Jessica Atkinson
May 10, 2017
EE310

Lab 8: The Complete Processor

Activity #1 Define the Full Control Unit

State Flowchart

Below is a flowchart to illustrate the flow/process happening inside of the control unit. I have separated the concurrent signals and sequential signals in different boxes.

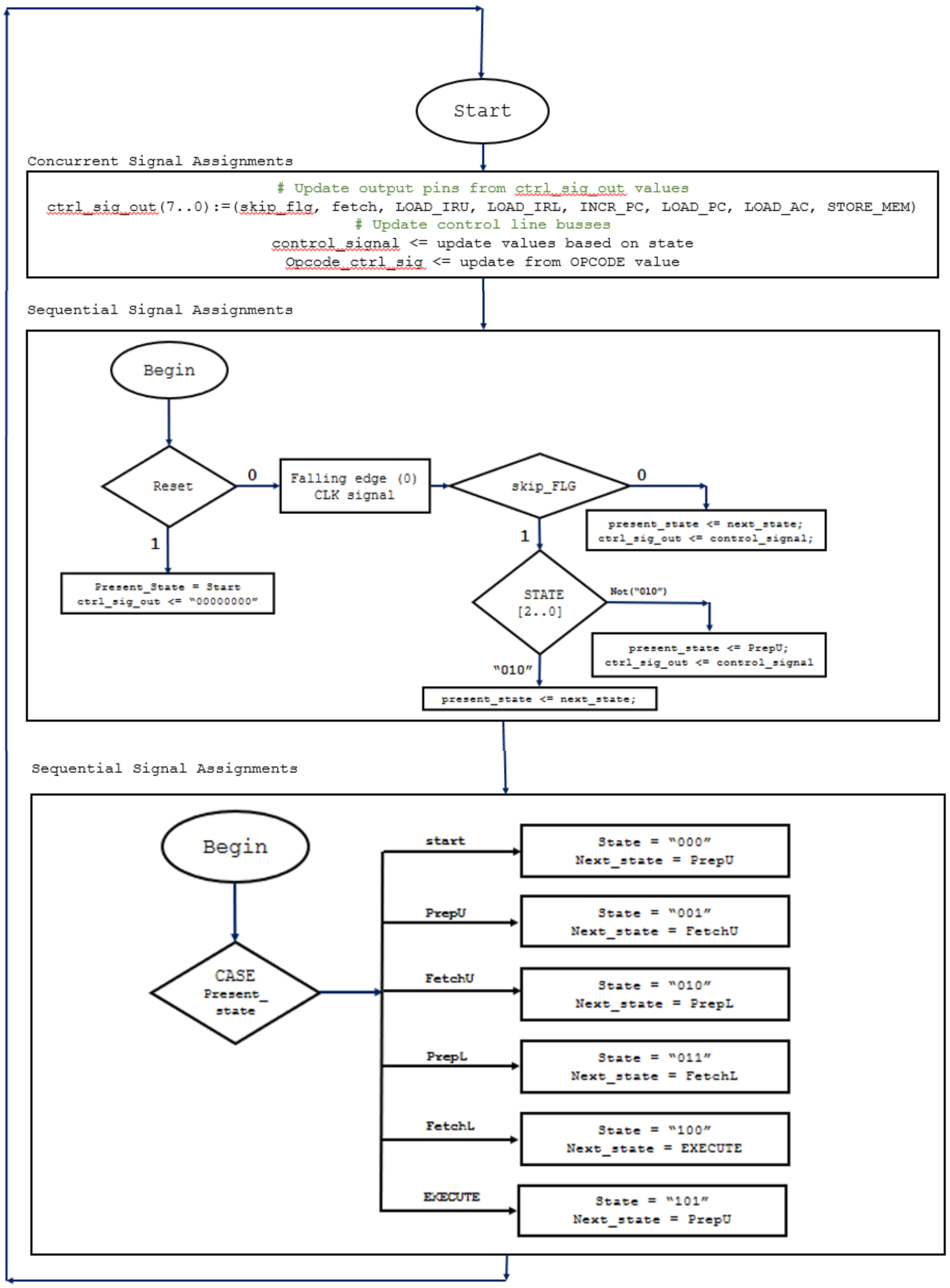

Start
Concurrent Signal Assignments
# Update output pins from ctrl_sig_out values
ctrl_sig_out(7..0):=(skip_flg, fetch, LOAD_IRU, LOAD_IRL, INCR_PC, LOAD_PC, LOAD_AC, STORE_MEM)
# Update control line busses
control_signal <= update values based on state
Opcode_ctrl_sig <= update from OPCODE value
Sequential Signal Assignments
Begin
Reset
0
1
Present_State = Start
ctrl_sig_out <= "00000000"
Falling edge (0) CLK signal
skip_FLG
0
present_state <= next_state;
ctrl_sig_out <= control_signal;
1
STATE [2..0]
Not("010")
present_state <= PrepU;
ctrl_sig_out <= control_signal
"010"
present_state <= next_state;
Sequential Signal Assignments
Begin
CASE Present_state
start
State = "000"
Next_state = PrepU
PrepU
State = "001"
Next_state = FetchU
FetchU
State = "010"
Next_state = PrepL
PrepL
State = "011"
Next_state = FetchL
FetchL
State = "100"
Next_state = EXECUTE
EXECUTE
State = "101"
Next_state = PrepU

Table of States

State Name	**State Code**	**Description/Action**	**Next State**
Start	0	Immediately on RESET No action	PrepU
PrepU	1	Prepare for upper byte instruction fetch MAR ← PC	FetchU
FetchU	2	Fetch upper byte of instruction IRU ← MDR, PC ← PC+1	Prep(L|U)
PrepL	3	Prepare for lower byte instruction fetch MAR ← PC	FetchL
FetchL	4	Fetch lower byte of instruction IRL ← MDR, PC ← PC+1	Execute
Execute	5	Determine OPCODE operation toggle control signal outputs	PrepU

Activity 2: Write VHDL and Simulate Processor in Full Operation

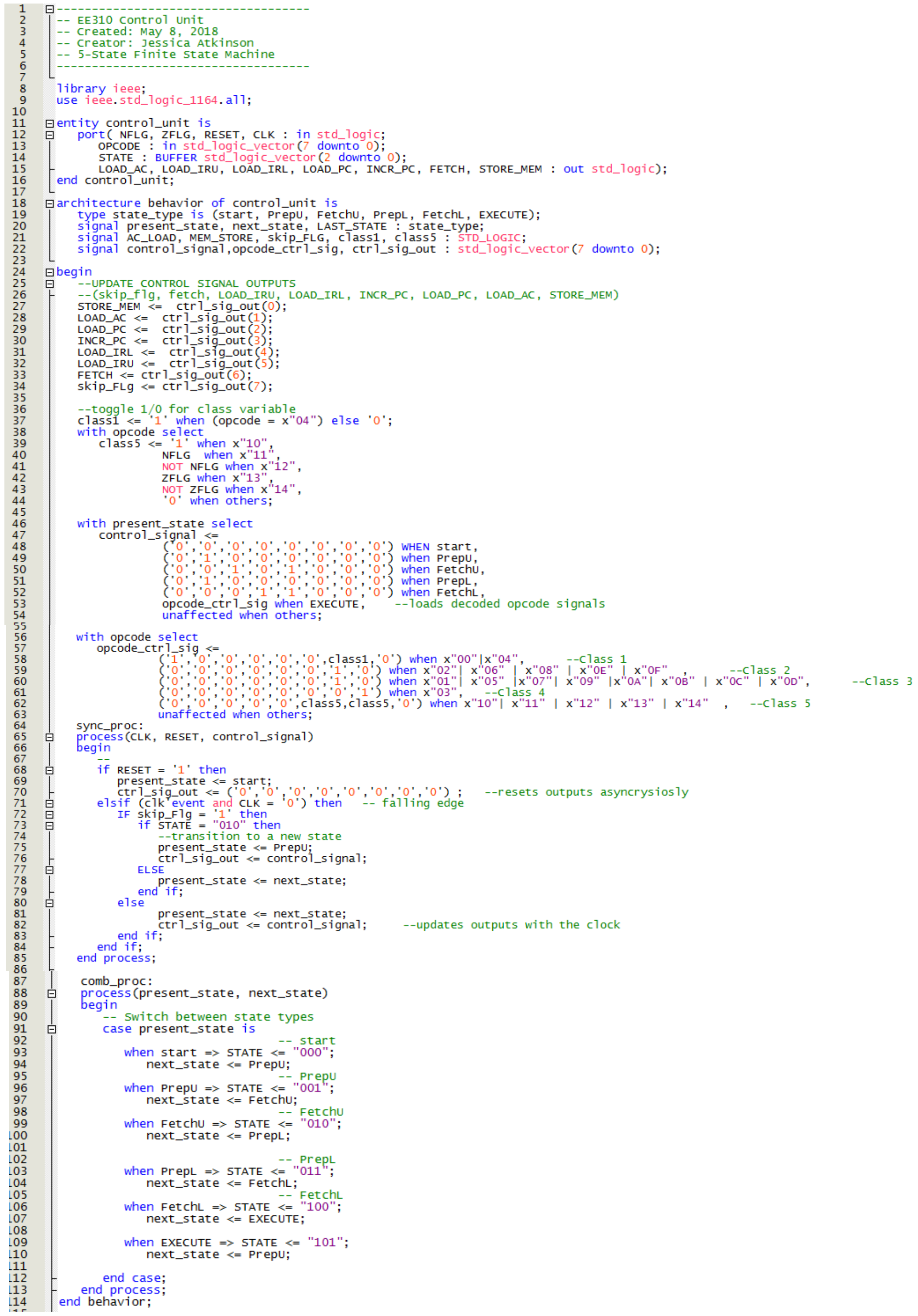
VHDL code

I reused the code form lab 7 and added some additional signals that will change and update the control unit outputs only when specified. I preset all the outputs per state and per opcode as it gets updated and it will only update when ctrl_sig_out is updated to the values of control_signal. These are 8 bit vectors where each bit represents a different output that are eventually stored into the outputs.

ctrl_sig_out[7..0]:=(skip_flg, fetch, LOAD_IRU, LOAD_IRL, INCR_PC, LOAD_PC, LOAD_AC, STORE_MEM)

Then from the lab 8 powerpoint I modeled this design by following the cases 1-5. Within each class is variables that may differ from one another so I created class variables that will update according to the OPCODE.

control_unit_tb.vhd

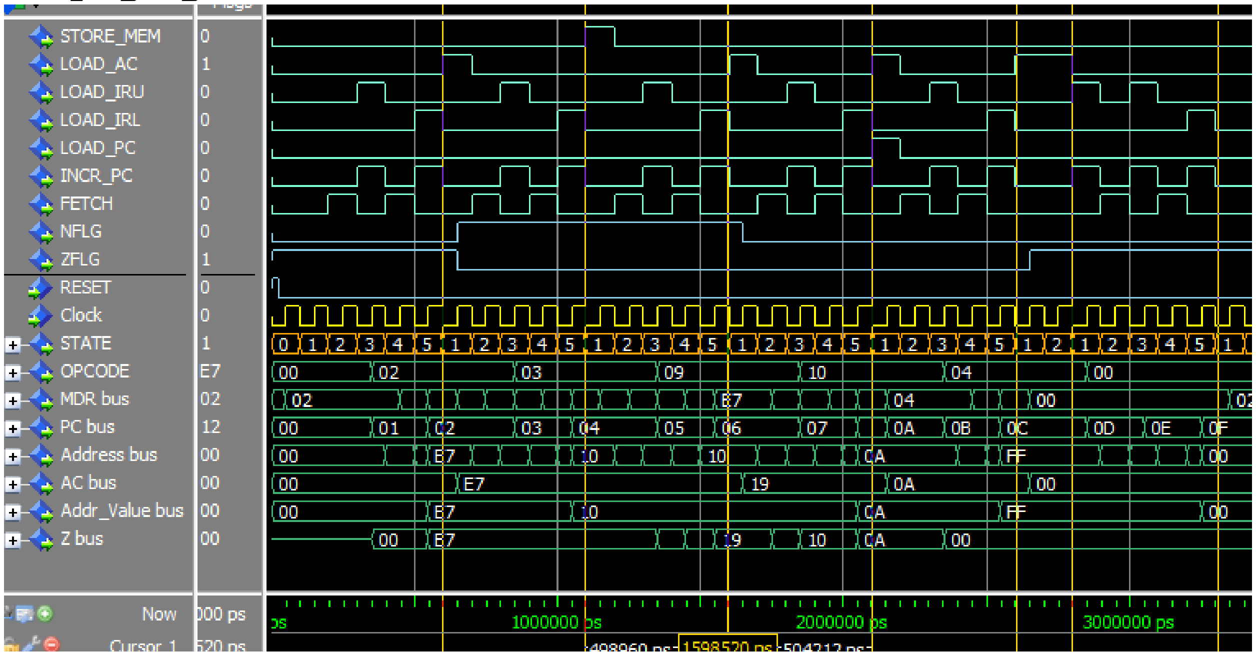


```vhdl
------------------------------------
-- EE310 Control Unit
-- Created: May 8, 2018
-- Creator: Jessica Atkinson
-- 5-State Finite State Machine
------------------------------------

library ieee;
use ieee.std_logic_1164.all;

entity control_unit is
   port( NFLG, ZFLG, RESET, CLK : in std_logic;
      OPCODE : in std_logic_vector(7 downto 0);
      STATE : BUFFER std_logic_vector(2 downto 0);
      LOAD_AC, LOAD_IRU, LOAD_IRL, LOAD_PC, INCR_PC, FETCH, STORE_MEM : out std_logic);
end control_unit;

architecture behavior of control_unit is
   type state_type is (start, PrepU, FetchU, PrepL, FetchL, EXECUTE);
   signal present_state, next_state, LAST_STATE : state_type;
   signal AC_LOAD, MEM_STORE, skip_FLG, class1, class5 : STD_LOGIC;
   signal control_signal,opcode_ctrl_sig, ctrl_sig_out : std_logic_vector(7 downto 0);

begin
   --UPDATE CONTROL SIGNAL OUTPUTS
   --(skip_flg, fetch, LOAD_IRU, LOAD_IRL, INCR_PC, LOAD_PC, LOAD_AC, STORE_MEM)
   STORE_MEM <=  ctrl_sig_out(0);
   LOAD_AC <=  ctrl_sig_out(1);
   LOAD_PC <=  ctrl_sig_out(2);
   INCR_PC <=  ctrl_sig_out(3);
   LOAD_IRL <=  ctrl_sig_out(4);
   LOAD_IRU <=  ctrl_sig_out(5);
   FETCH <= ctrl_sig_out(6);
   skip_FLg <= ctrl_sig_out(7);

   --toggle 1/0 for class variable
   class1 <= '1' when (opcode = x"04") else '0';
   with opcode select
      class5 <= '1' when x"10",
               NFLG  when x"11",
               NOT NFLG when x"12",
               ZFLG when x"13",
               NOT ZFLG when x"14",
               '0' when others;

   with present_state select
      control_signal <=
               ('0','0','0','0','0','0','0','0') WHEN start,
               ('0','1','0','0','0','0','0','0') when PrepU,
               ('0','0','1','0','1','0','0','0') when FetchU,
               ('0','1','0','0','0','0','0','0') when PrepL,
               ('0','0','0','1','1','0','0','0') when FetchL,
               opcode_ctrl_sig when EXECUTE,     --loads decoded opcode signals
               unaffected when others;

   with opcode select
      opcode_ctrl_sig <=
               ('1','0','0','0','0','0',class1,'0') when x"00"|x"04",        --Class 1
               ('0','0','0','0','0','0','1','0') when x"02"| x"06" | x"08" | x"0E" | x"0F"   ,       --Class 2
               ('0','0','0','0','0','0','1','0') when x"01"| x"05" |x"07"| x"09" |x"0A"| x"0B" | x"0C" | x"0D",       --Class 3
               ('0','0','0','0','0','0','0','1') when x"03",    --Class 4
               ('0','0','0','0','0',class5,class5,'0') when x"10"| x"11" | x"12" | x"13" | x"14"   ,   --Class 5
               unaffected when others;
   sync_proc:
   process(CLK, RESET, control_signal)
   begin
      --
      if RESET = '1' then
         present_state <= start;
         ctrl_sig_out <= ('0','0','0','0','0','0','0','0') ;   --resets outputs asyncrysiosly
      elsif (clk'event and CLK = '0') then   -- falling edge
         IF skip_Flg = '1' then
            if STATE = "010" then
               --transition to a new state
               present_state <= PrepU;
               ctrl_sig_out <= control_signal;
            ELSE
               present_state <= next_state;
            end if;
         else
               present_state <= next_state;
               ctrl_sig_out <= control_signal;     --updates outputs with the clock
         end if;
      end if;
   end process;

    comb_proc:
    process(present_state, next_state)
    begin
       -- Switch between state types
       case present_state is
                                 -- start
          when start => STATE <= "000";
             next_state <= PrepU;
                                 -- PrepU
          when PrepU => STATE <= "001";
             next_state <= FetchU;
                                 -- FetchU
          when FetchU => STATE <= "010";
             next_state <= PrepL;

                                 -- PrepL
          when PrepL => STATE <= "011";
             next_state <= FetchL;
                                 -- FetchL
          when FetchL => STATE <= "100";
             next_state <= EXECUTE;

          when EXECUTE => STATE <= "101";
             next_state <= PrepU;

       end case;
    end process;
end behavior;
```

Functional Simulation Results

Below is the ModelSim functional simulation results of my finite state machine. I have marked when the state machine changes from state 5 or state 2 back to state 1. When the opcode executed is x"04" there is only 8-bits that need to be stored so it changes from state 2 to state 1.

control_unit_force_file



Force File

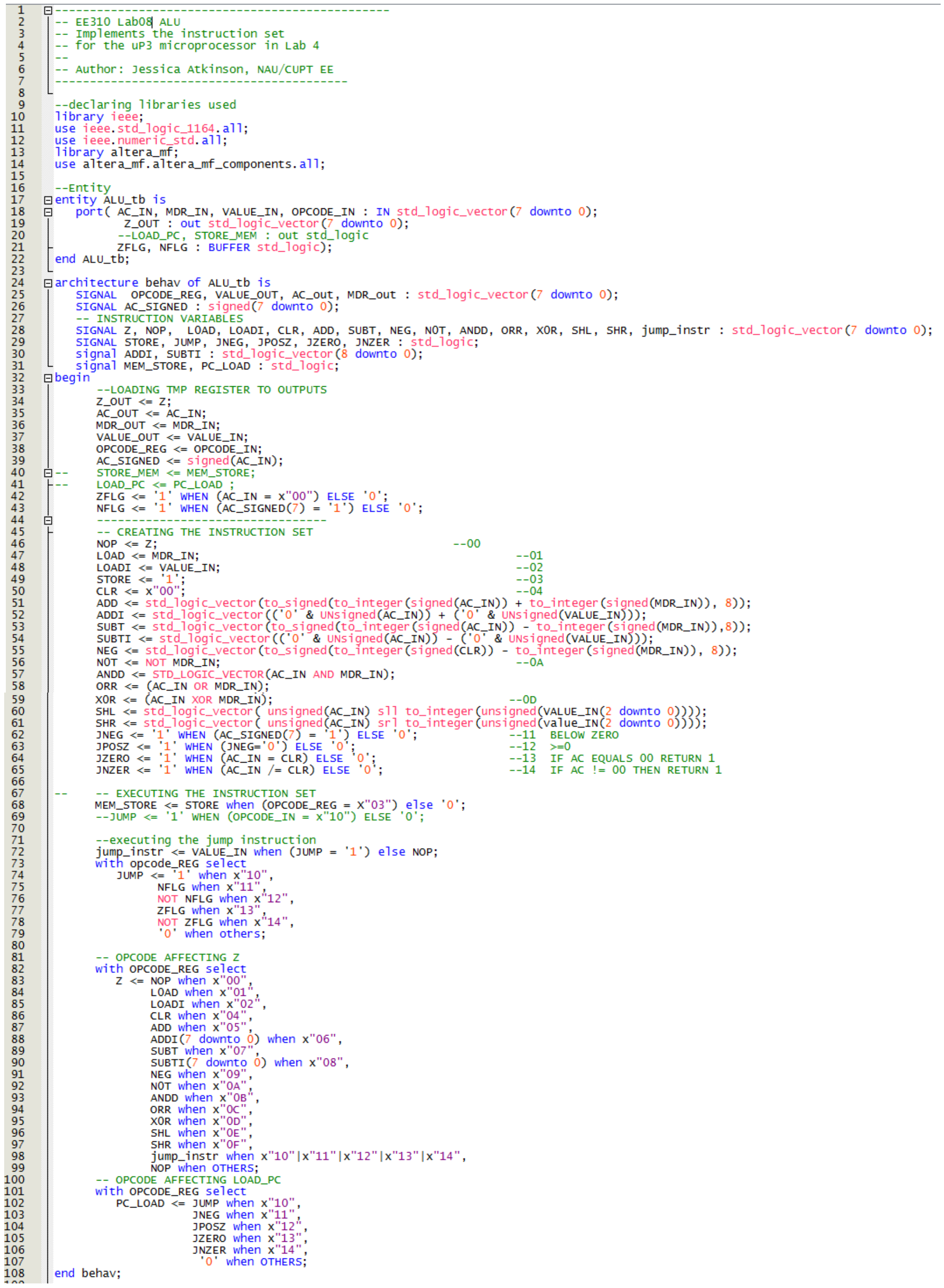
I was able to use the same force file from Lab7 Act#2. Because the VHDL code was the only thing that needed to be updated so the force file code remained unchanged.

```
# EE310 Lab08
# Force File
# Demonstrate Control Signals

restart -force

# Reset and clear all registers
force RESET 1 0ns ;
force clk 0 0ns, 1 50ns -r 100ns;

# Run one clock cycle
run 25ns;
force RESET 0 0ns;
run 75ns;
run 5000ns;
```

ALU

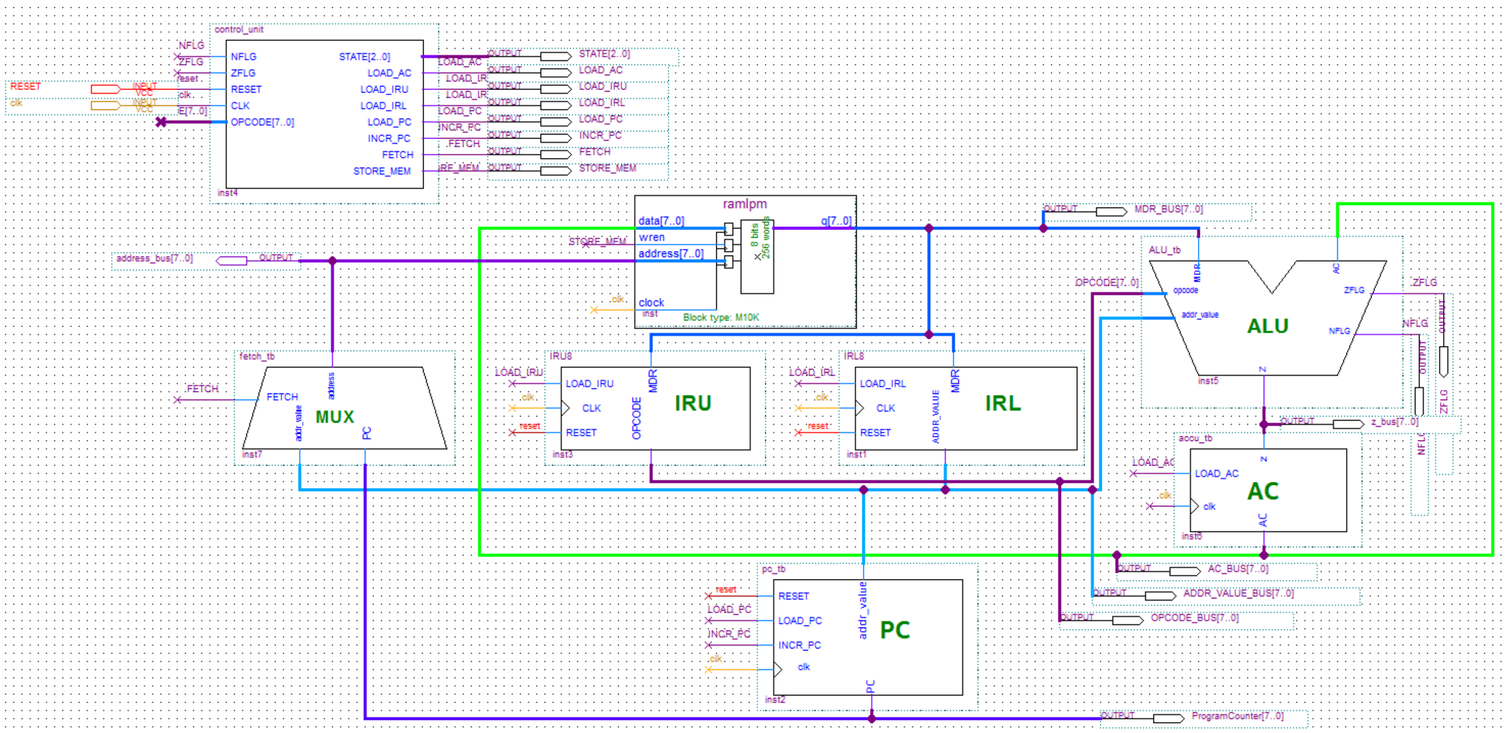
The ALU is the only other VHDL file that had to be changed to work with the control unit. I had to alter the code to know when to produce the control signal to load the value_IN to the output and send the signal flag to know when to assert LOAD_AC.

alu_tb.vhd



```vhdl
---------------------------------------------------
-- EE310 Lab08 ALU
-- Implements the instruction set
-- for the uP3 microprocessor in Lab 4
--
-- Author: Jessica Atkinson, NAU/CUPT EE
-------------------------------------------

--declaring libraries used
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library altera_mf;
use altera_mf.altera_mf_components.all;

--Entity
entity ALU_tb is
   port( AC_IN, MDR_IN, VALUE_IN, OPCODE_IN : IN std_logic_vector(7 downto 0);
          Z_OUT : out std_logic_vector(7 downto 0);
         --LOAD_PC, STORE_MEM : out std_logic
         ZFLG, NFLG : BUFFER std_logic);
end ALU_tb;

architecture behav of ALU_tb is
   SIGNAL  OPCODE_REG, VALUE_OUT, AC_out, MDR_out : std_logic_vector(7 downto 0);
   SIGNAL AC_SIGNED : signed(7 downto 0);
   -- INSTRUCTION VARIABLES
   SIGNAL Z, NOP,  LOAD, LOADI, CLR, ADD, SUBT, NEG, NOT, ANDD, ORR, XOR, SHL, SHR, jump_instr : std_logic_vector(7 downto 0);
   SIGNAL STORE, JUMP, JNEG, JPOSZ, JZERO, JNZER : std_logic;
   signal ADDI, SUBTI : std_logic_vector(8 downto 0);
   signal MEM_STORE, PC_LOAD : std_logic;
begin
      --LOADING TMP REGISTER TO OUTPUTS
      Z_OUT <= Z;
      AC_OUT <= AC_IN;
      MDR_OUT <= MDR_IN;
      VALUE_OUT <= VALUE_IN;
      OPCODE_REG <= OPCODE_IN;
      AC_SIGNED <= signed(AC_IN);
--    STORE_MEM <= MEM_STORE;
--    LOAD_PC <= PC_LOAD ;
      ZFLG <= '1' WHEN (AC_IN = x"00") ELSE '0';
      NFLG <= '1' WHEN (AC_SIGNED(7) = '1') ELSE '0';
      --------------------------------
      -- CREATING THE INSTRUCTION SET
      NOP <= Z;                                         --00
      LOAD <= MDR_IN;                                            --01
      LOADI <= VALUE_IN;                                         --02
      STORE <= '1';                                              --03
      CLR <= x"00";                                              --04
      ADD <= std_logic_vector(to_signed(to_integer(signed(AC_IN)) + to_integer(signed(MDR_IN)), 8));
      ADDI <= std_logic_vector(('0' & UNsigned(AC_IN)) + ('0' & UNsigned(VALUE_IN)));
      SUBT <= std_logic_vector(to_signed(to_integer(signed(AC_IN)) - to_integer(signed(MDR_IN)),8));
      SUBTI <= std_logic_vector(('0' & UNsigned(AC_IN)) - ('0' & UNsigned(VALUE_IN)));
      NEG <= std_logic_vector(to_signed(to_integer(signed(CLR)) - to_integer(signed(MDR_IN)), 8));
      NOT <= NOT MDR_IN;                                         --0A
      ANDD <= STD_LOGIC_VECTOR(AC_IN AND MDR_IN);
      ORR <= (AC_IN OR MDR_IN);
      XOR <= (AC_IN XOR MDR_IN);                                --0D
      SHL <= std_logic_vector( unsigned(AC_IN) sll to_integer(unsigned(VALUE_IN(2 downto 0))));
      SHR <= std_logic_vector( unsigned(AC_IN) srl to_integer(unsigned(value_IN(2 downto 0))));
      JNEG <= '1' WHEN (AC_SIGNED(7) = '1') ELSE '0';           --11  BELOW ZERO
      JPOSZ <= '1' WHEN (JNEG='0') ELSE '0';                    --12  >=0
      JZERO <= '1' WHEN (AC_IN = CLR) ELSE '0';                 --13  IF AC EQUALS 00 RETURN 1
      JNZER <= '1' WHEN (AC_IN /= CLR) ELSE '0';                --14  IF AC != 00 THEN RETURN 1

--    -- EXECUTING THE INSTRUCTION SET
      MEM_STORE <= STORE when (OPCODE_REG = X"03") else '0';
      --JUMP <= '1' WHEN (OPCODE_IN = x"10") ELSE '0';

      --executing the jump instruction
      jump_instr <= VALUE_IN when (JUMP = '1') else NOP;
      with opcode_REG select
         JUMP <= '1' when x"10",
               NFLG when x"11",
               NOT NFLG when x"12",
               ZFLG when x"13",
               NOT ZFLG when x"14",
               '0' when others;

      -- OPCODE AFFECTING Z
      with OPCODE_REG select
         Z <= NOP when x"00",
              LOAD when x"01",
              LOADI when x"02",
              CLR when x"04",
              ADD when x"05",
              ADDI(7 downto 0) when x"06",
              SUBT when x"07",
              SUBTI(7 downto 0) when x"08",
              NEG when x"09",
              NOT when x"0A",
              ANDD when x"0B",
              ORR when x"0C",
              XOR when x"0D",
              SHL when x"0E",
              SHR when x"0F",
              jump_instr when x"10"|x"11"|x"12"|x"13"|x"14",
              NOP when OTHERS;
      -- OPCODE AFFECTING LOAD_PC
      with OPCODE_REG select
         PC_LOAD <= JUMP when x"10",
                    JNEG when x"11",
                    JPOSZ when x"12",
                    JZERO when x"13",
                    JNZER when x"14",
                     '0' when OTHERS;
end behav;
```

Block Diagram


control_unit
NFLG
ZFLG
RESET
CLK
OPCODE[7..0]
STATE[2..0]
LOAD_AC
LOAD_IRU
LOAD_IRL
LOAD_PC
INCR_PC
FETCH
STORE_MEM
inst4
ramlpm
data[7..0]
wren
address[7..0]
clock
q[7..0]
8 bits 256 words
Block type: M10K
address_bus[7..0]
MDR_BUS[7..0]
fetch_tb
MUX
FETCH
address
addr_value
PC
inst7
IRU8
LOAD_IRU
CLK
RESET
MDR
OPCODE
IRU
inst3
IRL8
LOAD_IRL
ADDR_VALUE
IRL
inst1
ALU_tb
ALU
opcode
addr_value
ZFLG
NFLG
AC
Z
inst5
z_bus[7..0]
accu_tb
AC
LOAD_AC
clk
inst6
AC_BUS[7..0]
ADDR_VALUE_BUS[7..0]
OPCODE_BUS[7..0]
pc_tb
PC
RESET
LOAD_PC
INCR_PC
addr_value
inst2
ProgramCounter[7..0]