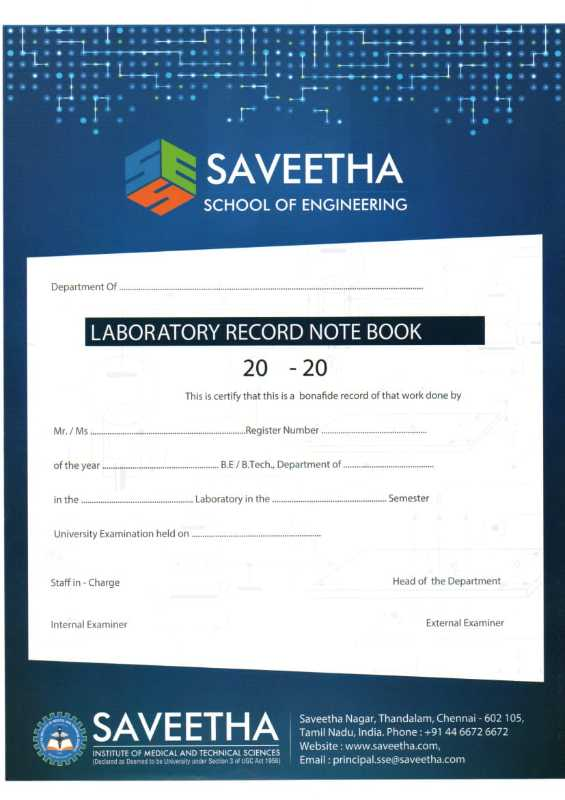
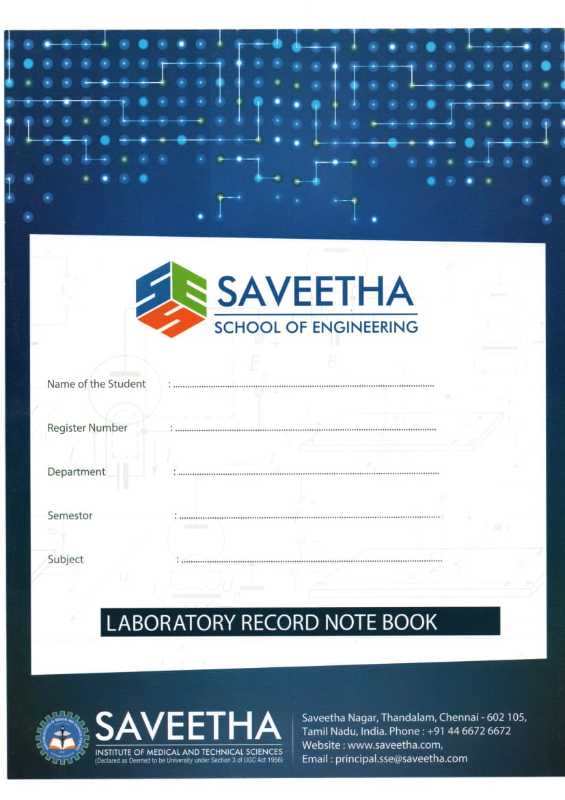
****

**Saveetha School of Engineering**

**Saveetha Institute of Medical and Technical Sciences**

**ITA01 - Computer Organization and Architecture**

**INDEX**

| **Exp.No.** | **TITLE** | **Page no.** | **Sign.** |
| --- | --- | --- | --- |
| **1.** | 8-BIT ADDITION | 4 |  |
| **2.** | 8-BIT SUBTRACTION | 5 |  |
| **3.** | 8-BIT MULTIPLICATION | 6 |  |
| **4.** | 8-BIT DIVISION | 8 |  |
| **5.** | 16 -BIT ADDITION | 10 |  |
| **6.** | 16-BIT SUBTRACTION | 11 |  |
| **7.** | 16-BIT MULTIPLICATION | 12 |  |
| **8.** | 16-BIT DIVISION | 14 |  |
| **9.** | FACTORIAL OF A GIVEN NUMBER USING 8085 MICROPROCESSOR | 15 |  |
| **10.** | LARGEST NUMBER IN AN ARRAY | 17 |  |
| **11.** | TWO BIT HALF ADDER USING LOGISIM SIMULATOR | 19 |  |
| **12.** | 32 BIT FULL ADDER USING LOGISIM SIMULATOR | 24 |  |
| **13.** | ADDITION OF TWO 4-BIT DATA USING IC 7483 | 27 |  |
| **14.** | 2’s COMPLEMENT SUBTRACTION USING IC 7485 | 29 |  |
| **15.** | 1’s & 2’s COMPLEMENT USING DIFFERENT LOCATIONS | 31 |  |

**8-BIT ADDITION**

**EXP NO : 1**

**AIM:**

**To write an Assembly language program to implement 8-Bit Addition using 8085 processor.**

**PROGRAM:**

| **ADDRESS** | **MNEMONICS** | **COMMENTS** |
| --- | --- | --- |
| 8100 | LDA 8500 | Load 8500 address in accumulator |
| 8103 | MOV B,A | Move Accumulator value in B register |
| 8104 | LDA 8501 | Load 8501 address in Accumulator |
| 8107 | ADD B | Add B register with Accumulator |
| 8108 | STA 8502 | Store the content of the Accumulator into 8502 |
| 810B | RST 1 | Break point (ACC VALUE DISPLAY) |

**INPUT:**

8500 03H

8501 04H

**OUTPUT:**

8502 07H

**RESULT:**

Thus the program was executed successfully using 8085 processor simulator.

**8-BIT SUBTRACTION**

**EXP NO: 2**

**AIM: To write an Assembly language program to implement 8-Bit Subtraction using 8085 processor.**

**PROGRAM:**

| **ADDRESS** | **MNEMONICS** | **COMMENTS** |
| --- | --- | --- |
| 8100 | LDA 8500 | Load 8500 address in accumulator |
| 8103 | MOV B,A | Move Accumulator value in B register |
| 8104 | LDA 8501 | Load 8501 address in Accumulator |
| 8107 | SUB B | Subtract B register with Accumulator |
| 8108 | STA 8502 | Store the content of the Accumulator into 8502 |
| 810B | RST 1 | Break point (ACC VALUE DISPLAY) |

**INPUT:**

8500 45H

8501 13H

**OUTPUT:**

8502 32H

**RESULT:**

Thus the program was executed successfully using 8085 processor simulator.

**8-BIT MULTIPLICATION**

**EXP NO: 3**

**AIM: To write an Assembly language program to implement 8-Bit Multiplication using 8085 processor.**

**PROGRAM:**

| **ADDRESS** | **MNEMONICS** | **COMMENTS** |
| --- | --- | --- |
| 8100 | LDA 8500 | Load 8500 address into accumulator |
| 8103 | MOV B,A | Move Accumulator value into B register |
| 8104 | LDA 8501 | Load 8501 address into Accumulator |
| 8107 | MOV C,A | Move accumulator value into C register |
| 8108 | CPI 00 | If X2=0 then result is 0 and exit |
| 810A | JZ LOOP | If carry is zero then jump into 8116 |
| 810D | XRA A | Clear ACC and flags ass the B as many as the C |
| 810E | LOOP1: ADD B | Add B register with Accumulator |
| 810F | DCR C | Decrement C register |
| 8110 | JZ LOOP | Store the result and exit |
| 8113 | JMP LOOP1 | Multiply start address=810E |
| 8116 | LOOP: STA 8502 | Store the content of the accumulator into 8502 |
| 8119 | RST 1 | Breakpoint (ACC VALUE DISPLAY) |

**INPUT:**

8500 06H

8501 02H

**OUTPUT:**

8502 0CH

**RESULT:**

Thus the program was executed successfully using 8085 processor simulator.

**8-BIT DIVISION**

**EXP NO: 4**

**AIM: To write an Assembly language program to implement 8-Bit Division using 8085 processor.**

**PROGRAM:**

| **ADDRESS** | **MNEMONICS** | **COMMENTS** |
| --- | --- | --- |
| 8100 | LDA 8501 | Load 8501 value into accumulator |
| 8101 | MOV B,A | Move Accumulator value into Register B |
| 8103 | LDA 8500 | Load 8500 value into Accumulator |
| 8104 | MVI C,00 | Count for quotient |
| 8107 | LOOP: CMP B | Check for A<B |
| 8109 | JC LOOP1 | If A<B then go to store |
| 810A | SUB B | Subtract the register b with Accumulator |
| 810D | INR C | Increment the C register |
| 810E | JMP LOOP | Jump into 8109 |
| 811F | STA 8503 | Store remainder value into 8503 |
| 8112 | DCR C | Decrement the C register |
| 8115 | MOV A,C | Move register c value into accumulator |
| 8116 | LOOP1: STA 8502 | Store the quotient value into 8502 |
| 8119 | RST 1 | Breakpoint |

**INPUT:**

8500 06H

8501 02H

**OUTPUT:**

8502 03H(Quotient)

8503 00H(Reminder)

**RESULT:**

Thus the program was executed successfully using 8085 processor simulator.

**16-BIT ADDITION**

**EXP NO: 5**

**AIM: To write an Assembly language program to implement 16-Bit Addition using 8086 processor.**

**PROGRAM:**

| **ADDRESS** | **MNEMONICS** | **COMMENTS** |
| --- | --- | --- |
| 1100 | MOV SI,1200H | Move 1200h into SI pointer |
| 1103 | LODSW | Load the first data into AX |
| 1104 | MOV BX,AX | Move AX value into BX |
| 1106 | LODSW | Load the second data into AX |
| 1107 | ADD BX,AX | ADD BX,AX registers |
| 1109 | MOV DI,1300H | Load 1300h address location into DI |
| 110C | MOV [DI],BX | Store BX value into memory |
| 110E | HLT | Stop the program |

**INPUT:**

1200 13H 1201 13H

1202 14H 1203 14H

**OUTPUT:**

1300 27H

1301 27H

**RESULT:**

Thus the program was executed successfully using 8086 processor simulator.

**16-BIT SUBTRACTION**

**EXP NO: 6**

**AIM: To write an Assembly language program to implement 16-Bit Subtraction using 8086 processor.**

**PROGRAM:**

| **ADDRESS** | **MNEMONICS** | **COMMENTS** |
| --- | --- | --- |
| 1100 | MOV SI,1200H | Load 1200 into SI |
| 1103 | LODSW | Load the first data |
| 1104 | MOV BX,AX | Move AX value into BX |
| 1106 | LODSW | Load the second data |
| 1107 | SUB BX,AX | Subtract AX from BX |
| 1109 | MOV DI,1300H | Load 1300 address into DI |
| 110C | MOV [DI],BX | Load BX value into DI |
| 110E | HLT | Stop the program |

**INPUT:**

1200 08H 1201 08H

1202 04H 1203 04H

**OUTPUT:**

1300 04H

1301 04H

**RESULT:**

Thus the program was executed successfully using 8086 processor simulator.

**16-BIT MULTIPLICATION**

**EXP NO: 7**

**AIM: To write an Assembly language program to implement 16-Bit Multiplication using 8086 processor.**

**PROGRAM:**

| **ADDRESS** | **MNEMONICS** | **COMMENTS** |
| --- | --- | --- |
| 8100 | LHLD 8500 | Load 8500 value into HL pair |
| 8103 | MOV D,H | Move the Highest Byte into D register |
| 8104 | MOV E,L | Move the lowest byte into E register |
| 8105 | LDA 8502 | Load 8502 value into accumulator |
| 8108 | MOV C,A | Move accumulator value into C register |
| 8109 | CPI 00 | If X2=0 then result is 0 and exit |
| 810B | JZ LOOP1 8116 | If carry is zero then jump into 8119 |
| 810E | LXI H,0000 | Intialize the HL and DE added value in HL reg |
| 8111 | DAD D | 16 Bit add bet HL and DE added value in HL reg |
| 8112 | DCR C | Decrement the C register |
| 8113 | JZ LOOP1 | Store the result and Exit |
| 8116 | JMP LOOP | Jump multiply(multiply=8111) |
| 8119 | SHLD 8503 | Store the content of accumulator into 8503 |
| 811C | HLT | End |

**INPUT:**

8501 04H

8502 02H

**OUTPUT:**

8503 08H

8504 08H

**RESULT:**

Thus the program was executed successfully using 8086 processor simulator.

**16-BIT DIVISION**

**EXP NO: 8**

**AIM: To write an Assembly language program to implement 16-Bit Division using 8086 processor.**

**PROGRAM:**

| **ADDRESS** | **MNEMONICS** | **COMMENTS** |
| --- | --- | --- |
| 1100 | MOV DX,0000 | Clear DX registers |
| 1103 | MOV AX,0006H | Load the dividend in AX |
| 1106 | MOV CX,0004H | Load the divisor value in BX |
| 1109 | DIV CX | Divide the two data's |
| 110B | MOV DI,1300H | Load 1300 address into DI |
| 110E | MOV [DI],AH | Load AL value into DI |
| 1110 | INC DI | Increment DI |
| 1111 | MOV [DI],AH | Load AH value into DI |
| 1113 | INC DI | Increment DI |
| 1114 | MOV [DI],DX | Load DX value into DI |
| 1116 | HLT | END |

**INPUT:**

AX 0006H

CX 0004H

**OUTPUT:**

1300 01H

1301 00H

1302 02H

**RESULT:**

Thus the program was executed successfully using 8086 processor simulator.

**FACTORIAL OF A GIVEN NUMBER USING 8085 MICROPROCESSOR**

**EXP NO: 9**

**AIM: To find the factorial of a given number using 8085 microprocessor.**

**PROGRAM:**

| **ADDRESS** | **MNEMONICS** | **COMMENTS** |
| --- | --- | --- |
| 8100 | LXI H,8500H | Load data from memory |
| 8103 | MOV B,H | Load data to B register |
| 8104 | MVI D,01H | Set D register with 1 |
| 8106 | CALL MULTIPLY | Subroutine call for multiplication |
| 8109 | DCR B | Decrement B |
| 810A | JNZ FACTORIAL | Call factorial till B becomes 0 |
| 810D | INX H | Increment memory |
| 810E | MOV M,D | Store result in memory |
| 810F | HLT | Halt |
| 8200 | MOV E,B | Transfer contents of B to C |
| 8201 | MVI A,00H | Clear accumulator to store result |
| 8203 | ADD D | Add contents of D to A |
| 8204 | DCR E | Decrement E |
| 8205 | JNZ MULTIPLYLOOP | Repeated addition |
| 8208 | MOV D,A | Transfer contents of A to D |
| 8209 | RET | Return from subroutine |

**INPUT:**

8500 05H

**OUTPUT:**

8501 78H

**RESULT:**

Thus the program was executed successfully using 8085 processor simulator.

**LARGEST NUMBER IN AN ARRAY**

**EXP NO: 10**

**AIM: To find the largest number from an array using 8086 processor.**

**PROGRAM:**

| **ADDRESS** | **LABELS** | **MNEMONICS** | **COMMENTS** |
| --- | --- | --- | --- |
| 1100 |  | LXI H, 1200H | Point to get array size |
| 1103 |  | MOV C, M | Get the size of array |
| 1104 |  | INX H | Point to actual array |
| 1105 |  | MOV B, M | Load the first number into B |
| 1106 |  | DCR C | Decrease C |
| 1107 | LOOP | INX H | Point to next location |
| 1108 |  | MOV A, M | Get the next number from memory to Acc |
| 1109 |  | CMP B | Compare Acc and B |
| 110A |  | JC SKIP | if B > A,then skip |
| 110D |  | MOV B, A | If CY is 0, update B |
| 110E | SKIP | DCR C | Decrease C |
| 110F |  | JNZ LOOP | When count is not 0, go to LOOP |
| 1112 |  | LXI H,1300H | Point to destination address |
| 1115 |  | MOV M, B | Store the MAXIMUM number |
| 1116 |  | HLT | Terminate the program |

**INPUT:**

**1200 H , 05 H, 09 H, 01 H, 02 H, 07 H, 03 H**

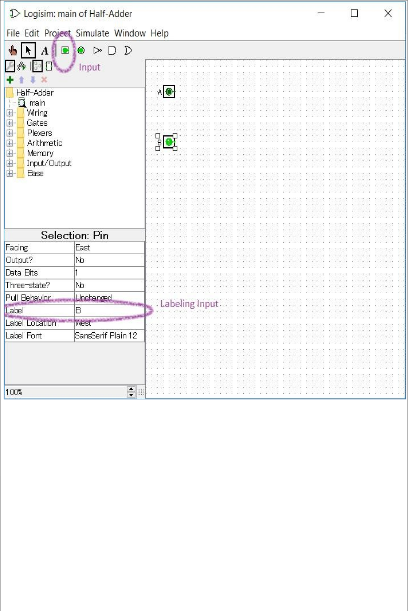
**OUTPUT:**

**1300 H 09 H**

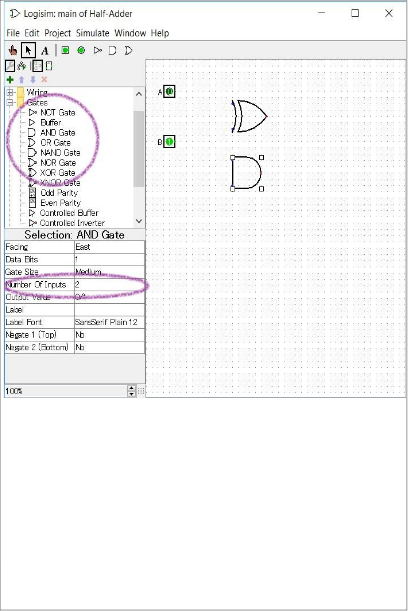
**RESULT:**

Thus the program was executed successfully using 8086 processor simulator.

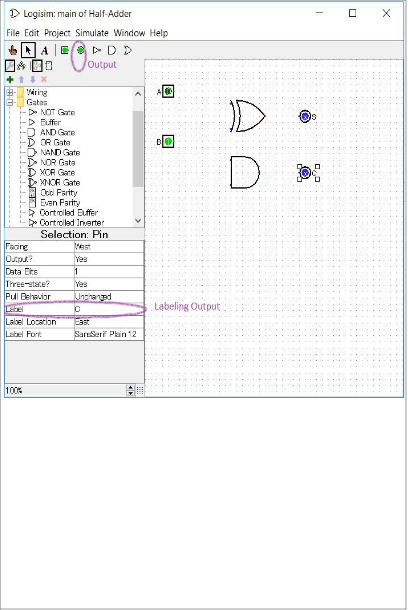
|  | **TWO BIT HALF ADDER USING LOGISIM SIMULATOR** |
| --- | --- |
| **EXP.NO: 11**  **AIM:**  To design and implement the two bit half adder using Logisim simulator.  **TRUTH TABLE:-**    S = A XOR B . C = A AND B .  THE FOLLOWING STEPS IS USED TO DRAW A HALF-ADDER CIRCUIT.   * Insert 2 inputs into the canvas.   + Label the inputs (A & B) by setting the attribute 'Label' in the attribute table.   + Note that both inputs have now 0s inside their green spots. These are the current bit value of the input. | |



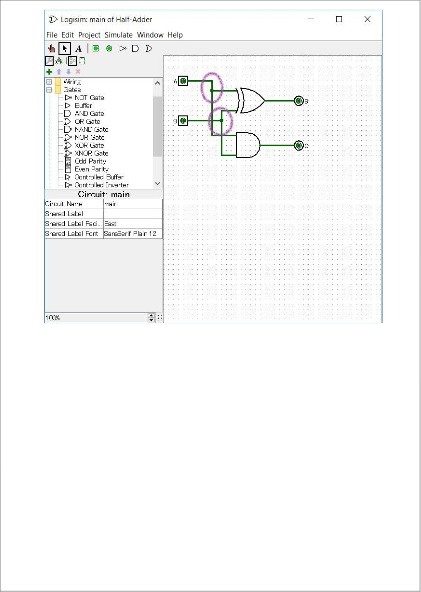
* Insert one XOR gate and one AND gate into the canvas.
  + The two gates are located inside the 'Gates' library in the explorer pane.
  + Change the 'Number of Inputs' in the attribute table to 2.

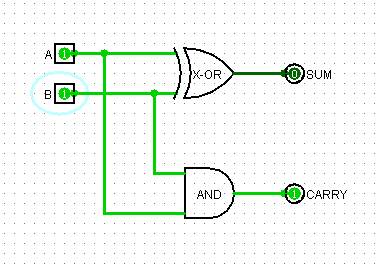


* Insert 2 outputs in the canvas
  + Label the outputs (S & C).
  + Note that both outputs have X inside the dots. X indicates an invalid value for the output.



* Connect the inputs to the XOR gate.
* Connect the inputs to the AND gate.
  + Start the connection on the AND gate input and finish it on the wire reaching the designated input.
* Connect the outputs to the gates





# 

# RESULT:-

Hence the designing of the 2-bit half adder using logisim simulator has been implemented successfully.

|  | **32-BIT FULL ADDER USING LOGISIM SIMULATOR** |
| --- | --- |
| **EXP.NO: 12** |
| **AIM:**  To design and implement 32-bit full adder by using logisim simulator.  **TRUTH TABLE:-**    **STEPS TO BE FOLLOWED:- STEP-0:-**  Make a truth table with input coloumns X,Y and cin,in one column give the result of X XOR Y.In another column,give the result for(x XOR y)XOR cin,you should now see that :   * S = x XOR y XOR Cin.   **STEP-1:-**  The sum-of-products equation for the carry output (Cout) is:Cout = x'·y·Cin + x·y'·Cin + x·y·Cin  + x·y·Cin', is not a minimal expression. Show step by step how you can reduce the expression for Cout to end up with:   * Cout = Cin·(x XOR y) + x·y.   **STEP-2:-**  It's now time to implement your 1-bit full adder in Logisim.   * Start Logisim. On the department Unix System, type logisim in a shell and press enter. If you work on a laptop or form home, downlad and install Logisim from here. | |

Open up add.circ in Logisim. Start by double-click on add1 to select the add1 circuit.

# STEP-3:-

Complete the add8 circuit by combining eight 1-bit adders.

* Add three splitters to the circuit. Each splitter should have an input bit width of 8 and a fan out of 8. Attach two east-facing splitters to the 8-bit inputs A and B. Attach a west-facing splitter to the 8-bit output S.
* Create eight instances of the add1 circuit.
* connect the S outputs of the eight add1 instances to the splitter for the 8-bit S output.
* Connect the carry inputs and outputs of the eight add1 instances so that carries will propagate appropriately from the Ci input, through the 1-bit adders, to the Co output.
* Connect the A inputs of the eight add1 instances to the splitter for the Ainput.
* Connect the B inputs of the eight add1 instances to the splitter for the Binput.

Change the values of the Ci, A, and B inputs and observe the Co and S outputs to verify the correct operation of the circuit.

# STEP-4:-

Complete the add32 circuit by combining four 8-bit adders.

* You will find three splitters in the circuit. Each splitter has an input bit width of 32 and a fan out of 4. Thus, each connection to a splitter represents 8 bits.
* Create four instances of the add8 circuit.
* Connect the 8-bit S outputs of the four add8 instances to the splitter for the 32-bit S output.
* connect the carry inputs and outputs of the four add8 instances so that carries will propagate appropriately from the Ci input, through the 8-bit adders, to the Co output.
* Connect the 8-bit A inputs of the four add8 instances to the splitter for the Ainput.
* Connect the 8-bit B inputs of the four add8 instances to the splitter for the Binput.

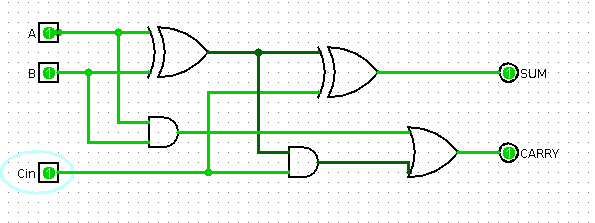
# STEP-5:-

* Within the main circuit, you will find a 32-bit adder connected side-by-side with your add 32 circuit.
* Change the values of the Ci, A, and B inputs and observe the Co and S outputs to verify the correct operation of your add32 circuit.

# STEP-6:-

* There is more than one way to implemt a logical function. An alternative expression for Cout can be found by reducing the expression using only min-terms.
* Use a Karnaugh map to reduce the expression for Cout.Note: using Karnaugh map to reduce the expression for the Sum will no be possible, it will result in the original sum-of-products for the Sum.
* Add a new circuit to the project named add1\_k and implement a new version of a 1 bit full adder using the new expression for Cout and the original sum-of-producs exreppsions for the SumSimlary, add new circuits named add8\_k and add32\_k to construct an alternate version of the 32 bit full adder.
* Add the add32\_k component to the main circuit along with the other adders and verify that they give the same results.

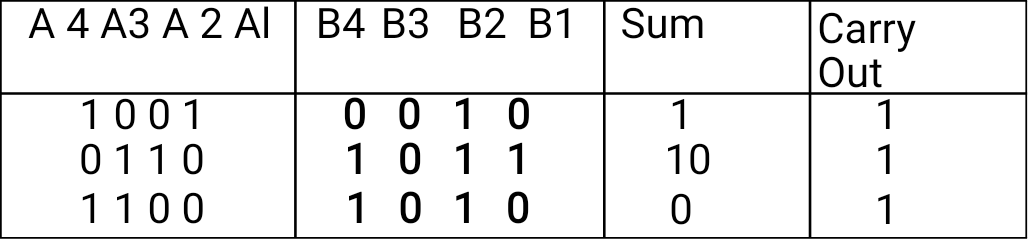
# CIRCUIT DIAGRAM:-

****

**RESULT:-**

Thus the designing of the 32-bit full adder using Logisim simulator has been implemented successfully.

|  | **ADDITION OF TWO 4-BIT DATA USING IC 7483** |
| --- | --- |
| **EXP.NO: 13** |
| **AIM:**  To design and implement the addition of two 4-bit data using IC7483.  **APPARATUS:-**   * IC type 7483 4-bit binary adder.   **ADDITION:-**   * IC type 7483 is a 4-bit binary adder with fast carry. The pin assignment is shownin Fig.l. The two 4-bit input binary numbers are Al through A4 and Bl through B4. * The 4-bit sum is obtained from S1 through S4. CO is the input carry and C4 the out carry. This IC can be used as an adder-subtractor and as a magnitude comparator.   **PROCEDURE:-**   * Design and construct a half adder circuit using only XOR gate and NAND gates (UseLogic Works). * Design and construct a full adder circuit using only XOR and NAND gates (Use Logic Works).   Use IC 7483 to add the following two 4-bit numbers A and B.  **TRUTH TABLE:-** | |

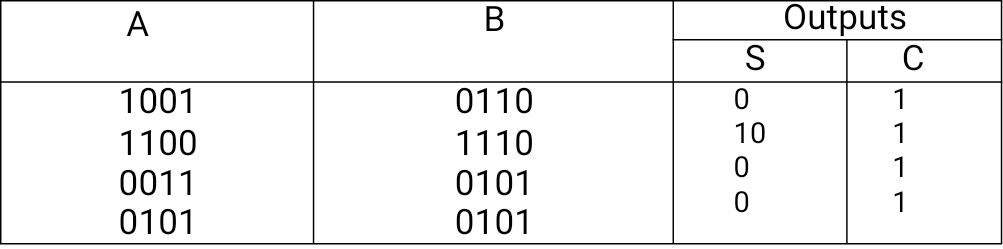


# CIRCUIT DIAGRAM:-

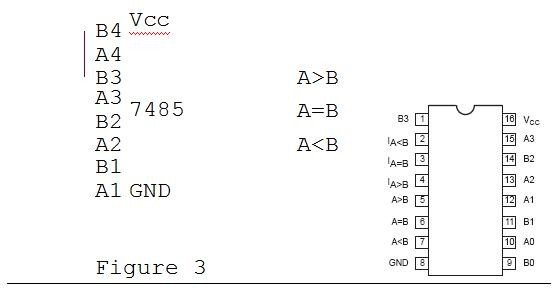
**RESULT:-**

Thus the addition of two 4-bit data using IC 7483 has been executed successfully.

|  | **2’s COMPLEMENT SUBTRACTION USING IC 7485** |
| --- | --- |
| **EXP.NO: 14** |
| **AIM:**  To design and implement 2’s complement subtraction using IC 7485.  **APPARATUS:-**   * IC type 7485 4-bit magnitude comparator.   **(A)Subtraction:-**  The subtraction of two binary numbers can be done by taking the 2's complement of the subtrahend and adding it to the minued. The 2's complement can be obtained by taking the 1's complement and adding 1. To perform A-B, we complement the four bits of B, add them to the four bits of A, and add 1 through the input carry. This is done as shown in Fig.4.Four XOR gates complement the bits of B when the mode select M=l and leave the bits of B unchaged when M = 0. Thus, when the mode select M is equal to 1, the input carry CO is equal to 1 and the sum output is A plus the 2's complement of B. When M is equal to 0, the input carry is equal to 0 and the sum generates A+B.  **(C)Magnitude Comparison:-**  The comparison of two numbers is an operation that determines whether one number is greater than, equal to, or less than the other number.The IC 7485 is a 4-bit magnitude comparator. It compares two 4-bit binary numbers (labeled as A & B), and generates an output of 1 at one of three outputs labeled A>B, A<B, A=3. Three inputs are available for cascading comparators.  **TRUTH TABLE:-** | |



# CIRCUIT DIAGRAM:-

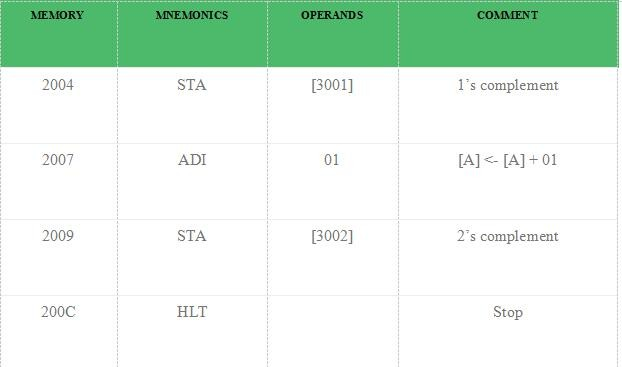


**RESULT:-**

Thus the design and implementation of 2’s complement subtraction using IC 7485

hasbeen implementation successfully.

|  | **1’s & 2’s COMPLEMENT USING DIFFERENT**  **LOCATIONS** |
| --- | --- |
| **EXP.NO: 15** |
| **AIM:**  The program to find 1’s and 2’s complement of 8-bit number where starting address is 2000 and the number is stored at 3000 memory address and store result into 3001 and 3002 memory address.  **ALGORITHM:-**  1.Load the data from memory 3000 into A (accumulator) 2.Complement content of accumulator  3.Store content of accumulator in memory 3001 (1’s complement) 4.Add 01 to Accumulator content  5.Store content of accumulator in memory 3002 (2’s complement) 6.Stop .  **EXPLANATION:-**  1.A is an 8-bit accumulator which is used to load and store the data directly 2.LDA is used to load accumulator direct using 16-bit address (3 Byte instruction) 3.CMA is used to complement content of accumulator (1 Byte instruction)  4.STA is used to store accumulator direct using 16-bit address (3 Byte instruction) 5.ADI is used to add data into accumulator immediately (2 Byte instruction) 6.HLT is used to halt the program  **PROGRAM:-** | |



# RESULT:-

Thus the 1’s & 2’s complement using different locations has been executed successfully.