Armondo Thomas

Partner: Grant Swenson

March 28, 2017

EE 310

Lab 5 Report

Introduction

This lab we began constructing the ALU. It builds off of a previous lab. All functionality is described by the opcodes and block diagrams.

Activity 1

1. (5pts) In the left box below, list the instructions that produce a data result on the Z bus of the ALU. In the right box, list all of the remaining instructions.

Instructions producing results on Z bus:	Instructions where Z bus does not matter:
OpCodes: x01,x02, x04-x0F	X00,x03,x10-x14

2. (5pts) There are some instructions that do not require a specific output from the ALU. You are free to choose the value of the ALU output (the Z bus) in these cases. What is it and why did you select it?

We are choosing to use whatever the existing value of Z is. We are choosing this method because it is the most straightforward and won't require any changes to the Z bus.

3. (5pts) What are the input and output signals of the ALU entity and how many bits are they?

Inputs:	Outputs:
MDR – 8 Bits	Z – 8 Bits
AC – 8 Bits	STORE_MEM – 1 Bit
Value -8 Bits	LOAD_PC – 1 Bit

4. As a class, we will discuss how to choose four good test cases to perform as thorough a test as we can. Once we decide these test cases together, you must determine the correct outputs (**Z**, **STORE_MEM**, **LOAD_PC**) for each instruction and each test case.
   - Use hexadecimal.
   - The test cases should verify the correct operation of each opcode as well as illustrate that connections are correct (correct bus numbering).
   - All inputs and all outputs should be '0' and '1' in at least one test case for each instruction.
   - For the ADDI instruction, test the carry from least significant bit (LSB) to most significant bit (MSB).
   - For the ADDI, SUBTI, and LOADI instructions, test that the *value* input works.
   - Some instructions do not need a specific **Z** output value because it is not used. Specify your **Z** outputs for these cases the same way you answered question 2 above.

   Note: Four test cases is a ridiculously low number. Many more would be included in a real design.

(15pts) In each box, the format is: **Z bus output** (in hex), **STORE_MEM**, **LOAD_PC**

Opcode, Instruction mnemonic	**Test case #1** **MDR: 00** **AC: 55** **address/value:** **36**	**Test case #2** **MDR: 55** **AC: AA** **address/value:** **AB**	**Test case #3** **MDR: AA** **AC: 00** **address/value:** **49**	**Test case #4** **MDR: FF** **AC: 9E** **address/value:** **C2**
00 NOP	xx, 0, 0	xx, 0, 0	xx, 0, 0	xx, 0, 0
01 LOAD	x00, 0, 0	x55, 0, 0	xAA, 0, 0	xFF, 0, 0
02 LOADI	x36, 0, 0	xAB, 0, 0	x49, 0, 0	xC2, 0, 0
03 STORE	xx, 1, 0	xAB, 1, 0	x49, 1, 0	xC2, 1, 0
04 CLR	x00, 0, 0	x00, 0, 0	x00, 0, 0	x00, 0, 0
05 ADD	x55,0,0	xFF, 0 0	xAA, 0, 0	x9D, 0, 0
06 ADDI	x8B, 0, 0	x55, 0, 0	x49, 0, 0	x60, 0, 0
07 SUBT	x55, 0, 0	x55, 0, 0	x56, 0, 0	x9F,0 ,0
08 SUBTI	x1F, 0, 0	xFF, 0, 0	xB7, 0, 0	xDC, 0, 0
09 NEG	x00, 0, 0	xAB, 0, 0	x56, 0, 0	x01, 0 , 0
0A NOT	xFF, 0, 0	xAA, 0, 0	x55, 0, 0	x00, 0, 0
0B AND	x00, 0, 0	x00, 0, 0	x00, 0, 0	x9E, 0, 0
0C OR	x55, 0, 0	x55, 0, 0	xAA, 0, 0	xFF, 0, 0
0D XOR	x55, 0, 0	xFF, 0, 0	xAA, 0, 0	x61, 0, 0
0E SHL	xA8, 0, 0	x50, 0, 0	x00, 0, 0	x78, 0, 0
0F SHR	x01, 0, 0	x15, 0, 0	x00, 0, 0	x27, 0, 0

10 JUMP	xx, 0, 1	xx, 0, 1	xx, 0, 1	xx, 0, 1
11 JNEG	xx, 0, 0	xx, 0, 1	xx, 0, 0	xx, 0, 1
12 JPOSZ	xx, 0, 1	xx, 0, 0	xx, 0, 1	xx, 0, 0
13 JZERO	xx, 0, 0	xx, 0, 0	xx, 0, 1	xx, 0, 0
14 JNZER	xx, 0 , 1	xx, 0 , 1	xx, 0 , 0	xx, 0 , 1

Activity 2

Source Code

VHDL Files

alu.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library altera_mf;
use altera_mf.altera_mf_components.all;



entity alu is
	port (
	-- put port list here, use type SIGNED for the data busses
		opcode,value_in,mdr, ac:in signed(7 downto 0);
		z: out signed(7 downto 0);
		store_mem, load_pc: out std_logic
	);
end alu;

architecture behav of alu is
	SIGNAL temp_z: signed(7 downto 0);
	begin
		z <= temp_z;
		process(opcode, value_in, mdr, ac)  -- include necessary signals in sensitivity list
			begin
			-- put your code here!
```

```
case opcode is
        when x"00" =>
                store_mem <= '0';
                load_pc <= '0';

        when x"01" =>
                temp_z <= mdr;

        when x"02" =>
                temp_z <= value_in;

        when x"03" =>
                store_mem <= '1';

        when x"04" =>
                temp_z <= x"00";

        when x"05" =>
                temp_z <= ac + mdr;

        when x"06" =>
                temp_z <= ac+value_in;

        when x"07" =>
                temp_z <=  ac - mdr;

        when x"08" =>
                temp_z <=  ac - value_in;

        when x"09" =>
                temp_z <= x"00" - mdr;

        when x"0A"=>
```

```
            temp_z <= not(mdr);

when x"0B" =>
            temp_z <= (ac and mdr);

when x"0C"=>
            temp_z <= (ac or mdr);

when x"0D"=>
            temp_z <= (ac xor mdr);

when x"0E" =>
            temp_z <= signed(std_logic_vector(ac sll to_integer(value_in(2 downto 0))));

when x"0F" =>
            temp_z <= signed(std_logic_vector(ac srl to_integer(value_in(2 downto 0))));

when x"10" =>
            load_pc <= '1';

when x"11" =>
            if (ac< x"00") then
                        load_pc <= '1';
            end if;

when x"12" =>
            if (ac >= x"00") then
                        load_pc <= '1';
            end if;

when x"13" =>
            if (ac = x"00") then
                        load_pc <= '1';
            end if;
```

```
                when x"14" =>
                        if (ac /= x"00") then
                                load_pc <= '1';
                        end if;
                --do nothing
                when others => temp_z <= temp_z;

                end case;
        end process;
end behav;
```

alu_switch.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library altera_mf;
use altera_mf.altera_mf_components.all;

entity alu_switch is
        port(
                key_opcode, key_ac, key_mdr, key_value: in std_logic;
                data_input: in std_logic_vector(7 downto 0);
                out_opcode, out_ac, out_mdr, out_value: out std_logic_vector(7 downto 0)
        );
end alu_switch;

architecture behav of alu_switch is
        begin
                process(key_opcode, key_ac, key_mdr, key_value)
                        begin

                        if key_opcode <= '0' then
                                out_opcode <= data_input;
```

```
                elsif key_ac <= '0' then
                        out_ac <= data_input;

                elsif key_mdr <= '0' then
                        out_mdr <= data_input;

                elsif key_value <= '0' then
                        out_value <= data_input;
                end if;
        end process;
end behav;
```

DO Files

alu_sim.txt

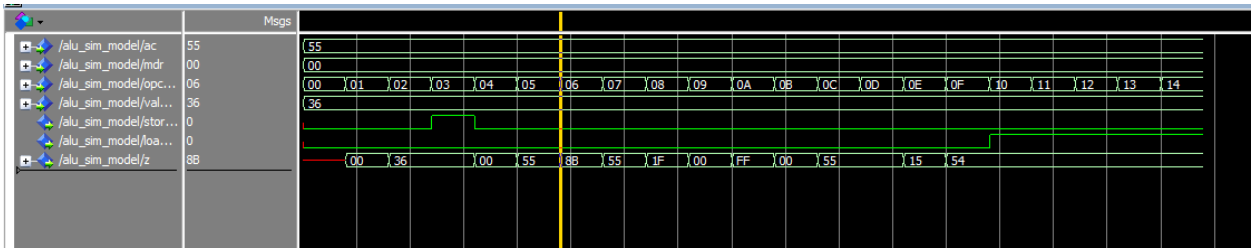
```
add wave -in *
add wave -out *
restart -f

force mdr x"00"
force ac x"55"
force value_in x"36"

force opcode x"00"
run 40ns
force opcode x"01"
run 40ns
force opcode x"02"
run 40ns
force opcode x"03"
run 40ns
force opcode x"04"
run 40ns
force opcode x"05"
run 40ns
force opcode x"06"
run 40ns
force opcode x"07"
run 40ns
force opcode x"08"
run 40ns
force opcode x"09"
run 40ns
force opcode x"0A"
run 40ns
force opcode x"0B"
run 40ns
force opcode x"0C"
run 40ns
force opcode x"0D"
run 40ns
force opcode x"0E"
run 40ns
```

```
force opcode x"0F"
run 40ns
force opcode x"10"
run 40ns
force opcode x"11"
run 40ns
force opcode x"12"
run 40ns
force opcode x"13"
run 40ns
force opcode x"14"
run 40ns
```

Screenshots



Conclusion

Overall, this was a great lab. Difficult to setup and was very time consuming. However, once the mistakes were realized, it put into perspective how simple this lab should have been.