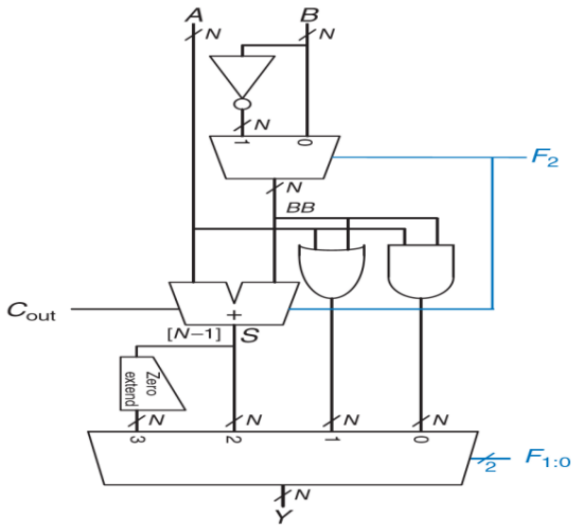
Name: Tanveer Singh Kahlon
Date: 11/29/2022
Lab 7: ALU Design

1. Short ALU Introduction paragraph

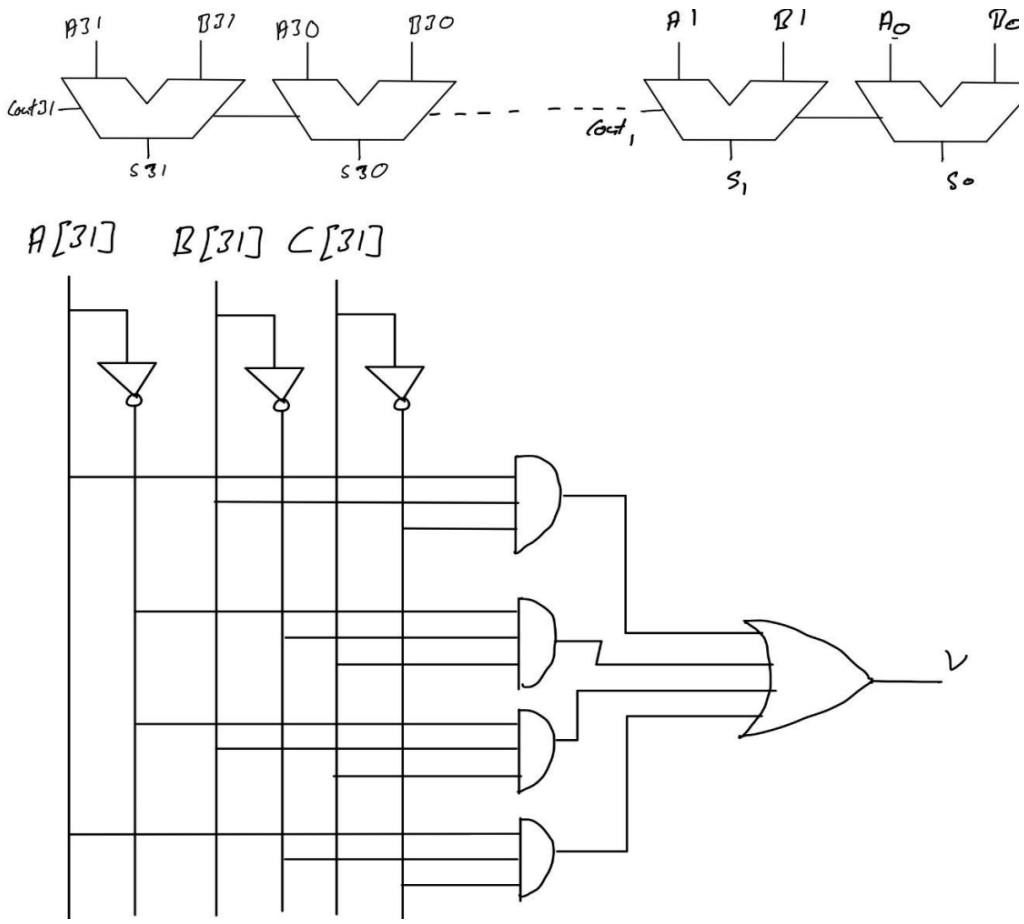
This lab is a designing an ALU with ability to perform certain functions to 32-bit number given the gates and multiplexers it has. This ALU performs AND, OR, addition, subtraction functions and also operates a set less than function on two different 32-bit number.

2. ALU Schematic and control signals table representing what combination of control signal performs which operation



F 2:0	Function
000	A AND B
001	A OR B
010	A + B
011	Not used
100	A AND B’
101	A OR B’
110	A – B
111	Set Less Than (SLT)

3. Schematic and explanation for overflow logic



The overflow flag is enabled in 4 scenarios:
1) Adding 2 positive numbers and getting a negative result
2) Adding 2 negative numbers and getting a positive result
3) Subtracting a positive from a negative number but getting a positive result
4) Subtracting a negative from a positive number but getting a negative result

4. Screenshot of Verilog code for 32 bit ALU with overflow and Zero output

```
module ALU_Design(A, B, F, Y, V, Z);
input[31:0] A, B;
input[2:0] F;
output reg[31:0] Y;
output reg V;
output Z;
wire[31:0] S, BB;

assign BB = F[2] ? ! B :B;
assign S = A + BB +F[2];

always @ (*)
```

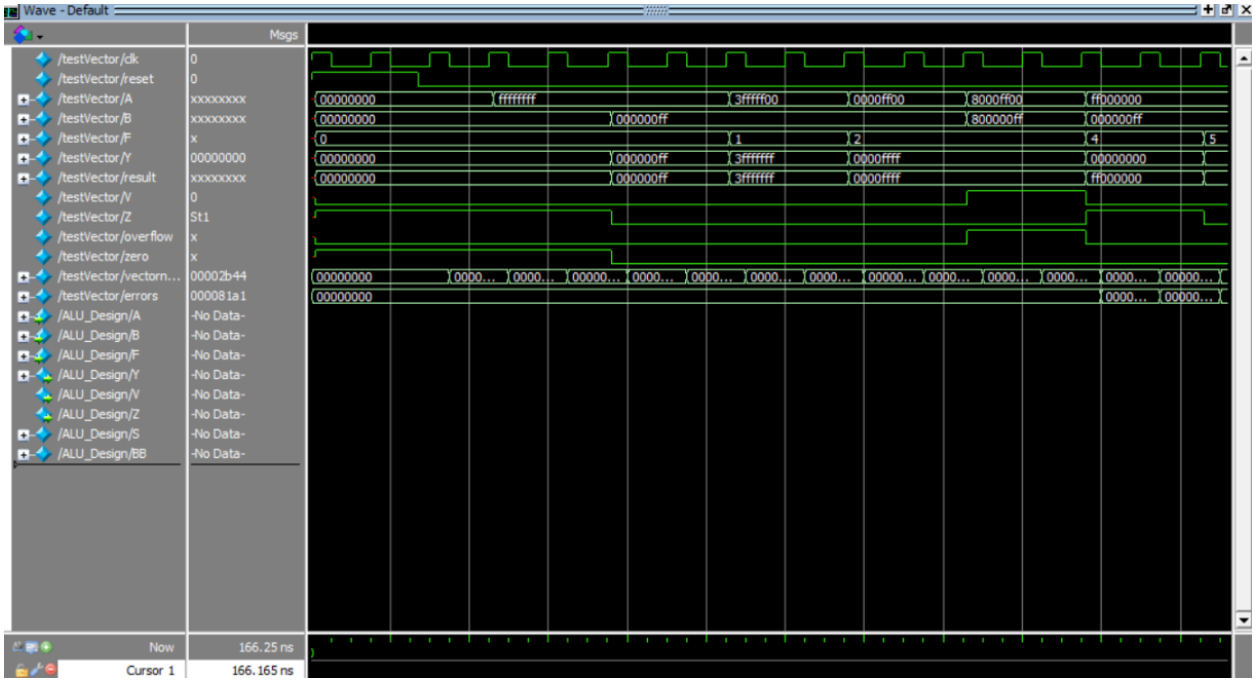
```
case (F[1:0])
2'b00: Y <= A & BB;
2'b01: Y <= A | BB;
2'B10: Y <= S;
2'B11: Y <= S[31];
endcase

assign Z = (Y == 32'b0);

always @ (*)

case (F[2])

1'b0: V <= (A[31] & B[31] & ~S[31]) | (~A[31] & ~B[31] & S[31]);
1'b1: V <= (~A[31] & B[31] & S[31]) | (A[31] & ~B[31] & ~S[31]);
default: V <= 1'b0;
endcase
endmodule
```

5. Screenshot of testbench

```
module testVector();
reg clk, reset;
reg[31:0] A, B;
reg[2:0] F;
wire [31:0] Y;
reg[31:0] result;
wire V, Z;
reg overflow, zero;
reg [31:0] vectornum, errors;
reg[100:0] testvectors[10000:0];

ALU_Design dut(A, B, F, Y, V,Z);

always
begin
clk = 1; #5; clk = 0; #10;
end

initial
begin

$readmemb("testVector.tv", testvectors);
vectornum = 0; errors = 0;
```

```
reset = 1; #27; reset = 0;
end

always @(posedge clk)
begin
#1; {A, B, F, overflow, zero, result} = testvectors[vectornum];
end

always @(negedge clk)
if(~reset) begin
if(result !== Y) begin
$display("result Error: inputs = %b", {Y});
$display("outputs = %b (%b expected)", Y, result);
errors = errors + 1;
end

if(zero !== Z) begin
$display("result Error: inputs = %b", {Z});
$display("outputs = %b (%b expected)", Z, zero);
errors = errors + 1;
end

if(overflow !== V) begin
$display("result Error: inputs = %b", {V});
$display("outputs = %b (%b expected)", V, overflow);
errors = errors + 1;
end

vectornum = vectornum + 1;
if(testvectors[vectornum] === 100'bx) begin
$display("%d tests completed with %d errors", vectornum, errors);
$finish;
end
end

endmodule
```

7. Simulation and console results screenshot of testbench



8. Summary

This lab is a design of ALU which is a learning of working with a large number of data. This lab is using 2 different multiplexers, AND, OR gates and adders. Also used the test bench and test vector to perform all the functions of ALU. The ALU was also able to perform 32 bit addition, subtraction, and other arithmetic that met all the requirements perfectly and did not return any errors due to unexpected results. Overall, it was a fun experiment and we learned that the functionality of the ALU can be expanded even further.