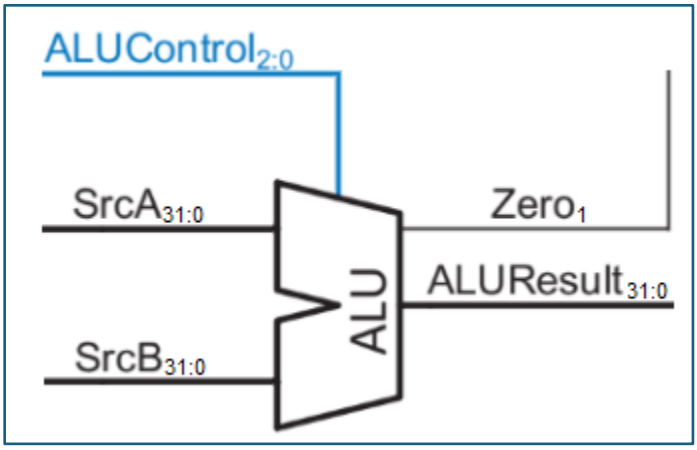
ALU Block

1- الهدف هوا اننا نعمل الـ Arithmetic and logic operation.

2- الـ ALU Block بيكون ليها كذا دخل وخرج واحد :-



Inputs :

1- **SrcA** : Frist operand [32 bits].

2- **SrcB** : Second operand [32 bits].

3- **ALUControl** : Operation Selector [3 bits].

Output :

➔ **ALUResult** : The operation result [32 bits].

➔ **Zero** : (Branch if equal) indicator [1 bit].

3- The RTL Code :-

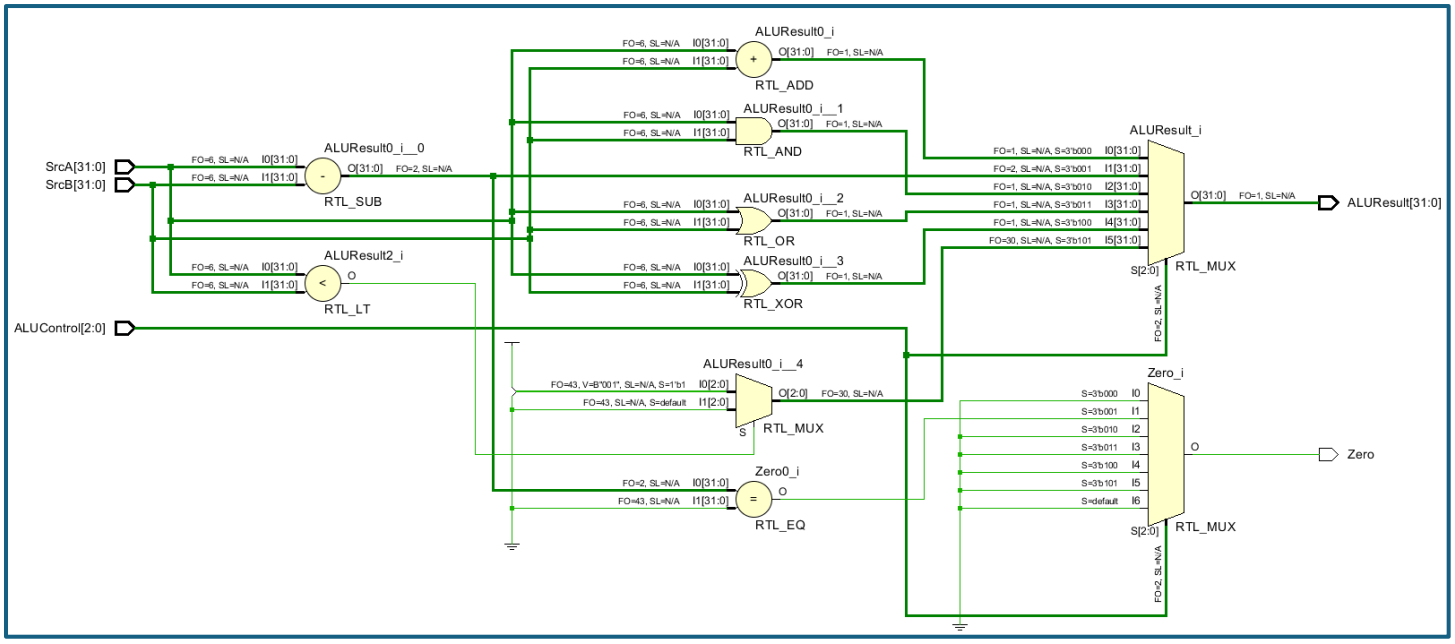
```
1   `timescale 1ns / 1ps
2   module ALU (SrcA, SrcB, ALUControl, ALUResult, Zero);
3
4       input [31:0] SrcA;              // First operand
5       input [31:0] SrcB;              // Second operand
6       input [2:0] ALUControl;         // (from ALU Decoder)
7       output reg [31:0] ALUResult;
8       output reg Zero;                 // Zero flag: 1 if ALUResult is zero
9
10      always @(*) begin
11          case (ALUControl)
12              3'b000:
13              begin
14                  ALUResult = SrcA + SrcB;          // Addition
15                  Zero = 0;
16              end
17              3'b001:
18              begin
19                  ALUResult = SrcA - SrcB;          // Subtraction
20                  Zero = (ALUResult == 32'b0);      // Zero flag (for beq)
21              end
22              3'b010:
23              begin
24                  ALUResult = SrcA & SrcB;          // Bitwise AND
25                  Zero = 0;
26              end
27              3'b011:
28              begin
29                  ALUResult = SrcA | SrcB;          // Bitwise OR
30                  Zero = 0;
31              end
32              3'b100:
33              begin
34                  ALUResult = SrcA ^ SrcB;          // Bitwise XOR
35                  Zero = 0;
36              end
37
38              3'b101:
39              begin
40                  ALUResult = (SrcA < SrcB) ? 1 : 0; // Set Less Than (slt)
41                  Zero = 0;                          // Zero flag should not be used here
42              end
43
44              default:
45              begin
46                  ALUResult = 32'bxxx;               // Default to unknown
47                  Zero = 0;
48              end
49          endcase
50      end
51  endmodule
```

← الـ **ALU** مصمم انه يعمل العمليات الاتيه : -

ALUControl	Operation
3'b000	Addition (ADD) / Add Immediate (ADDI)
3'b001	Subtraction (SUB)
3'b010	Bitwise AND (AND) / AND Immediate (ANDI)
3'b011	Bitwise OR (OR) / OR Immediate (ORI)
3'b100	Bitwise XOR (XOR)
3'b101	Set Less Than (SLT)

← مع كل **operation** بتتعمل بنحدد قيمة الـ **Zero** هتكون عامله ازاي , فهي الفكرة اننا لو طرحنا الـ **2 sources** ولقينا الناتج بـ **0** فدا معناه انهم كانوا متساوين ولازم يطلع **Flag** قيمة **1** يروح للـ**Control** , فإحنا مش فارق معانا الـ **Zero** في الـ **operation** التانيه المهم بالنسبالي هوا الطرح بس.

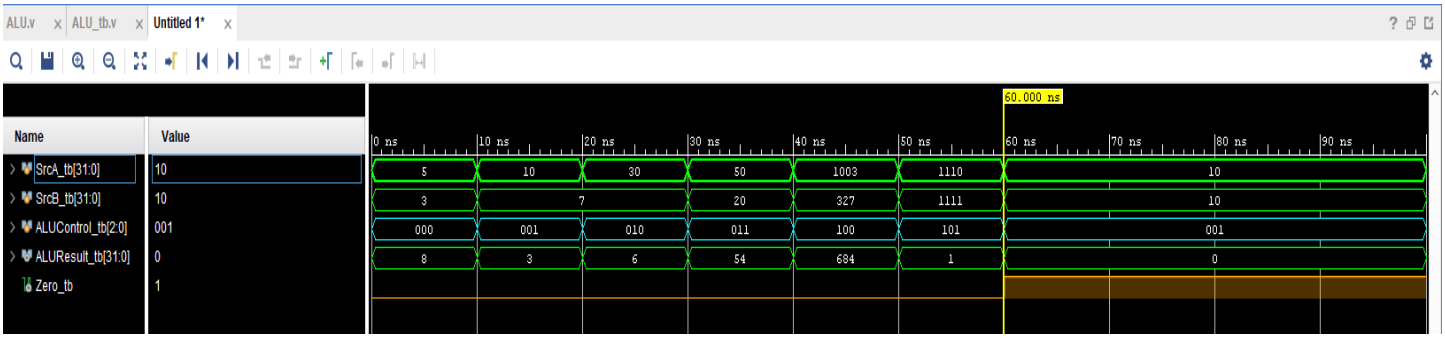
4- Elaborated design:-



5- Testbench :

```
`timescale 1ns / 1ps

module ALU_tb();

    // 1) Declare local reg and wire identifiers with _tb suffix
    reg  [31:0] SrcA_tb;
    reg  [31:0] SrcB_tb;
    reg  [2:0] ALUControl_tb;
    wire [31:0] ALUResult_tb;
    wire Zero_tb;

    // 2) Instantiate the ALU module under test
    ALU uut (
        .SrcA(SrcA_tb),
        .SrcB(SrcB_tb),
        .ALUControl(ALUControl_tb),
        .ALUResult(ALUResult_tb),
        .Zero(Zero_tb)
    );

    // 3) Generate stimuli using initial and always
    initial
     begin
        // Test case 0: Addition
        SrcA_tb = 32'd5;         SrcB_tb = 32'd3;      ALUControl_tb = 3'b000;  /* ADD operation*/    #10;

        // Test case 1: Subtraction
        SrcA_tb = 32'd10;        SrcB_tb = 32'd7;      ALUControl_tb = 3'b001;  /* SUB operation*/    #10;

        // Test case 2: Bitwise AND
        SrcA_tb = 32'd30;        SrcB_tb = 32'd7;      ALUControl_tb = 3'b010;  /* AND operation*/    #10;

        // Test case 3: Bitwise OR
        SrcA_tb = 32'd50;        SrcB_tb = 32'd20;     ALUControl_tb = 3'b011;  /* OR operation*/     #10;

        // Test case 4: Bitwise XOR
        SrcA_tb = 32'd1003;        SrcB_tb = 32'd327;  ALUControl_tb = 3'b100;  /* XOR operation*/    #10;

        // Test case 5: Set Less Than (SLT)
        SrcA_tb = 32'd1110;      SrcB_tb = 32'd1111;   ALUControl_tb = 3'b101;  /* SLT operation*/    #10;

        // Test case 6: Zero flag check for subtraction
        SrcA_tb = 32'd10;       SrcB_tb = 32'd10;      ALUControl_tb = 3'b001;   /* Branch if equal*/ #10;
    end

    // 4) Specify a stopwatch to stop the simulation
    initial
    begin
        #100 $stop;
    end
endmodule
```

←نتيجة الـ **Simulation** :



30 AND 7 =

6

50 OR 20 =

54

1003 XOR 327 =

684

Made by: Hossam Ahmed Seyam