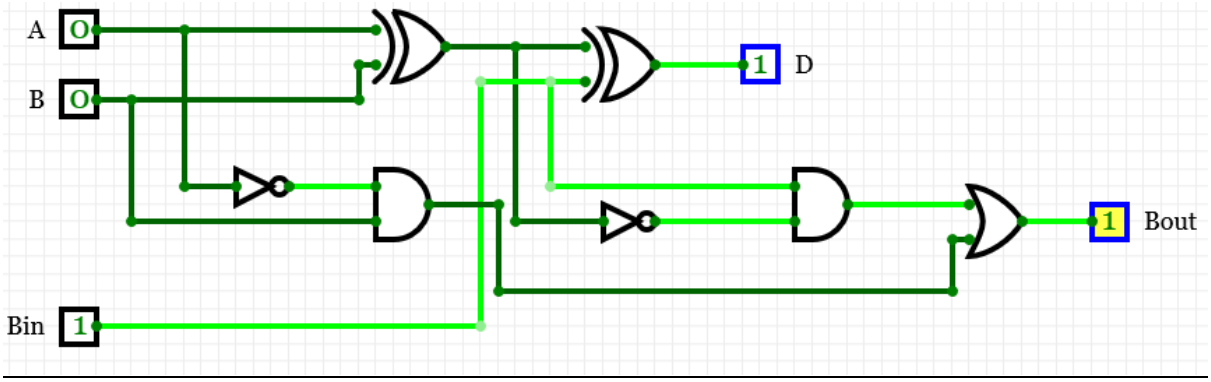
(B) Full Subtractor:

The circuit representation of Full Subtractor Circuit using:

(1) XOR, OR and AND Gates:

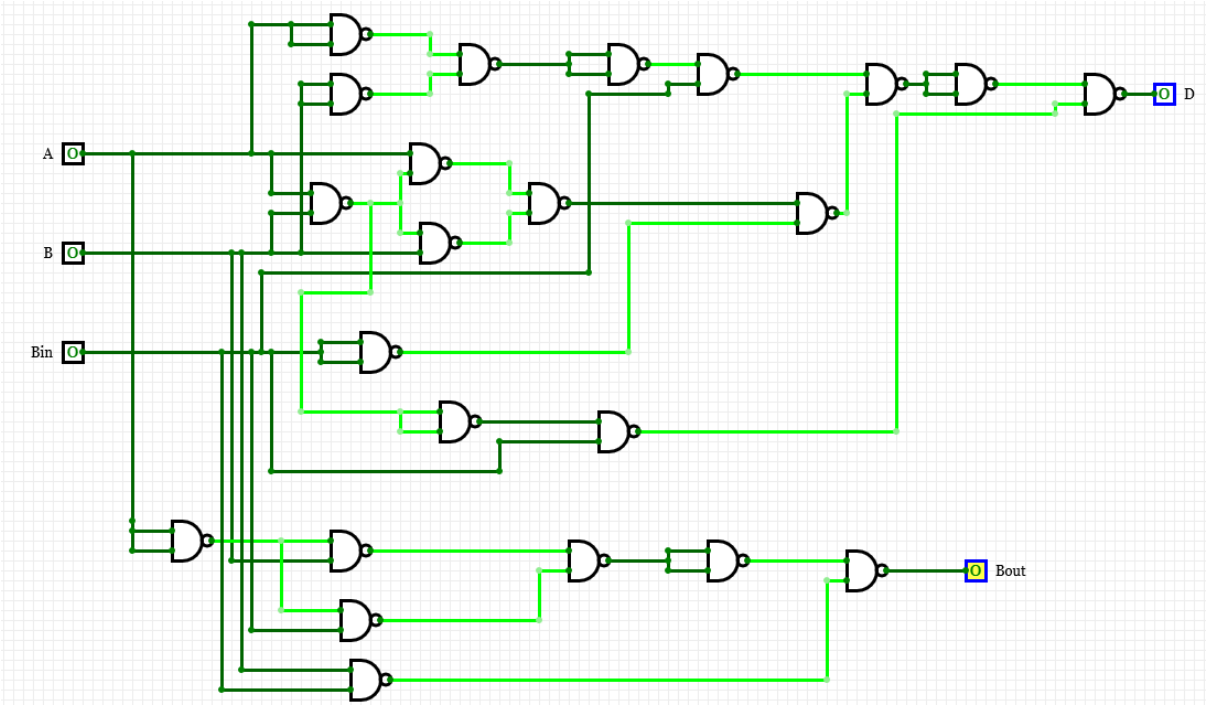


OR



(Using Half Subtractor Circuit)

(2) NAND Gate:



Result: The operation of Half Subtractor and Full Subtractor Circuit has been verified successfully.

CRITERIA	TOTAL MARKS	MARKS OBTAINED	COMMENTS
(A) CONCEPT	2		
(B) IMPLEMENTATION	2		
(C) PERFORMANCE	2		
TOTAL	6		

Experiment 5

Aim: To design the logical unit of A.L.U.

Theory: Arithmetic and Logical Units (or A.L.U.s) are found at the core of the of microprocessors, where they implement the arithmetic and logic functions offered by processor (e.g., addition, subtraction, AND'ing two values, etc.). An A.L.U. is a combinational circuit that combines many common logical circuits in one block. A.L.U. can be designed to perform a variety of different arithmetic and logical functions include AND, OR, XOR, XNOR, INV, CLR (for clear) and PASS (for passing a value unchanged). The data processing part of C.P.U. is responsible for executing arithmetic and logical instructions on various operand types including fixed point and floating-point numbers. Various circuits used to execute data processing instructions are usually combined in a single circuit called an Arithmetic Logic Unit (A.L.U.). The complexity of an A.L.U. is determined by the way in which its arithmetic instructions are realized.

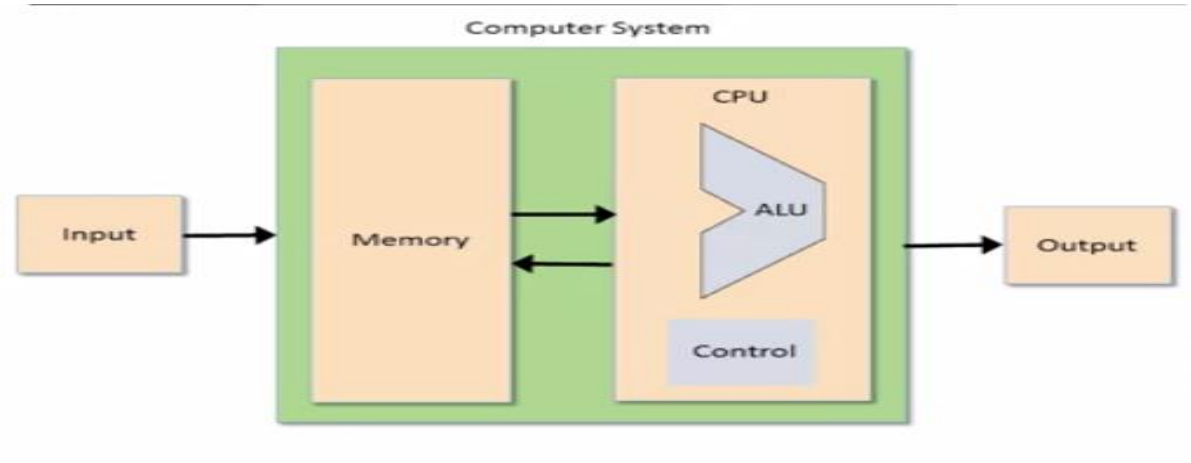


Fig 1: Structure of A.L.U.

In the following diagram:

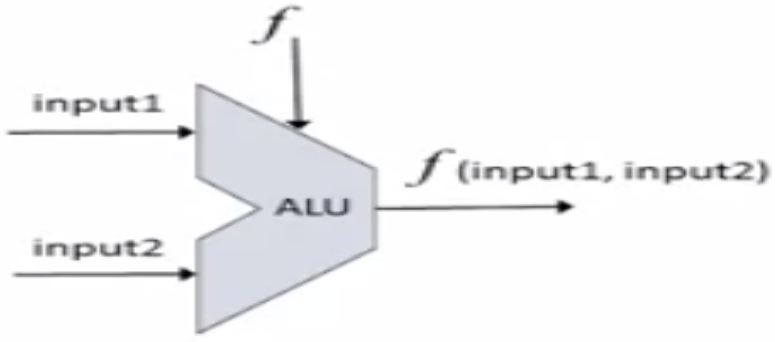


Fig 2: Representation of A.L.U.

- The ALU computes the function on two inputs and outputs the result.
- f: one out of the family of the predefined arithmetic and logical functions.

For constructing an A.L.U., one requires three control lines to identify any one of these operations. The input combination of these control lines are shown below:

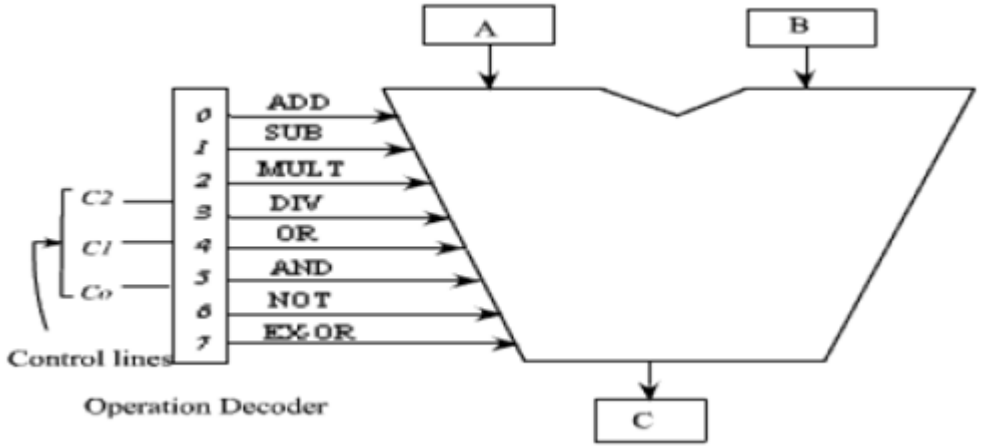


Fig 3: Block Diagram of A.L.U.

C_1	C_0	Arithmetic C2=0	Logical C2=1
0	0	Addition	OR
0	1	Subtraction	AND
1	0	Multiplication	NOT
1	1	Division	XOR

Table 1: Truth Table for A.L.U.

In this diagram:

- The control line C_2 is used to identify the group logical or arithmetic, i.e.
- C_2=0; arithmetic operation C_2=1 logical operation.
- Control lines C_0 and C_1 are used to identify any one of the four operations in a group.

In this experiment we are going to design the logical part of A.L.U. using Multiplexer and Decoder. The truth table of logical part of A.L.U. is as follows:

C_1	C_0	Logical Operations
0	0	AND
0	1	OR
1	0	NOT
1	1	XOR

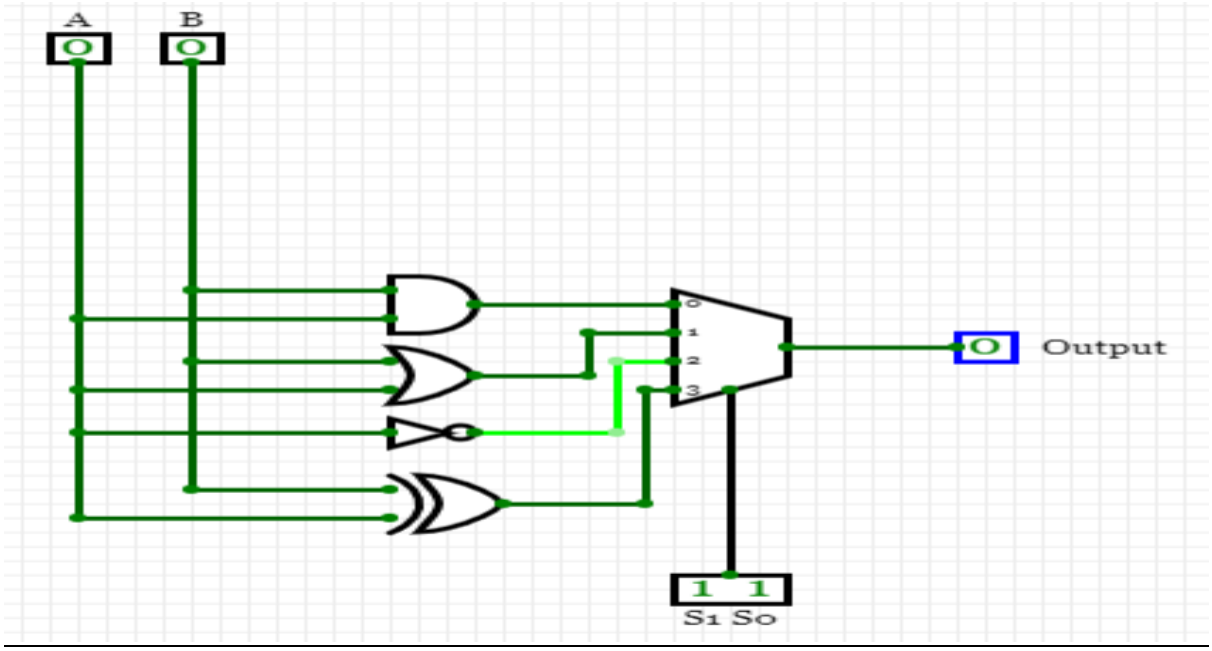
Table 2: Truth Table of Logical Unit of A.L.U.

Here, in this table, control lines C_0 and C_1 are used to identify any one of the four operations in a group.

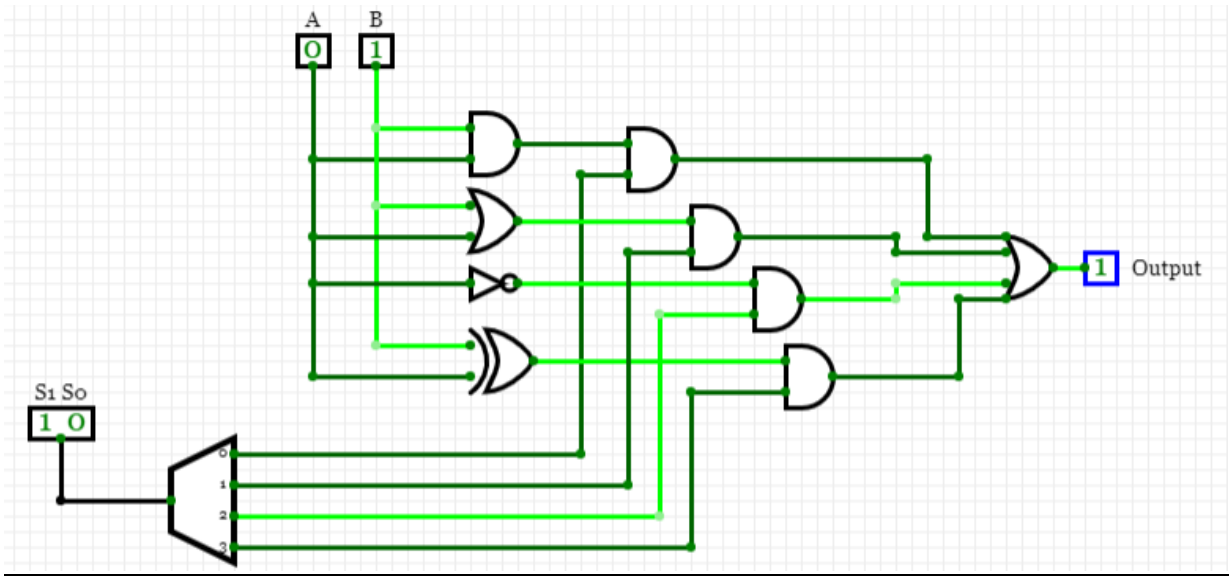
Observations:

Circuit Representation of Logical Unit of A.L.U. using:

(A) Multiplexer:



(B) Decoder:



Result: The logical unit of A.L.U. has been designed successfully.

CRITERIA	TOTAL MARKS	MARKS OBTAINED	COMMENTS
(A) CONCEPT	2		
(B) IMPLEMENTATION	2		
(C) PERFORMANCE	2		
TOTAL	6		

Experiment 6

Aim: To design a binary adder and subtractor using full adder circuit.

Tools Used: Circuit Verse.

Theory: In Digital Circuits, A Binary Adder-Subtractor is one which is capable of both addition and subtraction of binary numbers in one circuit itself. The operation being performed depends upon the binary value the control signal holds. It is one of the components of the ALU (Arithmetic Logic Unit). This Circuit Requires prerequisite knowledge of Exor Gate, Binary Addition and Subtraction, Full Adder. Here we will consider, two 4-bit binary numbers A and B as inputs to the Digital Circuit for the operation with digits

A0 A1 A2 A3 for A

B0 B1 B2 B3 for B

The circuit consists of 4 full adders since we are performing operation on 4-bit numbers. There is a control line M or we can say a mode bit M is used that which is used to hold a binary value of either 0 or 1 which determines that the operation being carried out is addition or subtraction. The addition and subtraction operations can be combined into one common circuit by including an XOR gate with each full - adder.

CASE 1: M=0

When M is 0 then C_{in} will be 0 and 0 XOR B_0 gives B_0, then S_0 will be sum of A_0 and B_0. Hence M=0 will perform addition.

CASE 2: M=1

When M is 1 then C_{in} will be 1 and 1 XOR B_0 gives B_0', then S_0 will be $A_0+B_0'+1= A_0-B_0+1$. Hence M=1 will perform subtraction.

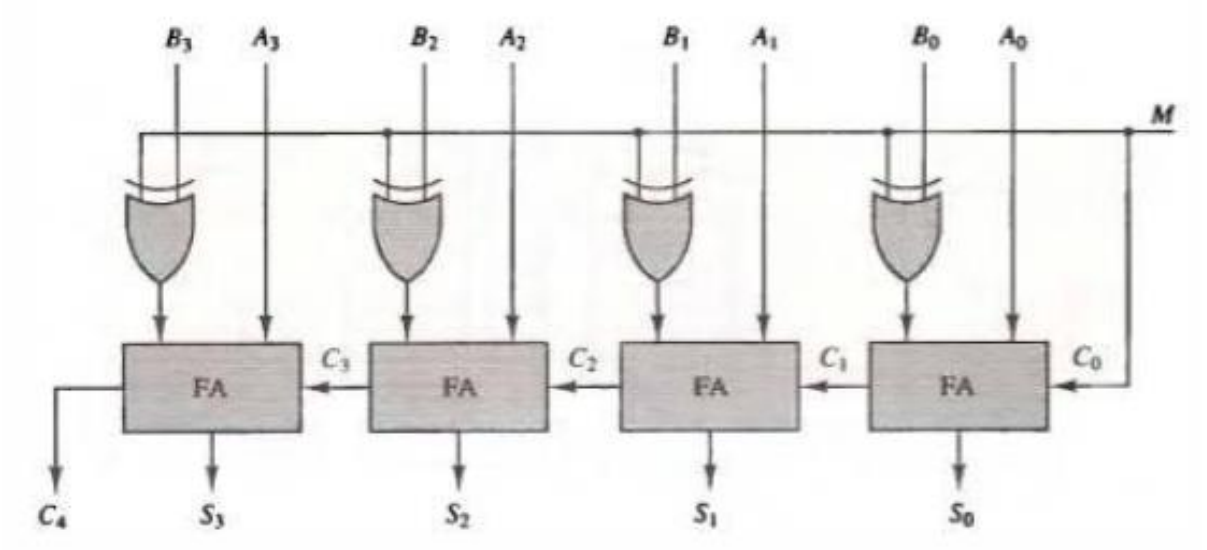


Fig 1: Blueprint of Binary Adder and Subtractor

As shown in the above figure, the first full adder has control line directly as its input (input carry C0), The input A0 (The least significant bit of A) is directly input in the full adder. The

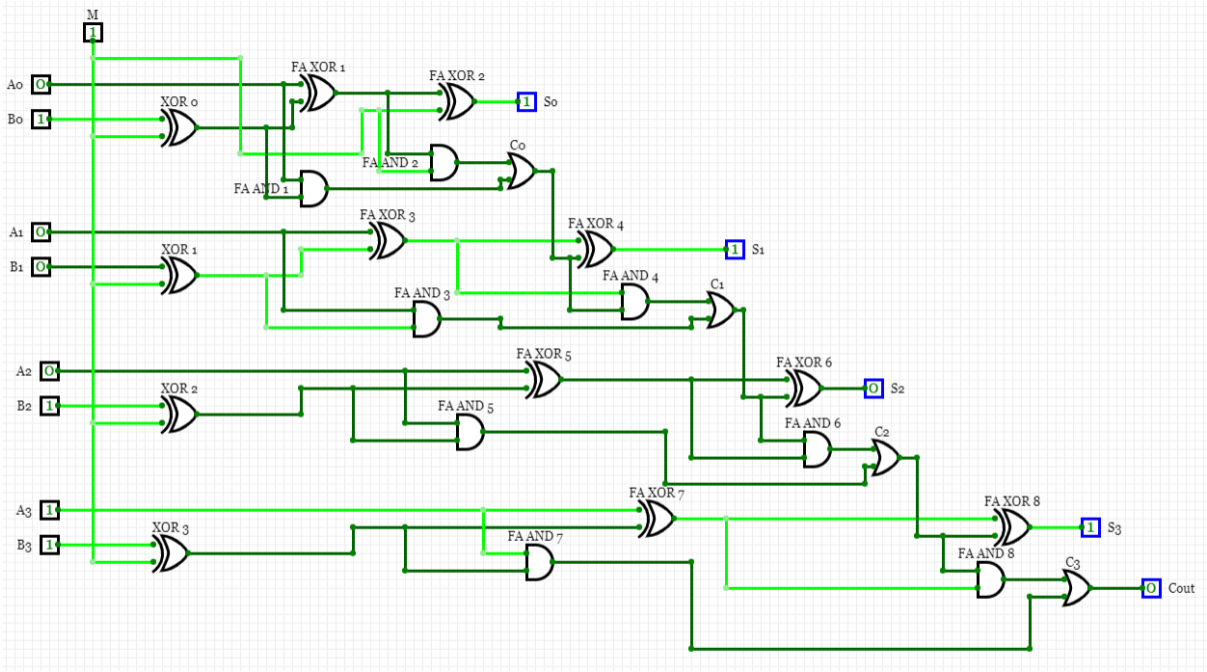
third input is the exor of B0 and M. The two outputs produced are Sum/Difference (S0) and Carry (C1).

If the value of M (Control line) is 1, the output of B0(EX-OR) M=B0′ (Complement B0). Thus, the operation would be A+(B0′). Now 2's complement subtraction for two numbers A and B is given by A+B'. This suggests that when M=1, the operation being performed on the four - bit numbers is subtraction. Similarly, if the value of M=0, B0 (Ex-OR) M=B0. The operation is A+B which is simple binary addition. This suggests that When M=0, the operation being performed on the four - bit numbers is addition.

Then C0 is serially passed to the second full adder as one of it's outputs. The sum/difference S0 is recorded as the least significant bit of the sum/difference. A1, A2, A3 are direct inputs to the second, third and fourth full adders. Then, the third input is the B1, B2, B3 (Ex-OR)ed with M to the second, third and fourth full adder respectively. The carry C1, C2 are serially passed to the successive full adder as one of the inputs. C3 becomes the total carry to the sum/difference. S1, S2, S3 are recorded to form the result with S0. For an n-bit binary adder-subtractor, we use n number of full adders.

Observations:

Circuit Representation of Binary Adder and Subtractor:



Result: The designing of the binary adder subtractor has been done successfully.

CRITERIA	TOTAL MARKS	MARKS OBTAINED	COMMENTS
(A) CONCEPT	2		
(B) IMPLEMENTATION	2		
(C) PERFORMANCE	2		
TOTAL	6		

Experiment 7

Aim: To design and simulate 4-bit BCD adder.

Tools Used: Circuit Verse.

Theory: BCD is Binary Coded Decimal number, where each digit of a decimal number is respected by its equivalent binary number. That means, LSB of a decimal number is represented by its equivalent binary number and similarly other higher significant bits of decimal number are also represented by their equivalent binary numbers.

Suppose, we have 4 bit numbers i.e. A and B which can vary from 0 (0000) to 9 (1001 in binary). For example:

Example 1:

```
Input :
A = 0111   B = 1000
Output :
Y = 1 0101

Explanation: We are adding A(=7) and B(=8).
The value of binary sum will be 1111(=15).
But, the BCD sum will be 1 0101,
where 1 is 0001 in binary and 5 is 0101 in binary.
```

Example 2:

```
Input :
A = 0101   B = 1001
Output :
Y = 1 0100

Explanation: We are adding A(=5) and B(=9).
The value of binary sum will be 1110(=14).
But, the BCD sum will be 1 0100,
where 1 is 0001 in binary and 4 is 0100 in binary.
```

If the decimal numbers are less than or equal to 9, then the value of BCD sum and the binary sum will be same otherwise they will be same otherwise they will be diff by 6 (0110 in binary).

Decimal	**Binary Sum**					**BCD Sum**				
	C’	S3’	S2’	S1’	S0’	C	S3	S2	S1	S0
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	1
2	0	0	0	1	0	0	0	0	1	0
3	0	0	0	1	1	0	0	0	1	1

4	0	0	1	0	0	0	0	1	0	0
5	0	0	1	0	1	0	0	1	0	1
6	0	0	1	1	0	0	0	1	1	0
7	0	0	1	1	1	0	0	1	1	1
8	0	1	0	0	0	0	1	0	0	0
9	0	1	0	0	1	0	1	0	0	1
10	0	1	0	1	0	1	0	0	0	0
11	0	1	0	1	1	1	0	0	0	1
12	0	1	1	0	0	1	0	0	1	0
13	0	1	1	0	1	1	0	0	1	1
14	0	1	1	1	0	1	0	1	0	0
15	0	1	1	1	1	1	0	1	0	1
16	1	0	0	0	0	1	0	1	1	0
17	1	0	0	0	1	1	0	1	1	1
18	1	0	0	1	0	1	1	0	0	0
19	1	0	0	1	1	1	1	0	0	1

Table 1: Truth Table of Binary to BCD Adder.

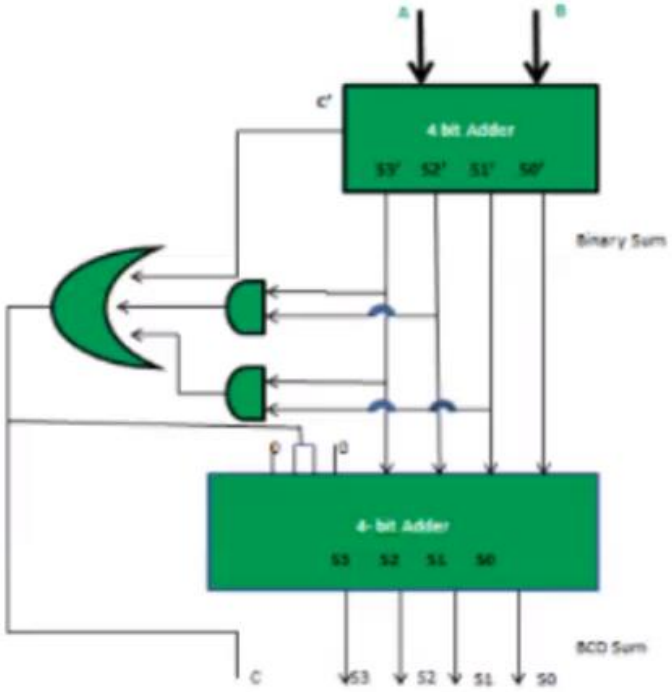
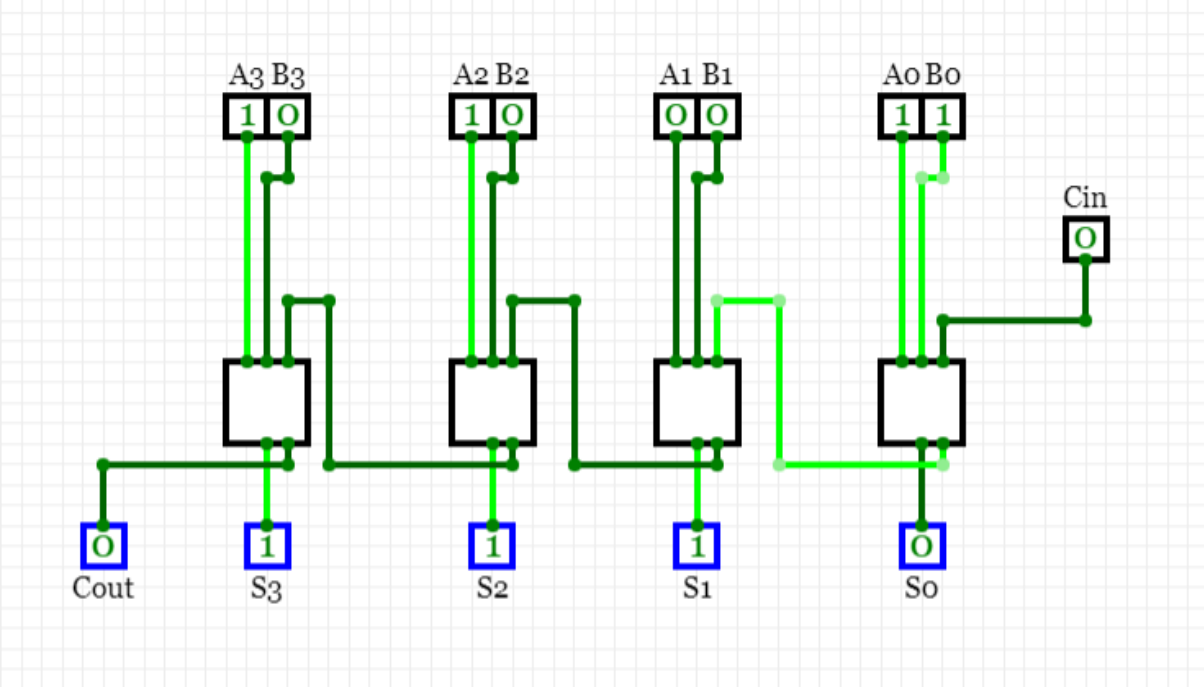


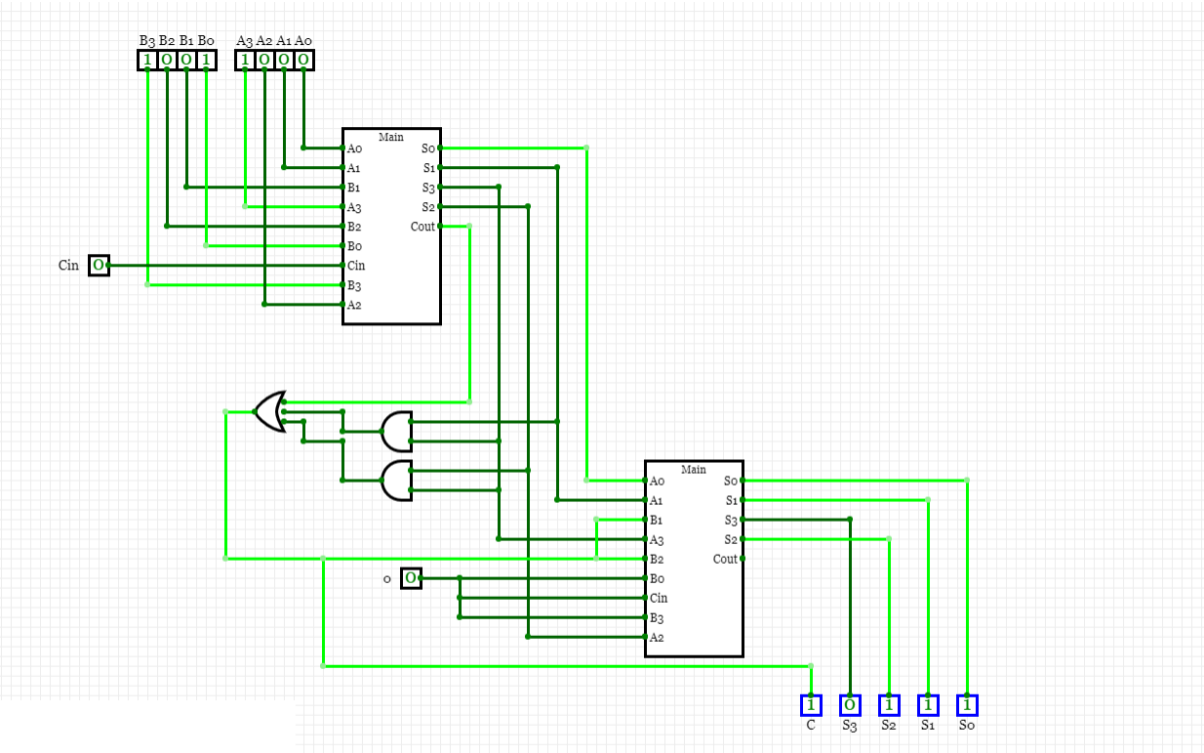
Fig 1: Blueprint of 3 Bit Binary to BCD Adder.

Observations:

Circuit Representation of Full Adder Circuit:



Circuit Representation of Binary to BCD:



Result: The designing and simulation of binary to BCD adder has been done successfully.

CRITERIA	TOTAL MARKS	MARKS OBTAINED	COMMENTS
(A) CONCEPT	2		
(B) IMPLEMENTATION	2		
(C) PERFORMANCE	2		
TOTAL	6		

Experiment 8

Aim: To design 4-bit bus using three-state buffer gate and decoder.

Tools Used: Circuit Verse

Theory: Usually any logic circuit has 2 states, i.e., in binary form (0 and 1). The buffer exhibits three states. It has 3 pins which include:
Input – accepts 1 or 0 (0 – disable and 1 – enable)
Output – if 3-state control is 0 then output follows input(according to the input 0 and 1.
Definition: A three state bus buffer is an integrated circuit that connects multiple data sources to a single bus. The open drivers can be selected to be either a logical high, a logical low, or high impedance which allows other buffers to drive the bus.
Now, let's see the more detailed analysis of a 3-state bus buffer in points:

1. As in a conventional gate, 1 and 0 are two states.
2. Third state is a high impedance state.
3. The third state behaves like an open circuit.
4. If the output is not connected, then there is no logical significance.
5. It may perform any type of conventional logic operations such as AND, OR, NAND, etc.

Difference Between Normal Buffer and Three-State Buffer: It contains both normal input and control input. Here, the output state is determined by the control input.

- When the control input is 1, the output is enabled and the gate will behave like conventional buffer.
- When the control input is 0, the output is disabled and the gate will be in high impedance state.

How to Form a Bus Line using Three-State Buffer:

1. To form a single bus line, all the outputs of the 4 buffers are connected together.
2. The control input will now decide which of the 4 normal inputs will communicate with the bus line.
3. The decoder is used to ensure that only one control input is active at a time.

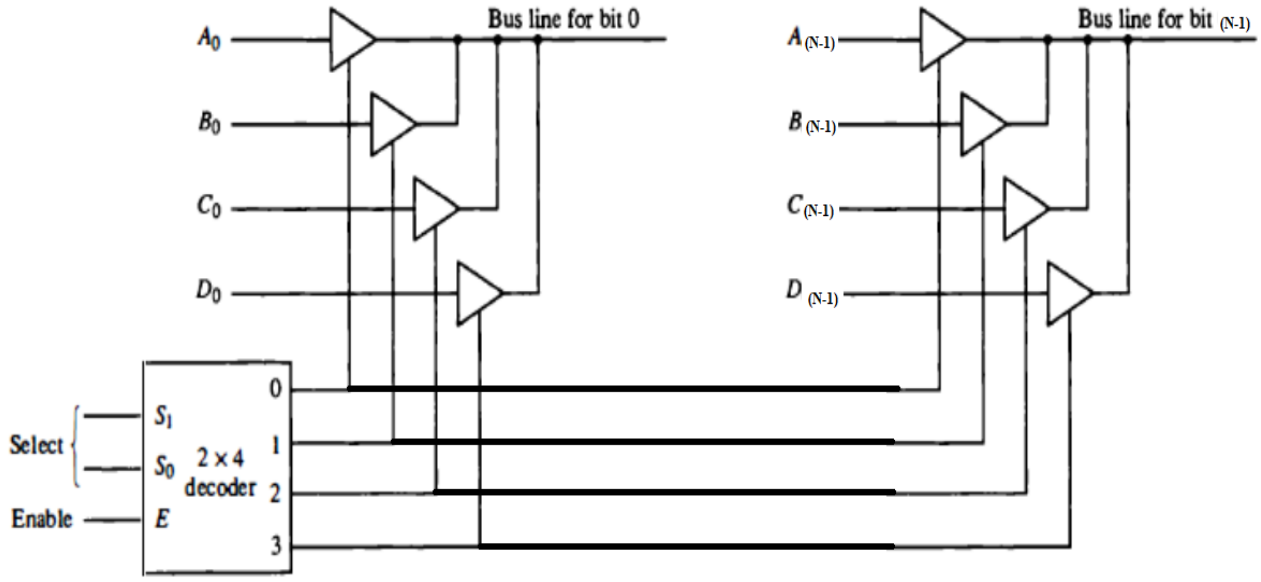
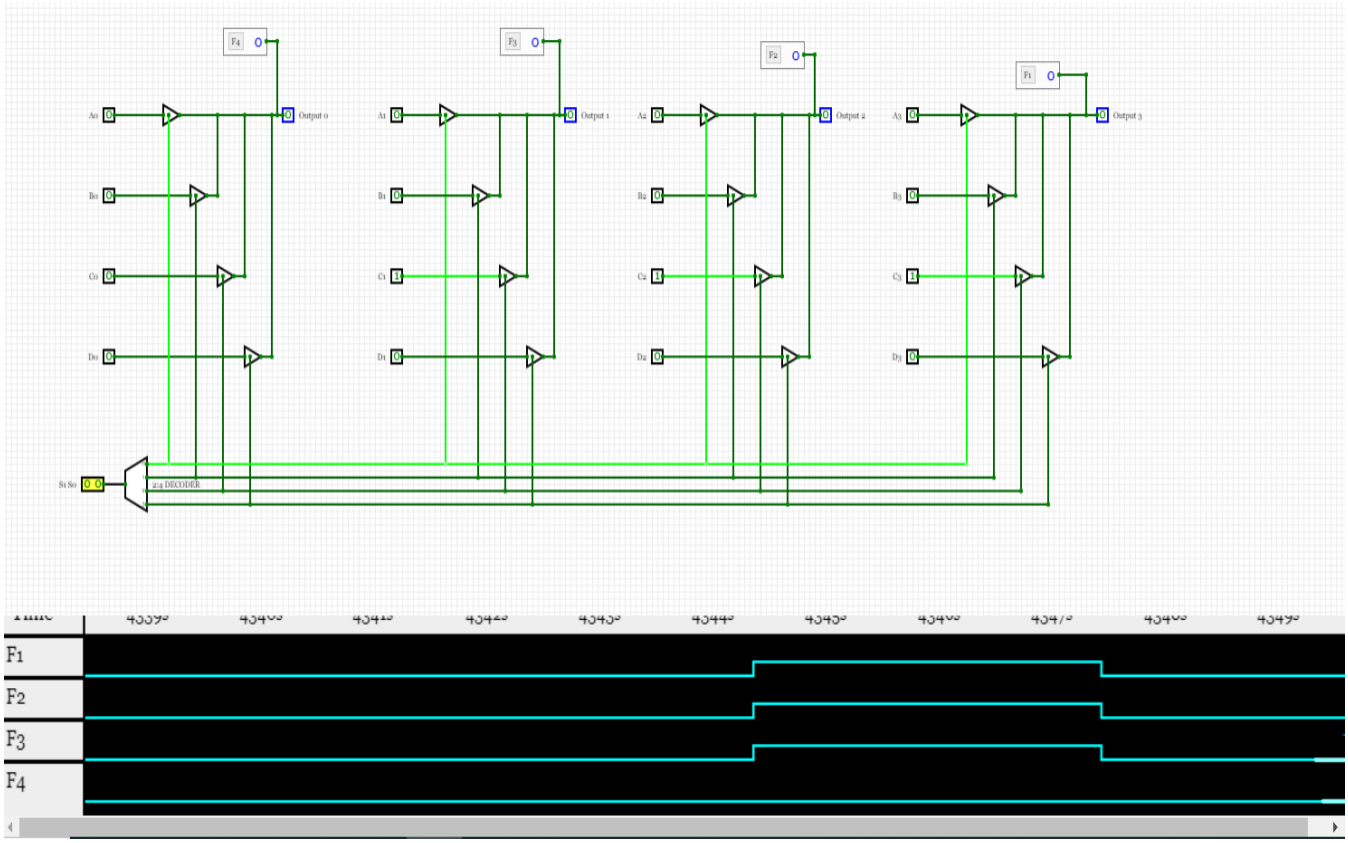


Fig 1: Bus Line with Three State Buffers and Decoder

Observations:

Circuit Representation of 4-Bit Bus System Using Three State Buffer Gate and Decoder:



Result: The designing of three-state buffer gate has been done successfully.

CRITERIA	TOTAL MARKS	MARKS OBTAINED	COMMENTS
(A) CONCEPT	2		
(B) IMPLEMENTATION	2		
(C) PERFORMANCE	2		
TOTAL	6		

Experiment 9

Aim: To design and simulate a 4-bit bus using multiplexers.

Theory: A typical computer has many registers and we need to transfer the information between these registers. A way to transfer the information is using the common bus system.

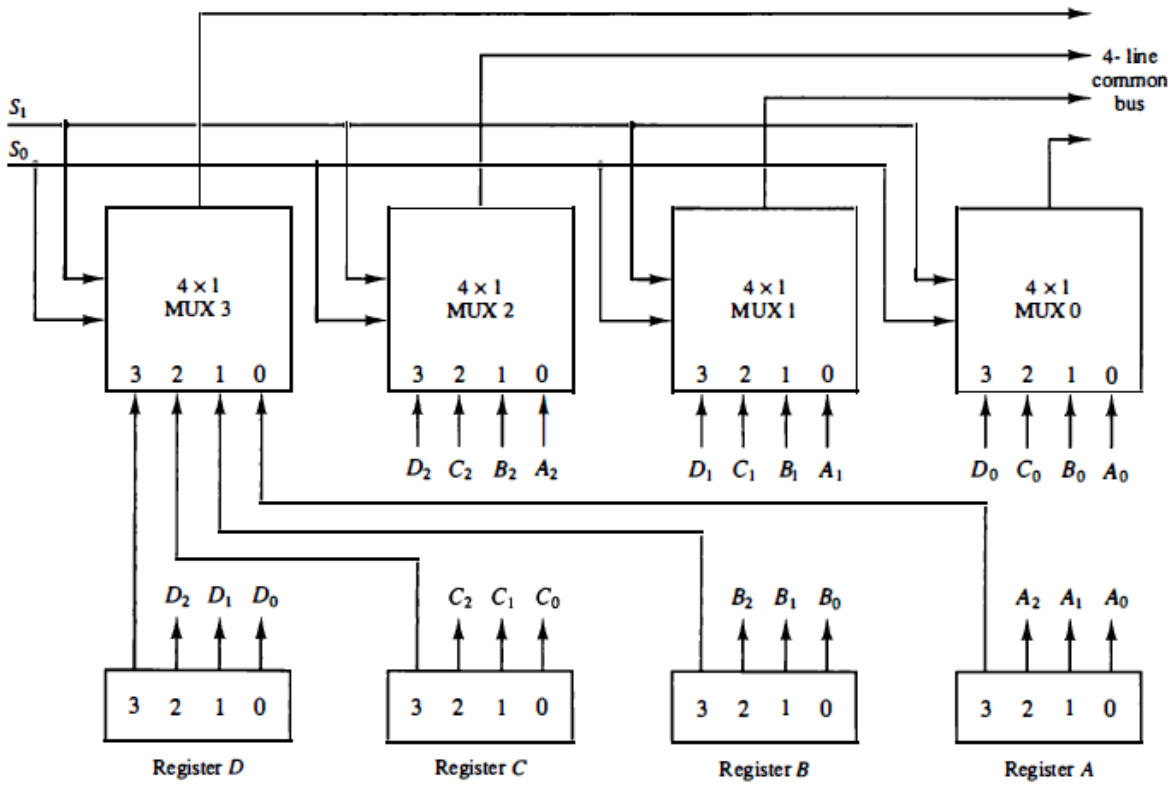


Fig 1: Bus Line with Multiplexers

The construction of this bus system for 4 registers is shown above. The bus consists of 4×1 multiplexers with 4 inputs and 1 output and 4 registers with bits numbered 0 to 3. There are 2 select inputs S0 and S1 which are connected to the select inputs of the multiplexers which will decide which register data must be transferred on common bus. The output 1 of register A is connected to input 0 of MUX 1 and similarly other connections are made as shown in the diagram. The data transferred to the bus depends upon the select lines. A table for the various combinations of select lines is shown below.

Select Lines Combination		**Register Selected**
S0	S1	
0	0	A
0	1	B
1	0	C
1	1	D

Table 1: Truth Table of Selection of Registers by Select Lines in Bus Line with Multiplexers

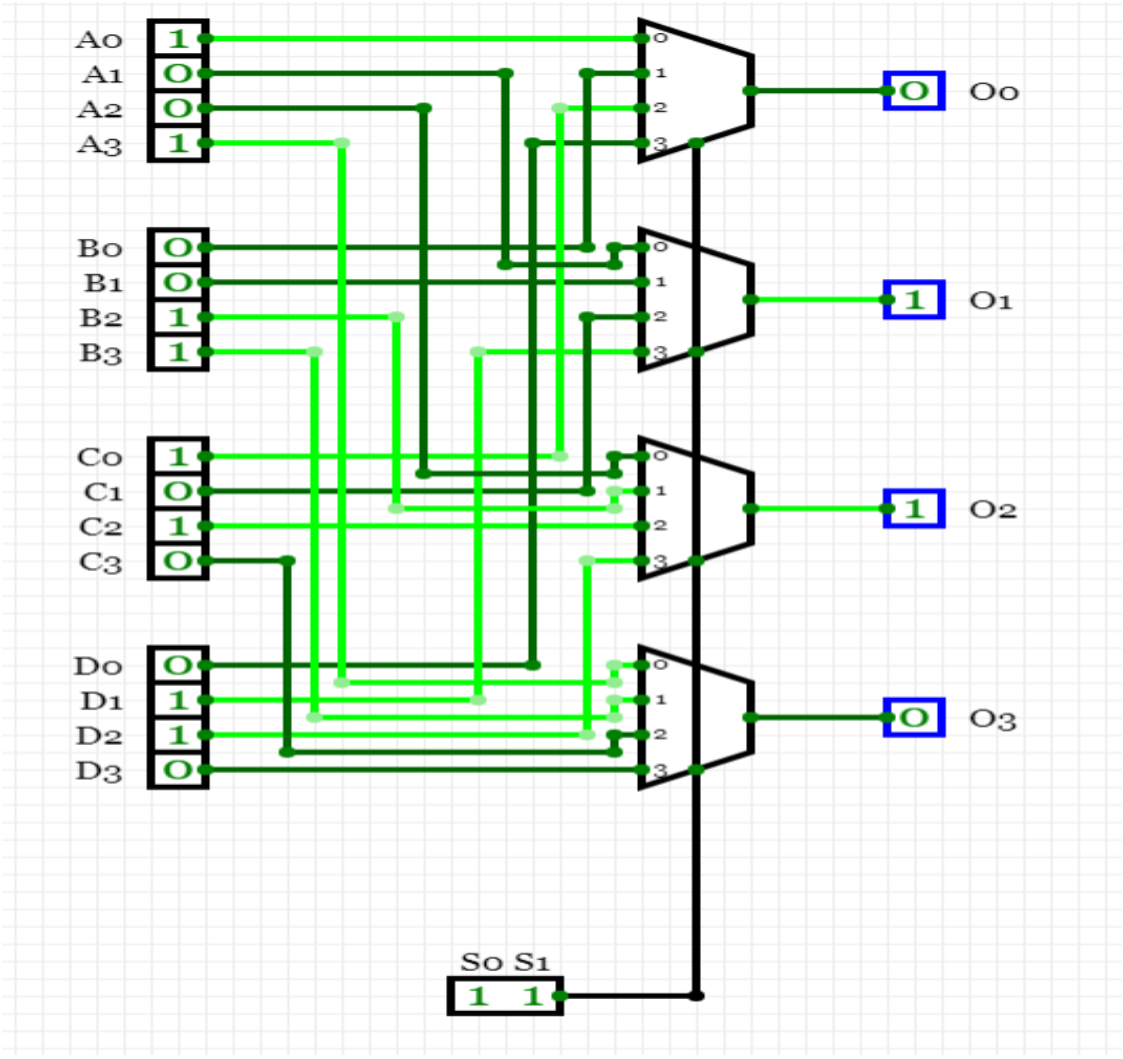
As we can see that when S1S0=00, register A is selected because on 00 the 0 data inputs of all the multiplexers are applied to the common bus. Since the 0 data inputs of all the multiplexers receive the inputs from the register A, thus register A gets selected. Similarly, for other combinations of S1S0 other register are selected.

Note:

- No. of multiplexers needed = No. of bits in each register
- Common bus system can be designed for a bus transfer of n registers using n MUX of n X 1 size.
- In general, a bus system will multiplex k registers of n bits each to produce an n-line common bus. The number of multiplexers needed to construct the bus is equal to n, no of bits in each register. The size of each multiplexer must be k × 1 since it multiplexer k data lines.

Observations:

Circuit Representation of 4-Bit Bus System Using Multiplexers:



Result: The designing of a 4-bit bus system with multiplexers has been done successfully.

CRITERIA	TOTAL MARKS	MARKS OBTAINED	COMMENTS
(A) CONCEPT	2		
(B) IMPLEMENTATION	2		
(C) PERFORMANCE	2		
TOTAL	6		

Experiment 10

Aim: To design and simulate 4-to2-Priority Encoder using logic gates.

Tools Used: Circuit Verse.

Theory: An encoder has 2^n inputs, and n output lines. Only one input can be at logic 1 at any given time (active input). All other inputs must be 0's. It accepts an active level on one of its inputs representing a digit, such as a decimal or octal digit, and converts it to a coded output, such as BCD or binary. The major problem with the encoder is that it can generate the wrong output code when there is more than one input present at logic level "1". To prevent this one can use priority encoder. A priority encoder is a device which has same functioning as encoders but it will produce the output corresponds to the currently active input which has the highest priority. So, when an input with a higher priority is present, all other inputs with a lower priority will be ignored.

It consists of:

- 4 input channels having sixteen input combinations.
- Three output variables A1, A0, and V.
- A valid indicator, V, is included to indicate whether or not the output is valid.
- Output is invalid when no inputs are active i.e. V = 0. Output is valid when at least one input is active V = 1.

In last row we have 1111 input combinations, whose output is 11. This is because E3 is the highest priority input, and it is equal to 1. Though the other inputs with smaller subscripts, namely, E2, E1, and E0 are also having values of one in some combinations, but they do not have the priority.

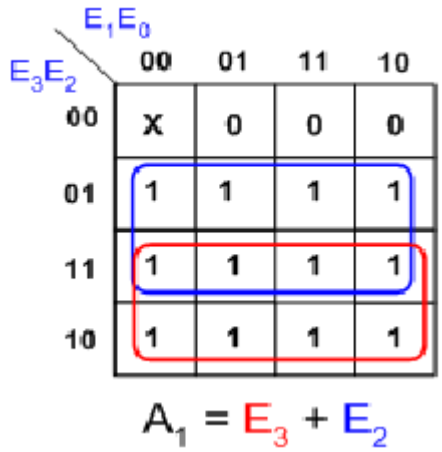


Fig 1: K-map Representation of the A_1 Output Line of the Priority Encoder

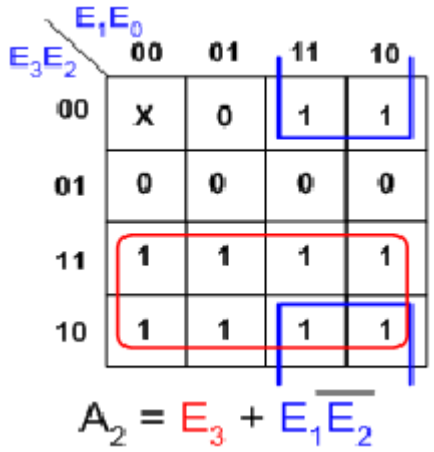


Fig 2: K-map Representation of the A_0 Output Line of the Priority Encoder

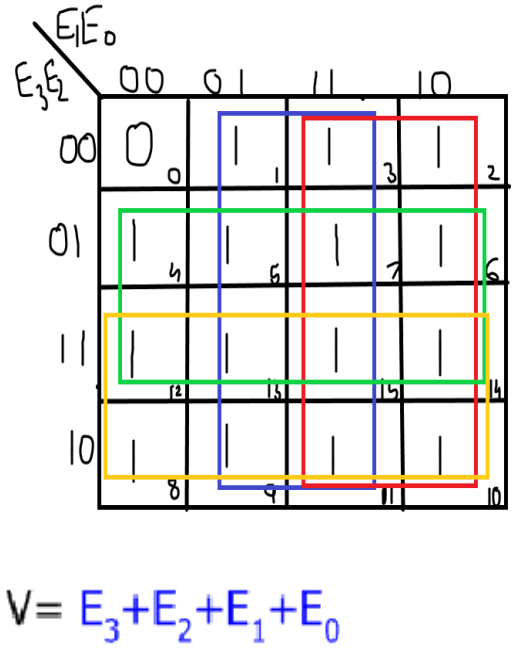


Fig 3: K-map Representation of the V Output Line of the Priority Encoder

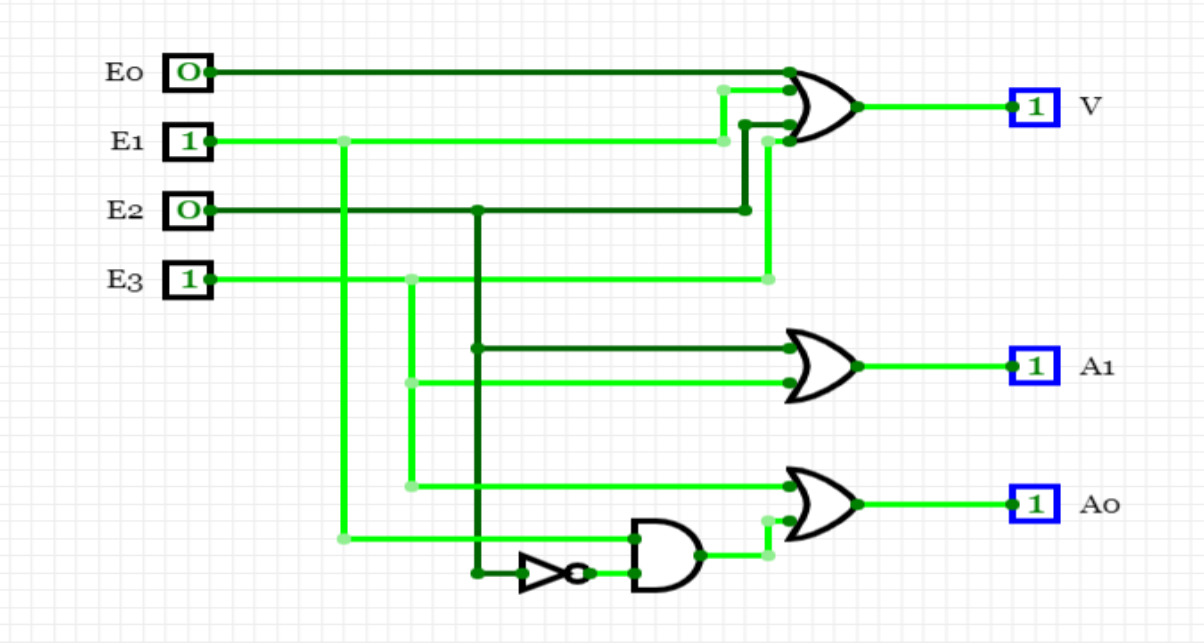
E3	E2	E1	E0	A1	A0	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	0	0	1	1
0	0	1	1	0	1	1
0	1	0	0	1	0	1
0	1	0	1	1	0	1
0	1	1	0	1	0	1
0	1	1	1	1	0	1
1	0	0	0	1	1	1
1	0	0	1	1	1	1
1	0	1	0	1	1	1
1	0	1	1	1	1	1

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

Table 1: Truth Table of Priority Encoder

Circuit Representation:

Circuit representation of the Priority Encoder circuit using Logic Gates is:



Result: The designing of the 4-to-2 Priority Encoder has been done successfully.

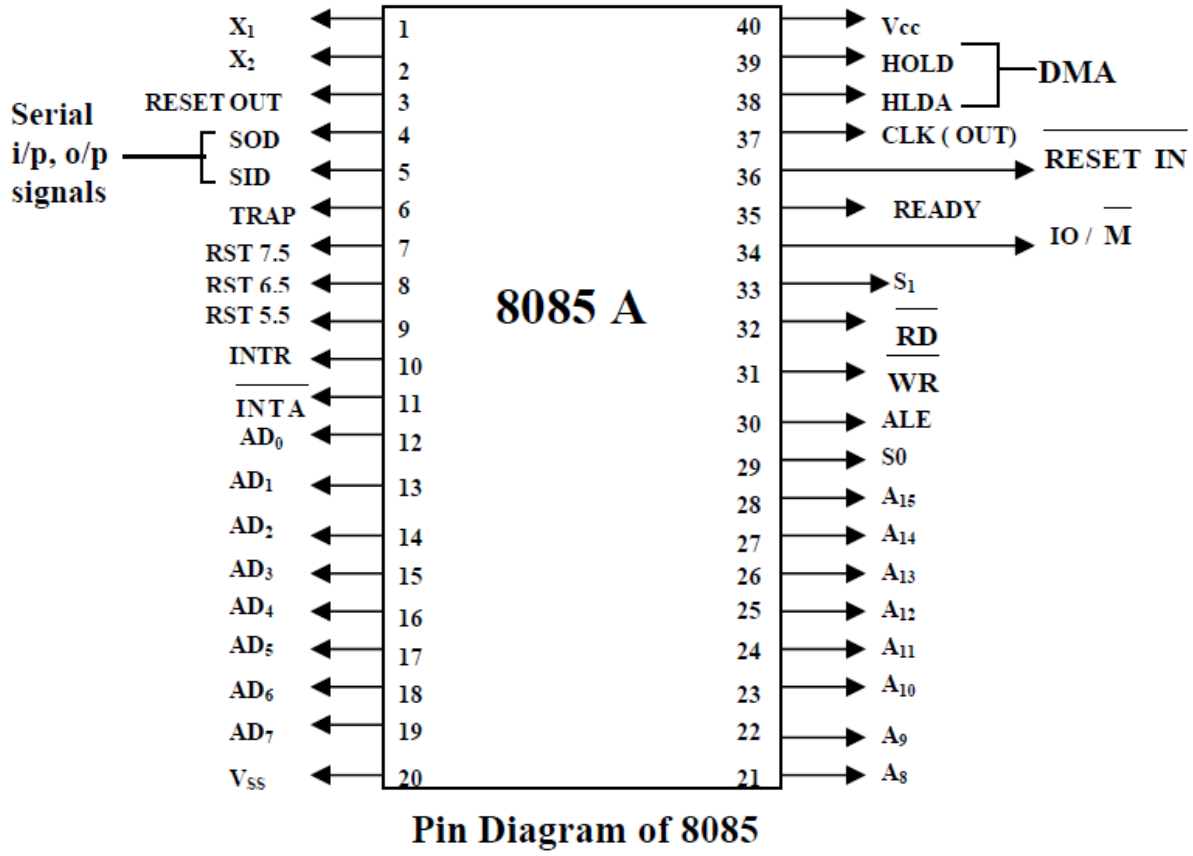
CRITERIA	TOTAL MARKS	MARKS OBTAINED	COMMENTS
(A) CONCEPT	2		
(B) IMPLEMENTATION	2		
(C) PERFORMANCE	2		
TOTAL	6		

Experiment 11

Aim: To study the architecture and instruction set of 8085 Microprocessor.

Tools Used: (Research)

Theory: The 8085 Microprocessor or Intel 8085 ("eighty-eighty-five") is an 8-bit microprocessor produced by Intel and introduced in March 1976. It is created with N-MOS technology. It has an 8-bit data bus. This means that 8 bits of data can flow around in the innards of the microprocessor. Apart from the data bus, it also has a 16-bit address bus, which addresses up to 64KB. It also has a 16-bit program counter & a 16-bit stack pointer. There are six 8-bit registers which are arranged in pairs: BC, DE, HL. It requires a voltage supply of +5V to operate at 3.2MHZ single-phase clock frequency. The Intel 8085 comes as a 40-pin IC package.



Pin Diagram of 8085

Fig 1: Pin Diagram of 8085 Microprocessor

The 8085 has extensions to support new interrupts, with three maskable interrupts (RST 7.5, RST 6.5 and RST 5.5), one non-maskable interrupt (TRAP), and one externally serviced interrupt (INTR).

Three control signals are available on chip:

- RD: it is a active low signal. Which indicate that the selected IO or Memory device is to be read and data is available on the data bus.

- WR: it is a active low signal which indicate that the data on the data bus are to be written into a selected memory or IO location.
- ALE: it is a +ve going pulse generated every time the 8085 begins an operation (machine cycle), which indicate that the bits on AD7-AD0 are address bits.

Three status signals are available on chip:

- IO/M: this is a status signal used to differentiate between IO and Memory operations. If it is high then IO operation and If it is low then Memory operation.
- S1 and S0: status signals similar to IO/M, can identify various operations that are rarely used in the systems.

Internal Architecture of 8085 Microprocessor: The architecture of 8085 consists of three main sections, ALU (Arithmetic and Logical Unit), timing and control unit and Registers.

Arithmetic and Logic Unit (ALU): The ALU performs the actual numerical and logical operations. The ALU performs the following arithmetic and logical operations.

- Addition
- Subtraction
- Logical AND
- Logical OR
- Logical Ex - OR
- Complement (logical NOT)
- Increment
- Decrement
- Left shift
- Right shift
- Clear

ALU includes the accumulator, the temporary register, the arithmetic and logic circuits and flags. It always stores result of operations in Accumulator.

Timing & Control Unit:

- It generates timing and control signals, which are necessary for the execution of instructions.
- It controls data flow between CPU and peripherals (including memory).
- It provides status, control and timing signals, which are required for the operation of memory and I/O devices.

8085 System Bus: Microprocessor communicates with memory and other devices (input and output) using three buses: Address Bus, Data Bus and Control Bus.

- Address Bus: The Address bus consists of 16 wires. The size of the address bus determines the size of memory, which can be used. To communicate with memory the microprocessor sends an address on the address bus to the memory. Address bus is unidirectional, i.e., numbers only sent from microprocessor to memory.
- Data Bus: Bus is bidirectional. Size of the data bus determines what arithmetic can be done. Data bus also carries instructions from memory to the microprocessor.

Memory size = 2A x D where, A denotes the address lines, and D denotes the data lines.

- Control Bus: Control bus are various lines which have specific functions for coordinating and controlling μP operations. The control bus carries control signals partly unidirectional, partly bidirectional. Control signals are things like read or write.

Registers: 8085 has six general purpose registers to store 8 - bit data, these are identified as B, C, D, E, H and L. They can be combined as register pairs BC, DE and HL to perform some 16 - bit operations.

- Accumulator: The accumulator is an 8 - bit register included as a part of Arithmetic Logic Unit (ALU). This register is used to store 8 - bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator.
- Flag Register: The ALU includes five flip-flops. They are called Zero (Z), Carry (CY), Sign (S), Parity (P) and Auxiliary Carry (AC) flags. The microprocessor uses these flags to test data conditions. The conditions (set or reset) of the flags are tested through the software instructions. The combination of the flag register and the accumulator is called Program Status Word (PSW) and PSW is the 16 - bit unit for stack operation.
- Program Counter (PC): This 16 - bit register deals with sequencing the execution of instruction. The microprocessor uses this register to sequence the execution of the instructions. The function of the program counter is to point to the memory address from which the next byte is to be fetched.
- Stack Pointer (SP): The stack pointer is also a 16 - bit register used as a memory pointer. It points to a memory location in read-write memory, called the stack.
- Instruction Register/Decoder: Temporary store for the current instructions of a program. Latest instruction sent here from memory prior to execution. Decoder then takes instruction and decodes or interprets the instruction. Decoded instruction then passed to next stage.
- Memory Address Register: Holds address, received from PC of next program instruction.
- Control Generator: It generates signal within μP to carry out the instructions which have been decoded.
- Register Selector: This block controls the use of the register stack.
- General Purpose Registers: μP requires extra registers for versatility. It can be used to store additional data during a program.

Operations of Microprocessor: The microprocessor performs the following four operations using address bus, data bus and control bus:

- Memory Read: Reads data (or instruction) from memory.
- Memory Write: Writes data (or instruction) into memory.
- I/O Read: Accepts data from input device.
- I/O Write: Sends data to output device.

The 8085 Instruction Format: An instruction is a command to the microprocessor to perform a given task on a specified data. Each instruction has two parts, one is task to be performed, called the operation code (opcode), and the second is the data to be operated on called the operand. The 8085 instruction set is classified according to word size.

- **One-Byte Instructions:** A 1-byte instruction includes the opcode and operand in the same byte. Operands are internal registers and are coded into the instruction.
- **Two-Byte Instructions:** In a two-byte instruction, the first byte specifies the operation code and the second byte specifies the operand. Source operand is a data byte immediately following the opcode.
- **Three-Byte Instructions:** In a three - byte instruction, the first byte specifies the op - code and the following two bytes specify the 16-bit address. Note that, the second byte is the low-order address and the third byte is the high-order address.

The 8085 Addressing Modes: The various formats for specifying operands are called the addressing modes. For 8085, they are

- **Immediate Addressing:**
  i) Data is provided in the instruction.
  ii) Load the immediate data to the destination provided.
  iii) Example: MVI A, 12 H.
- **Register Addressing:**
  i) Data is provided through the registers.
  ii) Example: MOV B, C.
- **Direct Addressing:**
  i) Used to accept data from outside devices to store in the accumulator or send the data stored in the accumulator to the outside device.
  ii) Example: MOV A, [1000].
- **Indirect Addressing:**
  i) Effective address is calculated by the processor and the contents of the address is used to form a second address. The second address is where the data is stored.
  ii) Example: MOV A, [[1000]].
- **Implicit addressing:**
  i) In this addressing mode the data itself specifies the data to be operated upon.
  ii) Example: CMA; Complement the contents of accumulator.

8085 Instruction set: An instruction is a binary pattern designed inside a microprocessor to perform a specific function. Each instruction is represented by 8 bit binary value. Instruction set can be categorised int0 5 types:

Data transfer instructions:

- These instructions are used to transfer data from one register to another register, from memory to register or register to memory.
- When an instruction of data transfer group is executed, data is transferred from the source to the destination without altering the contents of the source.
- Examples: MOV, MVI, LXI, LDA, STA, etc.

Arithmetic instructions:

- These instructions are used to perform arithmetic operations such as addition, subtraction, increment or decrement of the content of a register or memory.
- Examples: ADD, ADC, ADI, DAD, SUB, INR, DCR, etc.

Logical instructions:

- These instructions are used to perform logical operations such as AND, OR, compare, rotate etc.
- Examples: ANA, ANI, ORA, ORI, XRA, CMA, CMC , STC, CMP, RLC, RAL, RAR, etc.

Branching Instructions:

- These instructions are used to perform conditional and unconditional jump, subroutine call and return, and restart.
- Examples: JZ, JNZ, JC, JNC, JP, JM, JPE, JPO, CALL, RET, RST, etc.

Machine Control Instructions:

- These instructions control machine functions such as Halt, Interrupt, or do nothing.
- The microprocessor operations related to data manipulation can be summarized in four functions: copying data, performing arithmetic operations, performing logical operations, testing for a given condition and alerting the program sequence.
- Example: PUSH, POP, HLT, XTHL, NOP, EI, DI, etc.

Example-1: Write 8085 assembly program for multiplying two 8 - bit numbers.

- MVI A,00; Load immediate data into accumulator.
- MVI B,02; Load immediate data into register B.
- MVI C,04; Load immediate data into register C.
- LOOP: ADD B; Add the content of to accumulator.
- DCR C; Decrement the content of register C by 1.
- JNZ LOOP
- STA 1000 H; Store the content of accumulator to memory location 1000 H
- HLT; Halt

Example-2: Writh 8085 assembly program to find the largest number in an array of data.

- LXI H, 1000; Load the address of the first element of the array in HL pair
- MOV B, M; Load the Count
- INX H; Set first element as largest data
- MOV A, M; Get the first data in A
- DCR B; Decrements the count
- LOOP: INX H
- CMP M; Compare A and M
- JNC AHEAD; if no carry (A>M) then go to AHEAD
- MOV A, M; Set the new value as largest
- AHEAD: DCR B
- JNZ LOOP; Repeat comparisons till count = 0
- STA 2000; Store the largest value at 2000
- HLT

Direct Memory Access: Direct memory access (DMA) facilitates data transfer operations between main memory and I/O subsystems with limited CPU intervention. The majority of I/O devices provide two methods for transferring data between a device and memory.

- **Programmed I/O (PIO):** It is fairly easy to implement, but requires the processor to constantly read or write a single memory word (8-bits, 16-bits or 32-bits, depending on the device interface) until the data transfer is complete. Although PIO is not necessarily slower than DMA, it does consume more processor cycles and can be detrimental in a multi-processing environment.
- **DMA:** It allows a system to issue an I/O command to a device, initiate a DMA transaction and then place the process in a waiting queue. The system can now continue by selecting another process for execution, thereby utilizing the CPU cycles typically lost when using PIO.
- The **DMA controller** will inform the system when its current operation has been completed by issuing an interrupt signal. Although the data is still transferred 1 memory unit at a time from the device, the transfer to main memory now circumvents the CPU because the DMA controller can directly access the memory unit.
- Steps involved in the mode of DMA transfer are as follows.
  i) Device wishing to perform DMA asserts the processors bus request signal.
  ii) Processor completes the current bus cycle and then asserts the bus grant signal to the device.
  iii) The device then asserts the bus grant ack signal.
  iv) The processor senses in the change in the state of bus grant ack signal and starts listening to the data and address bus for DMA activity.
  v) The DMA device performs the transfer from the source to destination address.
  vi) During these transfers, the processor monitors the addresses on the bus and checks if any location modified during DMA operations is cached in the processor. If the processor detects a cached address on the bus, it can take one of the two actions:
    - Processor invalidates the internal cache entry for the address involved in DMA write operation
    - Processor updates the internal cache when a DMA write is detected
  vii) Once the DMA operations have been completed, the device releases the bus by asserting the bus release signal.
  viii) Processor acknowledges the bus release and resumes its bus cycles from the point it left off.

The 8085 microprocessor has two pins available for DMA mode of I/O communication: HOLD (Hold) and HLDA (Hold Acknowledge).

HOLD: This is an active high input signal to the 8085 from another master requesting the use of the address and data buses. After receiving the HOLD request, the Microprocessor relinquishes the buses in the following machine cycle. All buses are tri-stated and a Hold Acknowledge signal is sent out. The Microprocessor regains the control of buses after HOLD goes low.

HLDA: This is an active high output signal indicating that the MPU is relinquishing the control of the buses. Typically, an external peripheral such as DMA controller sends a request a high signal to the HOLD pin. The processor completes the execution of the current machine cycle;

floats (high impedance state) the address, the data, and the control lines; and sends the Hold Acknowledge (HLDA) signal. The DMA controller takes control of the buses and transfers data directly between source and destination, thus bypassing the microprocessor. At the end of data transfer, the controller terminates the request by sending a low signal to the HOLD pin, and the microprocessor regains control of the buses.

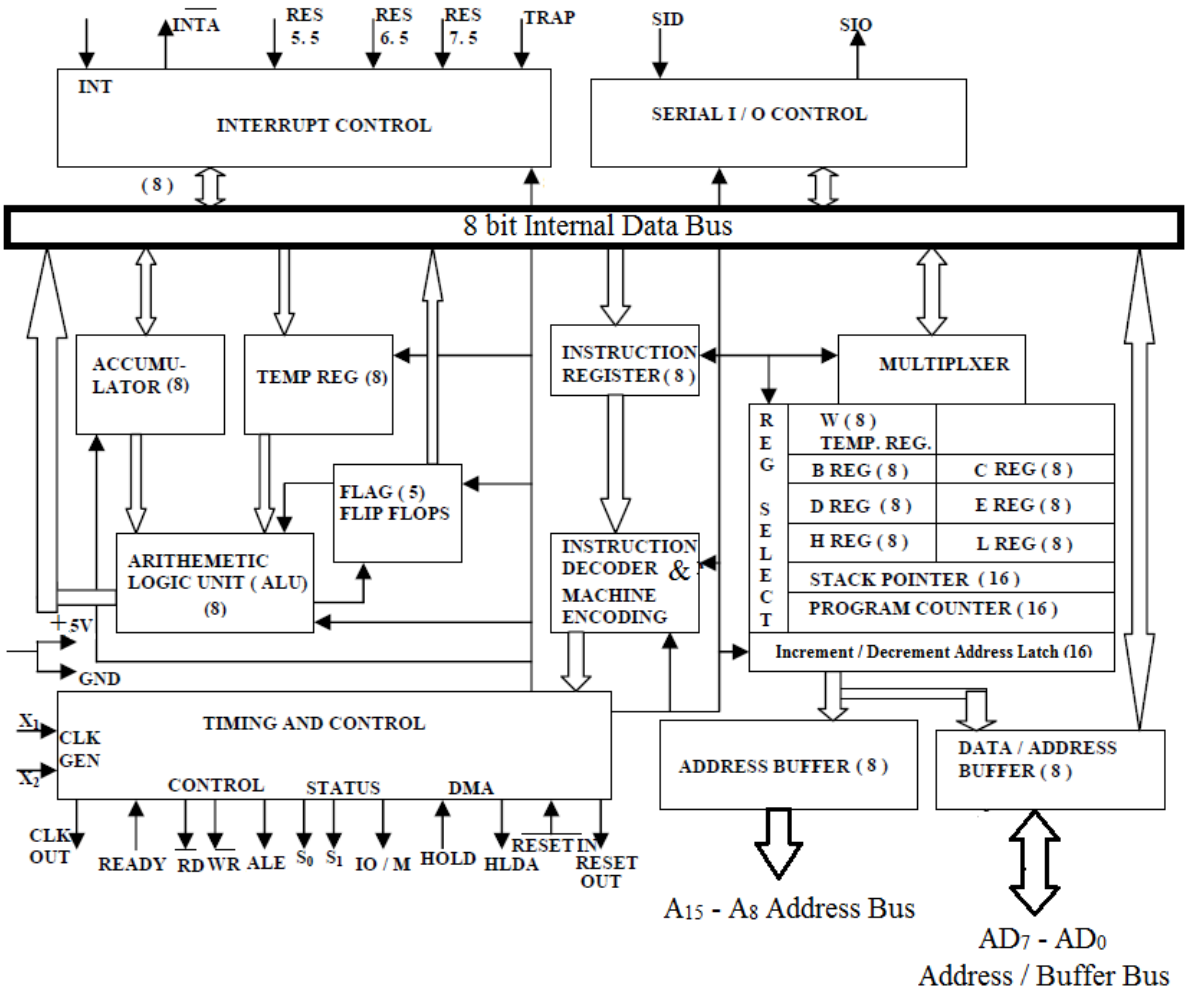


Fig 2: Internal Architecture of 8085 Microprocessor

Results and Conclusion: The 8085 Microprocessor has been studied successfully.

CRITERIA	TOTAL MARKS	MARKS OBTAINED	COMMENTS
(A) CONCEPT	2		
(B) IMPLEMENTATION	2		
(C) PERFORMANCE	2		
TOTAL	6		