Armondo Thomas

Partner: Grant Swenson

March 31, 2017

EE 310

# Lab 5 Report

## Introduction

In this lab, we began constructing the ALU. It builds from a previous lab. All functionality is described by the opcodes and block diagrams that were included in the lab description.

### 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:  OpCodes: x01,x02, x04-x0F | Instructions where Z bus does not matter:  X00,x03,x10-x14 |

1. (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.

1. (5pts) What are the input and output signals of the ALU entity and how many bits are they?

|  |  |
| --- | --- |
| Inputs:  MDR – 8 Bits  AC – 8 Bits  Value -8 Bits | Outputs:  Z – 8 Bits  STORE\_MEM – 1 Bit  LOAD\_PC – 1 Bit |

1. 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 File

**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 <= ac sll to\_integer(unsigned(value\_in(2 downto 0)));

when x"0F" =>

temp\_z <= ac srl to\_integer(unsigned(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 File

**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

### Screenshot

Though this screenshot is incorrect for opcode x0F (logical right shift) was reprogrammed and the screenshot not updated. The lab TA Michael was able to see a fully working demo for both board and simulation.

### Conclusion

This lab was very glitchy when it came to logical left and right shifting. I did not like the overhead and not knowing for sure what types each operation accepts. Overall, this lab was overcomplicated by the overhead about shifting. If that could be minimized, then this lab would have been much smoother.