Aidan Lambrecht

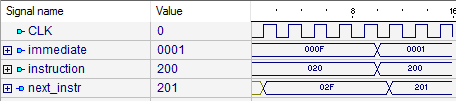
ELEC 5200

Dr. Harris

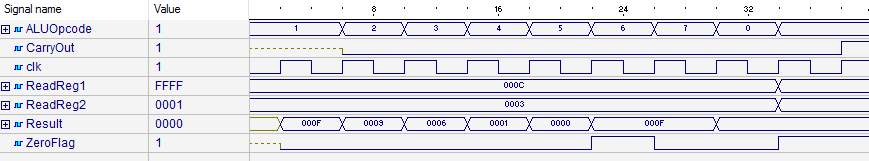
October 25, 2019

CPU Project Part 3

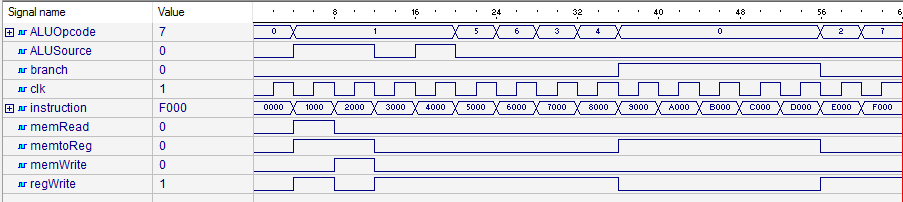
Nine components were tested for Part 3 of the design project, the first being the Address Adder. This component takes an address and adds an immediate to it, outputting the result in the signal 'next\_instr.' Code Base 1 in Appendix A displays the VHDL file for this component. This component was tested by inspection, and Figure 1 shows the component functioning with two different values for both the immediate and initial instruction. The addition of these values is successfully completed on the rising clock edge.



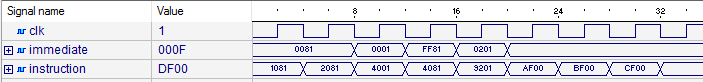
**Figure 1.** Address Adder Simulation



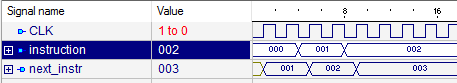
**Figure 2.** ALU Simulation



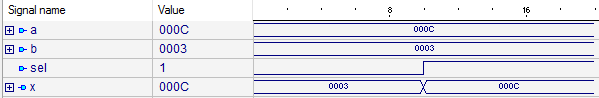
**Figure 3.** Control Unit Simulation



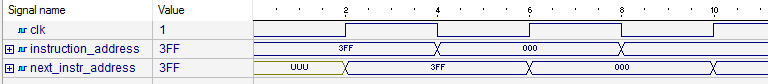
**Figure 4.** Immediate Generator/Sign Extender Simulation



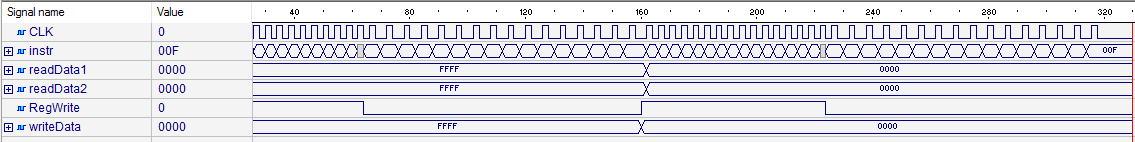
**Figure 5.** Instruction Adder Simulation



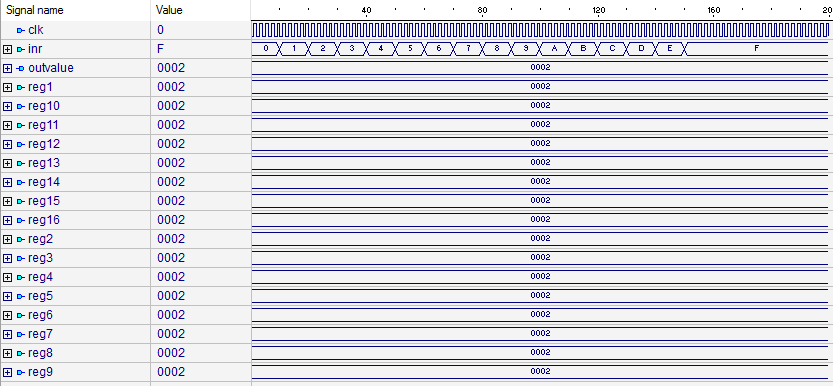
**Figure 6.** 2-1 Multiplexer Simulation



**Figure 7.** Program Counter Simulation



**Figure 8.** Register File Simulation



**Figure 9.** Testing Multiplexer Simulation

**Appendix A: VHDL Components**

**Code Base 1: Address Adder**

----------------------------------------------------------------------------------

-- Engineer: Aidan Lambrecht

--

-- Create Date: 10/21/2019

-- Module Name: Adder (Immediates)

-- Project Name: CPU Design Part 3

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.numeric\_std.all;

entity AddressAdder is

port (

CLK : in std\_logic;

instruction : in std\_logic\_vector(9 downto 0);

immediate : in std\_logic\_vector(15 downto 0);

next\_instr : out std\_logic\_vector(9 downto 0)

);

end AddressAdder;

architecture behav of AddressAdder is

begin

--figure out how to add / if we add positive and negative values through the immediate

process(CLK)

begin

if rising\_edge(CLK) then

next\_instr <= std\_logic\_vector(unsigned(instruction) + unsigned(immediate(9 downto 0)));

end if;

end process;

end behav; ----------------------------------------------------------------------------------

-- Engineer: Aidan Lambrecht

--

-- Create Date: 10/21/2019

-- Module Name: Adder (Immediates)

-- Project Name: CPU Design Part 3

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.numeric\_std.all;

entity AddressAdder is

port (

CLK : in std\_logic;

instruction : in std\_logic\_vector(9 downto 0);

immediate : in std\_logic\_vector(15 downto 0);

next\_instr : out std\_logic\_vector(9 downto 0)

);

end AddressAdder;

architecture behav of AddressAdder is

begin

--figure out how to add / if we add positive and negative values through the immediate

process(CLK)

begin

if rising\_edge(CLK) then

next\_instr <= std\_logic\_vector(unsigned(instruction) + unsigned(immediate(9 downto 0)));

end if;

end process;

end behav;

**Code Base 2: ALU**

----------------------------------------------------------------------------------

-- Engineer: Aidan Lambrecht

--

-- Create Date: 10/21/2019

-- Module Name: ALU

-- Project Name: CPU Design Part 3

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

entity ALU is

port (

CLK : in STD\_LOGIC;

ALUOpcode: in STD\_LOGIC\_VECTOR(2 downto 0);

ReadReg1 : in STD\_LOGIC\_VECTOR(15 downto 0);

ReadReg2 : in STD\_LOGIC\_VECTOR(15 downto 0);

CarryOut : out STD\_LOGIC;

ZeroFlag : out STD\_LOGIC;

Result : out STD\_LOGIC\_VECTOR(15 downto 0)

);

end ALU;

architecture behav of ALU is

signal r\_sig : STD\_LOGIC\_VECTOR(16 downto 0);

begin

operation : process (CLK)

begin

if rising\_edge(CLK) then

case ALUOpcode is

when "000" => --No operation

ZeroFlag <= '0';

r\_sig <= (others => '0');

when "001" => --Add

r\_sig <= ('0' & ReadReg1) + ('0' & ReadReg2);

when "010" => --Subtract

r\_sig <= ('0' & ReadReg1) - ('0' & ReadReg2);

when "011" => --Shift Left

r\_sig <= '0' & ReadReg2(14 downto 0) & '0';

when "100" => --Shift Right

r\_sig <= "00" & ReadReg2(15 downto 1);

when "101" => --AND

r\_sig <= ('0' & ReadReg1) and ('0' & ReadReg2);

when "110" => --OR

r\_sig <= ('0' & ReadReg1) or ('0' & ReadReg2);

when "111" => --XOR

r\_sig <= ('0' & ReadReg1) xor ('0' & ReadReg2);

when others =>

r\_sig <= "00000000000000000";

end case;

if (r\_sig(15 downto 0) = "0000000000000000") then

ZeroFlag <= '1';

else

ZeroFlag <= '0';

end if;

CarryOut <= r\_sig(16);

end if;

end process;

--assignment of outputs

Result <= r\_sig(15 downto 0);

end behav;

**Code Base 3: Control Unit**

----------------------------------------------------------------------------------

-- Engineer: Aidan Lambrecht

--

-- Create Date: 10/21/2019

-- Module Name: Control Unit

-- Project Name: CPU Design Part 3

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.numeric\_std.all;

entity ControlUnit is

port (

CLK : in std\_logic;

instruction : in std\_logic\_vector(15 downto 0);

branch : out std\_logic;

memRead : out std\_logic;

memtoReg : out std\_logic;

memWrite : out std\_logic;

ALUSource : out std\_logic;

ALUOpcode : out std\_logic\_vector(2 downto 0);

regWrite : out std\_logic

);

end ControlUnit;

architecture behav of ControlUnit is

begin

process(CLK)

begin

if instruction(15 downto 12) = "0000" then --halt

branch <= '0';

memRead <= '0';

memtoReg <= '0';

memWrite <= '0';

ALUSource <= '0';

ALUOpcode <= "000";

regWrite <= '0';

elsif instruction(15 downto 12) = "0001" then --load

branch <= '0';

memRead <= '1';

memtoReg <= '1';

memWrite <= '0';

ALUSource <= '1';

ALUOpcode <= "001";

regWrite <= '1';

elsif instruction(15 downto 12) = "0010" then --store

branch <= '0';

memRead <= '0';

memtoReg <= '1';

memWrite <= '1';

ALUSource <= '1';

ALUOpcode <= "001";

regWrite <= '0';

elsif instruction(15 downto 12) = "0011" then --ADDR

branch <= '0';

memRead <= '0';

memtoReg <= '0';

memWrite <= '0';

ALUSource <= '0';

ALUOpcode <= "001";

regWrite <= '1';

elsif instruction(15 downto 12) = "0100" then --ADDI

branch <= '0';

memRead <= '0';

memtoReg <= '0';

memWrite <= '0';

ALUSource <= '1';

ALUOpcode <= "001";

regWrite <= '1';

elsif instruction(15 downto 12) = "0101" then --AND

branch <= '0';

memRead <= '0';

memtoReg <= '0';

memWrite <= '0';

ALUSource <= '0';

ALUOpcode <= "101";

regWrite <= '1';

elsif instruction(15 downto 12) = "0110" then --OR

branch <= '0';

memRead <= '0';

memtoReg <= '0';

memWrite <= '0';

ALUSource <= '0';

ALUOpcode <= "110";

regWrite <= '1';

elsif instruction(15 downto 12) = "0111" then --SHL

branch <= '0';

memRead <= '0';

memtoReg <= '0';

memWrite <= '0';

ALUSource <= '0';

ALUOpcode <= "011";

regWrite <= '1';

elsif instruction(15 downto 12) = "1000" then --SHR

branch <= '0';

memRead <= '0';

memtoReg <= '0';

memWrite <= '0';

ALUSource <= '0';

ALUOpcode <= "100";

regWrite <= '1';

elsif instruction(15 downto 12) = "1001" then --JUMP

branch <= '1';

memRead <= '0';

memtoReg <= '1';

memWrite <= '0';

ALUSource <= '0';

ALUOpcode <= "000";

regWrite <= '0';

elsif instruction(15 downto 12) = "1010" then --BRE

branch <= '1';

memRead <= '0';

memtoReg <= '1';

memWrite <= '0';

ALUSource <= '0';

ALUOpcode <= "000";

regWrite <= '0';

elsif instruction(15 downto 12) = "1011" then --BRNE

branch <= '1';

memRead <= '0';

memtoReg <= '1';

memWrite <= '0';

ALUSource <= '0';

ALUOpcode <= "000";

regWrite <= '0';

elsif instruction(15 downto 12) = "1100" then --BRLT

branch <= '1';

memRead <= '0';

memtoReg <= '1';

memWrite <= '0';

ALUSource <= '0';

ALUOpcode <= "000";

regWrite <= '0';

elsif instruction(15 downto 12) = "1101" then --BRGE

branch <= '1';

memRead <= '0';

memtoReg <= '1';

memWrite <= '0';

ALUSource <= '0';

ALUOpcode <= "000";

regWrite <= '0';

elsif instruction(15 downto 12) = "1110" then --SUBR

branch <= '0';

memRead <= '0';

memtoReg <= '0';

memWrite <= '0';

ALUSource <= '0';

ALUOpcode <= "010";

regWrite <= '1';

elsif instruction(15 downto 12) = "1111" then --XOR

branch <= '0';

memRead <= '0';

memtoReg <= '0';

memWrite <= '0';

ALUSource <= '0';

ALUOpcode <= "111";

regWrite <= '1';

end if;

end process;

end behav;

**Code Base 4: Immediate Generator/Sign Extender**

----------------------------------------------------------------------------------

-- Engineer: Aidan Lambrecht

--

-- Create Date: 10/21/2019

-- Module Name: Immediate Generator

-- Project Name: CPU Design Part 3

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.numeric\_std.all;

entity ImmediateGenerator is

port (

CLK : in std\_logic;

instruction : in std\_logic\_vector(15 downto 0);

immediate : out std\_logic\_vector(15 downto 0)

);

end ImmediateGenerator;

architecture behav of ImmediateGenerator is

begin

process(CLK)

begin

if ((instruction(15 downto 12) = "0001") or (instruction(15 downto 12) = "0010")) then

immediate <= "00000000" & instruction(7 downto 0);

elsif (instruction(15 downto 12) = "0100") then

if instruction(7) = '0' then

immediate <= "00000000" & instruction(7 downto 0);

else

immediate <= "11111111" & instruction(7 downto 0);

end if;

elsif (instruction(15 downto 12) = "1001") then

immediate <= "000000" & instruction(9 downto 0);

elsif ((instruction(15 downto 12) = "1010") or (instruction(15 downto 12) = "1011") or (instruction(15 downto 12) = "1100") or (instruction(15 downto 12) = "1101")) then

immediate <= "000000000000" & instruction(11 downto 8);

end if;

end process;

end behav;

**Code Base 5: Instruction Adder**

----------------------------------------------------------------------------------

-- Engineer: Aidan Lambrecht

--

-- Create Date: 10/21/2019

-- Module Name: Adder (Instructions)

-- Project Name: CPU Design Part 3

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.numeric\_std.all;

entity InstructionAdder is

port (

CLK : in std\_logic;

instruction : in std\_logic\_vector(9 downto 0);

next\_instr : out std\_logic\_vector(9 downto 0)

);

end InstructionAdder;

architecture behav of InstructionAdder is

begin

process(CLK)

begin

if rising\_edge(CLK) then

next\_instr <= std\_logic\_vector(unsigned(instruction) + 1);

end if;

end process;

end behav;

**Code Base 6: 2-1 Multiplexer**

----------------------------------------------------------------------------------

-- Engineer: Aidan Lambrecht

--

-- Create Date: 10/21/2019

-- Module Name: Multiplexer 2-1

-- Project Name: CPU Design Part 3

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

entity mux2 is

generic ( N : integer := 16);

port (

sel : in std\_logic;

a : in std\_logic\_vector(N-1 downto 0);

b : in std\_logic\_vector(N-1 downto 0);

x : out std\_logic\_vector(N-1 downto 0)

);

end mux2;

architecture behav of mux2 is

begin

x <= a when (sel = '1') else b;

end behav;

**Code Base 7: Program Counter**

----------------------------------------------------------------------------------

-- Engineer: Aidan Lambrecht

--

-- Create Date: 10/21/2019

-- Module Name: Program Counter

-- Project Name: CPU Design Part 3

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

entity PC is

port (

clk : in std\_logic;

instruction\_address : in std\_logic\_vector(9 downto 0);

next\_instr\_address : out std\_logic\_vector(9 downto 0)

);

end PC;

architecture behav of PC is

begin

process(clk)

begin

if rising\_edge(clk) then

next\_instr\_address <= instruction\_address;

end if;

end process;

end behav;

**Code Base 8: Register File**

----------------------------------------------------------------------------------

-- Engineer: Aidan Lambrecht

--

-- Create Date: 10/21/2019

-- Module Name: Register File

-- Project Name: CPU Design Part 3

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.numeric\_std.all;

entity registerFile is

port (

CLK : in STD\_LOGIC;

instr : in STD\_LOGIC\_VECTOR(11 downto 0);

RegWrite : in STD\_LOGIC;

writeData: in STD\_LOGIC\_VECTOR(15 downto 0);

readData1: out STD\_LOGIC\_VECTOR(15 downto 0);

readData2: out STD\_LOGIC\_VECTOR(15 downto 0)

);

end registerFile;

architecture behav of registerFile is

--design type to facilitate registers

type regArray is array(15 downto 0) of STD\_LOGIC\_VECTOR(15 downto 0);

signal registers : regArray := (others => (others => '0')); --initialize all values to 0

begin

--handles write function

registerFile\_main : process(CLK)

begin

if rising\_edge(CLK) then

if (RegWrite = '1') then

registers(to\_integer(unsigned(instr(3 downto 0)))) <= writeData;

end if;

end if;

end process;

--read behavior

readData1 <= registers(to\_integer(unsigned(instr(7 downto 4))));

readData2 <= registers(to\_integer(unsigned(instr(11 downto 8))));

end behav;

**Code Base 9: Testing Multiplexer**

----------------------------------------------------------------------------------

-- Engineer: Aidan Lambrecht

--

-- Create Date: 10/21/2019

-- Module Name: testingMux

-- Project Name: CPU Design Part 3

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

entity testingMux is

generic ( N : integer := 16);

port (

clk : in std\_logic;

inr : in std\_logic\_vector(3 downto 0);

reg1 : in std\_logic\_vector(N-1 downto 0);

reg2 : in std\_logic\_vector(N-1 downto 0);

reg3 : in std\_logic\_vector(N-1 downto 0);

reg4 : in std\_logic\_vector(N-1 downto 0);

reg5 : in std\_logic\_vector(N-1 downto 0);

reg6 : in std\_logic\_vector(N-1 downto 0);

reg7 : in std\_logic\_vector(N-1 downto 0);

reg8 : in std\_logic\_vector(N-1 downto 0);

reg9 : in std\_logic\_vector(N-1 downto 0);

reg10 : in std\_logic\_vector(N-1 downto 0);

reg11 : in std\_logic\_vector(N-1 downto 0);

reg12 : in std\_logic\_vector(N-1 downto 0);

reg13 : in std\_logic\_vector(N-1 downto 0);

reg14 : in std\_logic\_vector(N-1 downto 0);

reg15 : in std\_logic\_vector(N-1 downto 0);

reg16 : in std\_logic\_vector(N-1 downto 0);

outvalue : out std\_logic\_vector(N-1 downto 0)

);

end testingMux;

architecture behav of testingMux is

begin

process(clk)

begin

case inr is

when "0000" => outvalue <= reg1;

when "0001" => outvalue <= reg2;

when "0010" => outvalue <= reg3;

when "0011" => outvalue <= reg4;

when "0100" => outvalue <= reg5;

when "0101" => outvalue <= reg6;

when "0110" => outvalue <= reg7;

when "0111" => outvalue <= reg8;

when "1000" => outvalue <= reg9;

when "1001" => outvalue <= reg10;

when "1010" => outvalue <= reg11;

when "1011" => outvalue <= reg12;

when "1100" => outvalue <= reg13;

when "1101" => outvalue <= reg14;

when "1110" => outvalue <= reg15;

when "1111" => outvalue <= reg16;

when others => outvalue <= (others => '0');

end case;

end process;

end behav;

**Appendix B: VHDL Testbenches**

**Code Base 10: ALU Testbench**

----------------------------------------------------------------------------------

-- Engineer: Aidan Lambrecht

--

-- Create Date: 10/21/2019

-- Module Name: ALU Testbench

-- Project Name: CPU Design Part 3

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

use IEEE.NUMERIC\_STD.ALL;

library work;

use work.all;

entity ALUTestbench is

end ALUTestbench;

architecture test of ALUTestbench is

component ALU is

port (

CLK : in STD\_LOGIC;

ALUOpcode: in STD\_LOGIC\_VECTOR(2 downto 0);

ReadReg1 : in STD\_LOGIC\_VECTOR(15 downto 0);

ReadReg2 : in STD\_LOGIC\_VECTOR(15 downto 0);

CarryOut : out STD\_LOGIC;

ZeroFlag : out STD\_LOGIC;

Result : out STD\_LOGIC\_VECTOR(15 downto 0)

);

end component;

signal clk : std\_logic;

signal ALUOpcode : std\_logic\_vector(2 downto 0);

signal ReadReg1 : std\_logic\_vector(15 downto 0);

signal ReadReg2 : std\_logic\_vector(15 downto 0);

signal CarryOut : std\_logic;

signal ZeroFlag : std\_logic;

signal Result : std\_logic\_vector(15 downto 0);

begin

UUT: ALU

port map (

clk => clk,

ALUOpcode => ALUOpcode,

ReadReg1 => ReadReg1,

ReadReg2 => ReadReg2,

CarryOut => CarryOut,

ZeroFlag => ZeroFlag,

Result => Result);

testing: process

begin

--set up inputs to test

ReadReg1 <= "0000000000001100";

ReadReg2 <= "0000000000000011";

clk <= '0';

ALUOpcode <= "001";

wait for 2 ns;

--add

clk <= '1';

wait for 2 ns;

clk <= '0';

assert(Result = "0000000000001111" and ZeroFlag = '0' and CarryOut = '0')

report "add failed"

severity warning;

wait for 2 ns;

--sub

ALUOpcode <= "010";

clk <= '1';

wait for 2 ns;

clk <= '0';

assert(Result = "0000000000001001" and ZeroFlag = '0' and CarryOut = '0')

report "subtract failed"

severity warning;

wait for 2 ns;

--shift left

ALUOpcode <= "011";

clk <= '1';

wait for 2 ns;

clk <= '0';

assert(Result = "0000000000000110" and ZeroFlag = '0' and CarryOut = '0')

report "shift left failed"

severity warning;

wait for 2 ns;

--shift right

ALUOpcode <= "100";

clk <= '1';

wait for 2 ns;

clk <= '0';

assert(Result = "0000000000000001" and ZeroFlag = '0' and CarryOut = '0')

report "shift right failed"

severity warning;

wait for 2 ns;

--and

ALUOpcode <= "101";

clk <= '1';

wait for 2 ns;

clk <= '0';

assert(Result = "0000000000000000" and ZeroFlag = '1' and CarryOut = '0')

report "and failed"

severity warning;

wait for 2 ns;

--or

ALUOpcode <= "110";

clk <= '1';

wait for 2 ns;

clk <= '0';

assert(Result = "0000000000001111" and ZeroFlag = '0' and CarryOut = '0')

report "or failed"

severity warning;

wait for 2 ns;

--xor

ALUOpcode <= "111";

clk <= '1';

wait for 2 ns;

clk <= '0';

assert(Result = "0000000000001111" and ZeroFlag = '0' and CarryOut = '0')

report "xor failed"

severity warning;

wait for 2 ns;

--no op

ALUOpcode <= "000";

clk <= '1';

wait for 2 ns;

clk <= '0';

assert(Result = "000000000000000" and ZeroFlag = '0' and CarryOut = '0')

report "no op failed"

severity warning;

wait for 2 ns;

--add test carry out

ReadReg1 <= "1111111111111111";

ReadReg2 <= "0000000000000001";

ALUOpcode <= "001";

clk <= '1';

wait for 2 ns;

clk <= '0';

assert(Result = "0000000000000000" and ZeroFlag = '1' and CarryOut = '1')

report "carryout failed"

severity warning;

wait for 2 ns;

clk <= '1';

wait;

end process;

end architecture;

**Code Base 11: Control Unit Testbench**

----------------------------------------------------------------------------------

-- Engineer: Aidan Lambrecht

--

-- Create Date: 10/21/2019

-- Module Name: ALU Testbench

-- Project Name: CPU Design Part 3

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

use IEEE.NUMERIC\_STD.ALL;

library work;

use work.all;

entity ControlUnitTestbench is

end ControlUnitTestbench;

architecture test of ControlUnitTestbench is

component ControlUnit is

port (

CLK : in std\_logic;

instruction : in std\_logic\_vector(15 downto 0);

branch : out std\_logic;

memRead : out std\_logic;

memtoReg : out std\_logic;

memWrite : out std\_logic;

ALUSource : out std\_logic;

ALUOpcode : out std\_logic\_vector(2 downto 0);

regWrite : out std\_logic

);

end component;

signal clk : std\_logic;

signal instruction : std\_logic\_vector(15 downto 0);

signal branch : std\_logic;

signal memRead : std\_logic;

signal memtoReg : std\_logic;

signal memWrite : std\_logic;

signal ALUSource : std\_logic;

signal ALUOpcode : std\_logic\_vector(2 downto 0);

signal regWrite : std\_logic;

begin

UUT: ControlUnit

port map (

clk => clk,

instruction => instruction,

branch => branch,

memRead => memRead,

memtoReg => memtoReg,

memWrite => memWrite,

ALUSource => ALUSource,

ALUOpcode => ALUOpcode,

regWrite => regWrite);

testing: process

begin

--halt

instruction <= "0000000000000000";

clk <= '0';

wait for 2 ns;

clk <= '1';

wait for 2 ns;

assert( branch = '0' and memRead = '0' and memtoReg = '0' and memWrite = '0' and ALUSource = '0' and ALUOpcode = "000" and regWrite = '0')

report "halt failed"

severity warning;

--load

instruction <= "0001000000000000";

clk <= '0';

wait for 2 ns;

clk <= '1';

wait for 2 ns;

assert( branch = '0' and memRead = '1' and memtoReg = '1' and memWrite = '0' and ALUSource = '1' and ALUOpcode = "001" and regWrite = '1')

report "load failed"

severity warning;

--store

instruction <= "0010000000000000";

clk <= '0';

wait for 2 ns;

clk <= '1';

wait for 2 ns;

assert( branch = '0' and memRead = '0' and memtoReg = '1' and memWrite = '1' and ALUSource = '1' and ALUOpcode = "001" and regWrite = '0')

report "store failed"

severity warning;

--addr

instruction <= "0011000000000000";

clk <= '0';

wait for 2 ns;

clk <= '1';

wait for 2 ns;

assert( branch = '0' and memRead = '0' and memtoReg = '0' and memWrite = '0' and ALUSource = '0' and ALUOpcode = "001" and regWrite = '1')

report "add failed"

severity warning;

--addi

instruction <= "0100000000000000";

clk <= '0';

wait for 2 ns;

clk <= '1';

wait for 2 ns;

assert( branch = '0' and memRead = '0' and memtoReg = '0' and memWrite = '0' and ALUSource = '1' and ALUOpcode = "001" and regWrite = '1')

report "addi failed"

severity warning;

--and

instruction <= "0101000000000000";

clk <= '0';

wait for 2 ns;

clk <= '1';

wait for 2 ns;

assert( branch = '0' and memRead = '0' and memtoReg = '0' and memWrite = '0' and ALUSource = '0' and ALUOpcode = "101" and regWrite = '1')

report "and failed"

severity warning;

--or

instruction <= "0110000000000000";

clk <= '0';

wait for 2 ns;

clk <= '1';

wait for 2 ns;

assert( branch = '0' and memRead = '0' and memtoReg = '0' and memWrite = '0' and ALUSource = '0' and ALUOpcode = "110" and regWrite = '1')

report "or failed"

severity warning;

--shl

instruction <= "0111000000000000";

clk <= '0';

wait for 2 ns;

clk <= '1';

wait for 2 ns;

assert( branch = '0' and memRead = '0' and memtoReg = '0' and memWrite = '0' and ALUSource = '0' and ALUOpcode = "011" and regWrite = '1')

report "shl failed"

severity warning;

--shr

instruction <= "1000000000000000";

clk <= '0';

wait for 2 ns;

clk <= '1';

wait for 2 ns;

assert( branch = '0' and memRead = '0' and memtoReg = '0' and memWrite = '0' and ALUSource = '0' and ALUOpcode = "100" and regWrite = '1')

report "shr failed"

severity warning;

--jump

instruction <= "1001000000000000";

clk <= '0';

wait for 2 ns;

clk <= '1';

wait for 2 ns;

assert( branch = '1' and memRead = '0' and memtoReg = '1' and memWrite = '0' and ALUSource = '0' and ALUOpcode = "000" and regWrite = '0')

report "jump failed"

severity warning;

--bre

instruction <= "1010000000000000";

clk <= '0';

wait for 2 ns;

clk <= '1';

wait for 2 ns;

assert( branch = '1' and memRead = '0' and memtoReg = '1' and memWrite = '0' and ALUSource = '0' and ALUOpcode = "000" and regWrite = '0')

report "bre failed"

severity warning;

--brne

instruction <= "1011000000000000";

clk <= '0';

wait for 2 ns;

clk <= '1';

wait for 2 ns;

assert( branch = '1' and memRead = '0' and memtoReg = '1' and memWrite = '0' and ALUSource = '0' and ALUOpcode = "000" and regWrite = '0')

report "brne failed"

severity warning;

--brlt

instruction <= "1100000000000000";

clk <= '0';

wait for 2 ns;

clk <= '1';

wait for 2 ns;

assert( branch = '1' and memRead = '0' and memtoReg = '1' and memWrite = '0' and ALUSource = '0' and ALUOpcode = "000" and regWrite = '0')

report "brlt failed"

severity warning;

--brge

instruction <= "1101000000000000";

clk <= '0';

wait for 2 ns;

clk <= '1';

wait for 2 ns;

assert( branch = '1' and memRead = '0' and memtoReg = '1' and memWrite = '0' and ALUSource = '0' and ALUOpcode = "000" and regWrite = '0')

report "brge failed"

severity warning;

--subr

instruction <= "1110000000000000";

clk <= '0';

wait for 2 ns;

clk <= '1';

wait for 2 ns;

assert( branch = '0' and memRead = '0' and memtoReg = '0' and memWrite = '0' and ALUSource = '0' and ALUOpcode = "010" and regWrite = '1')

report "subr failed"

severity warning;

--xor

instruction <= "1111000000000000";

clk <= '0';

wait for 2 ns;

clk <= '1';

wait for 2 ns;

assert( branch = '0' and memRead = '0' and memtoReg = '0' and memWrite = '0' and ALUSource = '0' and ALUOpcode = "111" and regWrite = '1')

report "xor failed"

severity warning;

wait;

end process;

end architecture;

**Code Base 12: Immediate Generator/Sign Extender Testbench**

----------------------------------------------------------------------------------

-- Engineer: Aidan Lambrecht

--

-- Create Date: 10/21/2019

-- Module Name: Immediate Generator Testbench

-- Project Name: CPU Design Part 3

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

use IEEE.NUMERIC\_STD.ALL;

library work;

use work.all;

entity ImmGenTestbench is

end ImmGenTestbench;

architecture test of ImmGenTestbench is

component ImmediateGenerator is

port (

clk : in STD\_LOGIC;

instruction: in std\_logic\_vector(15 downto 0);

immediate : out std\_logic\_vector(15 downto 0)

);

end component;

signal clk : std\_logic;

signal instruction : std\_logic\_vector(15 downto 0);

signal immediate : std\_logic\_vector(15 downto 0);

begin

UUT: ImmediateGenerator

port map (

clk => clk,

instruction => instruction,

immediate => immediate

);

testing: process

begin

--set up inputs to test

instruction <= "0001000010000001";

clk <= '0';

wait for 2 ns;

clk <= '1';

wait for 2 ns;

assert(immediate = "0000000010000001") --load check

report "load failed"

severity warning;

instruction <= "0010000010000001";

clk <= '0';

wait for 2 ns;

clk <= '1';

wait for 2 ns;

assert(immediate = "0000000010000001") -- store check

report "store failed"

severity warning;

instruction <= "0100000000000001";

clk <= '0';

wait for 2 ns;

clk <= '1';

wait for 2 ns;

assert(immediate = "0000000000000001") -- addi check positive

report "addi positive failed"

severity warning;

instruction <= "0100000010000001";

clk <= '0';

wait for 2 ns;

clk <= '1';

wait for 2 ns;

assert(immediate = "1111111110000001") -- addi negative check

report "addi negative failed"

severity warning;

instruction <= "1001001000000001";

clk <= '0';

wait for 2 ns;

clk <= '1';

wait for 2 ns;

assert(immediate = "0000001000000001") -- jump check

report "jump failed"

severity warning;

instruction <= "1010111100000000";

clk <= '0';

wait for 2 ns;

clk <= '1';

wait for 2 ns;

assert(immediate = "0000000000001111") -- BRE check

report "BRE failed"

severity warning;

instruction <= "1011111100000000";

clk <= '0';

wait for 2 ns;

clk <= '1';

wait for 2 ns;

assert(immediate = "0000000000001111") -- BRNE check

report "BRNE failed"

severity warning;

instruction <= "1100111100000000";

clk <= '0';

wait for 2 ns;

clk <= '1';

wait for 2 ns;

assert(immediate = "0000000000001111") -- BRLT check

report "BRLT failed"

severity warning;

instruction <= "1101111100000000";

clk <= '0';

wait for 2 ns;

clk <= '1';

wait for 2 ns;

assert(immediate = "0000000000001111") -- BRGE check

report "BRGE failed"

severity warning;

wait;

end process;

end architecture;

**Code Base 13: Program Counter Testbench**

----------------------------------------------------------------------------------

-- Engineer: Aidan Lambrecht

--

-- Create Date: 10/21/2019

-- Module Name: Program Counter Testbench

-- Project Name: CPU Design Part 3

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

use IEEE.NUMERIC\_STD.ALL;

library work;

use work.all;

entity PCTestbench is

end PCTestbench;

architecture test of PCTestbench is

component PC is

port (

clk : in std\_logic;

instruction\_address : in std\_logic\_vector(9 downto 0);

next\_instr\_address : out std\_logic\_vector(9 downto 0)

);

end component;

signal clk : std\_logic;

signal instruction\_address : std\_logic\_vector(9 downto 0);

signal next\_instr\_address : std\_logic\_vector(9 downto 0);

begin

UUT: PC

port map (

clk => clk,

instruction\_address => instruction\_address,

next\_instr\_address => next\_instr\_address);

testing: process

begin

--setup test

clk <= '0';

instruction\_address <= "1111111111";

wait for 2 ns;

--move clk

clk <= '1';

wait for 2 ns;

assert(next\_instr\_address = "1111111111")

report "load failed"

severity WARNING;

clk <= '0';

instruction\_address <= "0000000000";

wait for 2 ns;

assert(next\_instr\_address = "1111111111")

report "clk'd process failed"

severity WARNING;

clk <= '1';

wait for 2 ns;

assert(next\_instr\_address = "0000000000")

report "load failed"

severity WARNING;

clk <= '0';

instruction\_address <= "1111111111";

wait for 2 ns;

assert(next\_instr\_address = "0000000000")

report "clk'd process failed"

severity WARNING;

clk <= '1';

wait for 2 ns;

assert(next\_instr\_address = "1111111111")

report "load failed"

severity WARNING;

wait;

end process;

end architecture;

**Code Base 14: Register File Testbench**

----------------------------------------------------------------------------------

-- Engineer: Aidan Lambrecht

--

-- Create Date: 10/21/2019

-- Module Name: InstructionMemory Testbench

-- Project Name: CPU Design Part 3

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

use IEEE.NUMERIC\_STD.ALL;

library work;

use work.all;

entity RegisterFileTestbench is

end RegisterFileTestbench;

architecture test of RegisterFileTestbench is

component registerFile is

port (

CLK : in STD\_LOGIC;

instr : in STD\_LOGIC\_VECTOR(11 downto 0);

RegWrite : in STD\_LOGIC;

writeData: in STD\_LOGIC\_VECTOR(15 downto 0);

readData1: out STD\_LOGIC\_VECTOR(15 downto 0);

readData2: out STD\_LOGIC\_VECTOR(15 downto 0)

);

end component;

signal CLK : std\_logic;

signal instr : std\_logic\_vector(11 downto 0);

signal RegWrite : std\_logic;

signal writeData : std\_logic\_vector(15 downto 0);

signal readData1 : std\_logic\_vector(15 downto 0);

signal readData2 : std\_logic\_vector(15 downto 0);

begin

UUT: registerFile

port map (

clk => clk,

instr => instr,

RegWrite => RegWrite,

writeData => writeData,

readData1 => readData1,

readData2 => readData2

);

testing: process

begin

clk <= '0';

writeData <= "1111111111111111";

for A in 0 to 15 loop

clk <= '0';

wait for 2 ns;

instr <= "00000000" & std\_logic\_vector(to\_unsigned(A,4));

RegWrite <= '1';

clk <= '1';

wait for 2 ns;

end loop;

RegWrite <= '0';

clk <= '0';

for A in 0 to 15 loop

instr <= "0000" & std\_logic\_vector(to\_unsigned(A,4)) & "0000";

wait for 2 ns;

clk <= '1';

wait for 2 ns;

assert (readData1 = "1111111111111111")

report "not all registers loaded correctly"

severity WARNING;

clk <= '0';

wait for 2 ns;

end loop;

writeData <= "0000000000000000";

RegWrite <= '1';

for A in 0 to 15 loop

clk <= '0';

wait for 2 ns;

instr <= "00000000" & std\_logic\_vector(to\_unsigned(A,4));

RegWrite <= '1';

clk <= '1';

wait for 2 ns;

end loop;

RegWrite <= '0';

clk <= '0';

for A in 0 to 15 loop

instr <= "00000000" & std\_logic\_vector(to\_unsigned(A,4));

wait for 2 ns;

clk <= '1';

wait for 2 ns;

assert (readData1 = "0000000000000000")

report "not all registers loaded correctly"

severity WARNING;

clk <= '0';

wait for 2 ns;

end loop;

wait;

end process;

end architecture;