Experiment No. : 03

Aim: Study of flip flop IC.

Objective:

1. To examining the behavior of the working module to understand how flipflops

2. To design a flipflops using gates

Theory:

A flip flop is an electronic circuit with two stable states that can be used to store binary data. The stored data can be changed by applying varying inputs. Flip-flops and latches are fundamental building blocks of digital electronics systems used in computers, communications, and many other types of systems. Both are used as data storage elements. It is the basic storage element in sequential logic. But first, let's clarify the difference between a latch and a flip-flop.

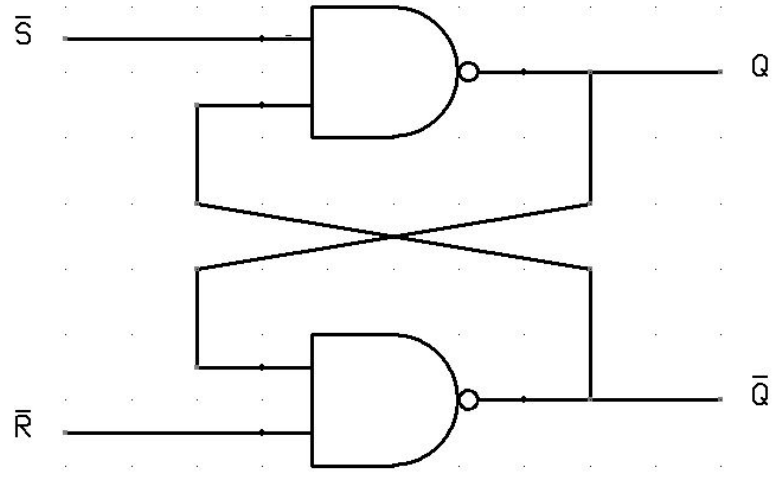
Flip flop v/s Latch

The basic difference between a latch and a flip-flop is a gating or clocking mechanism.

In Simple words. Flip Flop is edge-triggered and a latch is level triggered.

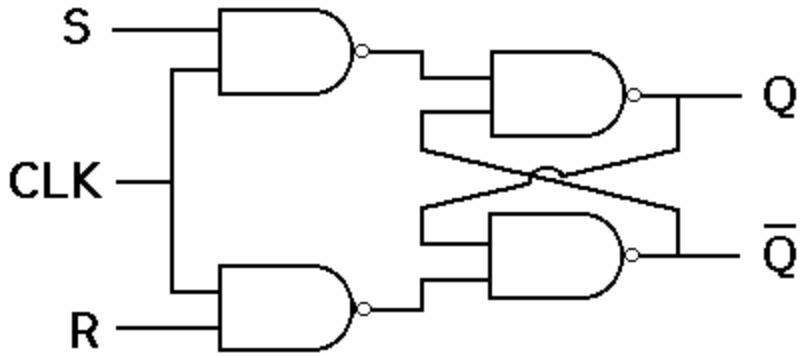
Read the full comparison of Flip-Flops v/s latch here

For example, let us talk about SR latch and SR flip-flops. In this circuit when you Set S as active the output Q would be high and Q' will be Low. This is irrespective of anything else. (This is an active-low circuit so active here means low, but for an active high circuit active would mean high)



SR Latch

A flip-flop, on the other hand, is synchronous and is also known as a gated or clocked SR latch.



SR Flip-Flop

In this circuit diagram, the output is changed (i.e. the stored data is changed) only when you give an active clock signal. Otherwise, even if the S or R is active the data will not change. Let's look at the types of flip-flops to understand better.

SR Flip Flop

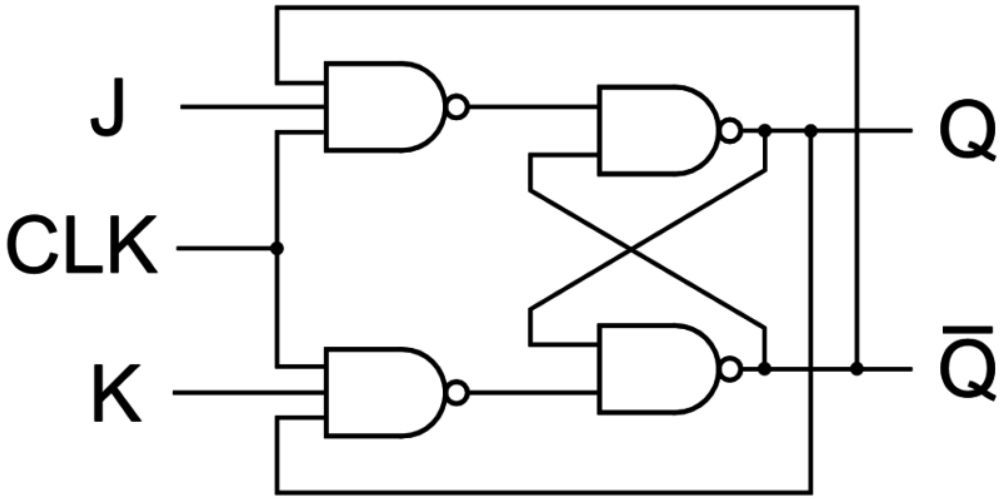
There are majorly 4 types of flip-flops, with the most common one being SR flip-flop. This simple flip-flop circuit has a set input (S) and a reset input (R). In this system, when you Set "S" as active the output "Q" would be high and "Q'"

will be low. Once the outputs are established, the wiring of the circuit is maintained until “S” or “R” go high, or power is turned off. As shown above, it is the simplest and easiest to understand. The two outputs, as shown above, are the inverse of each other. The truth table of SR Flip-Flop is highlighted below.

S	R	Q	Q’
0	0	0	1
0	1	0	1
1	0	1	0
1	1	∞	∞

JK Flip-flop

Due to the undefined state in the SR flip-flop, another flip-flop is required in electronics. The JK flip-flop is an improvement on the SR flip-flop where S=R=1 is not a problem.



JK Flip-Flop

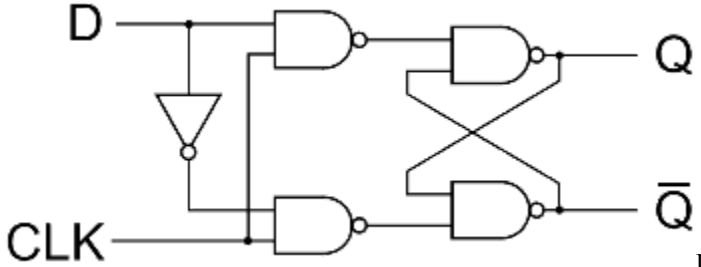
The input condition of J=K=1, gives an output inverting the output state. However, the outputs are the same when one tests the circuit practically.

In simple words, If J and K data input are different (i.e. high and low) then the output Q takes the value of J at the next clock edge. If J and K are both low then no change occurs. If J and K are both high at the clock edge then the output will toggle from one state to the other. JK Flip-Flops can function as Set or Reset Flip-flops

J	K	Q	Q’
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	1
0	1	1	0
1	0	1	1
1	1	1	0

D Flip Flop

D flip-flop is a better alternative that is very popular with digital electronics. They are commonly used for counters and shift-registers and input synchronisation.



D Flip-Flop

In this, the output can be only changed at the clock edge, and if the input changes at other times, the output will be unaffected.

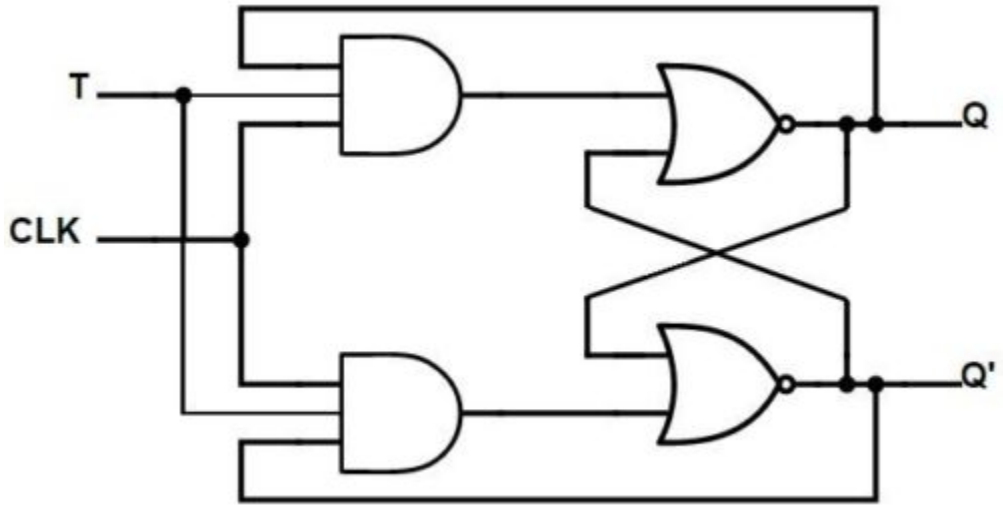
Clock	D	Q	Q’

↓ » 0	0	0	1
↑ » 1	0	0	1
↓ » 0	1	0	1
↑ » 1	1	1	0

The change of state of the output is dependent on the rising edge of the clock. The output (Q) is same as the input and can only change at the rising edge of the clock.

T Flip Flop

A T flip-flop is like a JK flip-flop. These are basically a single input version of JK flip-flops. This modified form of JK flip-flop is obtained by connecting both inputs J and K together. It has only one input along with the clock input.



These flip-flops are called T flip-flops because of their ability to complement its state (i.e.) Toggle, hence the name Toggle flip-flop.

T	**Q**	**Q (t+1)**
0	0	0
1	0	1
0	1	1
1	1	0

Applications of Flip-Flops

These are the various types of flip-flops being used in digital electronic circuits and the applications of Flip-flops are as specified below.

- Counters
- Frequency Dividers
- Shift Registers
- Storage Registers

Result:

Conclusion: Hence we have implemented, simulated, flipflops which is designed using NAND or NOR gate

QUESTIONNAIRE

Que 1. Which statement BEST describes the operation of a negative-edge-triggered D flip-flop?

- The Q output is ALWAYS identical to the CLK input if the D input is HIGH
- The logic level at the D input is transferred to Q on NGT of CLK
- The Q output is ALWAYS identical to the D input when CLK = PGT
- The Q output is ALWAYS identical to the D input

Que 2. Ring counter is similar to

- Stepping switch
- Latch
- SR flipflop
- Toggle flipflop

Que 3. How is a J-K flip-flop made to toggle?

- J = 0, K = 0
- J = 1, K = 1
- J = 1, K = 0
- J = 0, K = 1

Que 4. Where will you give the pulse input in a ripple counter, designed with edge triggered JK flipflop?

- J and K inputs of one flipflop
- Clock input of one flipflop
- Clock input of all the flipflops
- None of these

Experiment No. : 04

Aim: To implement 4-bit ripple carry full adder.

Objective:

3. To examining the behavior of the working module to understand how the carry ripples through the adder.

4. To design a ripple carry adder using full adders to mimic the behavior of the working module.

5. The adder will add two 4-bit numbers.

Theory : Multiple full adder circuits can be cascaded in parallel to add an N-bit number. For an N- bit parallel adder, there must be N number of full adder circuits. A ripple carry adder is a logic circuit in which the carry-out of each full adder is the carry in of the succeeding next most significant full adder. It is called a1ripple carry adder because each carry bit gets rippled into the next stage.

In a ripple carry adder the sum and carry out bits of any half adder stage is not valid until the carry in of that stage occurs.

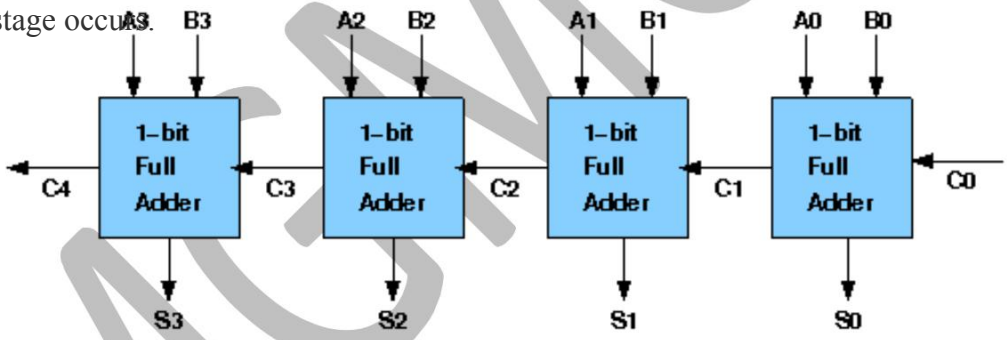


Fig1: Ripple Carry Adder

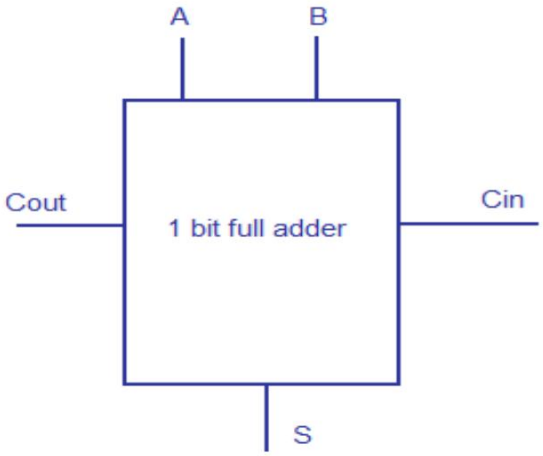
Full adder

To understand the working of a ripple carry adder completely, you need to have a look at the full adder too. Full adder is a logic circuit that adds two input operand bits1 plus a Carry in bit and outputs a Carry out bit and a sum bit.. The Sum out (Sout) of a full adder is the XOR of input operand bits A, B and the Carry in (Cin) bit. Truth table and schematic of a 1 bit Full adder is shown below.There is a simple trick to find results of a full adder. Consider the second last row of the truth table, here the operands are 1, 1, 0 ie (A, B, Cin). Add them together ie 1+1+0 = 10 . In

binary system, the number order is 0, 1, 10, 11……. and so the result of 1+1+0 is 10 just like we

get 1+1+0 =2 in decimal system. 2 in the decimal system corresponds to 10 in the binary system. Swapping the result "10" will give S=0 and Cout = 1 and the second last row is justified. This can be applied to any row in the table.

Inputs			Outputs	
A	B	Cin	Cout	S
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0
0	0	1	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1



1 bit full adder truth table & schematic

Fig 2: Full adder

A Full adder can be made by combining two half adder circuits together *(a half adder is a circuit that adds two input bits and outputs a sum bit and a carry bit).*

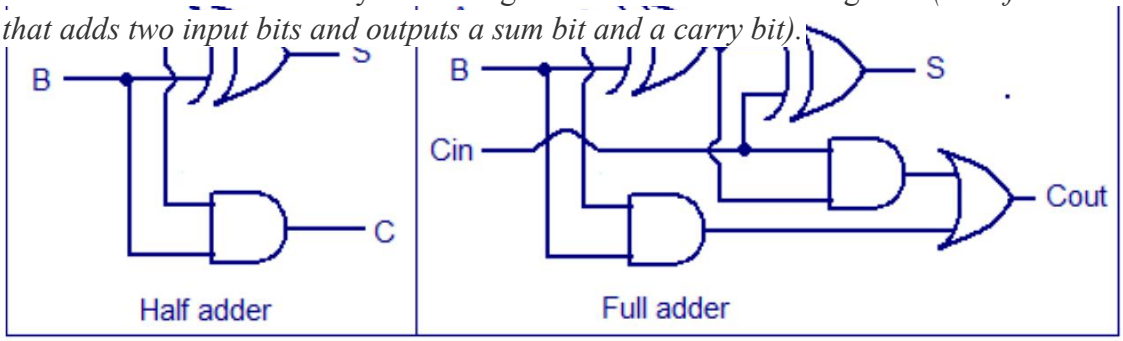


Fig 3: Full adder & half adder circuit

Full adder using NAND or NOR logic

Alternatively the full adder can be made using NAND or NOR logic. These schemes are universally accepted and their circuit diagrams are shown below.

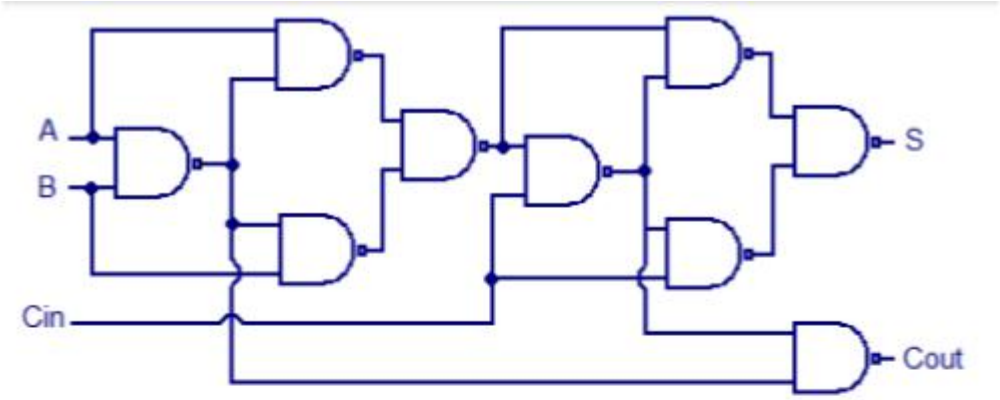


Fig 4: Full adder using NAND logic

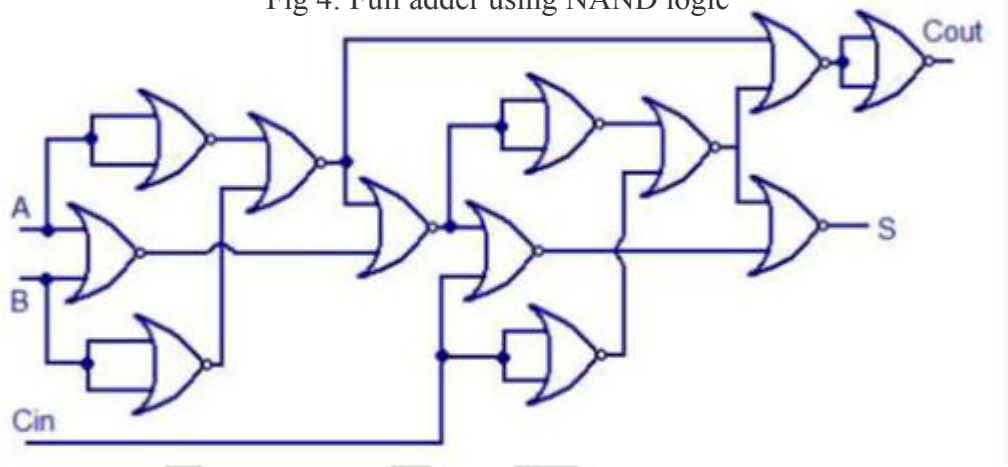


Fig 5: Full adder using NOR logic

Procedure:

1. Start the simulator as directed . This simulator tool support 5-valued logic.

2. To design the circuit we need 3 full adder, 1 half adder, 8 Bit switch(to give input), 3 Digital display(2 for seeing input and 1 for seeing output sum), 1 Bit display(to see the carry output), wires.

3. The pin configuration of a component is shown whenever the mouse is hovered on any canned component of the palette or press the 'show pinconfig' button. Pin numbering starts from 1 and from the bottom left corner(indicating with the circle) and increases anticlockwise.

4. For half adder input is in pin-5,8 output sum is in pin-4 and carry is pin-1, For full adder input is in pin-5,6,8output sum is in pin-4 and carry is pin-1.

5. Click on the half adder component(in the Adder drawer in the pallet) and then click on the position of the editor window where you want to add the component(no drag and drop, simple click will serve the purpose), likewise add 3 full adders(from the Adder drawer in the pallete), 8 Bit switches, 3 digital display and 1 bit Displays(from display and Input drawer of the pallet, if it is not seen scroll down in the drawer).

6. To connect any two components select the Connection menu of Palette, and then click on the Source terminal and click on the target terminal. According to the circuit diagram connect all the components, connect 4 bit switches to the 4 terminals of a digital display and another set of 4 bit switches to the 4 terminals of another digital display. connect the pin-1 of the full adder which will give the final carry output. connect the sum(pin-4) of all the adders to the terminals of the third digital display(according to the circuit diagram shown in screenshot). After the connection is over click the selection tool in the pallete.

7. To see the circuit working, click on the Selection tool in the pallet then give input by double clicking on the bit switch, (let it be 0011(3) and 0111(7)) you will see the output on the output(10) digital display as sum and 0 as carry in bit display.

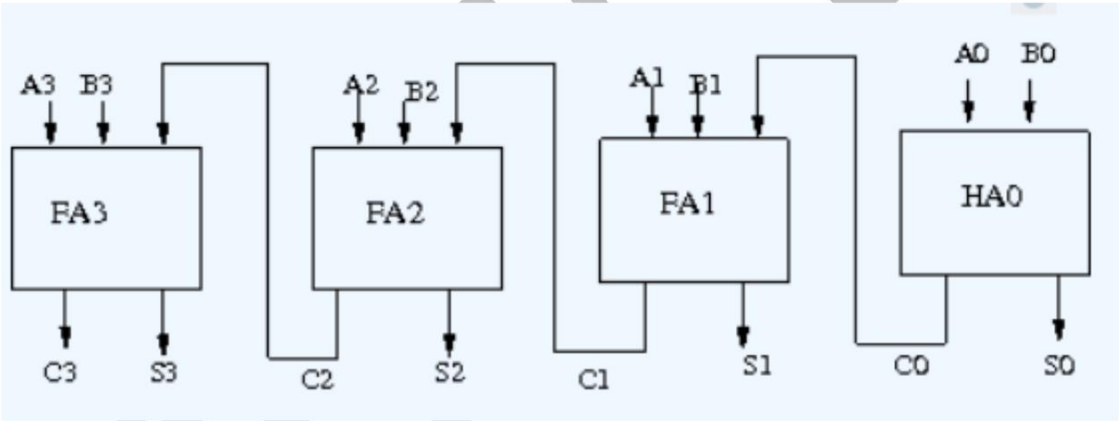


Fig 6: Circuit diagram of Ripple Carry Adder

Result:

Conclusion: Hence we have implemented, simulated, analyzed ripple carry full adder which is designed using NAND or NOR gate.

Industrial Application : Ripple Carry full adder is used in processor chip like Snapdragon, Exynous or Intel pentium for CPU part which consists of ALU (Arithmetic Block unit) . This Block is used to make operations like Add, subtract, Multiply etc. A full adder adds binary numbers and accounts for values carried in as well as out. . It is used in the ALU also.

QUESTIONNAIRE

Que 1. What is the main difference between half-adders and full-adder?

- Nothing basically; full-adders are made up of two half-adders.
- Full-adders can handle double digit numbers.
- Full-adders have a carry input capability.
- Half-adders can only handle single digit numbers.

Que 2. The MOST commonly used system for representing signed binary numbers is the

- 2's-complement system
- 1's-complement system
- 10's-complement system
- Sign-magnitude system

Que 3. One way to make a four-bit adder perform subtraction is by

- Inverting the output
- Inverting the carry-in
- Inverting the B inputs
- Invert both carry-in and B

Que 4. Which one of the following will give the sum of full adders as output?

- Three bit parity check
- Three point majority circuit
- Three bit counter
- Three bit comparator

Que 5. The number of full and half-adders required to add 16-bit numbers is

- 1 half-adder, 5 full-adders
- 4 half-adders, 12 full-adders
- 8 half-adders, 8 full-adders
- 16 half-adders, 0 full-adders

Que 6. What is the time complexity of ripple carry addition if the sum has n number of bits?

- O(nlogn)
- O(n)

- O(1)
- O(logn)

Que 7. What is the disadvantage of ripple-carry adder?

- More stages are required to a full-adder
- The interconnections are more complex
- It is slow due to propagation time
- All of the above are correct

Que 8. The space complexity of ripple carry adder are respectively

- O(nlogn)
- O(n)

- O(1)
- O(logn)

Que 9. What is the disadvantage of ripple-carry adder?

Que 10. What is the major difference between half-adders and full-adders?

Experiment No. : 05

Aim: To implement 4-bit carry lookahead adder.

Objective:

1. To reducing computation time with respect of ripple carry adder by using carry generate and propagate functions.
2. the adder will add two 4 bit numbers.

Theory : A **carry-lookahead adder** (CLA) or fast **adder** is a type of **adder** used in digital logic. A **carry-lookahead adder** improves speed by reducing the amount of time required to determine **carry** bits. It can be contrasted with the simpler, but usually slower, *ripple carry adder* for which the carry bit is calculated alongside the sum bit, and each bit must wait until the previous carry bit have been calculated to begin calculating its own result and carry bits (see adder for detail on ripple carry adders). The carry-lookahead adder calculates one or more carry bits before the sum, which reduces the wait time to calculate the result of the larger value bits of the adder. To reduce the computation time, there are faster ways to add two binary numbers by using carry lookahead adders. They work by creating two signals P and G known to be **Carry Propagator** and **Carry Generator**. The carry propagator is propagated to the next level whereas the carry generator is used to generate the output carry ,regardless of input carry. The block diagram of a 4-bit Carry Lookahead Adder is shown here below

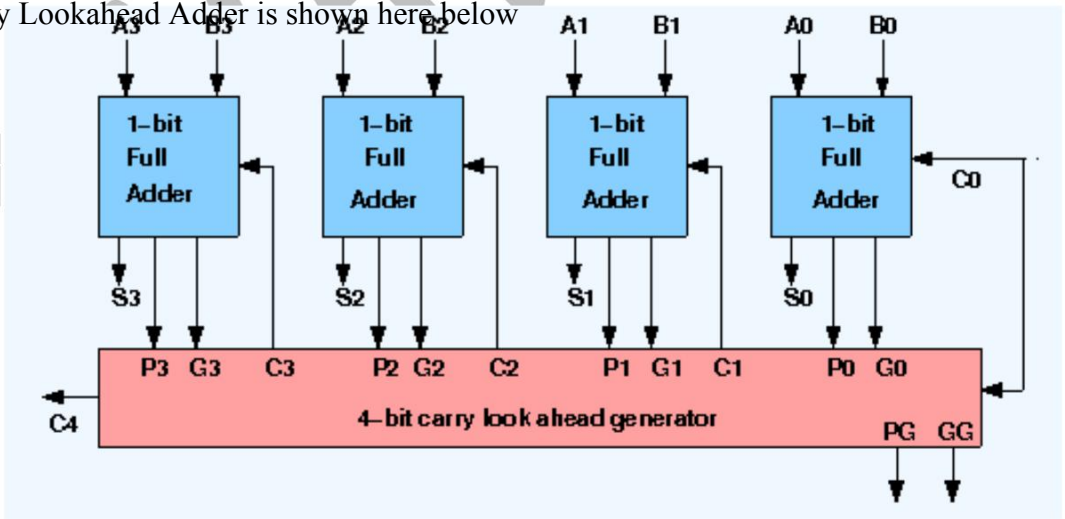


Fig1: 4-bit carry lookahead adder.

The number of gate levels for the carry propagation can be found from the circuit of full adder. The signal from input carry C_{in} to output carry C_{out} requires an AND gate and an OR gate, which constitutes two gate levels. So if there are four full adders in the parallel adder, the output carry C_5 would have 2 X 4 = 8 gate levels from C_1 to C_5. For an n-bit parallel adder, there are 2n gate levels to propagate through.

A	B	Ci	C i +1	Condition
0	0	0	0	No carry generate
0	0	1	0	
0	1	0	0	
0	1	1	1	No carry propagate
1	0	0	0	
1	0	1	1	
1	1	0	1	Carry generate
1	1	1	1	

Fig 2: carry lookahead adder truth table

Procedure:

1. Start the simulator as directed. This simulator supports 5-valued logic.

2. To design the circuit we need 7 half adder, 3 OR gate, 1 V+(to give 1 as input), 3 Digital display(2 for seeing input and 1 for seeing output sum), 1 Bit display(to see the carry output), wires.

3. The pin configuration of a component is shown whenever the mouse is hovered on any canned component of the palette or press the 'show pinconfig' button. Pin numbering starts from 1 and from the bottom left corner(indicating with the circle) and increases anticlockwise.

4. For half adder input is in pin-5,8 output sum is in pin-4 and carry is pin-1.

5. Click on the half adder component(in the Adder drawer in the pallet) and then click on the position of the editor window where you want to add the component(no drag and drop, simple click will serve the purpose), likewise add 6 more full adders(from the Adder drawer in the pallet), 3 OR gates(from Logic Gates drawer in the pallete), 1 V+, 3 digital display and 1 bit Displays(from Display and Input drawer of the pallete,if it is not seen scroll down in the drawer).

6. To connect any two components select the Connection menu of Palette, and then click on the Source terminal and click on the target terminal. According to the circuit diagram connect all the components, connect V+ to the upper input terminals of 2 digital displays according to you input. connect the OR gates according to the diagram shown in the screenshot connect the pin-1 of the half adder which will give the final carry output. connect the sum(pin-4) of those adders to the terminals of the third digital display which will give output sum. After the connection is over click the selection tool in the pallete.

7. See the output, in the screenshot diagram we have given the value 0011(3) and 0111(7) so get 10 as sum and 0 as carry.you can also use many bit switches instead of V+ to give input and by double clicking those bit switches can give different values and check the result.

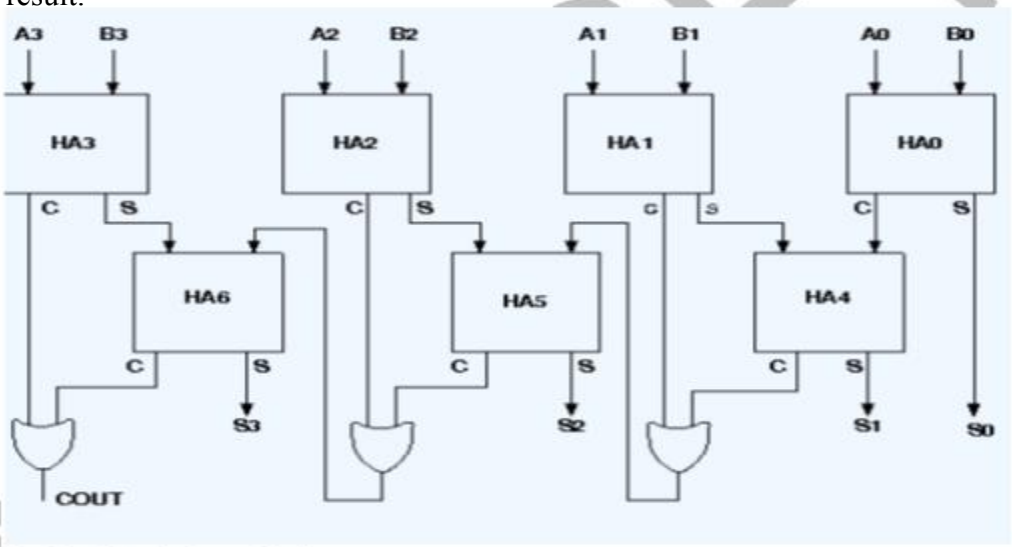


Fig 3: Circuit diagram of Carry lookahead Adder

Result:

Conclusion: Hence we have implemented, simulated, analyzed carry lookahead adder which gives less propagation delay as compared to ripple carry adder. The carry look ahead adder is used as propagation adder in various multipliers and gives less delay.

Industrial Application: Carry lookahead adder used in industry in digital logic. A carry-lookahead adder improves speed by reducing the amount of time required to determine carry bits. The carry-lookahead adder calculates one or more carry bits before the sum, which reduces the wait time to calculate the result of the larger value bits of the adder. The **Kogge-Stone adder** and **Brent-Kung adder** are examples of this type of adder.

QUESTIONNAIRE

Que 1 What distinguishes the look-ahead-carry adder?

- It requires advance knowledge of the final answer
- It is faster than a ripple-carry adder
- It is slower than the ripple-carry adder
- It is easier to implement logically than a full-adder

Que 2. The time and space complexity of carry look ahead adder are respectively

- O(n), O(nlogn)
- O(n), O(1)
- O(n), O(n)
- O(logn), O(nlogn)

Que 3. Ripple carry delay is increased in carry-look-ahead adder

- True
- False

Que 4. What is the worst case delay of an n-bit carry look ahead adder?

Que 5. What is the advantage of carry look head adder over ripple carry adder?

Que 6. What is the generate and propagate functions expression at level-2?

Que 7. What is the number of additions in carry look head adder for adding m numbers?

Que 8. What is the reason for using look ahead carry adder?

Que 9. What is the number of gate levels for output carry in a parrel adder composed of 4 full adders?

Que 10. What is the number of gate levels to propagate through for an n-bit parallel adder?

Experiment No. : 06

Aim: To implement 4 bit Booth's Multiplier.

Objective: 1. To design Booth's multiplier with a controller and a datapath. This will also help in the learning of control unit design as a finite state machine

2. To understand the advantages of Booth's multiplier

- It can handle signed integers in 2's complement notion
- It decreases the number of addition and subtraction
- It requires less hardware than combinational multiplier
- It is faster than straightforward sequential multiplier

Theory : Booth's Multiplier : Booth's multiplication algorithm is an algorithm which multiplies 2 signed integers in 2's complement. The algorithm is depicted in the figure 2 with a brief description. This approach uses fewer additions and subtractions than more straightforward algorithms.

The multiplicand and multiplier are placed in the m and Q registers respectively. A 1 bit register is placed logically to the right of the LSB (least significant bit) Q0 of Q register. This is denoted by Q-1. A and Q-1 are initially set to 0. Control logic checks the two bits Q0 and Q-1. If the twi bits are same (00 or 11) then all of the bits of A, Q, Q-1 are shifted 1 bit to the right. If they are not the same and if the combination is 10 then the multiplicand is subtracted from A and if the combination is 01 then the multiplicand is added with A. In both the cases results are stored in A, and after the addition or subtraction operation, A, Q, Q-1 are right shifted. The shifting is the arithmetic right shift operation where the left most bit namely, A_{n-1} is not only shifted into A_{n-2} but also remains in A_{n-1}. This is to preserve the sign of the number in A and Q. The result of the multiplication will appear in the A and Q.

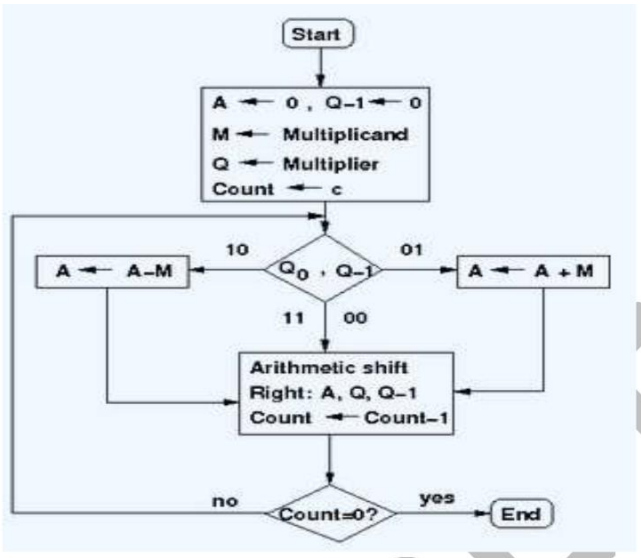


Fig 1:Flow chart of Booth's Multiplier

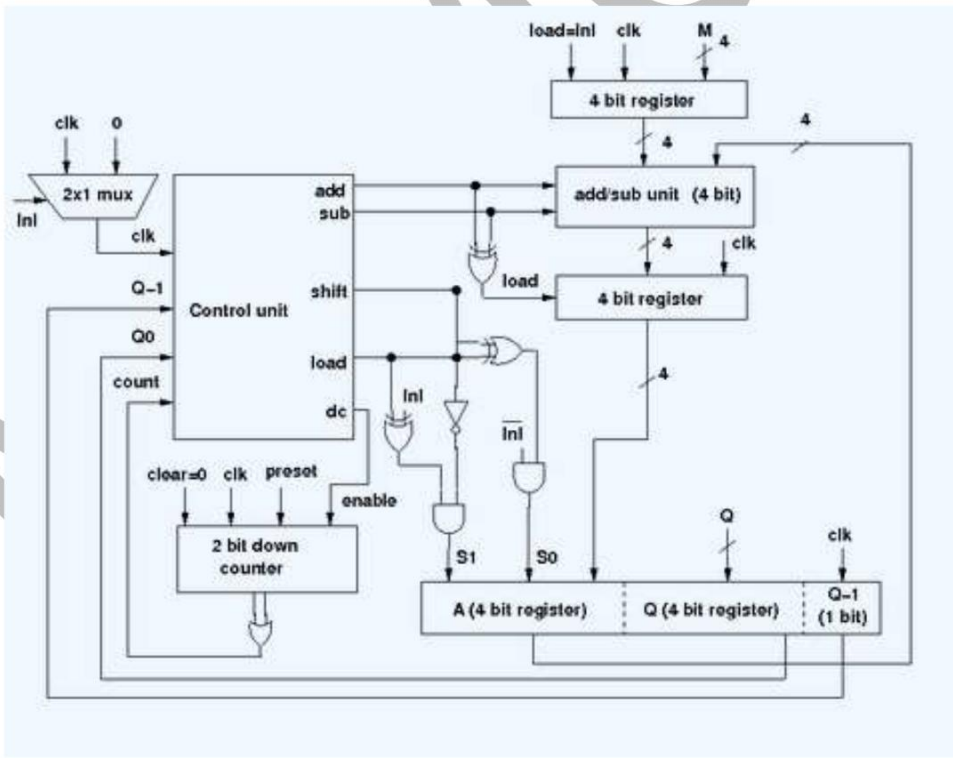


Fig 2:4 bit Booth's Multiplier

The combinational logic between the control signals of the controller and the control pins of the registers are to satisfy different operational conditions of individual components. In this design, the controller controls only the multiplication process, the initialization has been separated, so a

multiplexer is used to the clock port of the controller to deactivate ot during initialization. Except M, A, Q, Q-1 registers one more 4-bit register has been used which works as temporary storage. With the preset, the down counter is set to all one which is then decremented. The controller state chart is designed in such a way that at each state it activates only one control signal. The controller of this circuit behaves according to the state chart shown in the procedure section of this experiment. A and Q will hold the final multiplication result.

Design Issues :

Booth's algorithm can be implemented in many ways. This experiment is designed using a controller and a datapath. The operations on the data in the data path is controlled by the control signal received from the controller. The data path contains registers to hold multiplier, multiplicand, intermediate results, data processing units like ALU, adder/subtractor etc., counter and other combinational units. Following is the schematic diagram of the Booth's multiplier which multiplies two 4-bit numbers in 2's complement of this experiment. Here the adder/subtractor unit is used as data processing unit.M, Q, A are 4-bit and Q-1 is a 1-bit register. M holds the multiplicand, Q holds the multiplier, A holds the results of adder/subtractor unit. The counter is a down counter which counts the number of operations needed for the multiplication. The data flow in the data path is controlled by the five control signals generated from the controller. these signals are *load* (to load data in registers), *add* (to initiate addition operation), *sub* (to initiate subtraction operation), *shift* (to initiate arithmetic right shift operation), *dc* (this is to decrement counter). The controller generates the control signals according to the input received from the datapath. Here the inputs are the least significant Q_0 bit of Q register, Q-1 bit and count bit from the down counter.

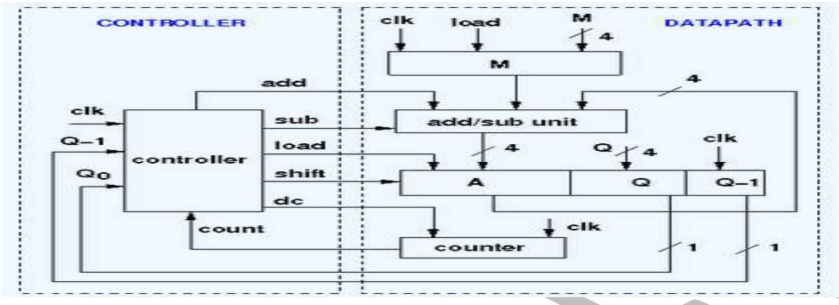
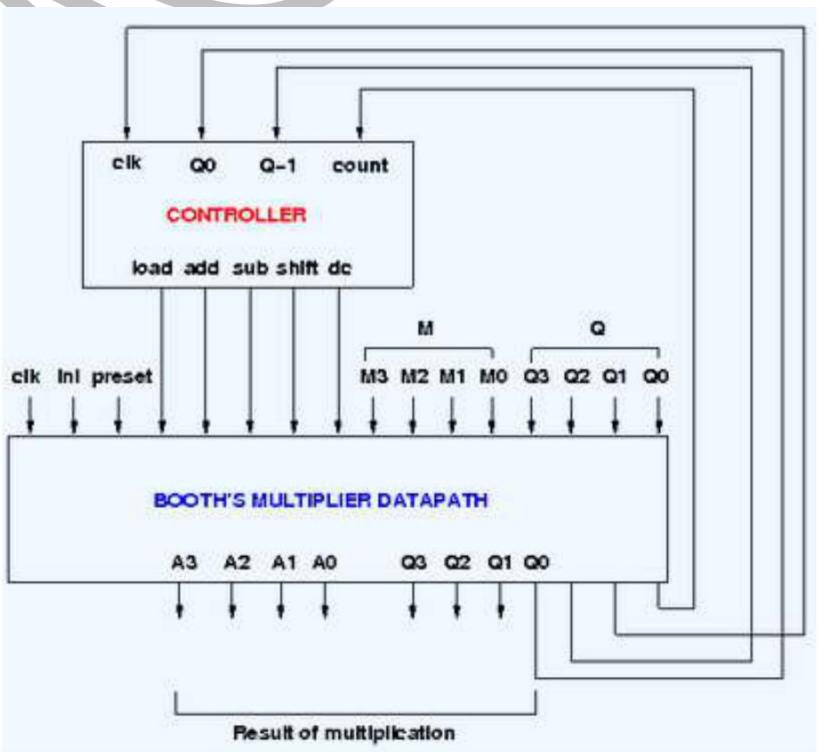
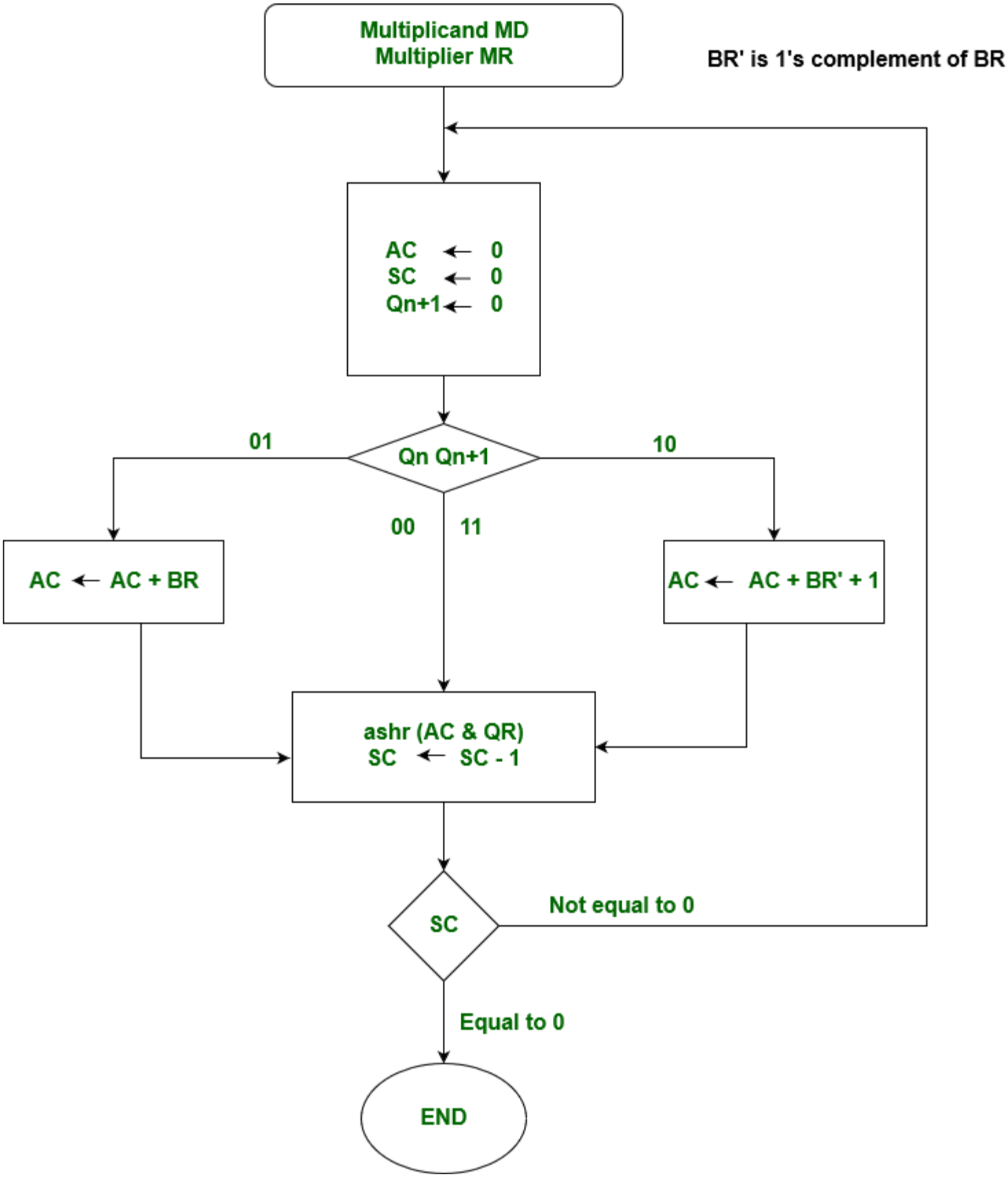
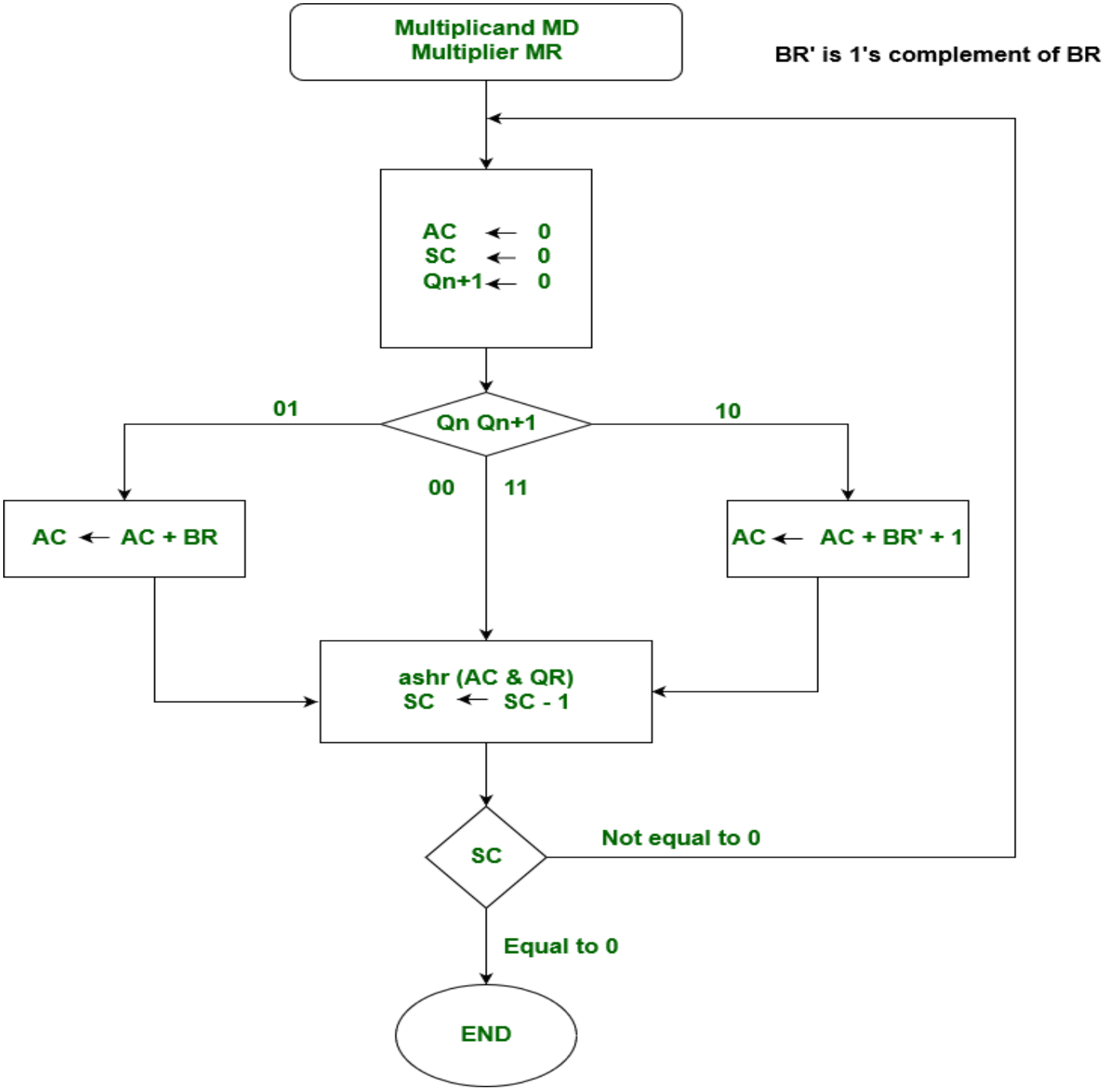


Fig 3: Design issue




Multiplicand MD
Multiplier MR
BR' is 1's complement of BR
AC ← 0
SC ← 0
Qn+1 ← 0
Qn Qn+1
01
10
00
11
AC ← AC + BR
AC ← AC + BR' + 1
ashr (AC & QR)
SC ← SC - 1
SC
Not equal to 0
Equal to 0
END



Program;

Result:

Conclusion: Hence we have implemented, simulated, analyzed 4 bit Booth's Multiplier.

Industrial Application: 4 bit Booth's Multiplier is used in many high performance systems such as calculators, digital signal processing applications, filters, microprocessors, etc. Performance of any system is generally determined by the performance of the multiplier used in it because the multiplier is generally the slowest element in any system.

QUESTIONNAIRE

Que 1. Which of the following gives best performance with Booth's algorithm?

- 0000 1111
- 1111 0000
- 1111 1111
- 1010 1010

Que 2. Booth's algorithm is faster du to less number of addition and subtraction and can handle signed numbers.

- True
- False

Que 3. Booth's algorithm increases space in case of fixed point signed multiplication

- True
- False

Que 4. 2's complement of 7 is ?

- 11100
- 11001
- 10111
- 10100

Que 5. Sign extension is a step in

- Arithmetic left shift
- Signed 16 bit integer addition
- Floating point multiplication

- None of these

Que 6. In which notation, 0 has two representations?

- Sign magnitude
- 1's complement
- 2's complement
- None of these

Que 7. What is the difference between combinational multiplier and Booth's multiplier?

Que 8. Why Booth's multiplier is faster than other approaches like shift and add multiplier?

Que 9. How Booth's multiplier handles the signed integer while others cannot?

Que 10. Prove that the multiplication of two m-bit numbers in base R gives a product of no more than 2m digits.

Experiment No. : 07

Aim; . To implement restoring division algorithm.

Objective: 1. To design restoring with a controller and a datapath. This will also help in the learning of control unit design as a finite state machine

2. To understand the advantages of restoring division .

Theory:

A division algorithm provides a quotient and a remainder when we divide two number. They are generally of two type **slow algorithm and fast algorithm**. Slow division algorithm are restoring. Restoring term is due to fact that value of register A is restored after each iteration. Here, register Q contain quotient and register A contain remainder. Here, n-bit dividend is loaded in Q and divisor is loaded in M. Value of Register is initially kept 0 and this is the register whose value is restored during iteration due to which it is named Restoring.
The step involved:

- **Step-1:** First the registers are initialized with corresponding values (Q = Dividend, M = Divisor, A = 0, n = number of bits in dividend)
- **Step-2:** Then the content of register A and Q is shifted left as if they are a single unit
- **Step-3:** Then content of register M is subtracted from A and result is stored in A
- **Step-4:** Then the most significant bit of the A is checked if it is 0 the least significant bit of Q is set to 1 otherwise if it is 1 the least significant bit of Q is set to 0 and value of register A is restored i.e the value of A before the subtraction with M
- **Step-5:** The value of counter n is decremented
- **Step-6:** If the value of n becomes zero we get of the loop otherwise we repeat from step 2
- **Step-7:** Finally, the register Q contain the quotient and A contain remainder

Examples:
Perform Division Restoring Algorithm

Dividend = 11

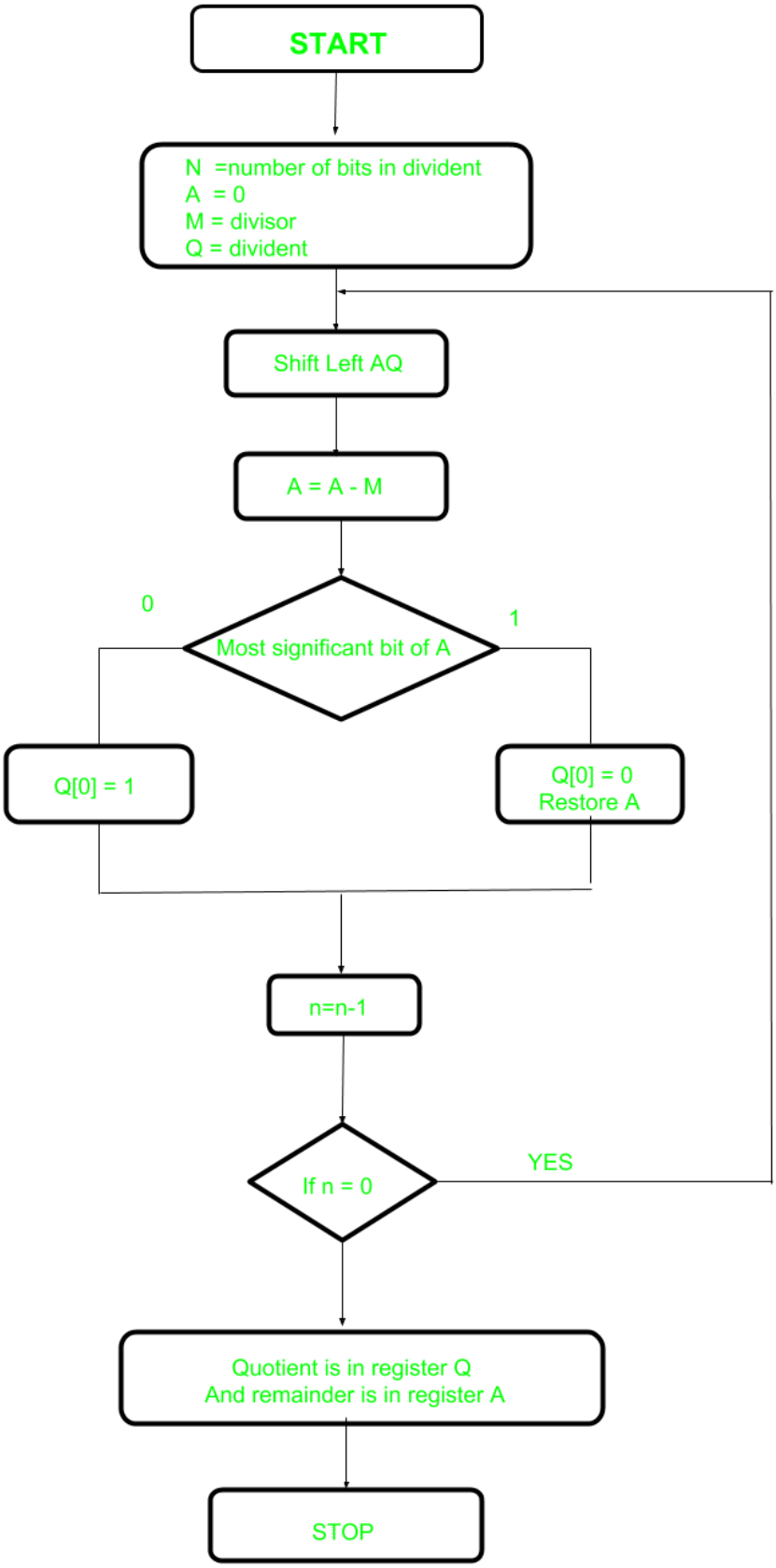
Divisor = 3

n	M	A	Q	Operation
4	00011	00000	1011	initialize
	00011	00001	011_	shift left AQ
	00011	11110	011_	A=A-M
	00011	00001	0110	Q[0]=0 And restore A
3	00011	00010	110_	shift left AQ

	00011	11111	110_	A=A-M
	00011	00010	1100	Q[0]=0
2	00011	00101	100_	shift left AQ
	00011	00010	100_	A=A-M
	00011	00010	1001	Q[0]=1
1	00011	00101	001_	shift left AQ
	00011	00010	001_	A=A-M
	00011	00010	0011	Q[0]=1

Remember to restore the value of A most significant bit of A is 1. As that register Q contain the quotient, i.e. 3 and register A contain remainder 2.

Flowchart:


START
N =number of bits in divident
A = 0
M = divisor
Q = divident
Shift Left AQ
A = A - M
0
Most significant bit of A
1
Q[0] = 1
Q[0] = 0
Restore A
n=n-1
If n = 0
YES
Quotient is in register Q
And remainder is in register A
STOP

Progrm:

Output:

Conclusion: Hence we have implemented -restoring division algorithm.

Question :

1.What is non restoring algorithm .

2How do you restore a division algorithm?

3.How restoring division algorithm is different from non-restoring division algorithm?

4.Which division algorithm for binary division is a faster one?

5.What are the limitations of the restoring algorithm?

Experiment No. : 08

Aim; . To implement non-restoring division algorithm.

Objective: 1. To design non- restoring with a controller and a datapath. This will also help in the learning of control unit design as a finite state machine

2. To understand the advantages of non-restoring division .

Theory:

To perform Non-Restoring division, it is less complex than the restoring one because simpler operation are involved i.e. addition and subtraction, also now restoring step is performed. In the method, rely on the sign bit of the register which initially contain zero named as A.This algorithm is different from the other algorithm because here, there is no concept of restoration and this algorithm is less complex than the restoring division algorithm. Let the dividend Q = 0110 and the divisor M = 0100.

The step involved:

- **Step-1:** First the registers are initialized with corresponding values (Q = Dividend, M = Divisor, A = 0, n = number of bits in dividend)
- **Step-2:** Check the sign bit of register A
- **Step-3:** If it is 1 shift left content of AQ and perform A = A+M, otherwise shift left AQ and perform A = A-M (means add 2's complement of M to A and store it to A)
- **Step-4:** Again the sign bit of register A
- **Step-5:** If sign bit is 1 Q[0] become 0 otherwise Q[0] become 1 (Q[0] means least significant bit of register Q)
- **Step-6:** Decrements value of N by 1
- **Step-7:** If N is not equal to zero go to **Step 2** otherwise go to next step
- **Step-8:** If sign bit of A is 1 then perform A = A+M
- **Step-9:** Register Q contain quotient and A contain remainder

Examples: Perform Non_Restoring Division for Unsigned Integer
Dividend =11

Divisor =3

-M =11101

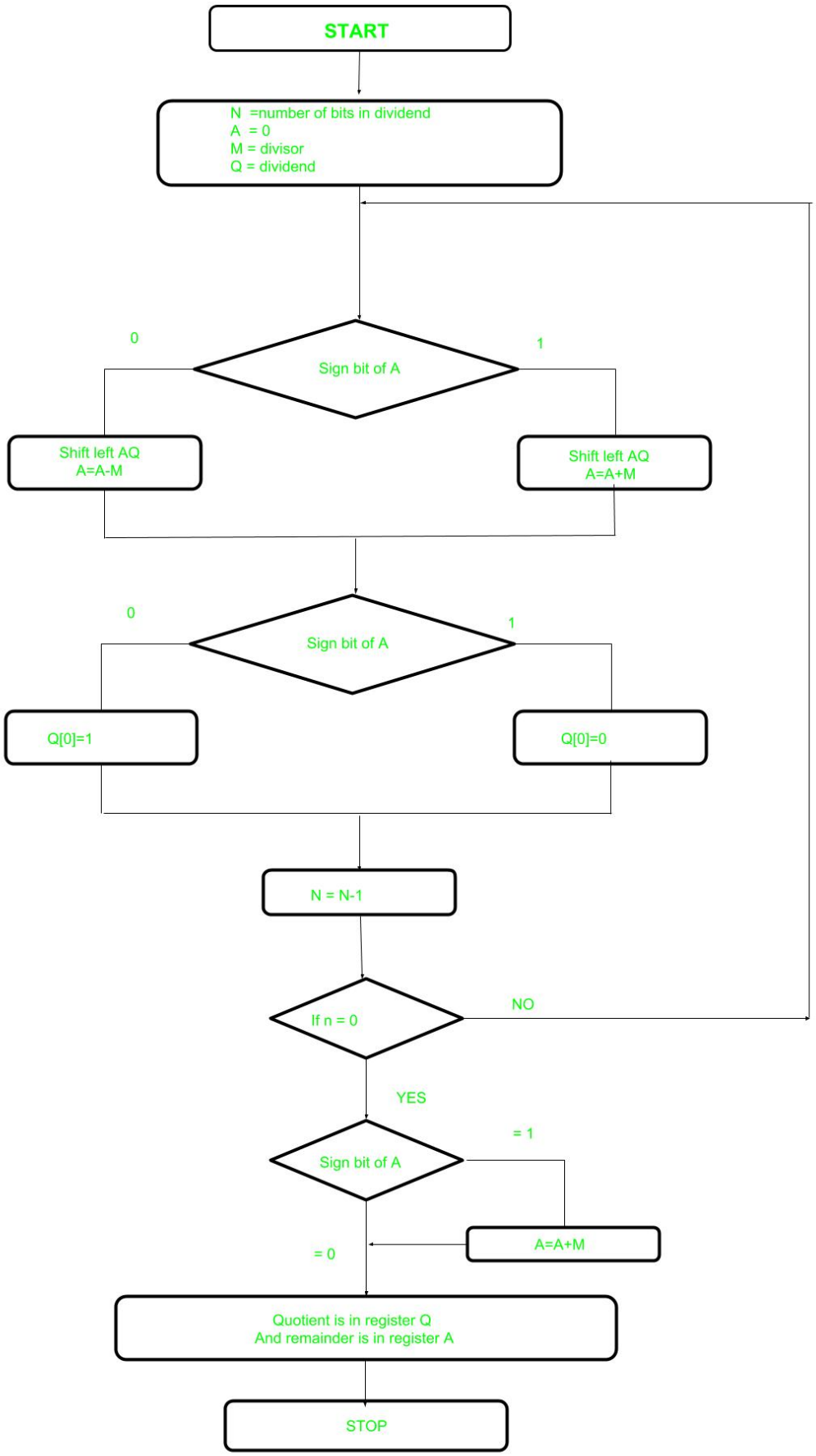
N	M	A	Q	Action
4	00011	00000	1011	Start
		00001	011_	Left shift AQ
		11110	011_	A=A-M
3		11110	0110	Q[0]=0
		11100	110_	Left shift AQ

	11111	110_	A=A+M
2	11111	1100	Q[0]=0
	11111	100_	Left Shift AQ
	00010	100_	A=A+M
1	00010	1001	Q[0]=1
	00101	001_	Left Shift AQ
	00010	001_	A=A-M
0	00010	0011	Q[0]=1

Quotient = 3 (Q)

Remainder = 2 (A)

Flowchart:


START
N =number of bits in dividend
A = 0
M = divisor
Q = dividend
0
Sign bit of A
1
Shift left AQ
A=A-M
Shift left AQ
A=A+M
0
Sign bit of A
1
Q[0]=1
Q[0]=0
N = N-1
If n = 0
NO
YES
Sign bit of A
= 1
A=A+M
= 0
Quotient is in register Q
And remainder is in register A
STOP

Progrm:

Output:

Conclusion: Hence we have implemented non-restoring division algorithm.

QUESTIONS :

1.What is non-restoring algorithm .

2.What are the advantages of non-restoring over restoring division?

3.What is the difference between restoring and non-restoring division algorithm?

4.Which is the faster division algorithm?

5..Which algorithm is used for division of integers?

6.What are the limitations of the non-restoring algorithm?

Experiment No. : 09

Aim: To implement 4-bit Arithmetic Logic Unit.

Objective: To understand behaviour of arithmetic logic unit from working module and designing an arithmetic logic unit for given parameter.

Theory :

Design of ALU:

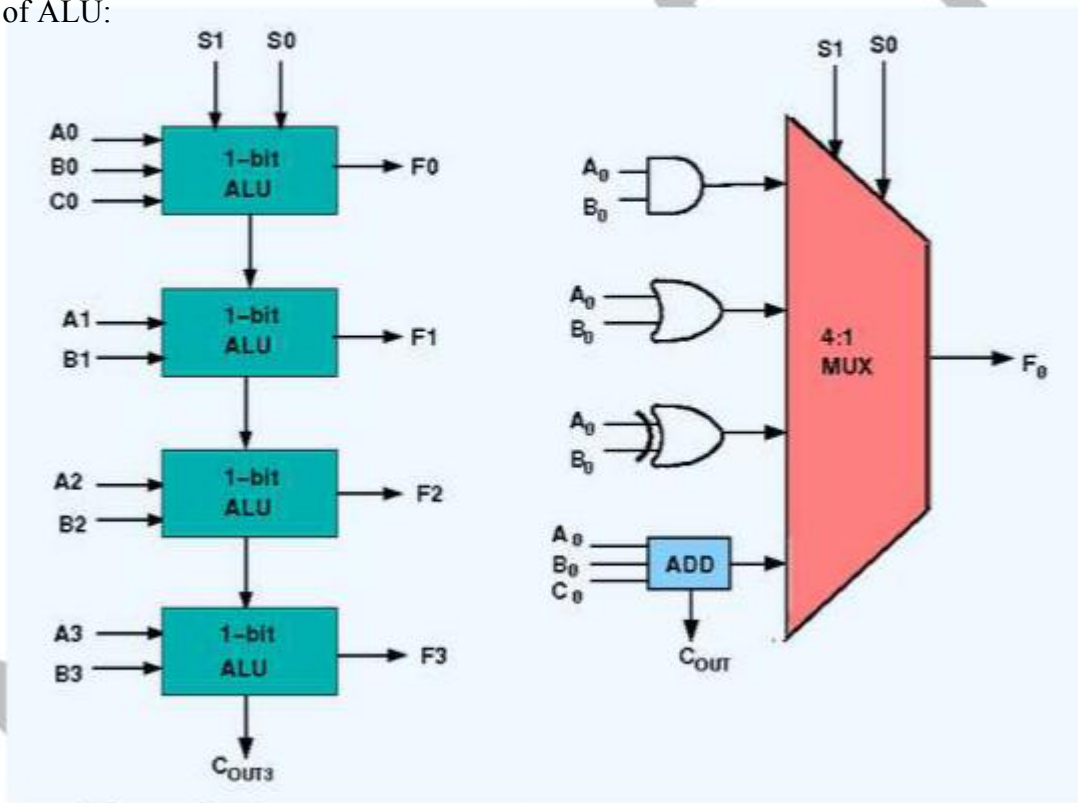


Fig 1: 4-bit ALU block is combined using 4 1-bit ALU block

Design Issues :

The circuit functionality of a 1 bit ALU is shown here, depending upon the control signal S_1 and S_0 the circuit operates as follows:

for Control signal $\mathbf{S_1 = 0}$, $S_0 = 0$, the output is **A And B**,
for Control signal $\mathbf{S_1 = 0}$, $S_0 = 1$, the output is **A Or B**, for

Control signal $\mathbf{S_1 = 1}$, $S_0 = 0$, the output is **A Xor B**, for
Control signal $\mathbf{S_1 = 1}$, $S_0 = 1$, the output is **A Add B**.

The truth table for 16-bit ALU with capabilities similar to 74181 is shown here:

Required functionality of ALU (inputs and outputs are active high)

Mode Select				F_n for active HIGH operands	
Inputs				Logic	Arithmetic (note 2)
S3	S2	S1	S0	(M = H)	(M = L) (C_n=L)
L	L	L	L	A'	A
L	L	L	H	A'+B'	A+B
L	L	H	L	A'B	A+B'
L	L	H	H	Logic 0	minus 1
L	H	L	L	(AB)'	A plus AB'
L	H	L	H	B'	(A + B) plus AB'
L	H	H	L	A ⊕ B	A minus B minus 1
L	H	H	H	AB'	AB minus 1
H	L	L	L	A'+B	A plus AB
H	L	L	H	(A ⊕ B)'	A plus B
H	L	H	L	B	(A + B') plus AB
H	L	H	H	AB	AB minus 1
H	H	L	L	Logic 1	A plus A (Note 1)
H	H	L	H	A+B'	(A + B) plus A
H	H	H	L	A+B	(A + B') plus A
H	H	H	H	A	A minus 1

The L denotes the logic low and H denotes logic high.

Fig 2: Truth Table of 4-bit ALU

Procedure:

1. Start the simulator as directed. This simulator supports 5-valued logic.

2. To design the circuit we need 4 1-bit ALU, 11 Bit switch(to give input, which will toggle its value with a double click), 5 Bit displays(for seeing output), wires.

3. The pin configuration of a component is shown whenever the mouse is hovered on any canned component of the palette. Pin numbering starts from 1 and from the bottom left corner(indicating with the circle) and increases anticlockwise.

4. For 1-bit ALU input A0 is in pin-9,B0 is in pin-10, C0 is in pin-11(this is input carry), for selection of operation, S0 is in pin-12, S1 is in pin-13, output F is in pin-8 and output carry is pin-7

5. Click on the 1-bit ALU component(in the Other Component drawer in the pallet) and then click on the position of the editor window where you want to add the component(no drag

and drop, simple click will serve the purpose), likewise add 3 more 1-bit ALU(from the other Component drawer in the pallet), 11 Bit switches and 5 Bit Displays(from Display and Input drawer of the pallet, if it is not seen scroll down in the drawer), 3 digital display and 1 bit Displays(from Display and Input drawer of the pallet,if it is not seen scroll down in the drawer)

6. To connect any two components select the Connection menu of Palette, and then click on the Source terminal and click on the target terminal. According to the circuit diagram connect all the components. Connect the Bit switches with the inputs and Bit displays component with the outputs. After the connection is over click the selection tool in the pallete.

7. See the output, in the screenshot diagram we have given the value of S1 S0=11 which will perform add operation and two number input as A0 A1 A2 A3=0010 and B0 B1 B2 B3=0100 so get output F0 F1 F2 F3=0110 as sum and 0 as carry which is indeed an add operation. you can also use many other combination of different values and check the result. The operations are implemented using the truth table for 4 bit ALU given in the theory.

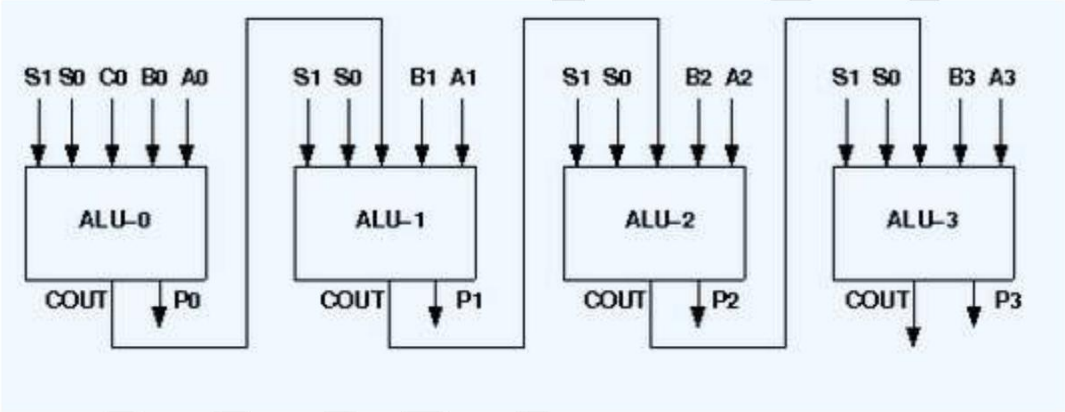


Fig 3: Circuit diagram of 4-bit ALU

Result :

Conclusion: Hence we have implemented, simulated, analyzed 4 bit ALU.

Industrial Application: ALU is designed to perform complex functions in a system, the resulting higher circuit complexity, cost, power consumption and larger size makes this impractical in many cases. ALU is a group of simple ALUs that calculates a square root in stages, with intermediate results passing through ALUs arranged like a factory production line.

QUESTIONNAIRE

Que 1. Both the ALU and control unit of CPU employ special purpose storage which is called

- Registers
- Buffers
- Decoders
- Multiplexers

Que 2. The number of levels in the design of a 64 X 1 mux using 4 X 1 mux

- 4
- 5
- 3
- 2

Que 3. On which factor the number of functions realized by a decoder depends?

- Number of gates
- Fanout capacity
- Both
- None of these

Que 4. A switching function is symmetric with respect to a set of literals if and only if the function remains unchanged after

- Any permutation of the literals
- All the literals are changed in clockwise order
- Two of these literals are interchanged
- All the literals are changed in anticlockwise order

Que 5. The number of canonical expressions that can be developed over a 3-valued boolean algebra is

- 16
- 32
- 64
- 8

Que 6. How many 3 to 8 line decoders are required for a 1-of-32 decoder?

- 8
- 1
- 4
- 2

Que 7. Design ALU with one select variable s and two data inputs A and B. when s=0, the circuit performs addition operation F=A+B. when s=1, the circuit performs increment operation F=A+1.

Que 8. How to design 16-bit ALU using 4-bit ALU ?

Que 9. What does the selector inputs to an arithmetic logic unit (ALU) determine ?

Que 10. Describe with a schematic diagram, how the ALU is connected to the processor registers through common bus.

Experiment No. : 10

Aim: To design CPU.

Objective: Our main objective for this experiment is to show the basic top level functionality, organization and architecture of a computer. This top level structure and function of a basic computer is important because of its explanatory power in understanding the nature of a computer.

Theory : At the top level a computer consists of a CPU (central processing unit), memory, I/O components, with one or more modules of each type. These modules are interconnected in a specific manner to achieve the basic functionality of a computer i.e. executing programs. At the top level a computer system can be described as follows:

- Describing the external behavior of each component i.e the data and the control signals that it exchanges with other components.
- Describing the interconnection structure and the controls required.

We are considering the Von Neumann architecture. Some of the basic features of this architecture are as follows:

- data and instructions are stored in a single read-write memory
- the contents of the memory are addressable by location
- execution occurs in a sequential manner (unless explicitly specified) from one instruction to the next top level components and interactions among them: CPU exchanges data with memory. For this CPU uses two internal resisters. Following is a block diagram of a basic computer system:

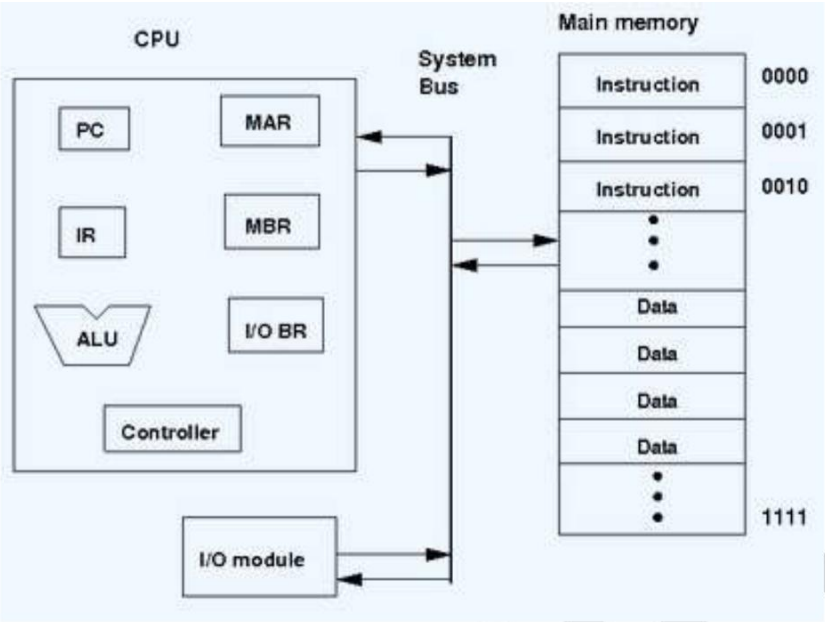
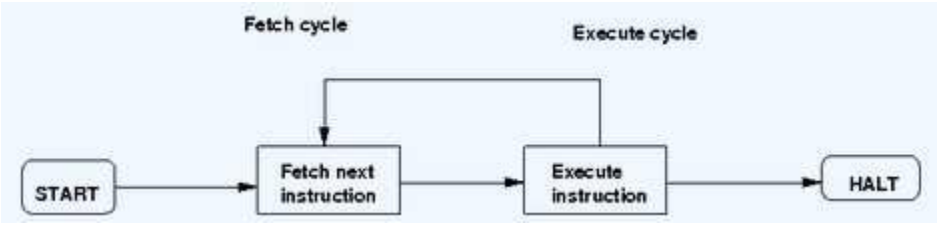


Fig 1:Computer System block Diagram

- Memory address register (MAR) which specifies the address for the next read or write.
- Memory buffer register (MBR) which contains the data to be written into memory or receives the data read from memory.
- I/O buffer (I/O BR) register is used for the exchange of data between an I/O module and the CPU.
- A memory module consists of a set of a locations defined by sequentially numbered addresses. Each location contains a binary number that can be interpreted as either an instruction or data.
- An I/O module transfers data from external devices to CPU and memory and vise versa

The basic function of a computer is to execute a program which consists of a set of instructions stored in the memory. Processing required for a single instruction is called an instruction cycle which consists instruction fetch and instruction execute. A register called program counter (PC) holds the address of the next instruction. Unless told otherwise the processor always increments PC after each instruction fetch so that the next instruction is fetched in sequence. The fetched instruction is fetched into a register called instruction register (IR).

The basic instruction cycle is shown in the following figure:



After fetching an instruction, processor execute the instruction by doing some processing on the data which may involve arithmetic and logic unit (ALU), specified by the instruction, then processor writes back the result (if any) to the memory.

This experiment provides a single instruction CPU with built-in controller. A working memory has been provided to check the working principle of the CPU. The single instruction which this CPU supports is SBN (subtract and branch if negative). The format of this instruction is:

- SBN a,b,c Mem[a] = Mem[a] - Mem[b] if(Mem[a] < 0) goto c
- a, b, c are 4 bit addresses

- Mem[a] denotes the contents of memory address a

In this experiment, no I/O module has been included for the purpose of simplicity. The working memory should contain the program and data in binary format. The CPU executes the program. For halting, it uses self loop, no other halt instruction is provided.

Procedure:

1. Start the simulator as directed. This simulator supports 5-valued logic.

2. To perform the experiment on the given modules, we need the CPU, the working memory with a program and data loaded, a clock input, Bit switch(to give input, which will toggle its value with a double click), Bit displays(for seeing output), wires.

3. Load memory: click on the *load memory* button in the left pane. you can either load the memory by filling the form or you can directly load the program form a text file. The memory provides 4-bit address space and 12 bit data word, thus providing 16 memory address starting from 0000 to 1111. For loading from file, the file should contain only binary values, it must contain 16 lines, each line containing the content to be stored in the corresponding memory address. For example, content of first line will be loaded to the 0000 address of the memory, similarly, the second line will correspond to the 0001 address and so on, finally the content of last line will be fed to the 1111 address. The progrm should use self loop for halting, for example, the instruction stored at address 1010 will cause self loop execution, if it content of a has -1 in binary format (in 2's complement), the content of b has 0 and c is 1010, then once the execution reaches to this 1010 address, it will finally point to itself.

4. Instantiating the memory: after loading the memory, click on the memory component from the computer design drawer in the palette of the simulator then click on the position of the design editor where you want to put the component(no drag and drop, simple click will serve the purpose). T

5. The pin configuration of the component is shown whenever the mouse is hovered on any canned component of the palette or pressing the *show pin configuration* button on the toolbar will show it constantly in the left pane. Pin numbering starts from 1 and from the bottom left corner(indicating with the circle) and increases anticlockwise.

6. Pin configuration of the memory module:

- Input pins (upper terminals): memory enable : 30, R/W' : 29, address : 25-28, data : 13-24 (13 is LSB)

- Output pins(lower terminals): data output : 1-12 (1 is MSB).

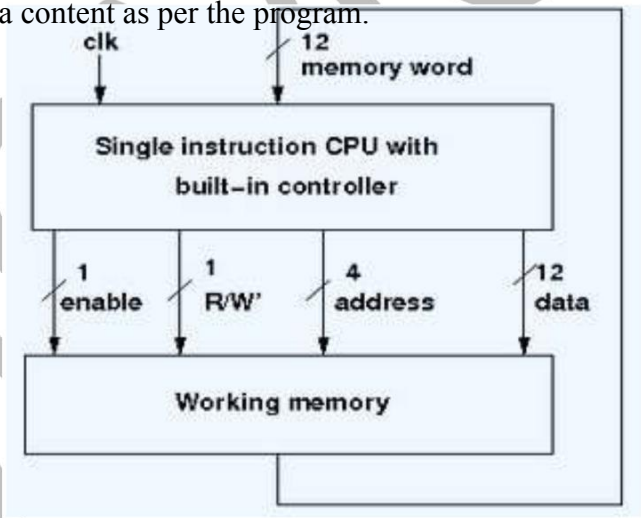
7. Instantiate the CPU from the computer design drawer in the palette of the simulator then click on the position of the design editor where you want to put the component.

8. Pin configuration of the CPU:

- Input pins (upper terminals): data input : 20-31 (20 is MSB) ,clock input : 19
- Output pins(lower terminals): memory enable : 1, R/W' : 2, address : 3-6(3 is MSB), data output : 7-18 (7 is MSB)

9. To connect any two components select the Connection menu of Palette, and then click on the Source terminal and click on the target terminal. According to the following diagram connect all the components. Connect the memory outputs to the input terminals of the CPU, specified datapath outputs to the inputs of the controller, the clock input, Bit switches with the inputs and Bit displays component with the outputs (from Display and Input drawer of the pallet,if it is not seen scroll down in the drawer). After the connection is over click the selection tool in the pallete.

10. Start clock and observe the behavior of the CPU. See the content of memory by clicking *show memory* button in the left pane. Observe how the program is executing sequentially and modify the data content as per the program.



Result:

Conclusion: Hence we have designed and analyzed CPU.

Industrial Application: CPU is designed for software development and developing project ,data storage, analysis of huge data .CPU is also designed for graphics.

QUESTIONNAIRE

Que 1. The program controller of digital computer works

- Synchronously
- Progressively
- Asynchronously
- At random

Que 2. General purpose internal registers are also called

- Stack
- Address register
- Status register
- Scratch pad

Que 3. Microprogramming is designing of

- ALU
- Control unit
- CPU
- None of these

Que 4. How many units in a single bus structure can communicate at a time?

- 4
- 3
- 2
- 1

Que 5. A single bus structure is found in generally

- Main frames
- Super computers
- Mini-and-micro computers
- High performance machines

Que 6. During read operation what is fetched by the CPU?

- Only instruction
- Only data
- Only address
- All of the above

Que 7. The technique which repeatedly uses the same block of internal storage during different stage of problem is called

- Swapping
- Overlapping
- Overlay
- None of these

Que 8. In a computer, the larger RAM increases the speed, because it reduces

- Disk I/Os
- Need for external memory
- Need for a data-wide path
- None of these

Que 9. What is the maximum memory address space that the processor can access directly if it is connected to a 16-bit memory address?

Que 10. What are the main components of any general purpose computer?

Que 11. What is the advantage of multiple-bus architecture rather than using single-bus architecture?