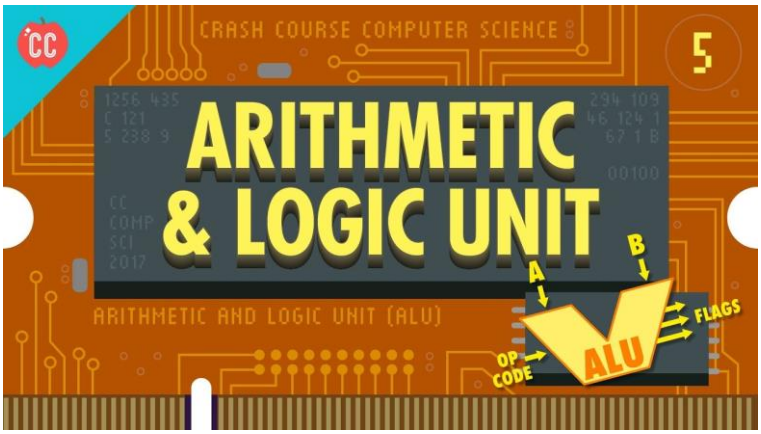


8-BIT ARITHMETIC AND LOGIC UNIT (ALU)

EEE 304 Project

GROUP NO: 04

Prepared By:
1706008 – Sheikh Munim Hussain
1706010 – Abdullah Al Mahmood
1706017 – Md. Jahidul Hoq Emon
1706020 – Shafin Shadman Ahmed
1706033 – Azazul Islam Razon

Submitted To:
Mr. Hamidur Rahman
Rifat Shahriar

Introduction:

An Arithmetic and Logic Unit (ALU) is an essential part of modern digital systems. In terms of computing and digital electronics, ALU is a combinational digital circuit that performs arithmetic and bitwise operations on integer binary numbers. ALU is a fundamental building block of CPU's, FPU's and GPU's used in computers and similar devices.

From the name, it can be easily understood that ALU has 2 parts: Arithmetic Unit and Logic Unit. If described separately, an Arithmetic Unit mainly deals with arithmetic operations such as addition, subtraction, multiplication, and division, while a Logic Unit deals with bitwise logic operations like logical AND, logical NOT, logical OR and so on.

This project aims to demonstrate how an 8-bit ALU works. The whole project was performed with the help of Proteus. The arithmetic functions implemented in this project are:

1. Addition
2. Subtraction
3. Increment (Addition by 1)
4. Decrement (Subtraction by 1)
5. Multiplication
6. Division

The logical functions implemented in this project are:

1. Logical AND
2. Logical OR
3. Logical NOT
4. Logical NAND
5. Logical NOR
6. Logical XOR
7. Logical XNOR
8. Comparator (Compares if a number is greater than the other, or both numbers are equal)

Objectives of the Project:

The objectives of this project are:

1. To implement an 8-bit Arithmetic and Logic Unit (ALU) using Proteus. The ALU will be able to perform different Arithmetic and Logic operations involving 8-bit binary numbers.

2. To examine the arithmetic and basic logic operations implemented using different 8-bit numbers. The inputs are taken in binary, and the results are mainly in binary too.

3. To show the outputs obtained using a 7-segment display. The 7-segment display shows the decimal representation of the binary output.

4. To explain the arithmetic and basic logic operations for 8-bit numbers implemented in this project.

Arithmetic Unit:

Addition Operation:

The addition block takes the two 8-bit numbers as input. The addition block consists of 8 full adder sub circuits. These sub circuits work as ripple carry adder. Thus, the addition operation done bit by bit continuously.

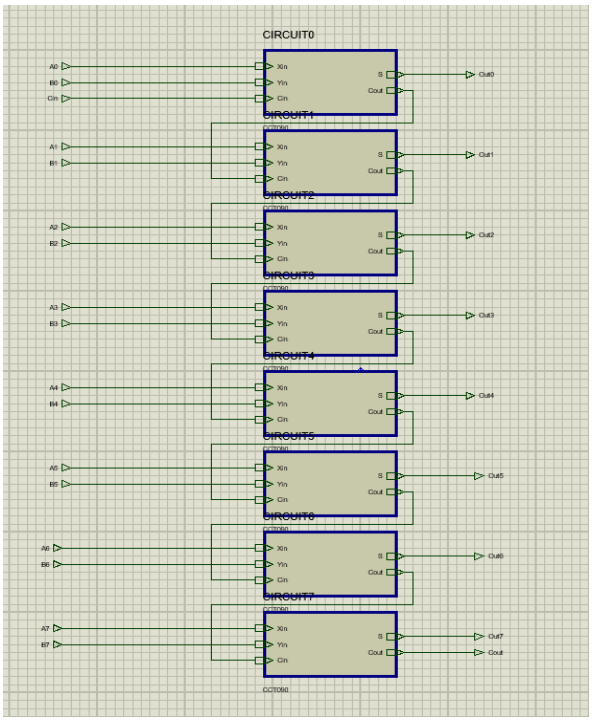


Figure: Addition Sub circuit

Each of the full adder block is consisted of some primitive gates, like and, xor, or etc. Each block takes 3 inputs, a bit from A, a bit from B and a carry input bit. It gives two outputs, a sum, and a carry. The sum bit is our output, and the carry is passed on to the next full adder block.

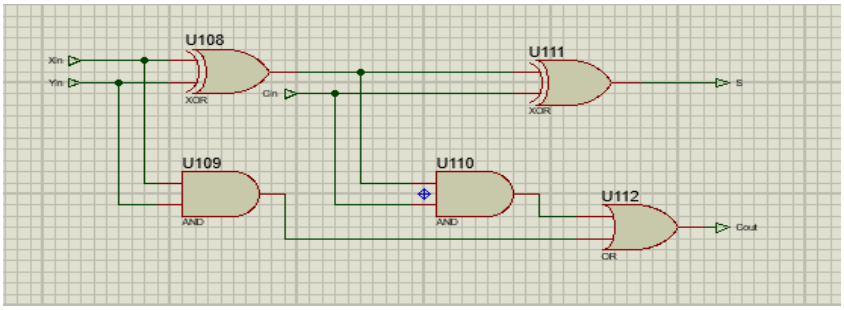


Figure: A single full adder sub circuit

Now, if we want to see how the addition performs, the inputs have to be given in the input portion of the main circuit. The inputs are given in binary. But their corresponding decimal values are also shown in the screen

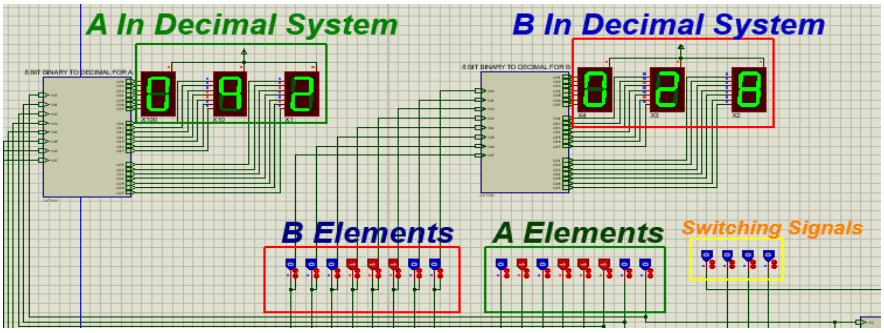


Figure: Input and Switching Sequence for A & B

If we want to add two numbers, let's say A=92 & B=28, their binary values are given as input. Our system will show the output 120.

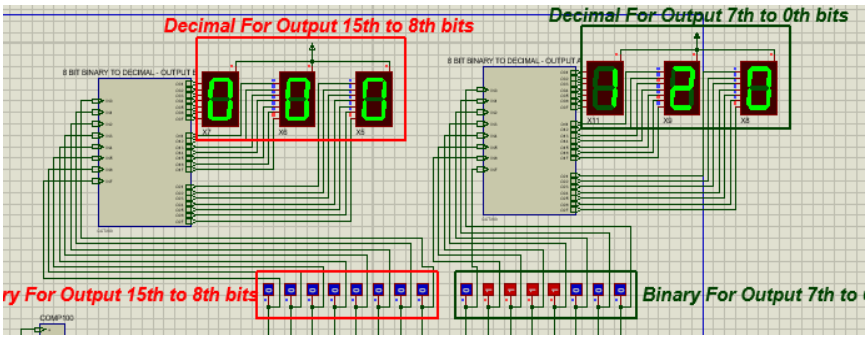


Figure: Output of addition operation

The outputs can be seen in both binary and decimal value.

Subtraction Operation:

Subtraction operation is done in almost similar way as addition. Subtraction is implemented by 2's complement method. Our system works in such a way that B is subtracted from A. And A has to be greater than B, as the display cannot show negative value. So for subtracting B from A, at first 1's complement is done on B. It is implemented by using not gates on the inputs of B. Then the carry input of LSB of B is given always 1. Thus 2's complement is done B.

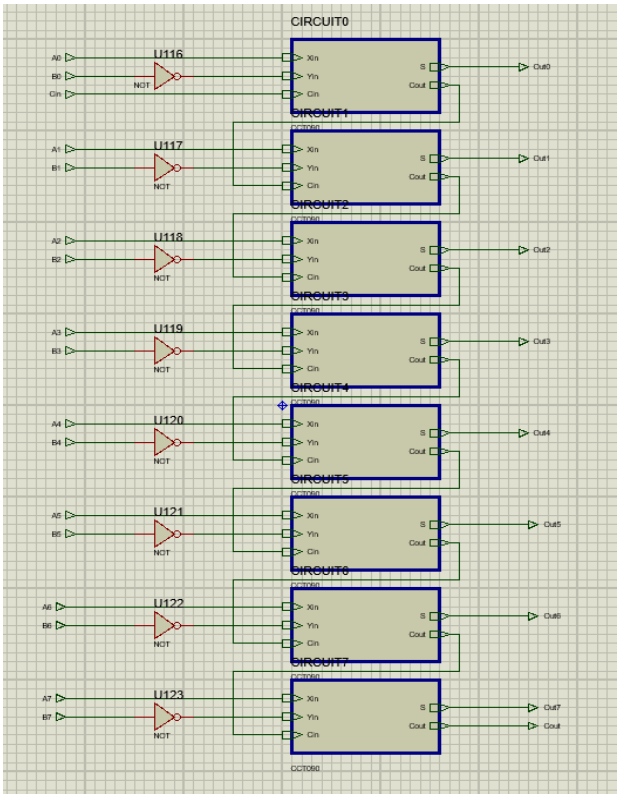


Figure: Subtraction sub circuit

Finally, the 2's complemented is added to A by 8-bit full adder circuit built in addition part. Each of the full bit adder blocks is same as before.

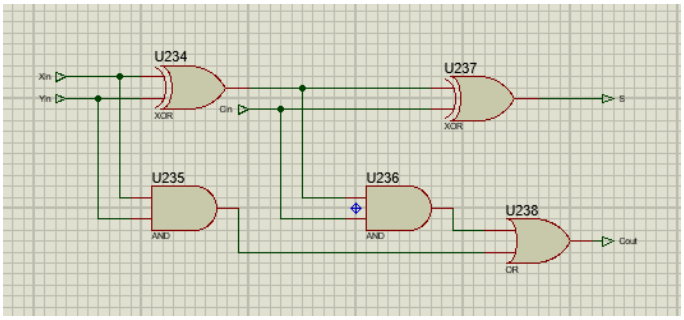


Figure: A Full Adder Block

For subtraction, input A & B are given as before. For doing this operation the switching sequence has to be given as 0001.

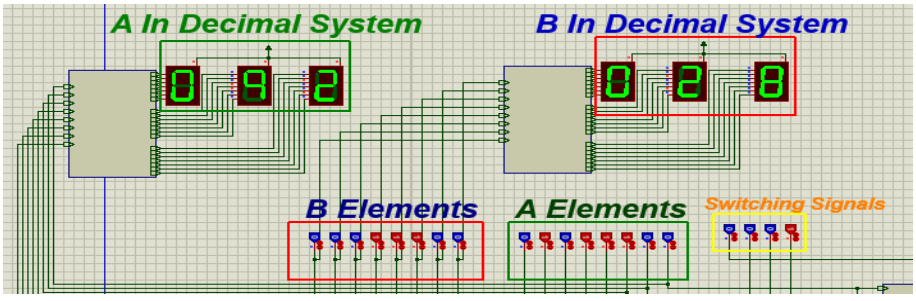


Figure: Input and Switching Sequence for Subtraction

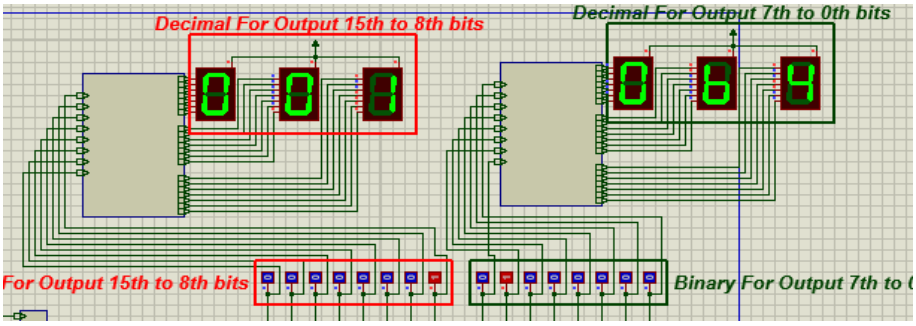


Figure: Output for Subtraction Operation

In our output board, the right most 3 digits are outputs, as the subtraction between two 3 digit decimal numbers will not be greater than 3 digits. The fourth digit will always show 1, which means the output is positive.

Increment Operation:

Increment operation increments the input by one, meaning it gives us the next decimal number of input. As increment operation can be done only at one number at a time, our system increments the value given in A.

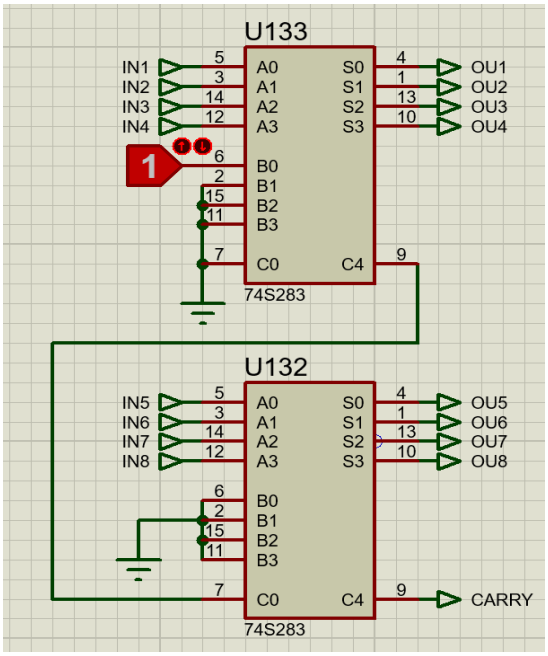


Figure: Increment Sub Circuit

Two 4-bit full adders are used in our system for incrementing. As increment means adding by one; 1 is added to the LSB of input. Then general ripple carry operation is done in rest of the bits.

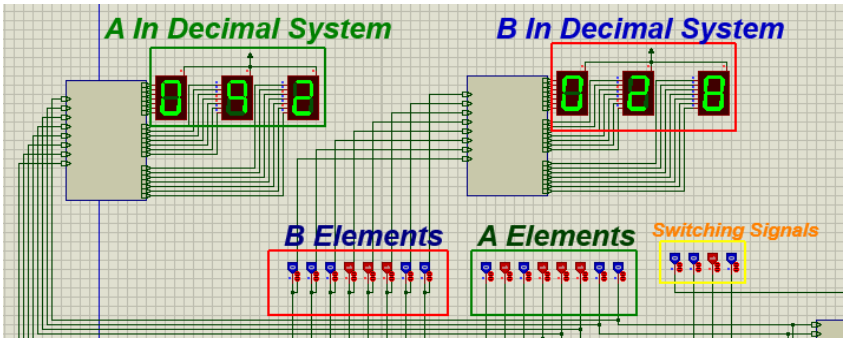


Figure: Input and Switching Sequence for Increment

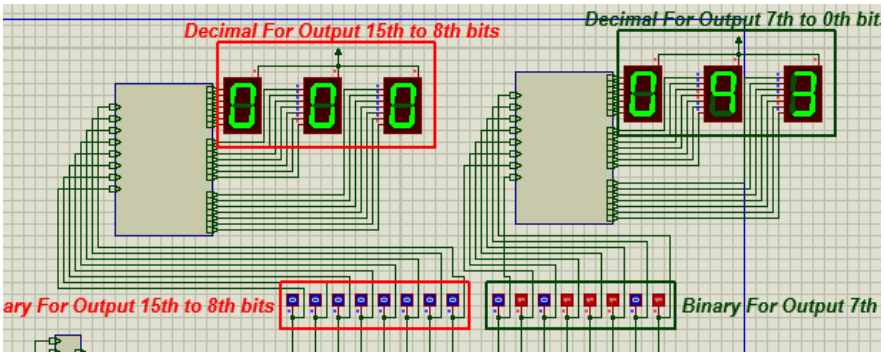


Figure: Output for Increment of A

As previous, the output is shown in both binary and decimal.

Decrement Operation:

Decrement operation decreases the given number by 1. Similarly, as increment, decrement is also done at a number. Decrement operation works in a similar way as subtraction, as decrement is also a kind of subtraction.

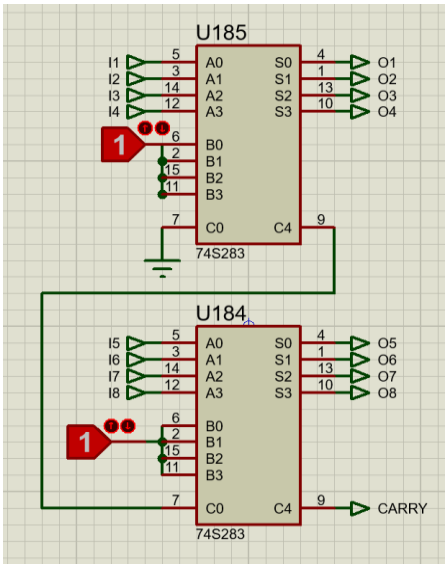


Figure: Decrement Sub Circuit

Decrement operation is also done as 2's complement. The decimal output presentation also contains the constrains faced in subtraction. The right 3 digits show the actual result. The left 3 digit must be ignored. Also, the fourth bit is also 1 which indicates positive number.

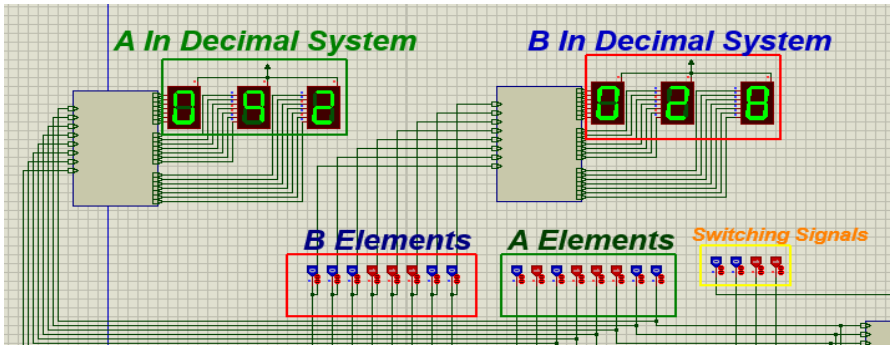


Figure: Input A and Switching Sequence for Decrement

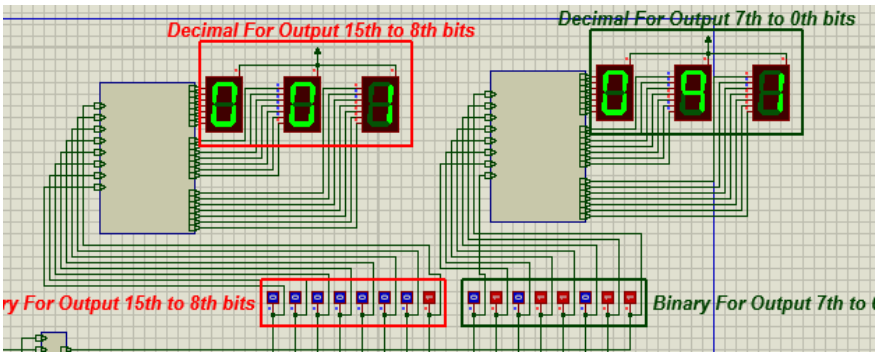


Figure: Output for Decrement

Multiplication: The addition block takes two 8-bit numbers as input and gives their multiplication as output.

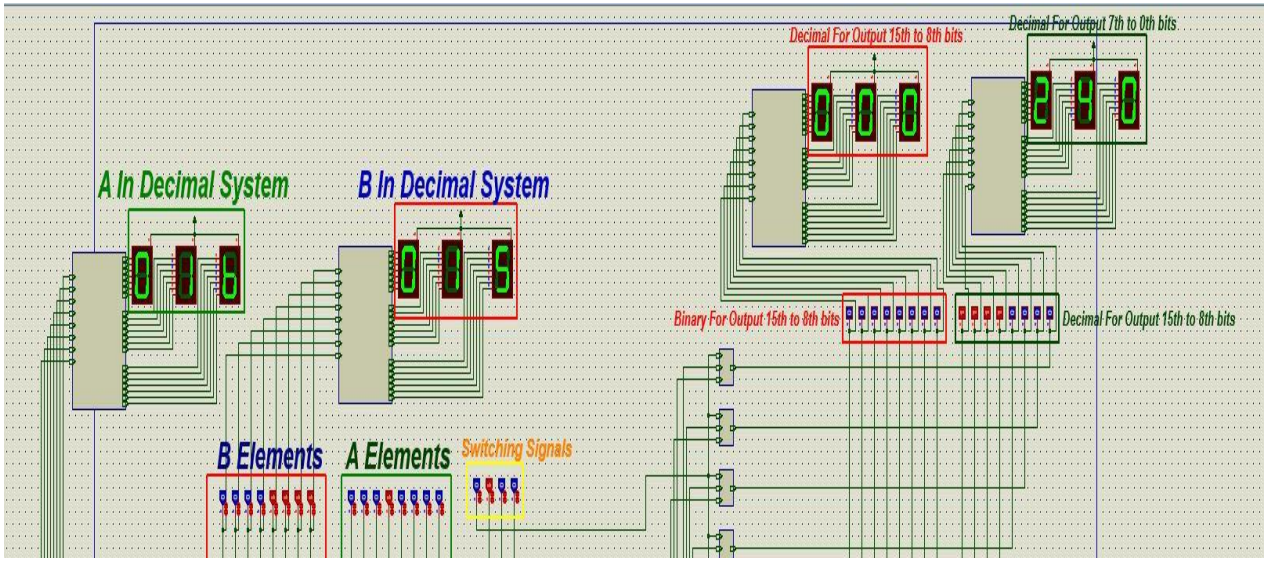


Figure: 8-bit multiplication in Proteus

The multiplication operation is done bit by bit. The algorithm of multiplication of two 4 bit numbers is shown hereby-

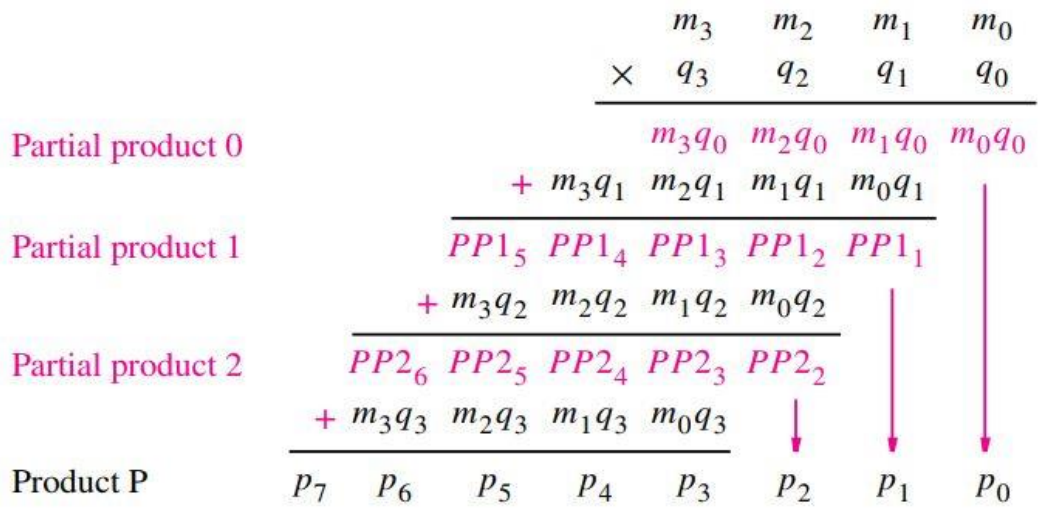


Figure: Step by step process of 4-bit multiplication

The same logic has been implemented in our circuit for 8-bit multiplication. For this manner, we used 7 full adder circuits and AND gates for bit-by-bit multiplication.

The multiplication block circuit is shown hereby-

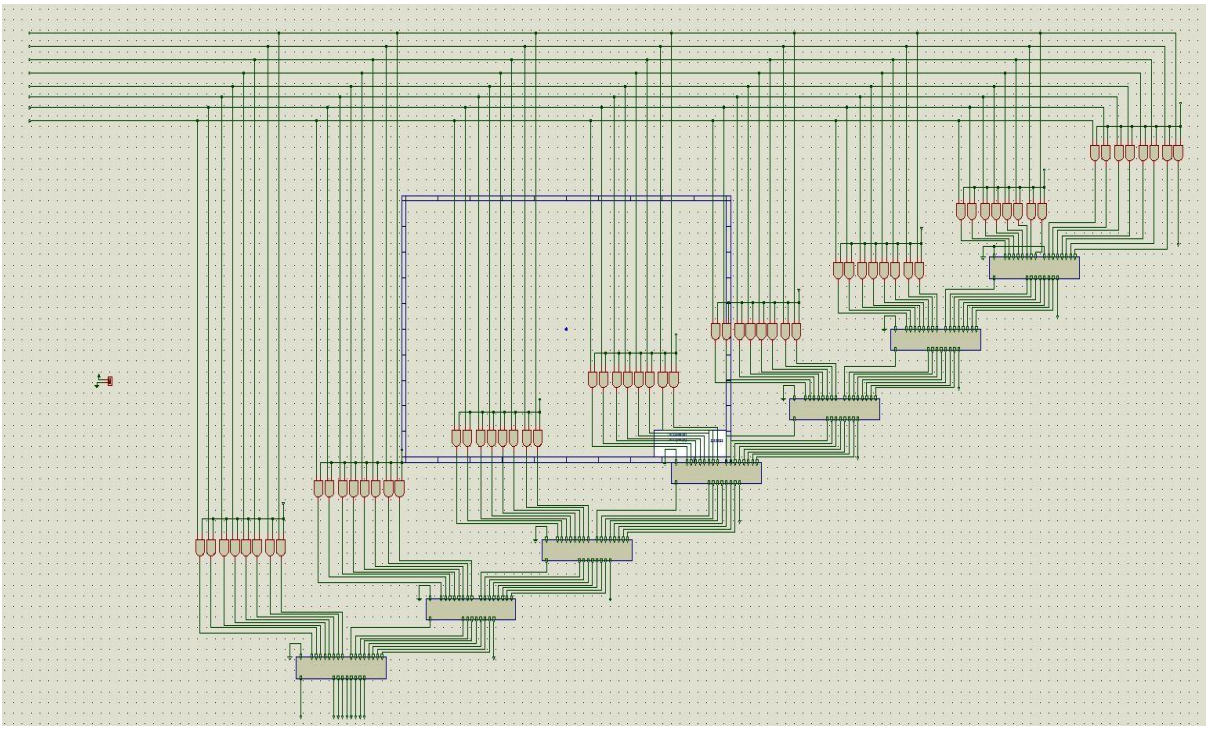


Figure: Multiplication block for 8-bit numbers in Proteus

The carryout of the last adder stage is the MSB of the output. By multiplying bit by bit and adding like the algorithm explained above, using adders we get our desired multiplication output.

Division:

Division is one of the fundamental arithmetic operations. It is the process of calculating how many times of a number another number contains. It can also be described as a repeated subtraction. If we divide a number by another number, we can know the maximum possible times a number contains another number; however, some quantities cannot be divided evenly, and the uneven remains are remainders.

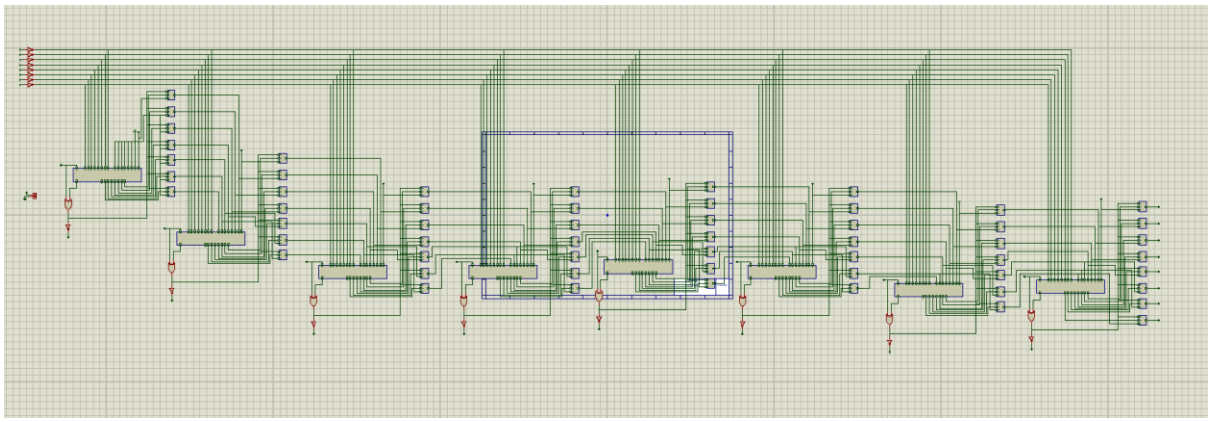


Figure: Overview of the Circuit used for Division

Now we will see the separate parts of the division circuit and how the circuit is implemented.

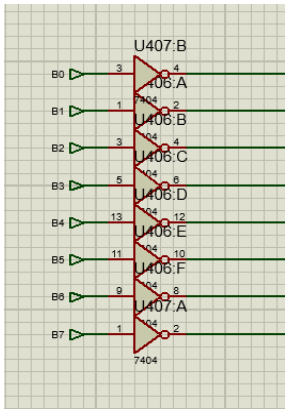


Figure: Divisor

Divisor: The divisor is the number whose multiplicity compared to the dividend s checked. The divisor is used as the sole subtrahend throughout the whole process. The 8 bits of the divisor are taken separately as shown, and they are turned into 1's complement at first. Later, a carry-bit of 1 is used to make it 2's complement.

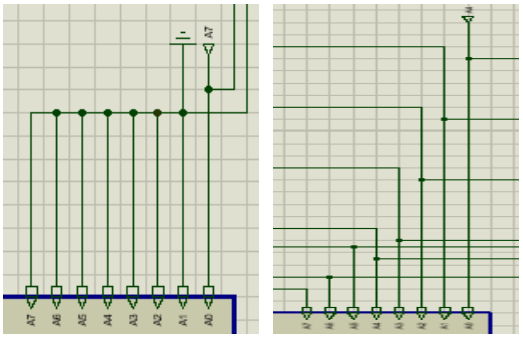


Figure: Dividend (A_7 and A_4 are shown)

Dividend: The dividend is the number which is divided and may or may not be a multiple of the divisor. In this process, the dividend is used to make minuends; throughout the whole process the dividend bits are used separately. Since this is an 8-bit operation, 8 subtractors are used, which needs all 8 dividend bits for making the minuends.

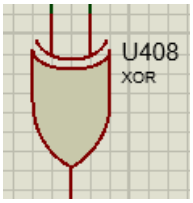


Figure: XOR Gate

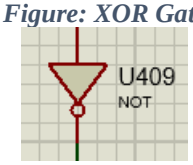


Figure: NOT Gate

XOR Gate: An Exclusive-Or gate, or an XOR gate, is a gate which gives an output of 0 if the inputs are all similar (for a 2-input XOR) or 1 is the input for an even number of times (for XOR having more than 2 inputs). In this circuit, XOR gates with 2 inputs are used to get switching values.

NOT Gate: A NOT gate inverts the input. The gate is used to find the quotient bits in this process.

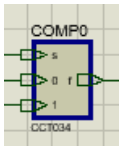


Figure: A 2×1 MUX

MUX: A multiplexer, or a MUX, is a circuit which decides which input to pass as an output depending on the switching logic. The simplest MUX is a 2×1 MUX which has a switching state of 0 or 1.

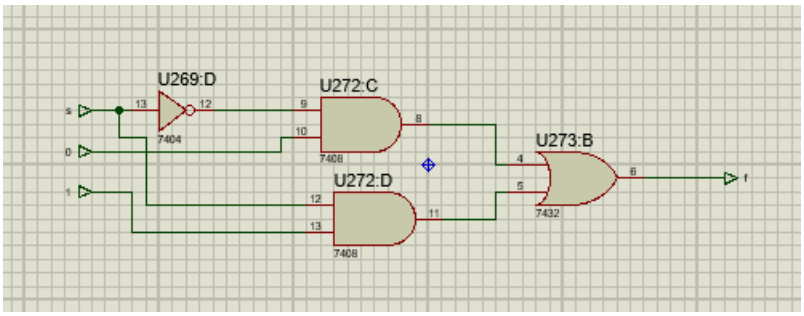


Figure: Inside the 2×1 MUX

The switch input, s, controls whether the input for 0 or 1 will be the output If s = 0, the input for 0 will be the output, otherwise the input for 1 will be the output.

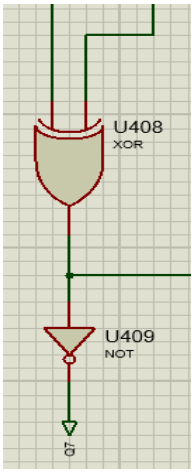


Figure: Quotient (Q_7)

Quotient: When a division is done, the main output is the quotient. It is the required number of times of the divisor the dividend contains. It may either be the exact number of times of divisor a dividend contains, or the nearest possible times.

In this process, the quotient bits are collected at the end of each subtraction. The XOR and NOT gates are used to get the quotient bits. Since this is an 8-bit division, the quotient is determined 8 times, for 8 bits. The quotients are in the order from Q_7 to Q_0.

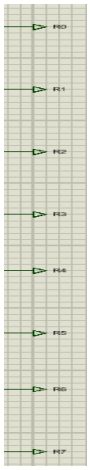


Figure: Remainder

Remainder: In some cases, the dividend may not be a perfect multiple of the divisor. In such cases, the part which cannot be eliminated by the divisor remains as a separate entity and is thus called a remainder. A remainder is always less than the divisor.

In this process, the remainders are obtained at the end of the process. Depending on the switching operations, either the minuend used in the last process, or the difference obtained in the last process will be the remainder.

Performing the Division Operation:

The operation of division done in this project is explained as follows:

1. Subtraction: The divisor always stays as the subtrahend, while the minuend is constructed by using the dividend bits. The dividend bits are taken from the MSB, successively to the LSB. At first, for the MSB A_7, the minuend is constructed by taking A_7 to the LSB and adding 0 to the other positions. For the remaining dividend bits, the minuend is constructed by comparing the difference and the previous minuend.

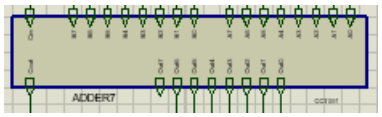


Figure: The 8-bit Adder/Subtractor Circuit. In the process of division, the circuit performs subtraction.

The portion A in the circuit indicates the minuend, while the portion B indicates the subtrahend. There is input carry and output carry. The difference is obtained from the Out portion. Now, we can have a look inside this circuit:

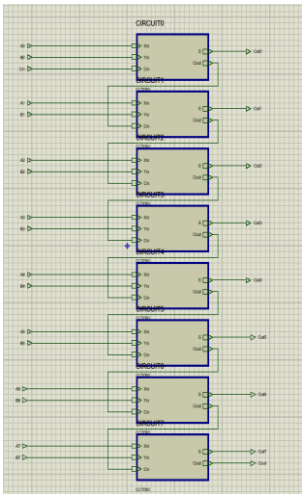


Figure: The Full Adder Operation

There are 8 full-adder operations. The bits of the minuend and the subtrahend are used one by one, along with the carry bits. The result bits are taken as outputs, while the carry bits are used until the final carry, or the output carry, comes.

Inside the circuit, the full-adder operation is maintained. Although the circuit of adder is used, it actually does subtraction, since the subtrahend is converted into 2's complement beforehand.

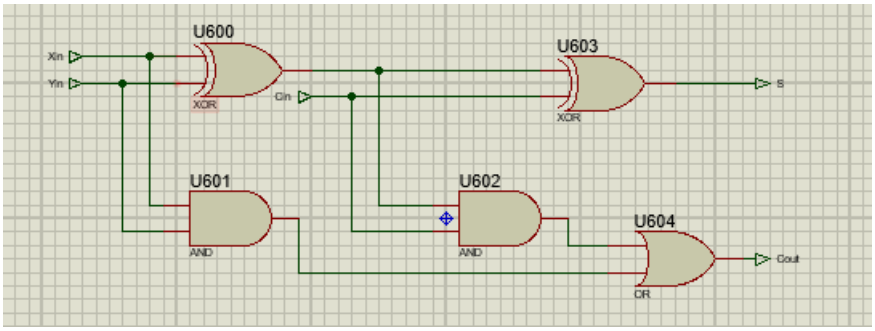


Figure: Inside a Full-Adder Circuit

This process of subtraction remains unchanged throughout the whole process. It makes sense since the division is a faster form of subtraction. When the outputs from the subtraction process are obtained, the next steps are performed.

2. Finding the Quotient Bit and the Switching Value: After the subtraction is done, the output carry-bit is sent to an XOR circuit, whose another input is the input carry-bit (which has a value of 1). The output of this XOR gate is used as a switch to work on the next minuend. This output is inverted to get the quotient bit. The quotient bit follows the order of the dividend bit, so for A_7 we get the quotient bit of Q_7 and for A_4 we get the quotient of Q_4.

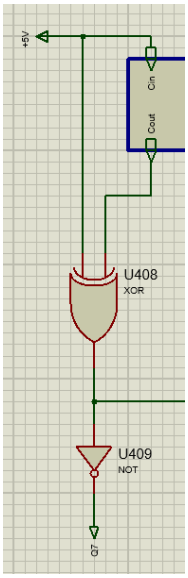


Figure: The XOR and the NOT gates, used to find out the quotient and the switching value of the selector

3. Making the Next Minuend:

The difference and the minuend are compared using a series of 2×1 MUX's. Except for the last case, the MSB's for both the minuend and the difference are ignored. Depending on the switching value, the next minuend bits are prepared. If the switching value is 0, the difference bits are used; otherwise, the previous minuend bits are used. Since the next minuend already has the LSB used up from the dividend bit, the remaining places are filled up by the new bits.

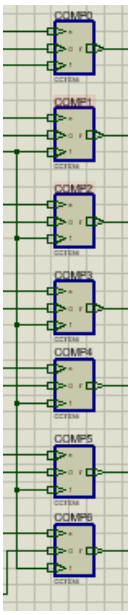


Figure: The MUX circuits used to make the next minuend by comparing the previous minuend and difference, based on the switching value obtained from the XOR gate.

4. Finding the Remainder Bits:

In the eighth step which is also the last step, the 2×1 MUXs are used, now to decide the remainder bits. Similarly, to the previous switching operations, if the switching value is 0, the

difference bits become the remainder bits, otherwise the minuend bits become the remainder bits. Differently from the previous steps, all eight bits are checked since the continuous process of subtraction is completed here.

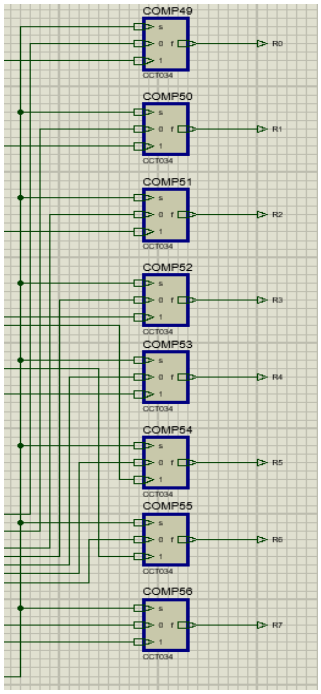


Figure: Finding the Remainder using the MUX circuits; the MUX circuits operate like before.

Implementation of Division:

Now some examples of division in this circuit will be shown. In the ALU, the division is performed by assigning 0101 to the Switching Signals.

Case 1: The dividend is fully divisible by the divisor. So, a quotient is obtained and there will be no remainder.

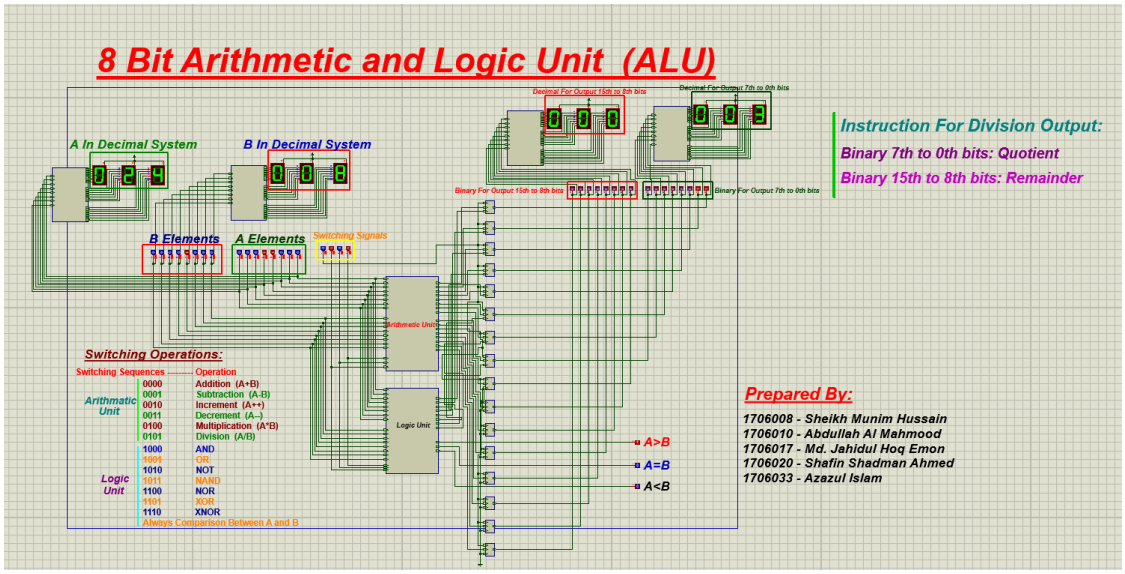


Figure: Dividing 24 by 8; the quotient is 3 and the remainder is 0.

We can see that A = 24 is being divided by B = 8. Since 24 is a multiple of 8 and 3 times of 8 is 24, the quotient will be 8 and the remainder will be 0 as expected.

Case 2: The dividend is not fully divisible by the divisor and the divisor is less than the dividend. So, both a quotient and a remainder are obtained.

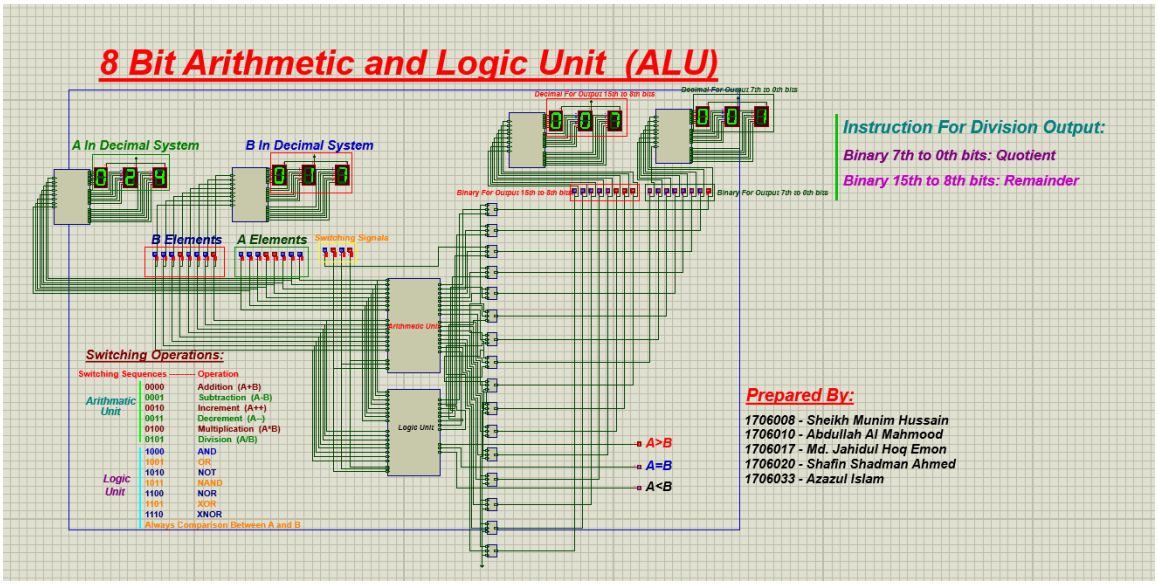


Figure: Dividing 24 by 17; the quotient is 1 and the remainder is 7.

We can see that A = 24 is being divided by B = 17. Since 24 is not a multiple of 17, the quotient will be 1 and the remainder will be 7 as expected. It is also seen that the remainder is less than the divisor.

Case 3: The dividend is not fully divisible by the divisor and the divisor is larger than the dividend. So, there will be no quotient and a remainder is obtained.

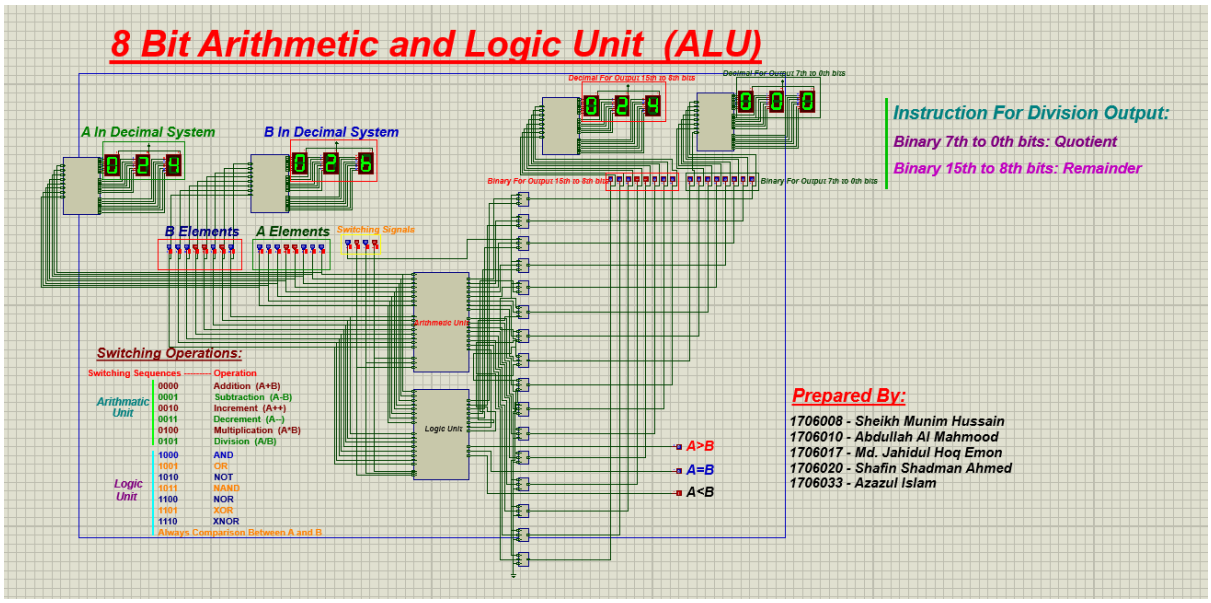


Figure: Dividing 24 by 26; the quotient is 0 and the remainder is 24.

We can see that A = 24 is being divided by B = 26. Since 24 is not a multiple of 26 and 26 is a larger number, the quotient will be 0 and the remainder will be 24 as expected. It is also seen that the remainder is less than the divisor and equal to the dividend itself.

Logic Unit:

The next integrated part of the project was the logic unit. In total it performs eight (8) different operations. In sequence they are:

i) Bitwise AND

ii) Bitwise OR

iii) Bitwise NOT

iv) Bitwise NAND

v) Bitwise NOR

vi) Bitwise XOR

vii) Bitwise XNOR

viii) Comparator

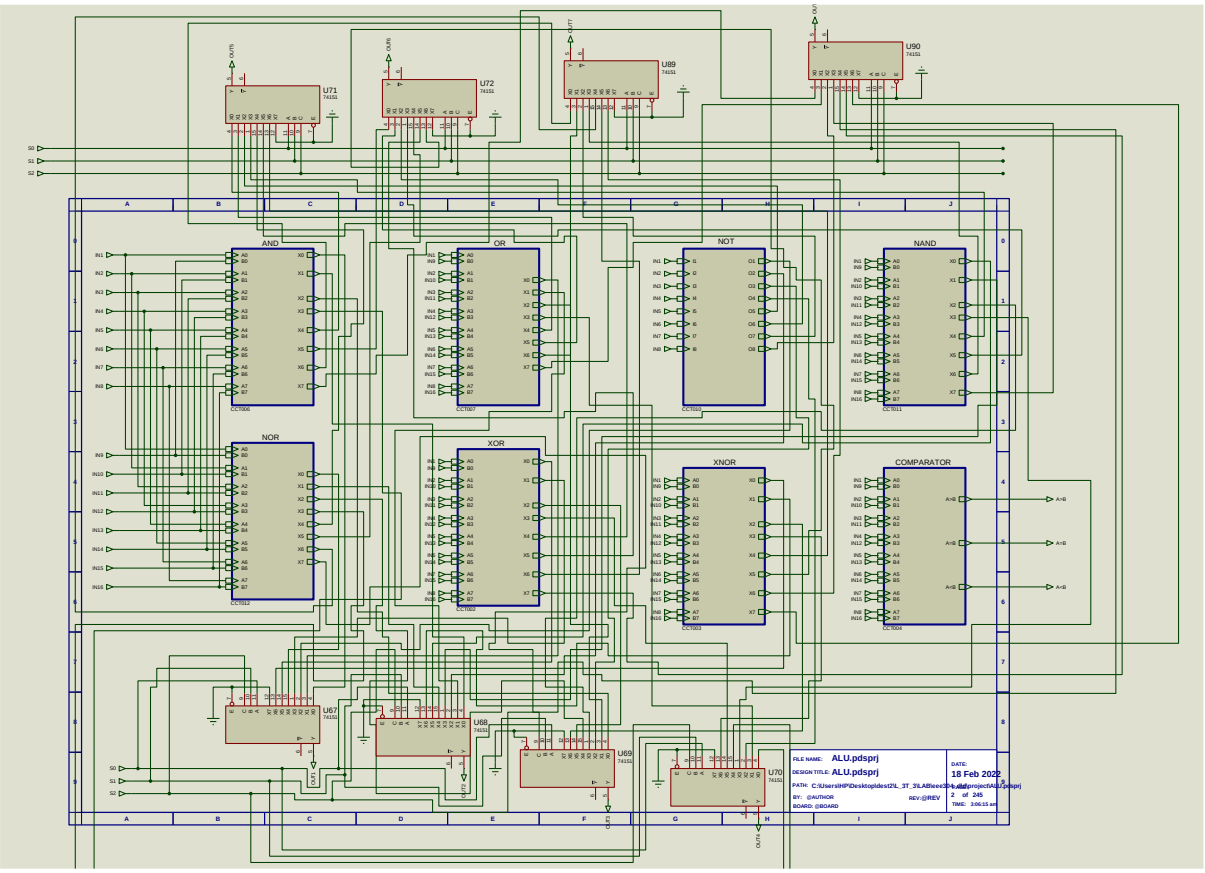


Figure: Logic Unit

Each of them takes two inputs and perform bitwise operations between them except comparator. Then which output will be selected depends on the switch operation.

The outputs relate to eight different 8 to 1 MUX so that the switch selection will activate corresponding operation.

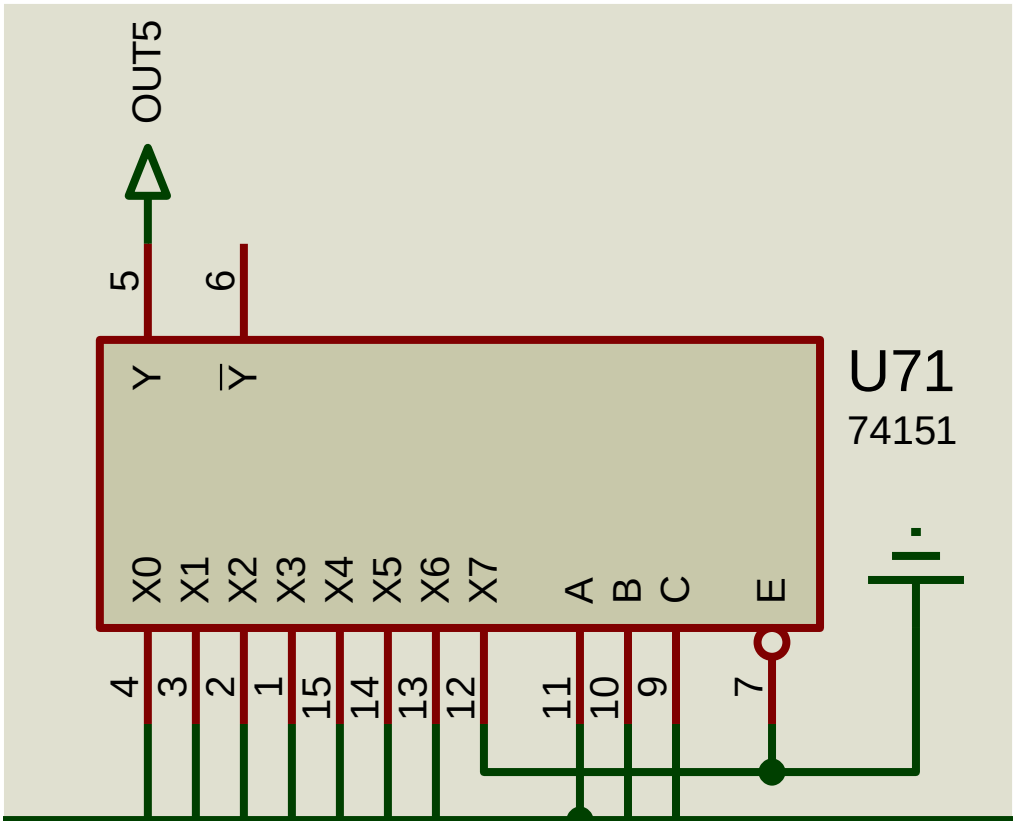


Figure: 8 to 1 Multiplexer

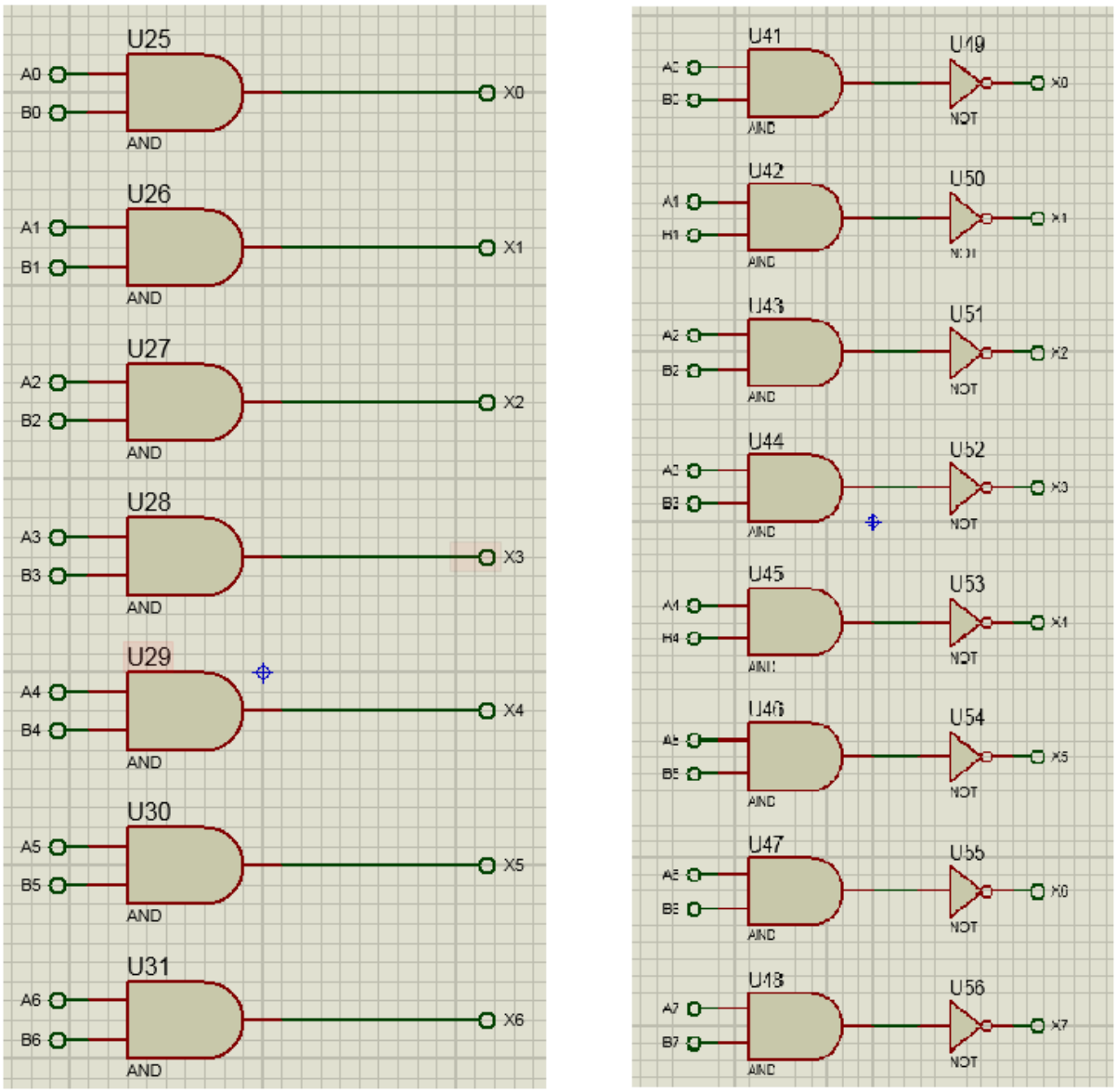
Table: Switch Connection Sequence

Switch Sequence	Operation
1000	AND
1001	OR
1010	NOT
1011	NAND
1100	NOR
1101	XOR
1110	XNOR

Now description of these operations will be analyzed.

Bitwise AND and Bitwise NAND:

Their basic operation is to compare between each bits of input A and B and produce their bitwise AND and NAND values.



AND **NAND**

Figure: Bitwise AND and NAND operation

Table: A and B possible 4 operations

A	B	AND	NAND
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

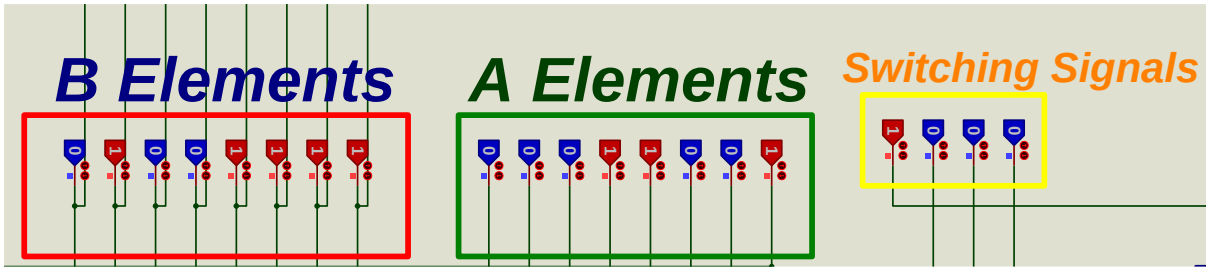


Figure: AND Input

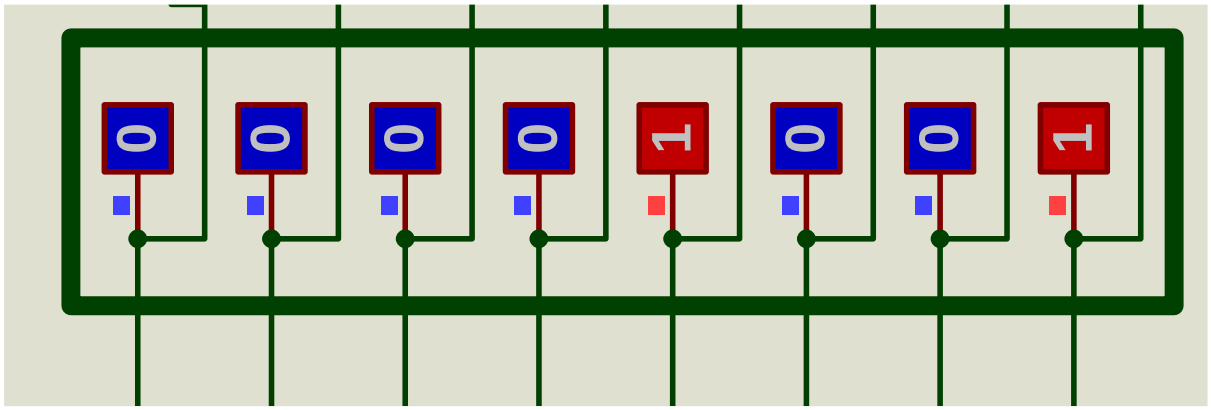


Figure: AND Output

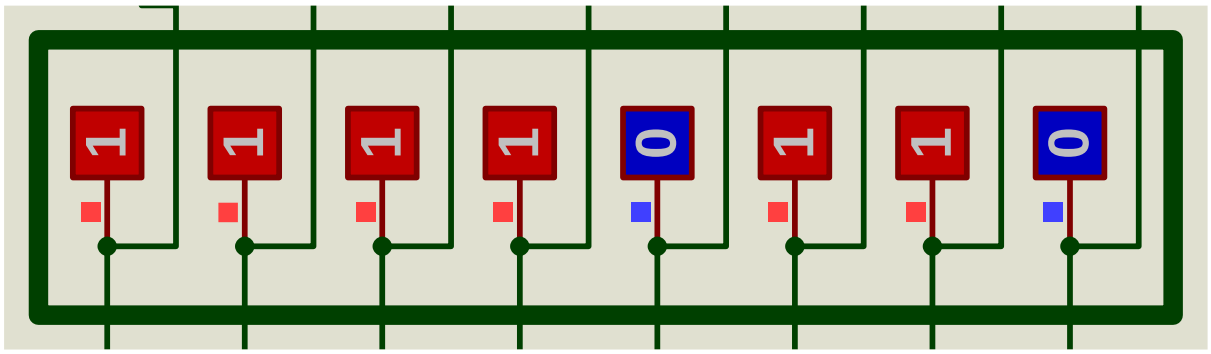
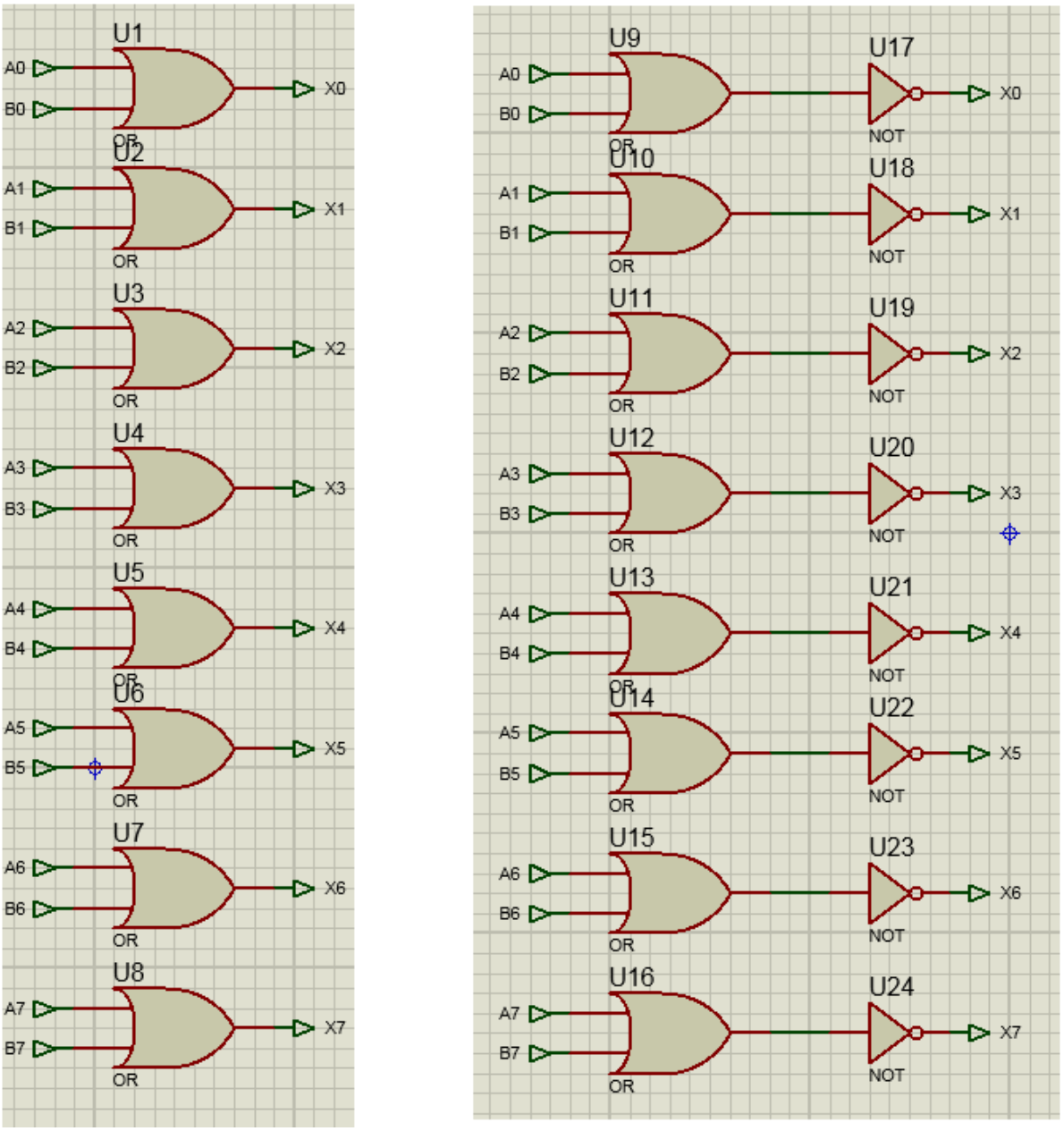


Figure: NAND Output

So, NAND operation generates logic invert of AND operation.

Bitwise OR and NOR Operation:

Similarly for each bit of two input, their OR and NOR operation will be executed.



OR **NOR**

Figure: Bitwise OR and NOR operation

Table: A and B possible 4 operations

A	B	OR	NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

For the same A and B from previous section,

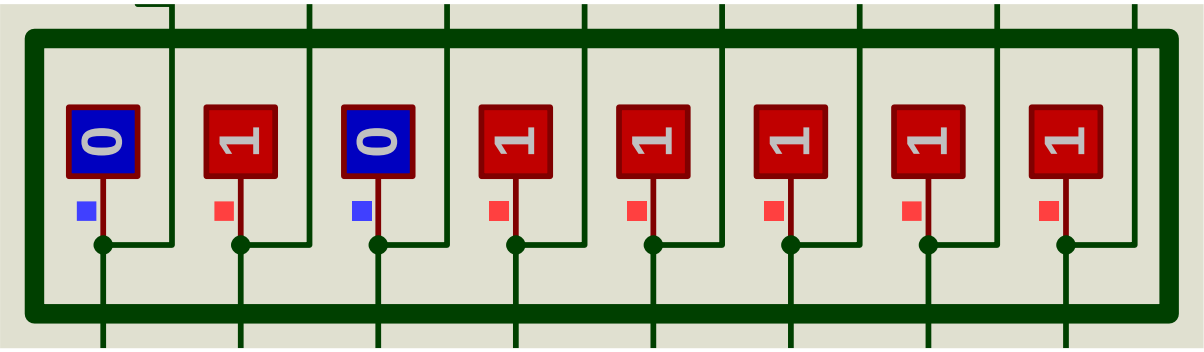


Figure: OR Operation Output

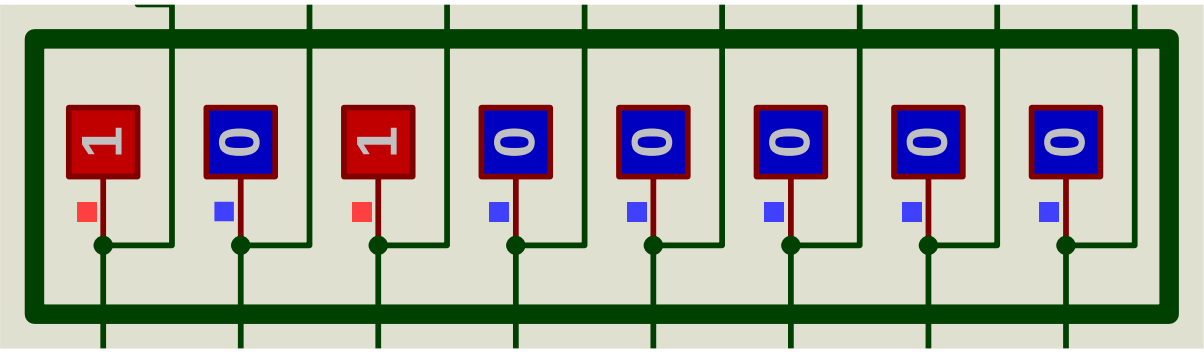
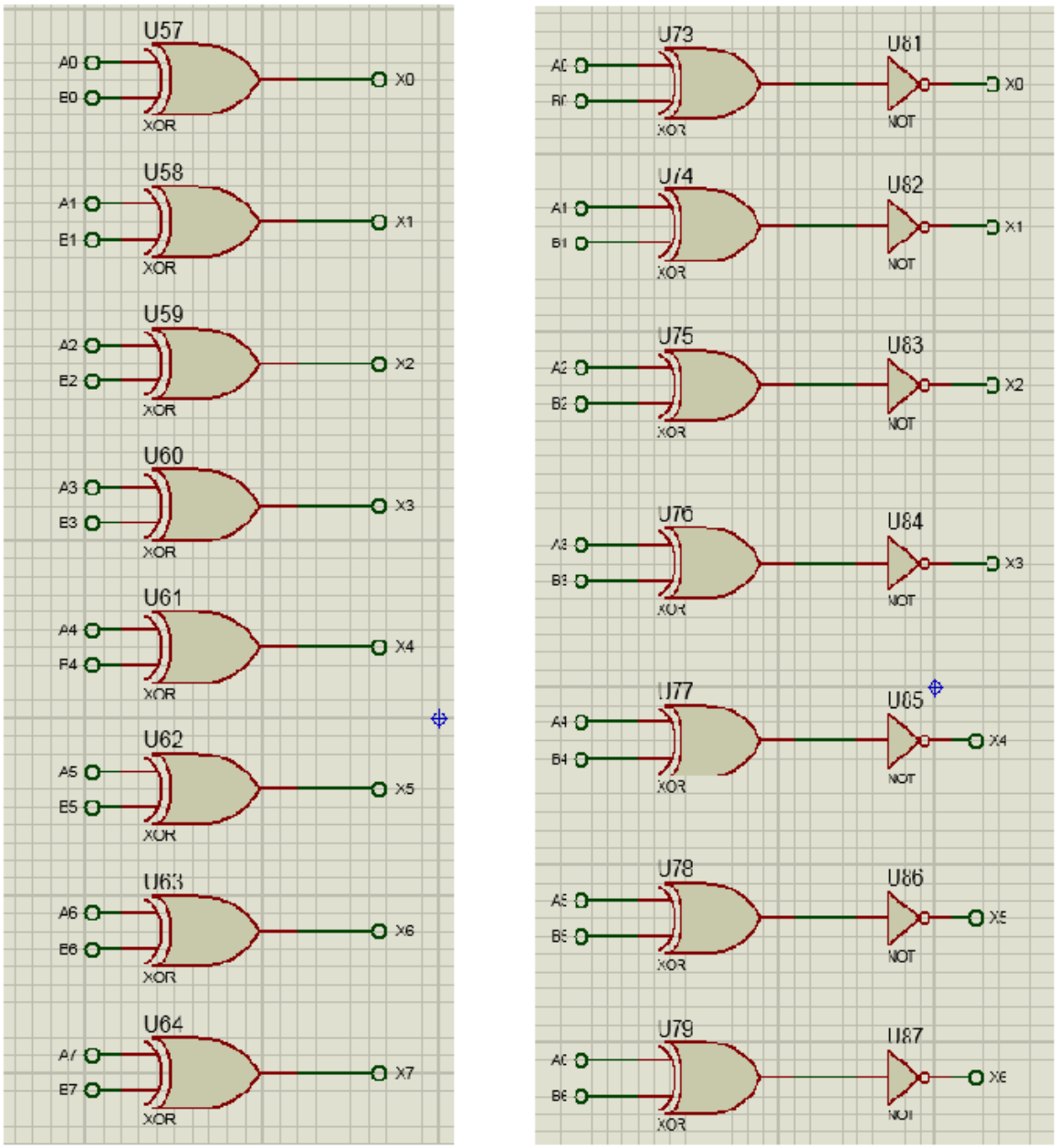


Figure: NOR Operation Output

Again, logic invert output of OR operation is executed by NOR operation.

Bitwise XOR and XNOR:

In case of XOR, for two similar bits from the input, it generates logical false and two different bits, output will be zero. For XNOR, opposite scenario will appear.



XOR *XNOR*

FIGURE: Bitwise XOR and NXOR operation

Table: A and B possible 4 operations

A	B	XOR	XNOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

For the same A and B,

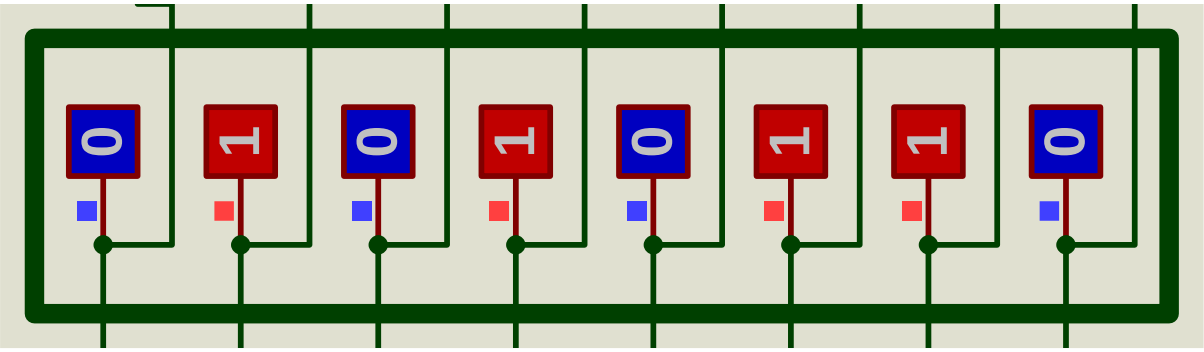


Figure: XOR Operation Output

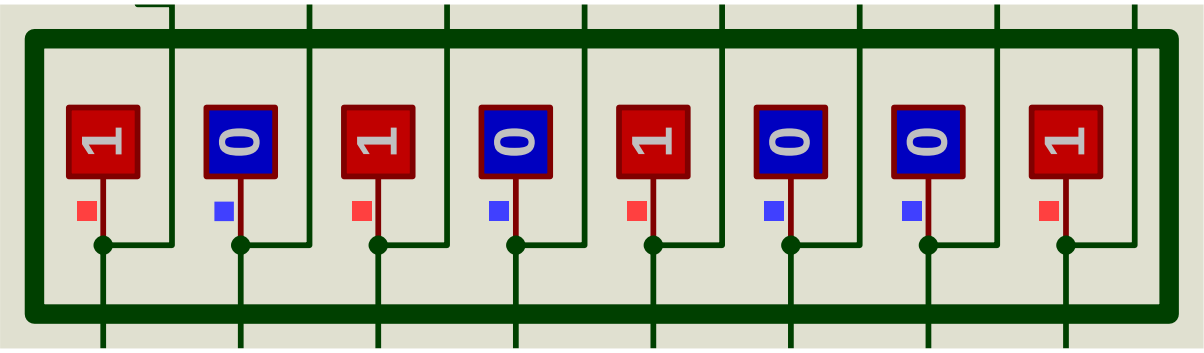


Figure: XNOR Operation Output

As it was said, logical invert operation was done by these two gates.

Bitwise NOT:

Unlike previous ones, NOT gate will take a single input and yields logical invert of each bit.

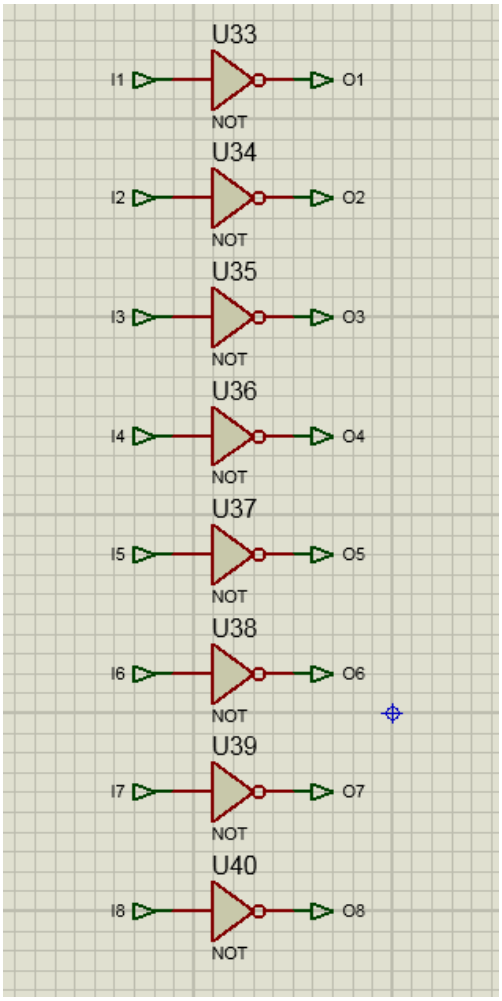


Figure: Bitwise NOT Operation

Table: A possible 2 operations

A	NOT
0	1
1	0

For same A,

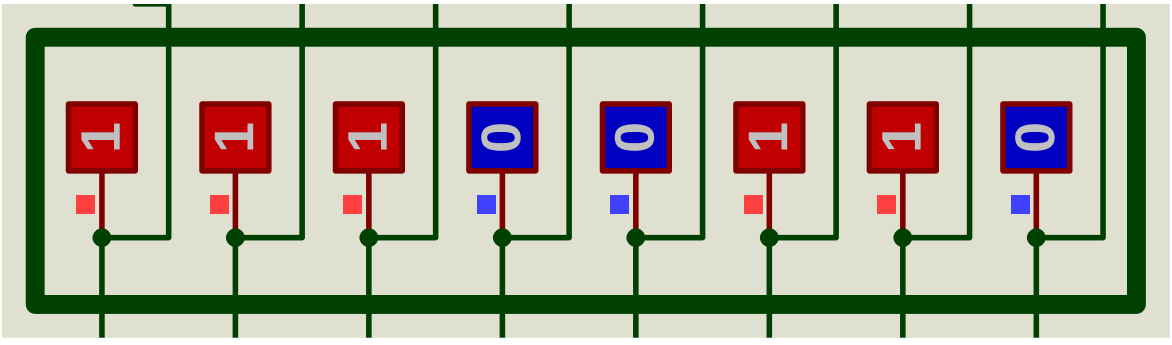


Figure: NOT Operation Output

COMPARATOR:

Instead of pursuing a bitwise operation, a comparator simply compares two input A and B and indicates whether they are equal, or which one is larger.

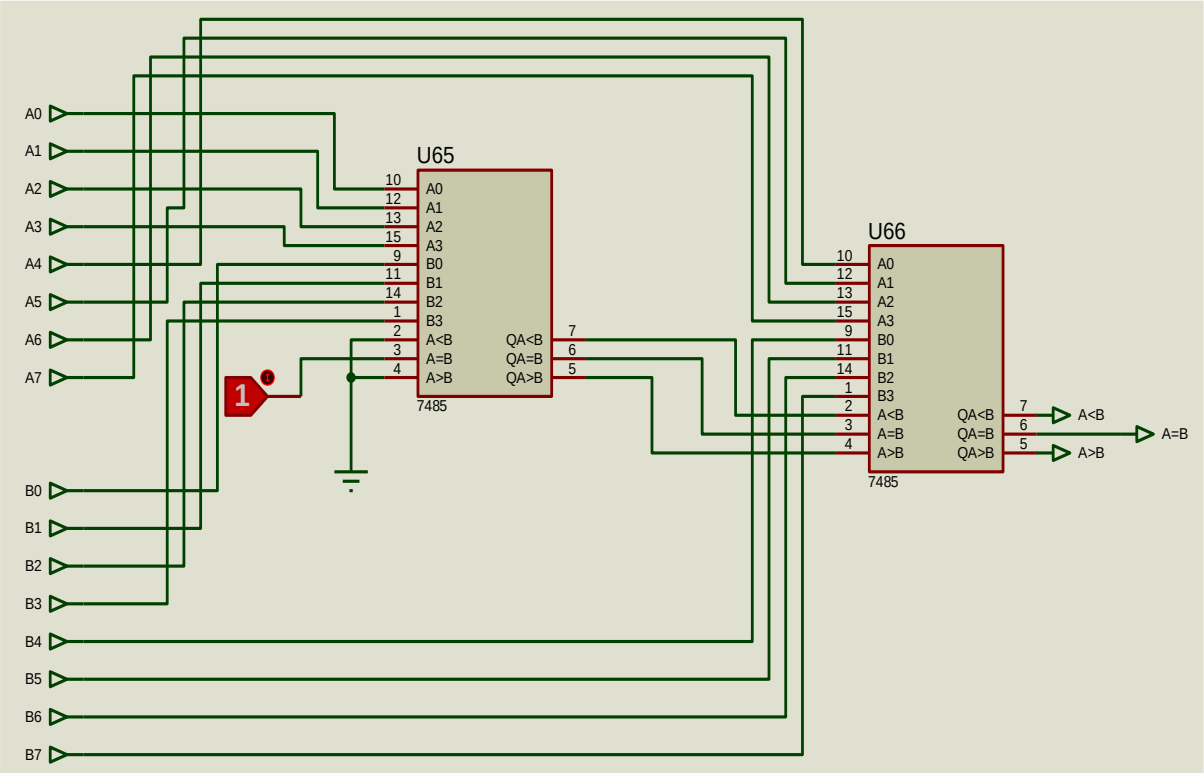
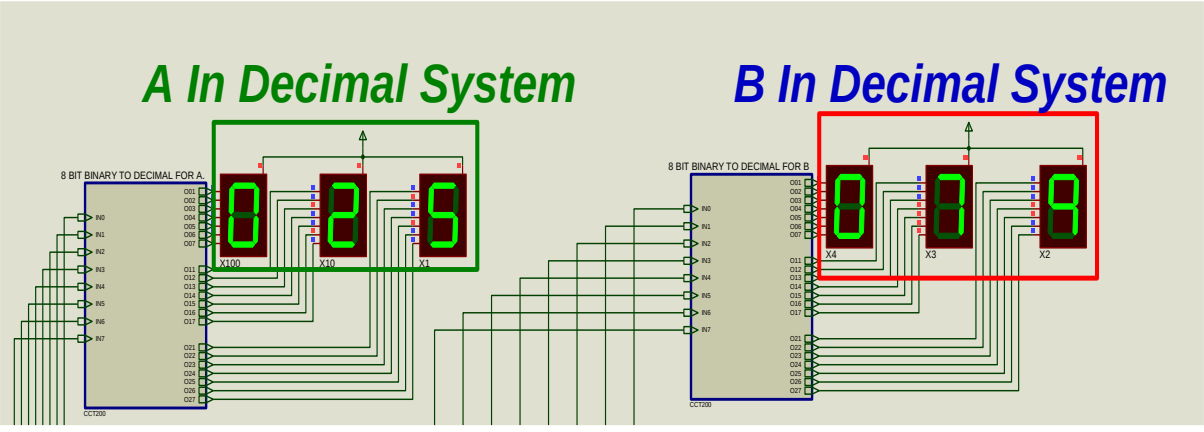


Figure: Comparator

Here we use IC7485 to implement the comparator. As the IC can take at most 4 bits at once, for our 8-bit operation, two IC 7485 was needed. Fist one is comparing between 4 LSB of the two inputs. Its output is passed to the next IC and then it compares between the 4 MSB bits. Final output will indicate the comparison. In our demonstration, comparator wasn't made exclusive by any switch sequence. Instead, the comparison will be executed all the time.

Now if we take A = 25 and B = 79, output should indicate A is smaller than B.



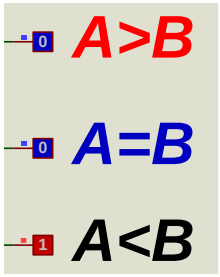
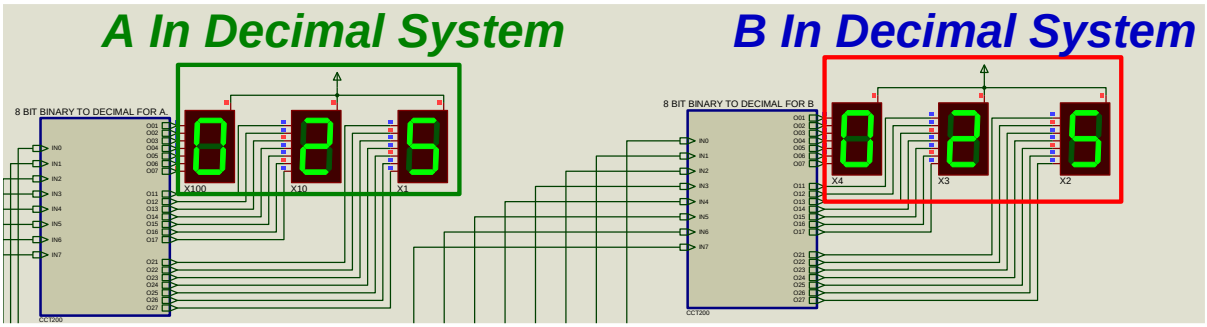


Figure: Input and Output for A<B

Now for a different case when A = B = 25, output should display A is equal to B.



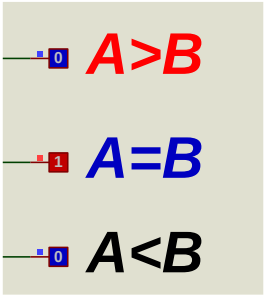
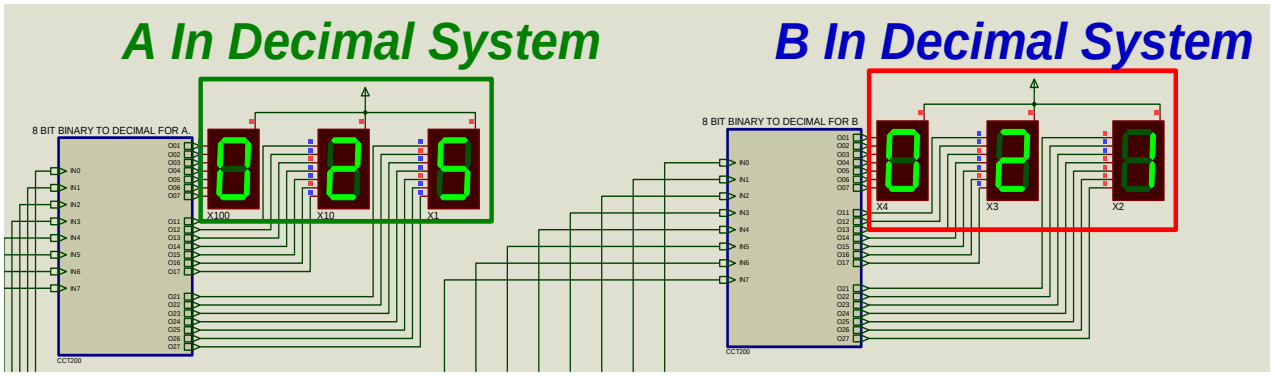


Figure: Input and Output for A=B

Finally for A = 25 and B = 21, output should display A is greater than B.



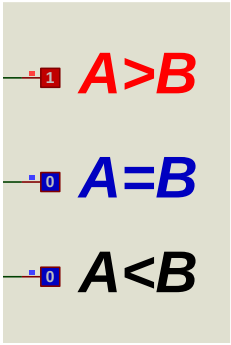


Figure: Input and Output for A=B

Thus, by selecting proper switching sequence logic unit for two 8-bit number can be performed.

Final Layout:

View of different parts:

At this point we will investigate the final layout in detail. Here we used two subcircuits for entire 'Arithmetic' and 'Logic' unit which were huge.

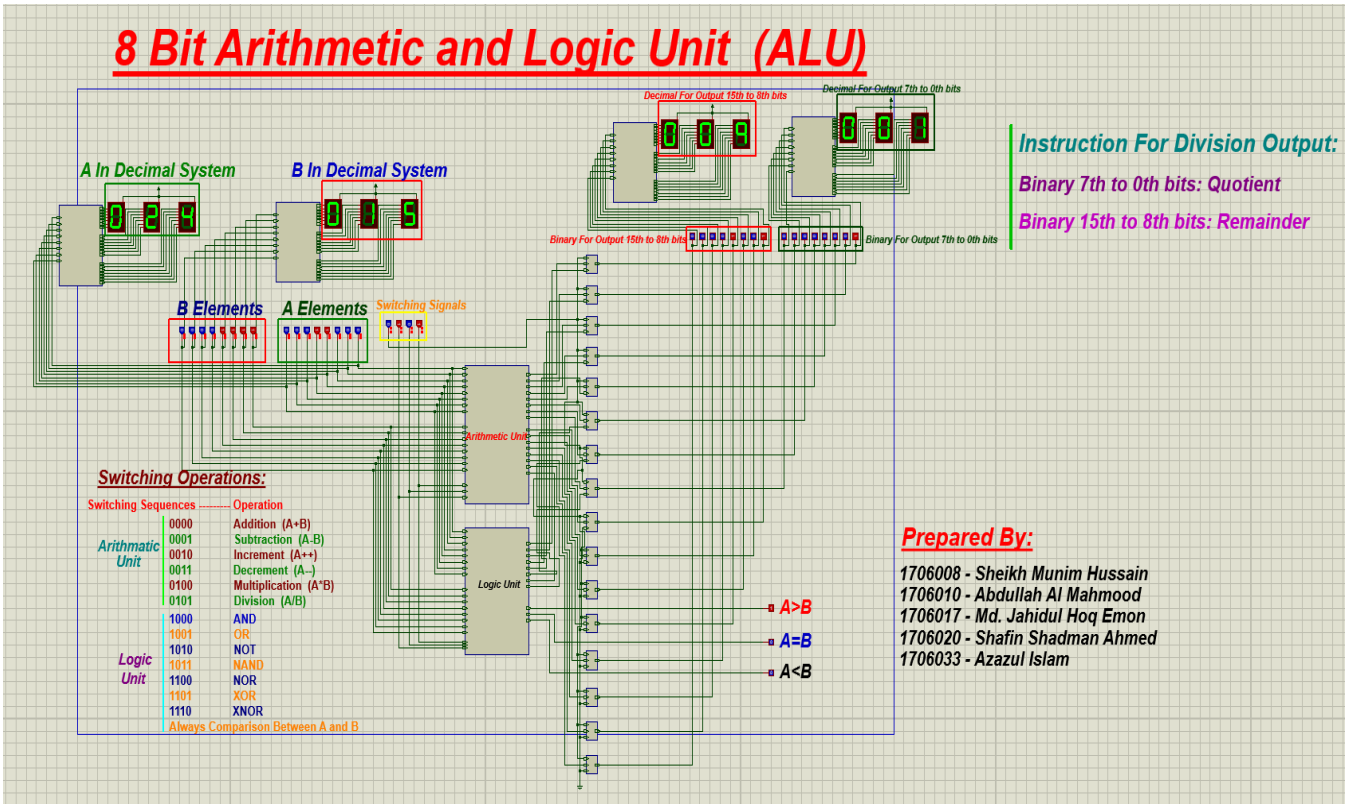


Figure: User Interface when we first enter the program (Division Operation is shown)

In arithmetic unit, we added all the components described above namely: addition, subtraction, multiplication, division, increment, and decrement.

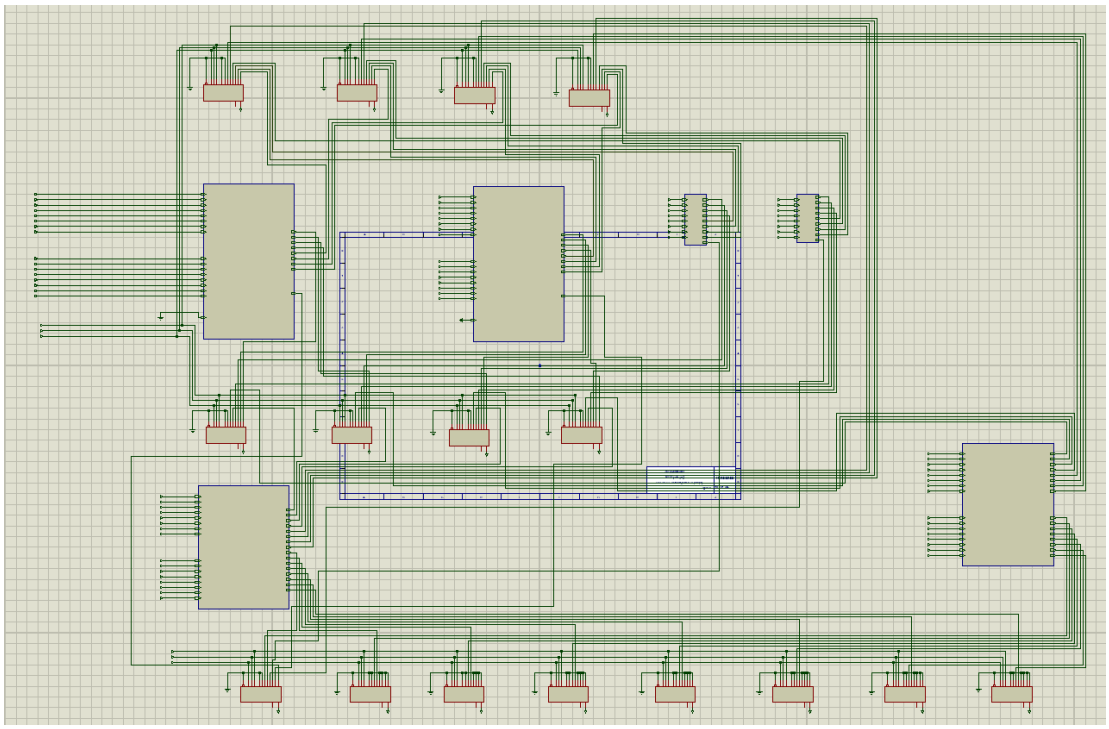


Figure: Inner View of Arithmetic Unit

Like the arithmetic one, we included all the operations of logic unit that we discussed before.

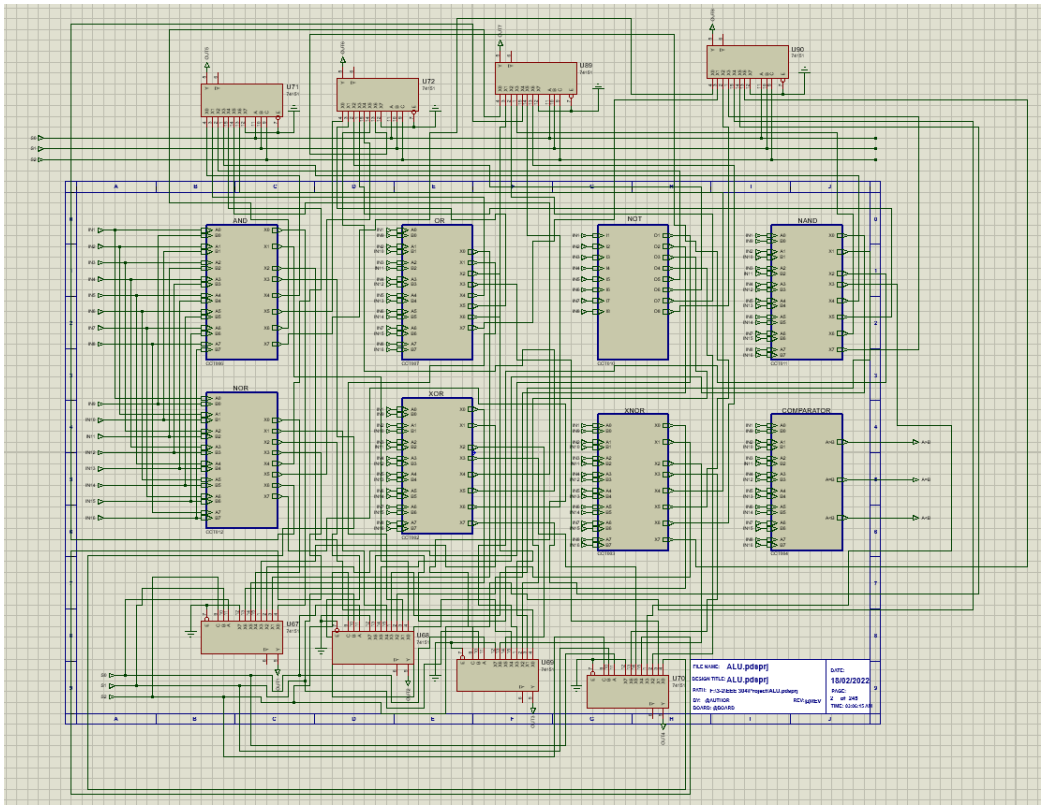


Figure: Inner View of Logic Unit

Switching Sequences:

For switching action, we used 2 step switching sequence. At first step, we used last 3 switching bits namely S0, S1, and S2. This switching sequence goes to both arithmetic and logic unit. The last switch bit or S3 is used to select which output we will show in the output probes and displays. This bit is used externally in the main layout.

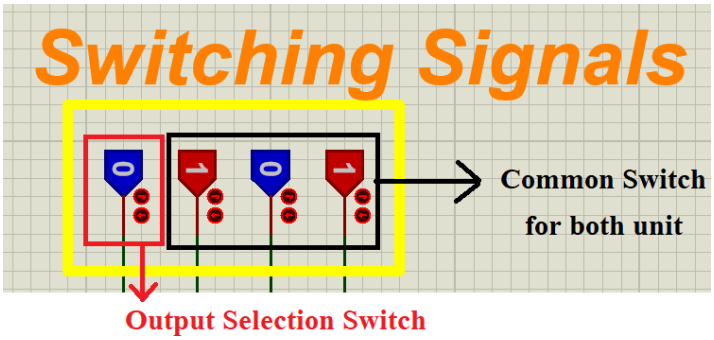


Figure: Switching sequence described

The last three bits enter the two units, and they determine which output will go from each unit. This decision making is done by IC74151 namely 8 to 1 Multiplexer. Here X0, X1, X2, X3, X4, X5 are single output bits from Adder, Subtractor, Increment Module, Decrement Module, Multiplier and Divider respectively. We need only 6 bits to get output so X6 and X7 are grounded. As the input of enable pin is inverted, we also gave ground to E.

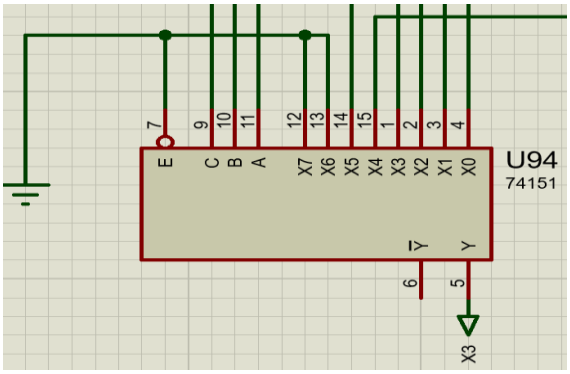


Figure: Switching Connection in both the module

A, B, and C are S0, S1, and S2 respectively. We got the single bit desired output from Y corresponding to our input switching sequences.

In Logic unit, we used the same multiplexing switching technique with the same switching sequences. However, in input pins, we used 7 pins as we have 7 selection dependent feature in logic unit along with 1 independent feature i.e., comparison which is shown always. The used input bits are: X0, X1, X2, X3, X4, X5, and X6 for bitwise AND, OR, NOT, NAND, NOR, XOR, and XNOR respectively. The pin X7 was grounded for its don't care condition.

As we used single multiplexer for getting single bit of the output, we used 16 8-to-1 MUXs in arithmetic unit and 8 8-to-1 MUXs in logic unit.

We should mention 1 thing here. Adding two 8 bits or incrementing an 8-bit number may result in a 9-bit number. So, we need to design all the outputs except multiplication and division as 9-bit output. Again, multiplying two 8-bit numbers will give us a 16-bit number at most. If we

divide an 8-bit number by an 8-bit number, then we get 8-bit quotient and 8-bit remainder. So, we need 16 bits for both multiplier and divider output.

As a result, our 10^{th} to 16^{th} MUX in arithmetic unit has only two input bits for multiplication and division at X4 and X5 respectively. All the other input ports are set to ground for don't care condition.

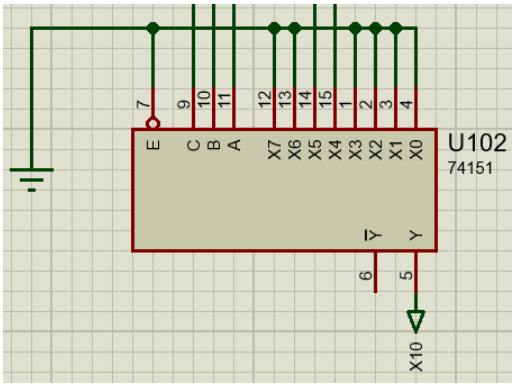


Figure: Only Multiplication and Division output bits are used in one of the last 7 MUXs

Now let's look at the last digit of switching sequence or namely S4. After getting bits from the two subcircuits of Arithmetic and Logic unit, we processed the single bits one by one. As the bits gave us the output from one desired operation of two units, we now need to choose which type of operation we need to select. If we want to get one of the arithmetic operations, we need to set S=0. If we need to get one of the logical operations, we need to set S=1.

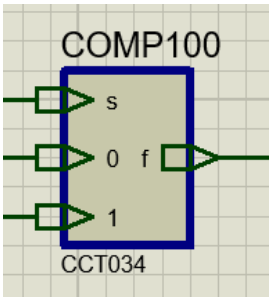


Figure: Main Layout 2 to 1 MUX

In the MUX setup, we gave S4 to s port, output from arithmetic part to 0 port and output from logic part to 1 port.

As Logic unit output is of 8 digits, our last 8 2-to-1 MUXs' port 1 is grounded.

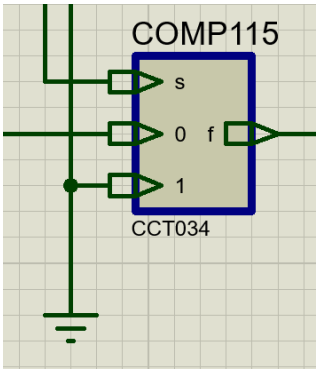


Figure: Grounded second input pin

Now we will look at the switching manual list quickly. Here, the last line suggest that comparison operation always gives output irrespective of the switching signal.

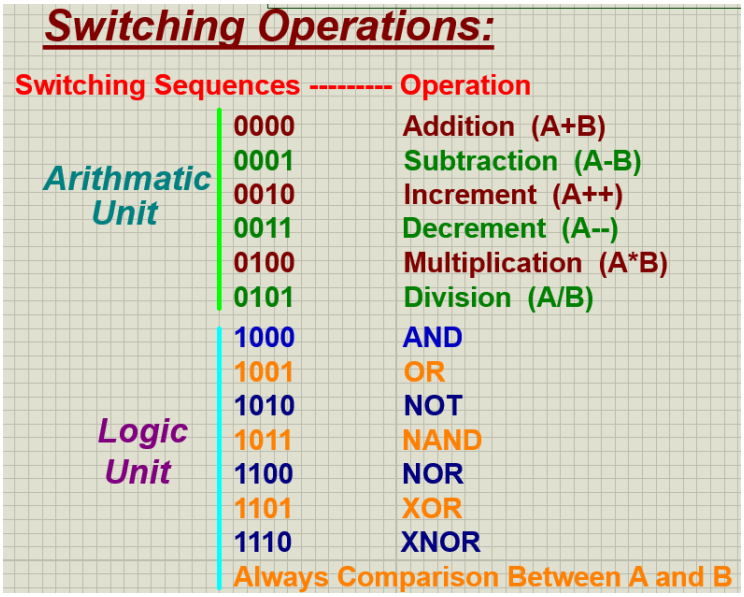


Figure: Switching operation manual given in the circuit

Input Signal Description:

Here we used two input signals. Both are of 8 bits. They are placed in binary digits as MSB at the leftmost position and LSB at the rightmost position.

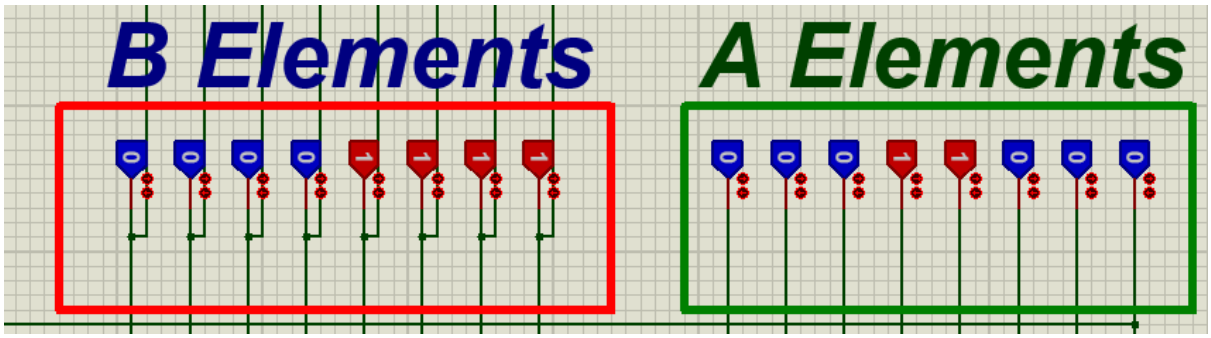


Figure: A and B input binary elements which are equivalent to 24 and 15 in Decimal respectively

We know that, using an 8-bit binary number, we can express at most $2^8 - 1 = 255$ in decimal. So, we need 3 seven segment displays to show decimal equivalent of our binary input bits. To convert the 8-bit binary digit into decimal, we made a subcircuit whose input is 8 bit of the input, and the output is 3 7-segment display output values (21 in total).

Now, we will look into the conversion circuit in details. For converting binary bits into BCD values, we used an IC 2732 which is called EPROM (Erasable Programmable Read Only Memory). A conversion code was written and used in EPROM, The converted BCD values

then were sent through IC 7447 which converts the BCD values to seven segment display input values.

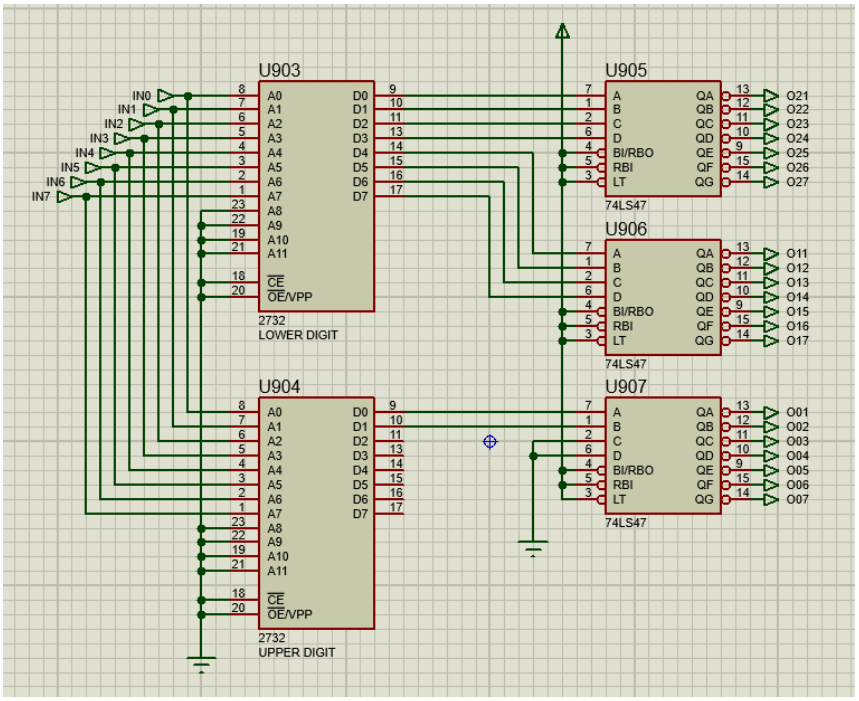


Figure: Inside of an 8-bit binary to Decimal Converter

The outputs from the converter then fed into 3 seven segment displays according to their position of hundreds, tens, and units.

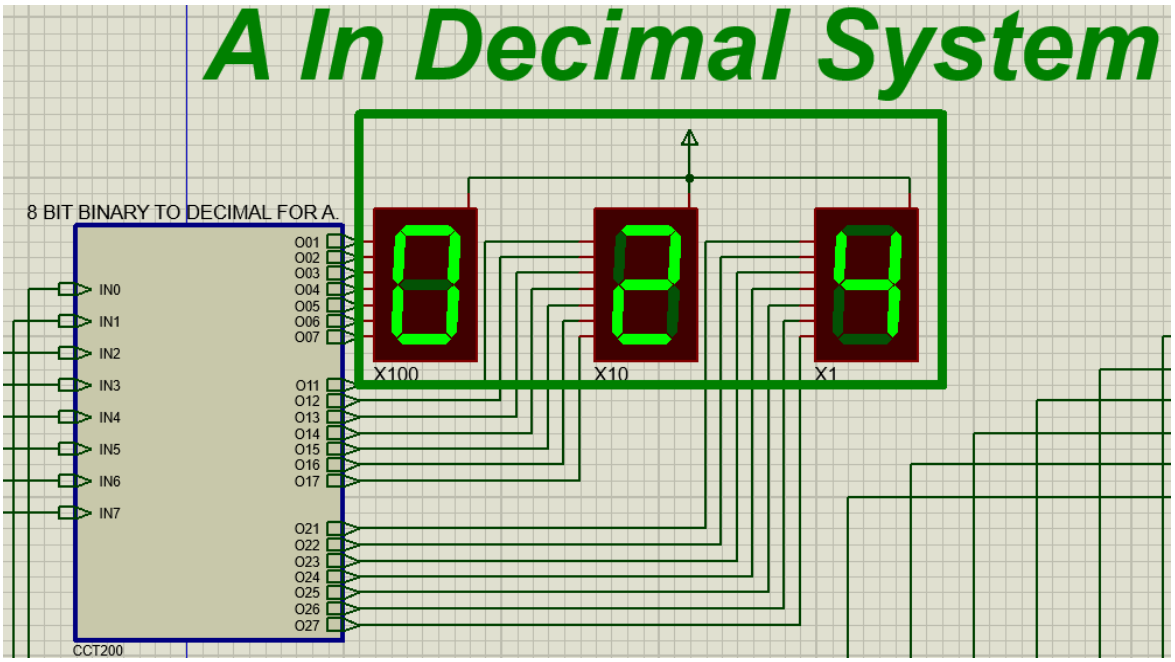


Figure: 8 Bit to Decimal Converter and output Displays

Output Signal Description:

There are 16 bits in the output in total. Generally, multiplication and division need 16 bits for output. So, we used 16 logic probes in output to get the output binary bits.

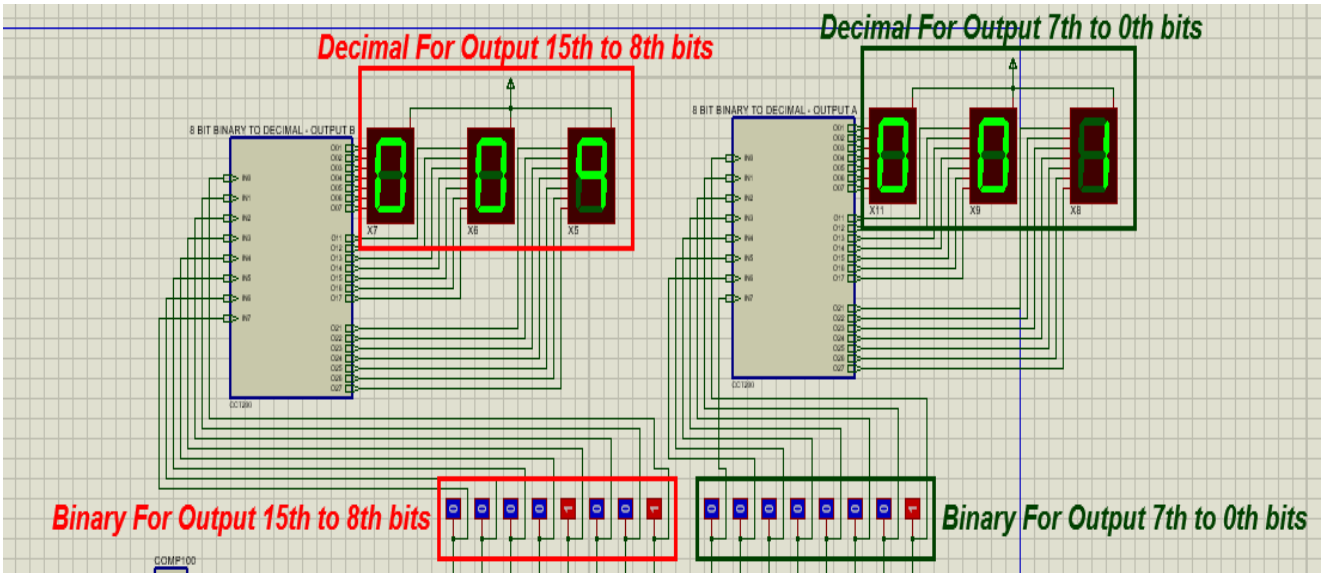


Figure: Output binary digits along with equivalent decimal value presenting 24÷ 15, resulting 1 as quotient and 9 as remainder

Here, the MSB bit is placed at the leftmost position and LSB is placed at the rightmost position. However, logic unit only used last 8 bits to show the output. Arithmetic unit shows output in 9 to 16 bits.

For binary to decimal conversion process, we used the same algorithm and the same subcircuit as the input ones.

There is another thing we need to mention about the division operation. Division operation always has 2 different outputs. One is quotient and another is remainder. In the output bits, 7 to 0 output bit positions show binary bits for quotient and corresponding 7-segment displays (right set) show decimal equivalent of the 8-bit quotient. Another one is remainder. In the output bits, 15 to 8 bits represent the 8-bit remainder and corresponding 7-segment displays (left set) show the decimal equivalent number of remainders.

Discussion:

In this project, we implemented a much simple prototype of arithmetic and logic unit (ALU) used in a computer. Both arithmetic and logic operations have been performed, and all operations involve 8 bits. We used 2 8-bit numbers and done different operations on them. In arithmetic part, we implemented addition, subtraction, increment, decrement, division, multiplication operation. Along with the binary bits, we also showed the inputs and the outputs in decimal. This would help to verify easily and quickly that our operation that was done, is correct. It will also help a learner to understand digital logic design easily.

In logic unit, we implemented bitwise AND, OR, NOT, NAND, NOR, XOR, XNOR, and Comparison operation. When 2 big binary numbers like 8-bit numbers are loaded, sometimes it becomes very difficult to identify which is smaller or bigger or whether they are same or not duo to a greater number of zeros and ones. So, this comparison operation will help a learner to quickly view about the two numbers not only when someone is working on logical operation but also on arithmetic operation.

All these arithmetic and logic operations are implemented in this project. Different switching codes are used to perform different arithmetic and logic operations. Inputs are taken in binary, and the results obtained are shown as the decimal equivalents of 8-bit binary numbers with the help of a 7-segment display system.

When designing such prototype, using software simulation is an important job. Unless someone gets better result in simulation, it can never be implemented on hardware. Proteus is a very good software to get an output which is very close to the real-life outputs. However, our software model gave a better output which satisfies our proposal.