Spring 2021

California State University, Northridge

Department of Electrical & Computer Engineering

Experiment 8

Arithmetic-Logic Unit Modeling

April 17, 2021

ECE 526L

Written By: Jose Luis Martinez

Introduction

In this lab we are tasked to design an ALU with as shown in **Fig. 1** with A, B, and ALU_OUT being eight bit inputs and OP_CODE being four bits. The ALU will perform an operation on the next clock cycle based on what OPCODE is given to it. For this ALU will be basing the operations on the table in **Fig. 2**. The ALU EN will enable the ALU to accept and display results. The OE will enable the output and when it is not enabled the output will be set to high impedance. The OF flag is an indication that a signed operation exceeded the ALU's output size. The Carry flag is for when unsigned numbers when adding the sum would exceed the output's bit width or when subtracting the number that is being subtracted is smaller than the number that you are subtracting by. The zero flag is when the output is zero and the negative flag is when the MSB of the output is 1. ALU_OUT will simply display the results of the operation based on the OPCODE.

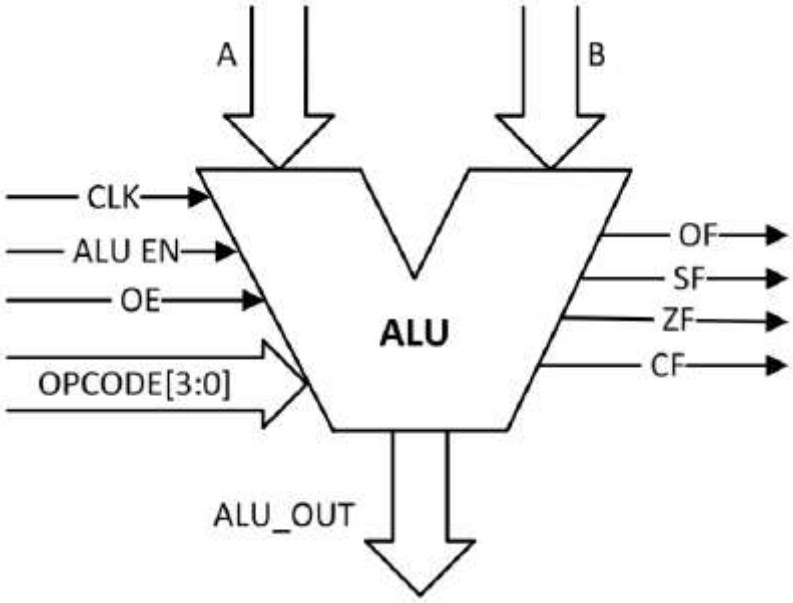

Fig. 1 ALU Model

Lab 8 Table 1: Opcodes and Actions

Opcode	Action
4'b0010	A + B
4'b0011	A - B
4'b0100	A and B
4'b0101	A or B
4'b0110	A xor B
4'b0111	not A

Fig. 2 ALU OPCODE Table

Methodology

I followed these steps in order to complete the lab:

1. The ALU module was created as shown in **Fig. 3.**
    a. The module contains a WIDTH parameter with a default value of 8.
    b. The module also has an ifdef compiler directive to easily switch between signed and unsigned.
2. A module was created to hold all the constants and timescales for the alu and testbench to use as shown in **Fig. 4.**
3. The test bench file was created as shown in **Fig. 5**.
    a. The module also contains an ifdef compiler directive to easily switch between signed and unsigned.
    b. The test bench module tests the ALU module to ensure that the ALU_OUT and the flags are working properly.
4. For compiling two scripts were used to facilitate the process.
    a. One script had +define+SIGNED appended it to at the end of the line.
    b. Screenshots of the results were taken for both signed and unsigned versions.

Results/Verilog Files

```
/*********************************************************************
***                                                                ***
*** ECE 526 L Experiment #8            Jose Luis Martinez, Spring, 2021    ***
***                                                                ***
*** Experiment 8 - Arithmetic Logic Unit                           ***
***                                                                ***
*********************************************************************
*** Filename: alu.v          Created by Jose Luis Martinez, April 16, 2021 ***
***                                                                ***
*********************************************************************/

`include "definitions.v"

module alu(CLK, EN, OE, OPCODE, A, B, ALU_OUT, CF, ZF, SF, OF);
    parameter WIDTH = 8;
    input CLK, EN, OE;
    input [3:0] OPCODE;

    `ifdef SIGNED
        input signed [WIDTH-1:0]A, B;
    `else
        input [WIDTH-1:0]A, B;
    `endif

    output [WIDTH-1:0]ALU_OUT;
    output CF, ZF, SF, OF;

    reg [WIDTH:0]temp;
    reg [WIDTH-1:0]A_temp, B_temp;
    reg [3:0] opcode_temp;
    wire [WIDTH-1:0]twos_complement_B;

    localparam ALU_SUM = 4'b0010;
    localparam ALU_SUB = 4'b0011;
    localparam ALU_AND = 4'b0100;
    localparam ALU_OR  = 4'b0101;
    localparam ALU_XOR = 4'b0110;
    localparam ALU_NOT = 4'b0111;

    assign twos_complement_B = ~B + 1'b1;

    assign ZF = (ALU_OUT == 0) ? 1 : 0;
    assign ALU_OUT = (OE&EN) ? temp[WIDTH-1:0] : {WIDTH{1'bz}};
    assign CF = (((opcode_temp == ALU_SUM) && temp[WIDTH]) || ((opcode_temp == ALU_SUB)
                && A_temp<B_temp)) ? 1'b1 : 1'b0;
    assign SF = temp[WIDTH-1];
    assign OF = ((A_temp[WIDTH-1] == B_temp[WIDTH-1])&&(A_temp[WIDTH-1] != temp[WIDTH-1])
                && (opcode_temp == ALU_SUM)) || ((A_temp[WIDTH-1] == twos_complement_B[WIDTH-1])
```

```
                &&(A_temp[WIDTH-1] != temp[WIDTH-1]) && (opcode_temp == ALU_SUB))? 1'b1 : 1'b0;

    always@(posedge CLK) begin
        if(EN) begin
            A_temp <= A;
            B_temp <= B;
            opcode_temp <= OPCODE;
            case (OPCODE)
                ALU_SUM: begin
                    temp <= A + B;
                end
                ALU_SUB: begin
                    temp <= A + twos_complement_B;
                end
                ALU_AND: temp <= {1'b0, A & B};
                ALU_OR:  temp <= {1'b0, A | B};
                ALU_XOR: temp <= {1'b0, A ^ B};
                ALU_NOT: temp <= {1'b0, ~A};
                default: temp <= 0;
            endcase
        end
    end

endmodule
```

Fig. 3 alu.v

```
/*************************************************************************
***                                                                    ***
*** ECE 526 L Experiment #8            Jose Luis Martinez, Spring, 2021    ***
***                                                                    ***
*** Experiment 8 - Arithmetic Logic Unit                               ***
***                                                                    ***
*************************************************************************
*** Filename: definitions.v   Created by Jose Luis Martinez, April 16, 2021 ***
***                                                                    ***
*************************************************************************/

`timescale 1ns/100ps
`define    WIDTH_TB   8
`define    CLK_PER    20
`define    ALU_ADD_TB 4'b0010
`define    ALU_SUB_TB 4'b0011
`define    ALU_AND_TB 4'b0100
`define    ALU_OR_TB  4'b0101
`define    ALU_XOR_TB 4'b0110
`define    ALU_NOT_TB 4'b0111
`define    STRING     "%d, OPCODE = %h, A = %d, B = %d, EN = %b, OE = %b, CLK = %b| CF = %b, ZF = %b, SF = %b, OF = %b, ALU_OUT = %d"
```

Fig. 4 definitions.v

```
/*********************************************************************
***                                                                ***
*** ECE 526 L Experiment #8            Jose Luis Martinez, Spring, 2021    ***
***                                                                ***
*** Experiment 8 - Arithmetic Logic Unit                           ***
***                                                                ***
*********************************************************************
*** Filename: lab8_tb.v       Created by Jose Luis Martinez, April 16, 2021 ***
***                                                                ***
*********************************************************************/

`include "definitions.v"

module lab8_tb();
    reg CLK, EN, OE;
    reg [3:0] OPCODE;

    `ifdef SIGNED
        reg signed [`WIDTH_TB-1:0]A, B;
        wire signed [`WIDTH_TB-1:0] ALU_OUT;
    `else
        reg [`WIDTH_TB-1:0]A, B;
        wire [`WIDTH_TB-1:0] ALU_OUT;
    `endif

    wire CF, ZF, SF, OF;

    alu #(.WIDTH(`WIDTH_TB)) alu1(.CLK(CLK), .EN(EN), .OE(OE), .OPCODE(OPCODE), .A(A),
          .B(B), .ALU_OUT(ALU_OUT), .CF(CF), .ZF(ZF), .SF(SF), .OF(OF));

    initial begin
        CLK <= 0;
        forever begin
            #(`CLK_PER/2) CLK <= ~CLK;
        end
    end

    initial begin
        $vcdpluson;
        EN <= 1;
        OE <= 1;
        A <= 8'b0000_0111;
        B <= 8'b0000_0111;
        OPCODE <= `ALU_ADD_TB;
        $strobe(`STRING, $time, OPCODE, A, B, EN, OE, CLK, CF, ZF, SF, OF, ALU_OUT);
        #(`CLK_PER)
        OPCODE <= `ALU_SUB_TB;
        $strobe(`STRING, $time, OPCODE, A, B, EN, OE, CLK, CF, ZF, SF, OF, ALU_OUT);
```

```
49          #(`CLK_PER)
50          B <= 8'b000_1000;
51          $strobe(`STRING, $time, OPCODE, A, B, EN, OE, CLK, CF, ZF, SF, OF, ALU_OUT);
52          #(`CLK_PER)
53          OPCODE <= `ALU_ADD_TB;
54          $strobe(`STRING, $time, OPCODE, A, B, EN, OE, CLK, CF, ZF, SF, OF, ALU_OUT);
55          #(`CLK_PER)
56          A <= 8'b1000_0000;
57          B <= 8'b1000_1001;
58          $strobe(`STRING, $time, OPCODE, A, B, EN, OE, CLK, CF, ZF, SF, OF, ALU_OUT);
59          A <= 8'b1000_1010;
60          B <= 8'b1000_1001;
61          OPCODE <= `ALU_SUB_TB;
62          #(`CLK_PER)
63          OPCODE <= `ALU_ADD_TB;
64          A <= 8'b1100_1010;
65          B <= 8'b1100_1001;
66          $strobe(`STRING, $time, OPCODE, A, B, EN, OE, CLK, CF, ZF, SF, OF, ALU_OUT);
67          #(`CLK_PER)
68          A <= 8'b1000_0001;
69          B <= 8'b1100_1001;
70          $strobe(`STRING, $time, OPCODE, A, B, EN, OE, CLK, CF, ZF, SF, OF, ALU_OUT);
71          #(`CLK_PER)
72          A <= 8'b1000_0001;
73          B <= 8'b0011_0111;
74          OPCODE <= `ALU_SUB_TB;
75          $strobe(`STRING, $time, OPCODE, A, B, EN, OE, CLK, CF, ZF, SF, OF, ALU_OUT);
76          #(`CLK_PER)
77          $strobe(`STRING, $time, OPCODE, A, B, EN, OE, CLK, CF, ZF, SF, OF, ALU_OUT);
78          #(`CLK_PER)
79          OE <= 0;
80          $strobe(`STRING, $time, OPCODE, A, B, EN, OE, CLK, CF, ZF, SF, OF, ALU_OUT);
81          #(`CLK_PER)
82          EN <= 0;
83          $strobe(`STRING, $time, OPCODE, A, B, EN, OE, CLK, CF, ZF, SF, OF, ALU_OUT);
84          #(`CLK_PER)
85          $strobe(`STRING, $time, OPCODE, A, B, EN, OE, CLK, CF, ZF, SF, OF, ALU_OUT);
86          #(`CLK_PER)
87          OE <= 1;
88          EN <= 1;
89          OPCODE <= `ALU_ADD_TB;
90          $strobe(`STRING, $time, OPCODE, A, B, EN, OE, CLK, CF, ZF, SF, OF, ALU_OUT);
91          #(`CLK_PER)
92          A <= 8'b1100_0001;
93          B <= 8'b0111_0111;
94          $strobe(`STRING, $time, OPCODE, A, B, EN, OE, CLK, CF, ZF, SF, OF, ALU_OUT);
95          #(`CLK_PER)
96          A <= 8'b0110_1001;
97          B <= 8'b0101_0111;
98          $strobe(`STRING, $time, OPCODE, A, B, EN, OE, CLK, CF, ZF, SF, OF, ALU_OUT);
99          #(`CLK_PER)
100         $strobe(`STRING, $time, OPCODE, A, B, EN, OE, CLK, CF, ZF, SF, OF, ALU_OUT);
101         #(`CLK_PER)
102         $finish;
103     end
104
105 endmodule
```

Fig. 5 lab8_tb.v

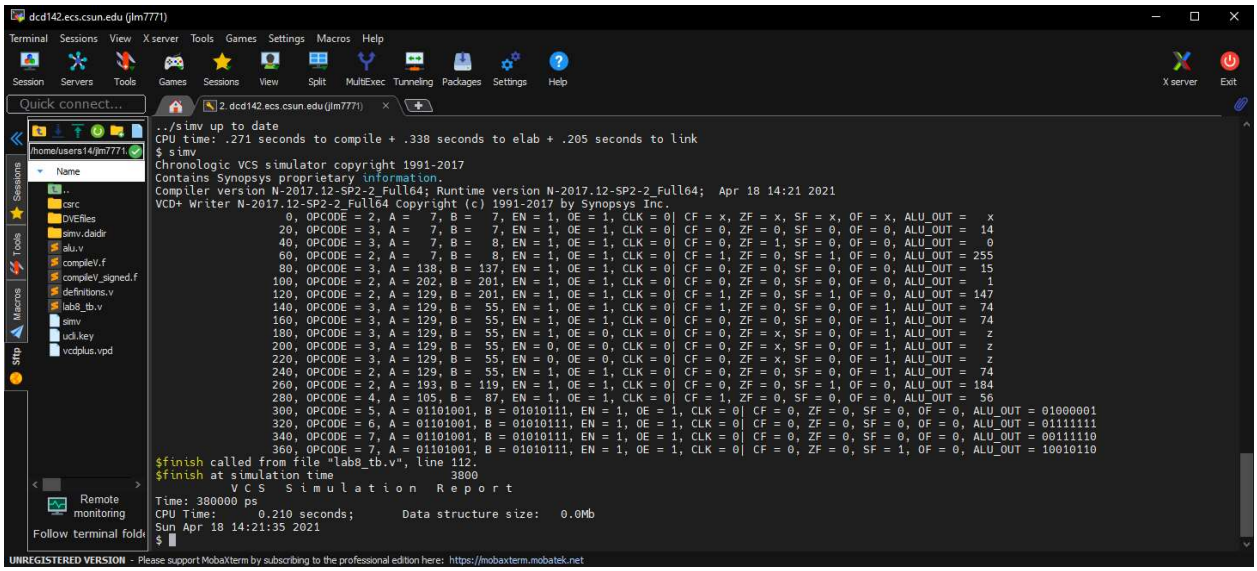

Fig. 6 simv unsigned output

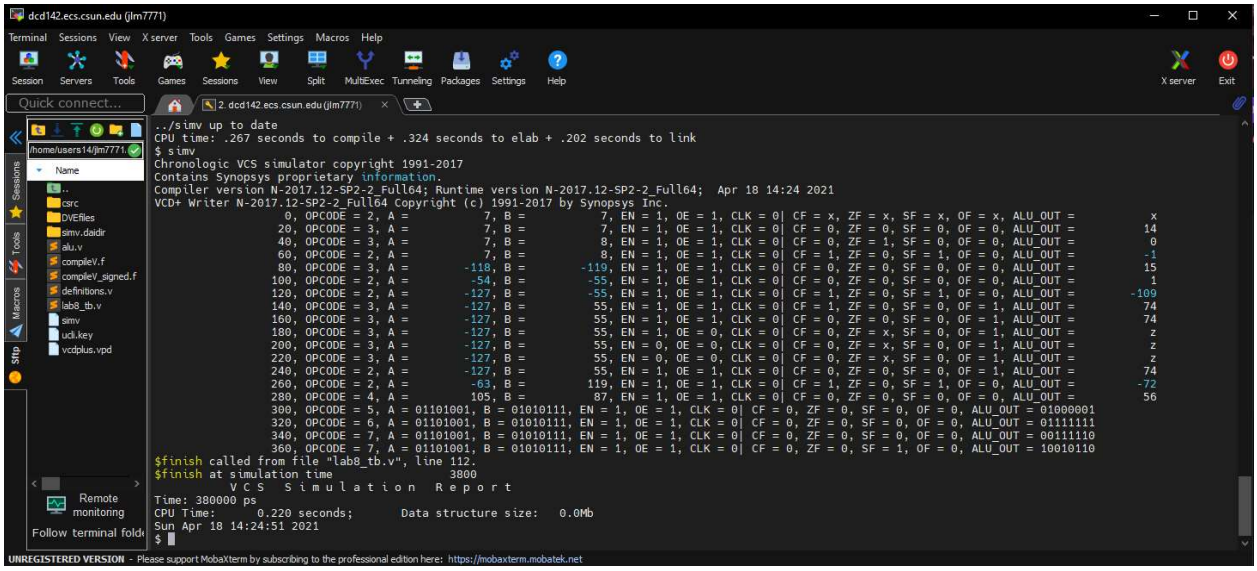

Fig. 7 simv signed output

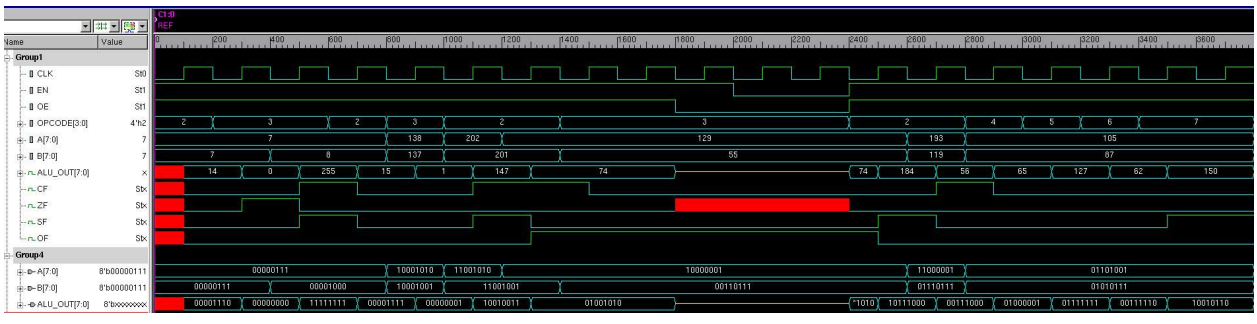

Fig. 8 waveforms unsigned

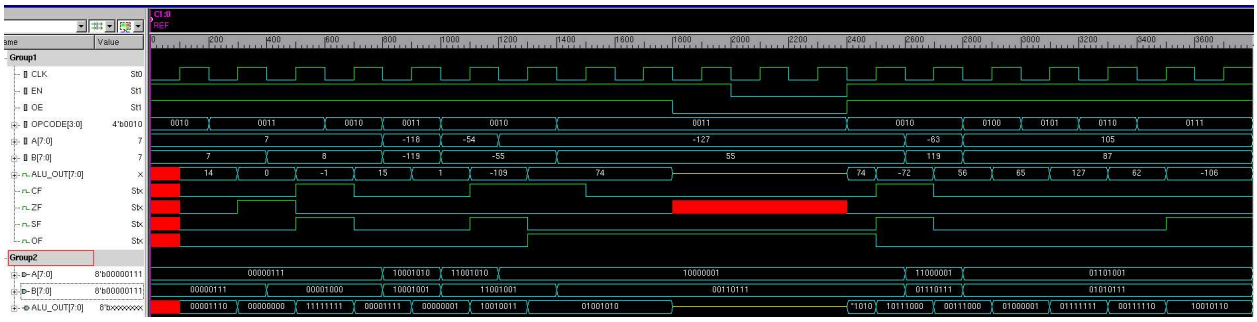

Fig. 9 waveforms signed

Analysis

The only difference between the signed and unsigned versions of the Alu and testbench module is that the inputs and outputs were declared as either signed or unsigned. The rest of the logic remained the same. Subtraction was made not using the verilog "-" operator but rather taking the 2's complement of the second number and adding it to the first number. That way we are able to set our overflow flag correctly. So the outputs shown here in **Figures 6-9** should be the same if the radix was changed to the same format but for the sake of verifying our values they were shown with their corresponding radix.

Flags that apply to both unsigned and signed parts are the zero flag(ZF) and the negative flag (SF). The ZF will be 1'b1 when all of the bits in ALU_OUT are 1'b0. For the SF, it will be 1'b1 when the MSB of the ALU_OUT is 1'b1.

For the unsigned part of the ALU the carry flag (CF) will be set to 1'b1 whenever adding two numbers results in a carry out bit or whenever subtracting, the subtractor is higher than the number you are subtracting. In **Fig. 6 & 8** we can see various examples of when this happens. One of these examples is when A = 7, A = 8, OPCODE = 4'b0011 which means it will subtract. After the next clock cycle the CF = 1 and ALU_OUT = 255. This means that subtracting when B>A CF will flag as a carry. Another example is when A = 192, B = 119, and OPCODE = 4'b0010 which means it will add. After the next clock cycle the CF = 1 and ALU_OUT = 56. This means that adding two numbers that would result in a number bigger than 8 bits our CF will flag. An example that the adder is working properly is when A = 7, B = 7, and OPCODE = 4'b010. In the next clock cycle, CF = 0 and ALU_OUT = 14. An example that the subtractor is working properly is when A = 7, B = 7, and OPCODE = 4'b011. In the next clock cycle, CF = 0, ZF = 0, and ALU_OUT = 0

For the signed part of the ALU the overflow flag (OF) will be set to 1'b1 whenever adding two negative or positive numbers and getting the opposite sign. The same works for subtracting except you look at the first number and the second numbers 2's complement. We can use **Fig. 7 & 9** to verify that the OF, adding, and subtracting are working properly for the signed numbers. One example of OF is when A = -127, B = -55, and OPCODE = 4'b0010. In the next clock cycle we can see that OF = 1 and ALU_OUT = 74. Another example of OF is when our A = -127, B= 55, and OPCODE = 4'b0011. In the next clock cycle we can see that OF = 1 and ALU_OUT = 74. Which indicates that our overflow flag for both addition and subtraction works. An example of addition is when A = 7, B = 7, and OPCODE = 4'b0010. In the next clock cycle we can see that OF = 0 and ALU_OUT = 14. An example of subtraction is when A = 7, B = 8, and OPCODE = 4'b0011. In the next clock cycle we can see that OF = 0, SF = 1, and

ALU_OUT = 14. Which also proves that the SF flag works properly as well.

For the other operations they are shown in **Figures 6-9** and are the same. For AND, A = 01101001, B = 01010111, and OPCODE = 4'b0100. In the next clock cycle our output is 01000001, which is correct. For OR, A = 01101001, B = 01010111, and OPCODE = 4'b0101. In the next clock cycle our output is 01111111, which is correct. For XOR, A = 01101001, B = 01010111, and OPCODE = 4'b0110. In the next clock cycle our output is 00111110, which is correct. For NOT, A = 01101001 and OPCODE = 4'b0111. In the next clock cycle our output is 10010110, which is correct.

Conclusion

In conclusion, an ALU module was made to compute an operation based on the given OPCODE. We were to test for both unsigned and signed values and besides the output format the ALU should work the same regardless of the inputs being signed or unsigned. I was able to verify that my module design was working through a test bench. I learned how to design an ALU and how the flags work.

Academic Dishonesty

Submitting any report that is not entirely your own work is a form of academic dishonesty and will not be tolerated. Each and every lab report must include the following statement, signed and dated by the student. Lab reports without the statement will be summarily rejected.

I hereby attest that this lab report is entirely my own work. I have not copied either code or text from anyone, nor have I allowed or will I allow anyone to copy my work.

Name (printed) Jose L Martinez

Name (signed) Jose Martinez Date 4/18/2021