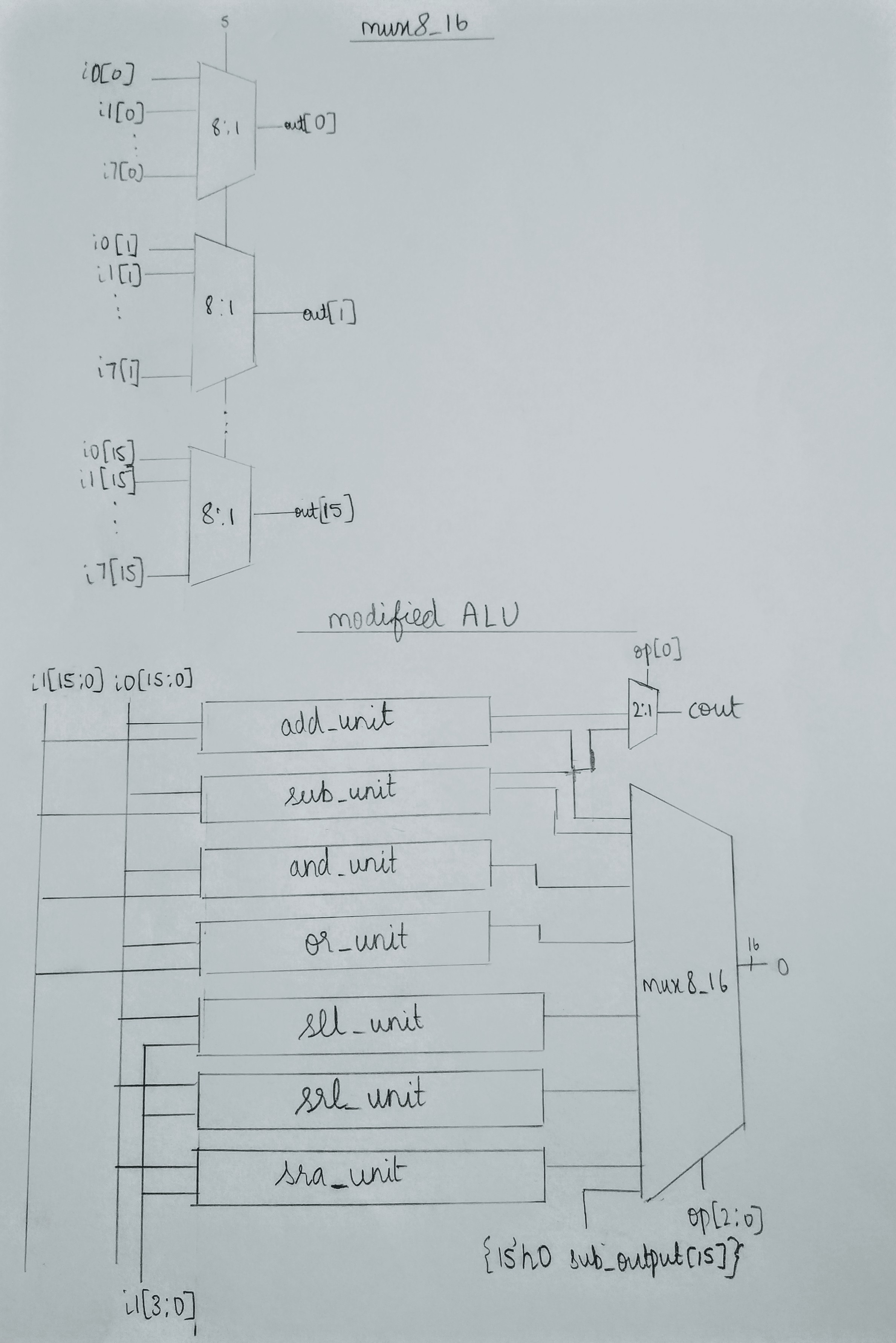
SHIFT OPERATIONS

PROBLEM DESCRIPTION:

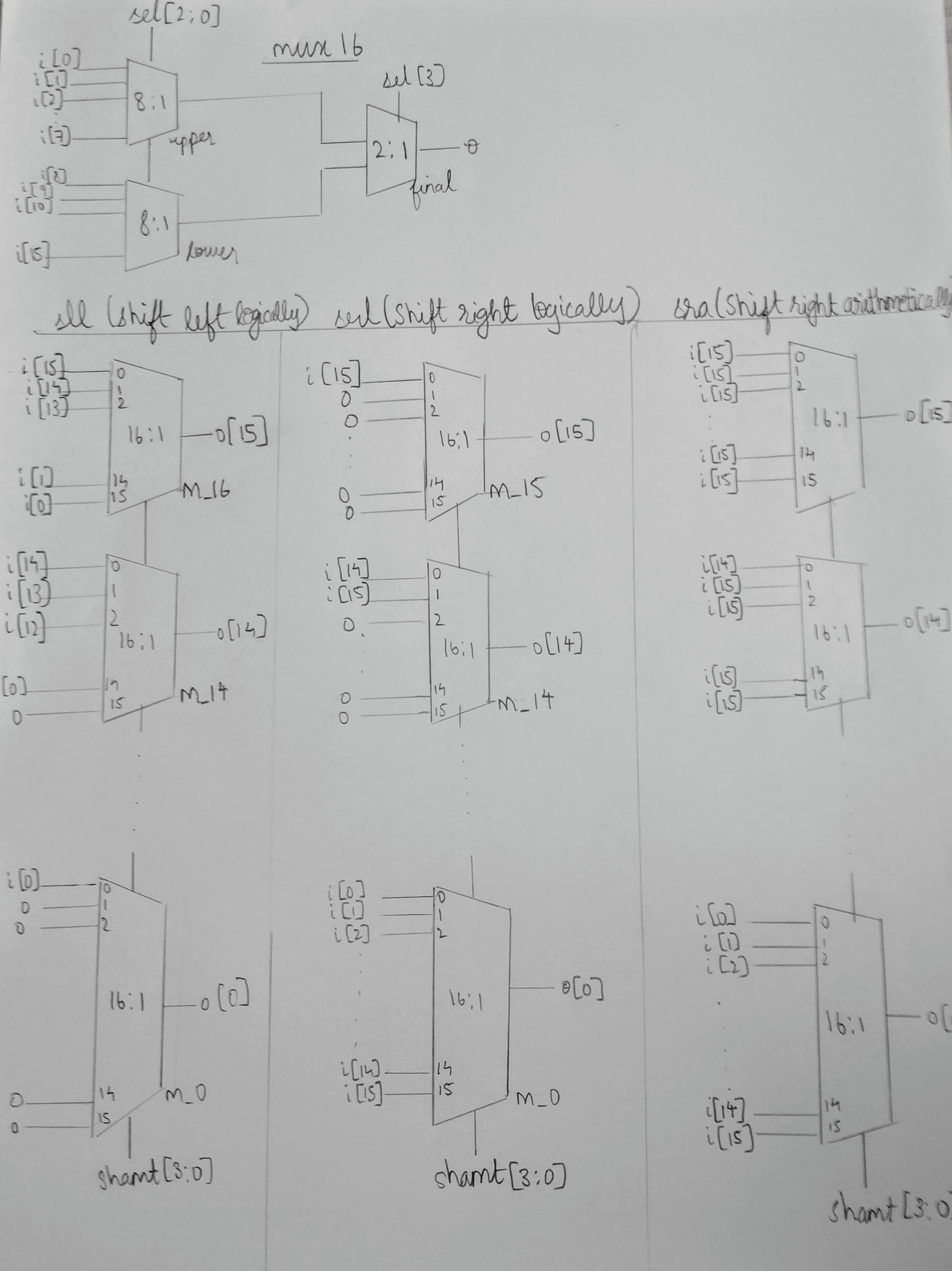
Implementing a left logical shifter, right logical shifter, arithmetic right shifter and a set less than operation. Finally integrating it with existing ALU.

LITERATURE SURVEY:

We searched the internet to understand the structure and working of a Logical shifter, arithmetic shifter and a set less than operation. After a bit of brainstorming, we decided to solve the problem using Multiplexers.

 Circuit Diagram for ALU

Circuit Diagram for Shifters and SLT



VERILOG CODE:

*module sll(input wire [15:0] i, input wire [3:0] shamt, output wire [15:0] o);*

*mux16 m\_15( { i[15], i[14], i[13], i[12], i[11], i[10], i[9], i[8], i[7], i[6], i[5], i[4], i[3], i[2], i[1], i[0] }, shamt, o[15]);*

*mux16 m\_14( { i[14], i[13], i[12], i[11], i[10], i[9], i[8], i[7], i[6], i[5], i[4], i[3], i[2], i[1], i[0], 1'b0 }, shamt, o[14]);*

*mux16 m\_13( { i[13], i[12], i[11], i[10], i[9], i[8], i[7], i[6], i[5], i[4], i[3], i[2], i[1], i[0], 1'b0, 1'b0 }, shamt, o[13]);*

*mux16 m\_12( { i[12], i[11], i[10], i[9], i[8], i[7], i[6], i[5], i[4], i[3], i[2], i[1], i[0], 1'b0, 1'b0, 1'b0 }, shamt, o[12]);*

*mux16 m\_11( { i[11], i[10], i[9], i[8], i[7], i[6], i[5], i[4], i[3], i[2], i[1], i[0], 1'b0, 1'b0, 1'b0, 1'b0 }, shamt, o[11]);*

*mux16 m\_10( { i[10], i[9], i[8], i[7], i[6], i[5], i[4], i[3], i[2], i[1], i[0], 1'b0, 1'b0, 1'b0, 1'b0, 1'b0 }, shamt, o[10]);*

*mux16 m\_9( { i[9], i[8], i[7], i[6], i[5], i[4], i[3], i[2], i[1], i[0], 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0 }, shamt, o[9]);*

*mux16 m\_8( { i[8], i[7], i[6], i[5], i[4], i[3], i[2], i[1], i[0], 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0 }, shamt, o[8]);*

*mux16 m\_7( { i[7], i[6], i[5], i[4], i[3], i[2], i[1], i[0], 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0 }, shamt, o[7]);*

*mux16 m\_6( { i[6], i[5], i[4], i[3], i[2], i[1], i[0], 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0 }, shamt, o[6]);*

*mux16 m\_5( { i[5], i[4], i[3], i[2], i[1], i[0], 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0 }, shamt, o[5]);*

*mux16 m\_4( { i[4], i[3], i[2], i[1], i[0], 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0 }, shamt, o[4]);*

*mux16 m\_3( { i[3], i[2], i[1], i[0], 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0 }, shamt, o[3]);*

*mux16 m\_2( { i[2], i[1], i[0], 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0 }, shamt, o[2]);*

*mux16 m\_1( { i[1], i[0], 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0 }, shamt, o[1]);*

*mux16 m\_0( { i[0], 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0 }, shamt, o[0]);*

*endmodule*

The module sll is abbreviation for Shift Left Logic. We accept the input 16 bit number and depending on the value of second input “shamt” we decide how much the bit is shifted by.

The output is directed to ‘o’. By using a MUX 16 that is repeated 16 times, with shamt as our control lines. The output of each of the MUXes, is a particular bit of the output number.The inputs are the bits of input number. the shamt control line, which also decides how much to shift by, chooses appropriate input bit and puts it onto output. there are also several zeros in the MUX, when we shift so much that there are no numbers to the right remaining. this increases as we increase shift value.

*module srl(input wire [15:0] i, input wire [3:0] shamt, output wire [15:0] o);*

*mux16 m\_0( { i[0], i[1], i[2], i[3], i[4], i[5], i[6], i[7], i[8], i[9], i[10], i[11], i[12], i[13], i[14], i[15] }, shamt, o[0]);*

*mux16 m\_1( { i[1], i[2], i[3], i[4], i[5], i[6], i[7], i[8], i[9], i[10], i[11], i[12], i[13], i[14], i[15], 1'b0 }, shamt, o[1]);*

*mux16 m\_2( { i[2], i[3], i[4], i[5], i[6], i[7], i[8], i[9], i[10], i[11], i[12], i[13], i[14], i[15], 1'b0, 1'b0 }, shamt, o[2]);*

*mux16 m\_3( { i[3], i[4], i[5], i[6], i[7], i[8], i[9], i[10], i[11], i[12], i[13], i[14], i[15], 1'b0, 1'b0, 1'b0 }, shamt, o[3]);*

*mux16 m\_4( { i[4], i[5], i[6], i[7], i[8], i[9], i[10], i[11], i[12], i[13], i[14], i[15], 1'b0, 1'b0, 1'b0, 1'b0 }, shamt, o[4]);*

*mux16 m\_5( { i[5], i[6], i[7], i[8], i[9], i[10], i[11], i[12], i[13], i[14], i[15], 1'b0, 1'b0, 1'b0, 1'b0, 1'b0 }, shamt, o[5]);*

*mux16 m\_6( { i[6], i[7], i[8], i[9], i[10], i[11], i[12], i[13], i[14], i[15], 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0 }, shamt, o[6]);*

*mux16 m\_7( { i[7], i[8], i[9], i[10], i[11], i[12], i[13], i[14], i[15], 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0 }, shamt, o[7]);*

*mux16 m\_8( { i[8], i[9], i[10], i[11], i[12], i[13], i[14], i[15], 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0 }, shamt, o[8]);*

*mux16 m\_9( { i[9], i[10], i[11], i[12], i[13], i[14], i[15], 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0 }, shamt, o[9]);*

*mux16 m\_10( { i[10], i[11], i[12], i[13], i[14], i[15], 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0 }, shamt, o[10]);*

*mux16 m\_11( { i[11], i[12], i[13], i[14], i[15], 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0 }, shamt, o[11]);*

*mux16 m\_12( { i[12], i[13], i[14], i[15], 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0 }, shamt, o[12]);*

*mux16 m\_13( { i[13], i[14], i[15], 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0 }, shamt, o[13]);*

*mux16 m\_14( { i[14], i[15], 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0 }, shamt, o[14]);*

*mux16 m\_15( { i[15], 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0 }, shamt, o[15]);*

*endmodule*

Using the same logic as in the above module, we implement right shift. All we change is the position of values inputted to the MUX.

*module sra(input wire [15:0] i, input wire [3:0] shamt, output wire [15:0] o);*

*mux16 m\_0( { i[0], i[1], i[2], i[3], i[4], i[5], i[6], i[7], i[8], i[9], i[10], i[11], i[12], i[13], i[14], i[15] }, shamt, o[0]);*

*mux16 m\_1( { i[1], i[2], i[3], i[4], i[5], i[6], i[7], i[8], i[9], i[10], i[11], i[12], i[13], i[14], i[15], i[15] }, shamt, o[1]);*

*mux16 m\_2( { i[2], i[3], i[4], i[5], i[6], i[7], i[8], i[9], i[10], i[11], i[12], i[13], i[14], i[15], i[15], i[15] }, shamt, o[2]);*

*mux16 m\_3( { i[3], i[4], i[5], i[6], i[7], i[8], i[9], i[10], i[11], i[12], i[13], i[14], i[15], i[15], i[15], i[15] }, shamt, o[3]);*

*mux16 m\_4( { i[4], i[5], i[6], i[7], i[8], i[9], i[10], i[11], i[12], i[13], i[14], i[15], i[15], i[15], i[15], i[15] }, shamt, o[4]);*

*mux16 m\_5( { i[5], i[6], i[7], i[8], i[9], i[10], i[11], i[12], i[13], i[14], i[15], i[15], i[15], i[15], i[15], i[15] }, shamt, o[5]);*

*mux16 m\_6( { i[6], i[7], i[8], i[9], i[10], i[11], i[12], i[13], i[14], i[15], i[15], i[15], i[15], i[15], i[15], i[15] }, shamt, o[6]);*

*mux16 m\_7( { i[7], i[8], i[9], i[10], i[11], i[12], i[13], i[14], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15] }, shamt, o[7]);*

*mux16 m\_8( { i[8], i[9], i[10], i[11], i[12], i[13], i[14], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15] }, shamt, o[8]);*

*mux16 m\_9( { i[9], i[10], i[11], i[12], i[13], i[14], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15] }, shamt, o[9]);*

*mux16 m\_10( { i[10], i[11], i[12], i[13], i[14], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15] }, shamt, o[10]);*

*mux16 m\_11( { i[11], i[12], i[13], i[14], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15] }, shamt, o[11]);*

*mux16 m\_12( { i[12], i[13], i[14], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15] }, shamt, o[12]);*

*mux16 m\_13( { i[13], i[14], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15] }, shamt, o[13]);*

*mux16 m\_14( { i[14], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15] }, shamt, o[14]);*

*mux16 m\_15( { i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15], i[15] }, shamt, o[15]);*

*endmodule*

This is almost the same as logical right shift operation, with the only difference being, that when we shift for a value such that there are no digits remaining on the left to shift to the right, instead of keeping such digits as zero, we replace them with the value of the MSB( most significant bit) of the input number.

module mux16(input wire [0:15] i, input wire [3:0] shamt, output wire o);

mux8 upper(i[0:7], shamt[0], shamt[1], shamt[2], o1);

mux8 lower(i[8:15], shamt[0], shamt[1], shamt[2], o2);

mux2 final(o1, o2, shamt[3], o);

endmodule

just a 16:1 MUX implemented using 2 8:1 MUXs and a 2:1 MUX.

SLT is implemented in the main ALU. In this, we use existing subtract operation and subtract two numbers. The MSB of this number gives us the sign of the number(2’s complement notation).

If it is 1, number is negative, and the second number is greater than the first, but if the MSB is 0, then the first number is greater than or equal to second number. we just put this MSB as the LSB of the output answer, leaving the other 15 bits of the output number as zero.