ECE 424
DUE: Wednesday, Nov 18, 12:30 pm

Design Project #3: Single-Cycle Datapath Design (100 points)

COVER SHEET

Honor Code: _______I have neither given or received, nor have I tolerated others’_________

_______________use of unauthorized aid._________

Name: ____Joe Leveille____ Signature: ____Joseph Leveille_____________

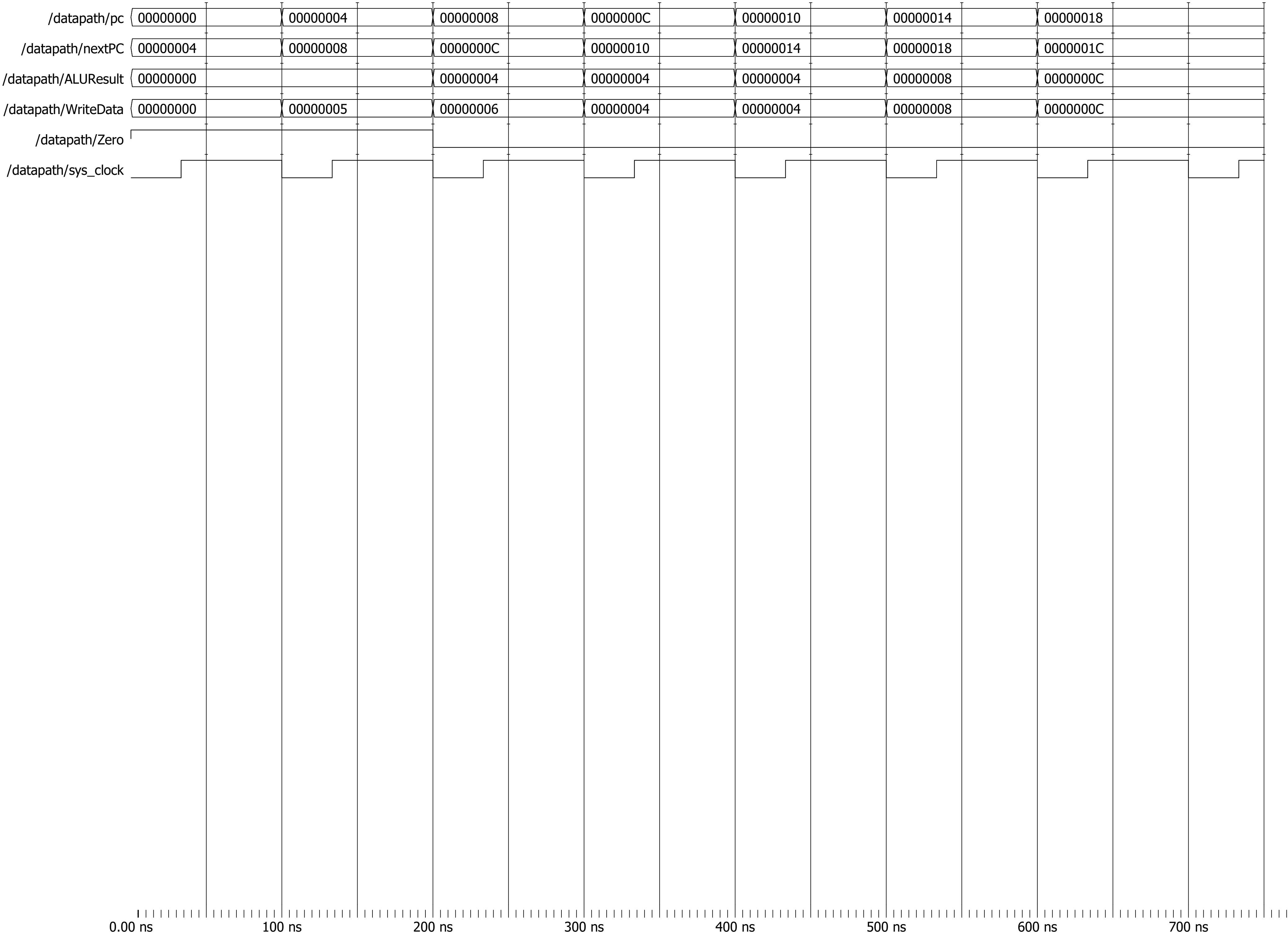
Honor Code: _____I have neither given or received, nor have I tolerated _______________

_______________others’ use of unauthorized aid. ________________

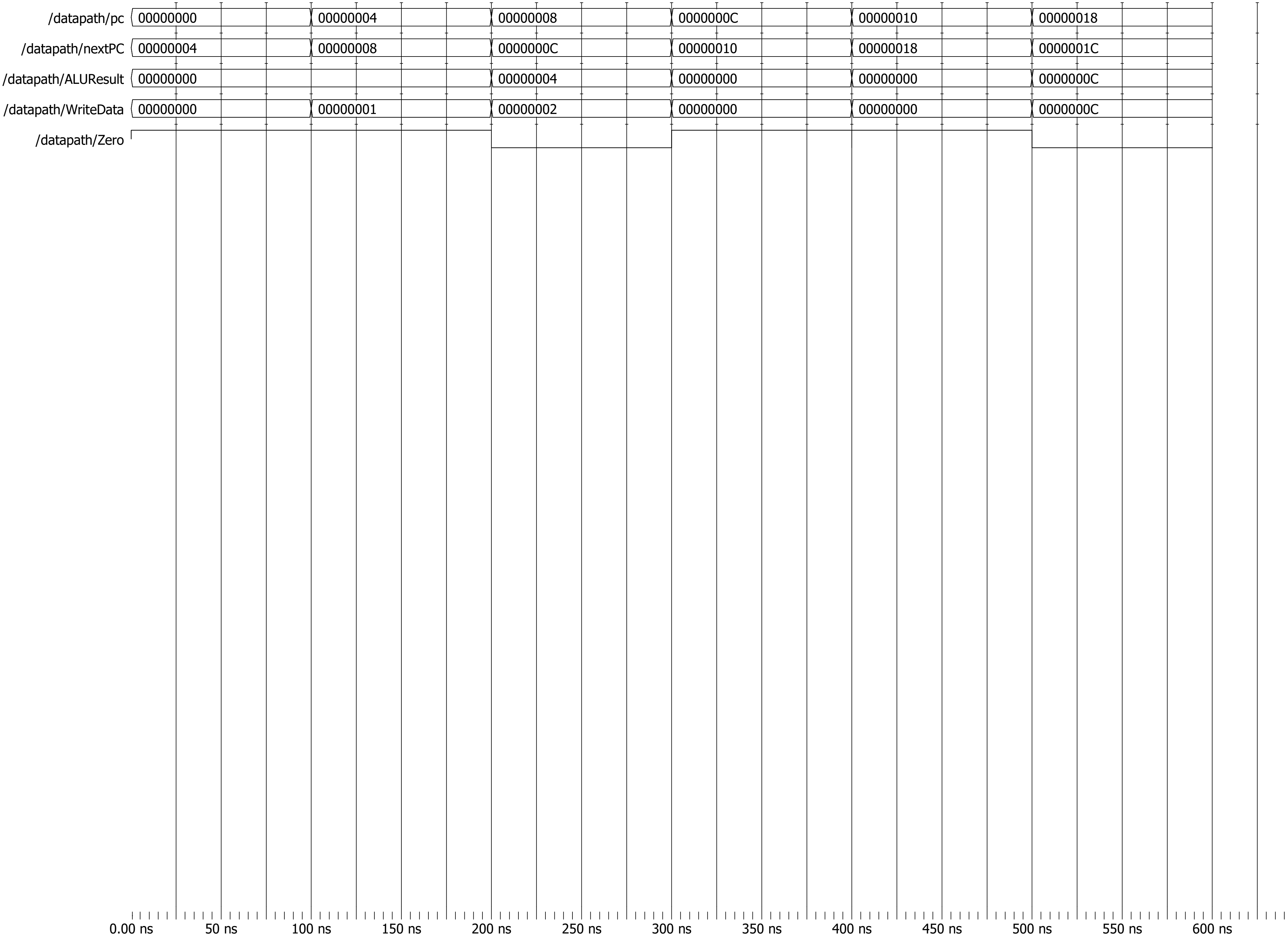
Name: ___Jon Bayert_____ Signature: ____Jonathon Bayert_________________

A complete assignment will contain:

1. This cover sheet
2. A printout of your VHDL code including your main file, modified imem.vhd, modified alu-control.vhd, and modified dmem.vhd.
3. Waveforms for the two different cases, showing the five (and only the five) signals in hex format.



Entity:datapath Architecture:databoi Date: Sun Nov 15 18:44:41 CST 2020   Row: 1 Page: 1



Entity:datapath Architecture:databoi Date: Sun Nov 15 18:55:50 CST 2020 Row: 1 Page: 1

```
-- Joe Leveille & Jon Bayert
-- Final VHDL Assignment - ECE 424 - Computer Architecture
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.mips_types.all;

entity datapath is
    PORT ( pc : IN STD_LOGIC_VECTOR ( 31 downto 0 );
           nextPC: OUT STD_LOGIC_VECTOR (31 downto 0) );

END datapath;

architecture dataBoi of datapath is
    COMPONENT alu_control
        port( INSTR: in std_logic_vector(5 DOWNTO 0);
            ALUOP: in std_logic_vector(1 DOWNTO 0);
            ALUCTRL : OUT std_logic_vector(3 DOWNTO 0));
    END COMPONENT;

    COMPONENT control
        port( INSTR: in std_logic_vector(5 DOWNTO 0);
             REGDST: OUT std_logic;
             BRANCH: OUT std_logic;
             MEMREAD: OUT std_logic;
             MEMTOREG: OUT std_logic;
             ALUOP: OUT std_logic_vector(1 DOWNTO 0);
             MEMWRITE: OUT std_logic;
             ALUSRC: OUT std_logic;
            REGWRITE: OUT std_logic);
    END COMPONENT;

    COMPONENT dmemory
       port(
          DIN: in mips_data;
          ADDR: in mips_address;
          DOUT: out mips_data;
          WE, RE: in std_logic;
          CLK: in std_logic
        );
    END COMPONENT;

    COMPONENT imemory
       port(
          ADDR: in mips_address;
          DOUT: out mips_data
        );
    END COMPONENT;

    COMPONENT alu32
        PORT( A, B: in std_logic_vector (31 downto 0);
            ALUOp: in std_logic_vector (3 downto 0);
            RESULT: out std_logic_vector (31 downto 0);
            Z, V, C: out std_logic );
    END COMPONENT;

    COMPONENT mux2a
       port( IN0: in mips_reg_addr;
          IN1: in mips_reg_addr;
          SEL: in STD_LOGIC;
          DOUT: out mips_reg_addr);
    end component;

    COMPONENT reg_file
      port (a1 : in mips_reg_addr;
            q1 : out mips_data;
        a2 : in mips_reg_addr;
        q2 : out mips_data;
        a3 : in mips_reg_addr;
        d3 : in mips_data;
        write_en : in std_logic;
        clk : in std_logic);
end COMPONENT;

COMPONENT signextend
   port( INDATA: in STD_LOGIC_VECTOR(15 DOWNTO 0);
      OUTDATA: out mips_data);
end COMPONENT;

COMPONENT mux2
   port( IN0: in mips_data;
      IN1: in mips_data;
      SEL: in std_logic;
      DOUT: out mips_data);
end COMPONENT;

COMPONENT shiftleft
   port( INDATA: in mips_data;
      OUTDATA: out mips_data);
end COMPONENT;

SIGNAL sys_clock : std_logic := '0';
CONSTANT Tcycle : time := 100 ns;

SIGNAL inst, ALUResult, outMuxIn1, branchALUin1, wDataMuxin1 : std_logic_vector( 31 downto 0);
SIGNAL defNextPC, WriteData, aluIn0, aluIn1, mux2in0, mux2in1  : std_logic_vector(31 downto 0);
SIGNAL REGDST, BRANCH, MEMREAD, MEMTOREG, MEMWRITE, ALUSRC, REGWRITE, Zero, outMuxSel : std_logic;
SIGNAL ALUOpDatapath : std_logic_vector(1 downto 0);
SIGNAL wRegIn : std_logic_vector(4 downto 0);
SIGNAL mainALUctl : std_logic_vector(3 downto 0);

BEGIN

  -- create clock process
  clk_gen: process
  begin
    sys_clock <= '1' after Tcycle/3, '0' after Tcycle;
    wait until sys_clock = '0';
  end process clk_gen;

Imem: imemory PORT MAP(
    ADDR => pc, DOUT => inst
);

PCplus4: alu32 PORT MAP(
    A => pc, B=> X"00000004", ALUOp => "0010", RESULT=>defNextPC, Z => oPeN, V=> OPen, C=>oPEn
);

CTL: control PORT MAP(
    INSTR => inst(31 downto 26), REGDST => RegDst,BRANCH => Branch,
    MEMREAD => MEMREAD,MEMTOREG => MEMTOREG,MEMWRITE => MEMWRITE,ALUSRC => ALUSRC,
    REGWRITE => REGWRITE,ALUOP => ALUOPDatapath
);

writeRegSel: mux2a PORT MAP(
    IN0 => inst(20 downto 16), IN1 => inst(15 downto 11), SEL => REGDST, DOUT => wRegIn
);

reg: reg_file PORT MAP(
    a1=> inst(25 downto 21), a2 => inst(20 downto 16), a3 => wRegIn,
    q1 => aluIn0, q2 => mux2in0, d3=> WriteData, write_en=> REGWRITE, CLK => sys_clock
  );

  sign: signextend PORT MAP(
    INDATA => inst(15 downto 0), OUTDATA => mux2in1
  );

  muxTheSecond: mux2 PORT MAP(
    IN0=>mux2in0, IN1=> mux2in1, SEL=>ALUSRC, DOUT => aluIn1
  );

  ALUctl: alu_control PORT MAP(
    INSTR=> inst(5 downto 0), ALUop => ALUOpDatapath, ALUctrl => mainALUctl
  );

  mainALU: alu32 PORT MAP(
    a=>aluIn0, b=>aluIn1, ALUOp=> mainALUctl, RESULT => ALUResult, Z=>Zero, V=>open, C=>oPEN
  );

  shift: shiftleft PORT MAP(
    INDATA=>mux2in1, OUTDATA=>branchALUin1
  );

  branchALU: alu32 PORT MAP(
    a=>defNextPC, b=>branchALUin1, ALUOP=>"0010", RESULT=>outMuxIn1, Z=>open, C=>opEn, V=>OpEN
  );

  outMuxSel <= branch and Zero;

  outMux: mux2 PORT MAP(
    IN0=>defNextPC,IN1=>outMuxIn1, SEL=> outMuxSel, DOUT => nextPC
  );

  dmem: dmemory PORT MAP(
    ADDR => ALUResult, DIN => mux2in0, DOUT => wDataMuxin1, CLK => sys_clock, WE => memWrite, RE =>memRead
  );

  wDataMux: mux2 PORT MAP(
    IN0 => ALUResult, IN1=>wDataMuxin1, sel=>memToReg, DOUT=>WriteData
  );

end dataBoi;
```

```vhdl
--
-- Copyright Jay Brockman
--
-- MIPs Processor Developement
-- Eric W. Johnson
-- Valparaiso University
--
--
-- Updated Jeffrey Will

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.mips_types.all;


-- memory Entity Description
entity imemory is
   port(
      ADDR: in mips_address;
      DOUT: out mips_data
   );
end imemory;

-- memory Architecture Description
architecture rtl of imemory is
   subtype ramword is bit_vector(31 DOWNTO 0);
   type rammemory is array (0 to 1024) of ramword;

   --****************************************************
   -- Students:  You will have to hand-assemble the instructions
   -- Given in the assignment, and fill in the values
   -- below.  (It is like you are "flashing" the instruction
   -- memory with your assembly code
   --*****************************************************

   signal ram : rammemory := (
                x"00000820", -- 00: add $1,$0,$0
                x"8C220000", -- 01: lw $2,0($1)
                x"8C230004", -- 02:
                x"00432024", -- 03:
                x"10800001", -- 04:
                x"AC240008", -- 05:
                x"AC24000C", -- 06:
                others => x"00000000");
begin

   read_Process: process(ram, ADDR)
      variable raddr1 : integer range 0 to 1024;
      variable tempdata : ramword;
   begin
      -- convert address to integer
      raddr1 := conv_Integer(ADDR);
      raddr1 := raddr1/4;
      tempdata  := (ram(raddr1));
      DOUT  <= to_stdlogicvector(tempdata);
   end process read_Process;

end rtl;
```

```
--
-- MIPs Processor Developement
-- Eric W. Johnson
-- Valparaiso University
--
--
-- Modified by Jeffrey D. Will


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.mips_types.all;

entity control is
   port( INSTR: in std_logic_vector(5 DOWNTO 0);
         REGDST: OUT std_logic;
         BRANCH: OUT std_logic;
         MEMREAD: OUT std_logic;
         MEMTOREG: OUT std_logic;
         ALUOP: OUT std_logic_vector(1 DOWNTO 0);
         MEMWRITE: OUT std_logic;
         ALUSRC: OUT std_logic;
         REGWRITE: OUT std_logic);
end control;

architecture behavioral of control is
begin
  PROCESS( INSTR )
     variable temp : std_logic_vector ( 7 downto 0 );
     variable instr_int : integer range 0 to 64;
  BEGIN
     temp := "00000000";
     temp := temp + instr;
     instr_int := conv_Integer(temp);
     case instr_int is

       -- ********************************************
       -- Students:  YOU MUST FILL IN THESE CONTROL LINES
       -- FOR EACH INDIVIDUAL INSTRUCTION... they are
       -- set to all zeroes below, but the zeroes are
       -- just placeholders...
       -- ********************************************
    when 0 => REGDST <= '1';
                  BRANCH <= '0';
                  MEMREAD <= '0';
                  MEMTOREG <= '0';
                  ALUOP <= "10";           -- ALUOP ORDER OF BITS????
                  MEMWRITE <= '0';
                  ALUSRC <= '0';
                  REGWRITE <= '1';
        when 35 => REGDST <= '0';
                  BRANCH <= '0';
                  MEMREAD <= '1';
                  MEMTOREG <= '1';
                  ALUOP <= "00";
                  MEMWRITE <= '0';
                  ALUSRC <= '1';
                  REGWRITE <= '1';
        when 43 => REGDST <= '0';
                  BRANCH <= '0';
                  MEMREAD <= '0';
                  MEMTOREG <= '0';
                  ALUOP <= "00";
                  MEMWRITE <= '1';
                  ALUSRC <= '1';
                  REGWRITE <= '0';
        when 4 => REGDST <= '0';
```

```
                  BRANCH <= '1';
                  MEMREAD <= '0';
                  MEMTOREG <= '0';
                  ALUOP <= "01";
                  MEMWRITE <= '0';
                  ALUSRC <= '0';
                  REGWRITE <= '0';
        when others => REGDST <= '0';
                  BRANCH <= '0';
                  MEMREAD <= '0';
                  MEMTOREG <= '0';
                  ALUOP <= "00";
                  MEMWRITE <= '0';
                  ALUSRC <= '0';
                  REGWRITE <= '0';

     END CASE;

  END PROCESS;

end behavioral;
```

```vhdl
--
-- MIPs Processor Developement
-- Eric W. Johnson
-- Valparaiso University
--
--
--
-- Modified by Jeffrey Will

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity alu_control is
   port( INSTR: in std_logic_vector(5 DOWNTO 0);
         ALUOP: in std_logic_vector(1 DOWNTO 0);
         ALUCTRL : OUT std_logic_vector(3 DOWNTO 0));
end alu_control;

--**************************************************
--Students: it is your job to fill in
-- each of the "---" with the appropriate 1's and 0's
--**************************************************
architecture behavioral of alu_control is
begin
  PROCESS(INSTR,ALUOP)
  BEGIN
     case ALUOP is
        when "00" => ALUCTRL <= "0010";
        when "01" => ALUCTRL <= "0110";
        when "10" => case INSTR is
                        when "100000" => ALUCTRL <= "0010";
                        when "100010" => ALUCTRL <= "0110";
                        when "100100" => ALUCTRL <= "0000";
                        when "100101" => ALUCTRL <= "0001";
                        when "101010" => ALUCTRL <= "0111";
                        when others => ALUCTRL <= "0000";
             end case;
        when others => ALUCTRL <= "0000";

     END CASE;

  END PROCESS;

end behavioral;
```

```
--
-- MIPs Processor Developement
-- Eric W. Johnson
-- Valparaiso University
--
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.mips_types.all;

--
-- Copyright Jay Brockman, Feb 1997
--  Updated Eric W. Johnson, Feb 1998
--
-- Modified June 2011 Jeffrey Will

-- memory Entity Description
entity dmemory is
   port(
      DIN: in mips_data;
      ADDR: in mips_address;
      DOUT: out mips_data;
      WE, RE: in std_logic;
      CLK: in std_logic

   );
end dmemory;

-- memory Architecture Description
architecture rtl of dmemory is
   subtype ramword is bit_vector(31 DOWNTO 0);
   type rammemory is array (0 to 4096) of ramword;

   --**************************************************
   -- Students: This is where you modify the contents of
   -- Data memory.
   --**************************************************
   signal ram : rammemory := (
            x"00000001", -- 00 through 03
                x"00000002", -- 04 through 07
                x"00000003", --
                x"00000004", --
                x"00000000", --
                others => x"00000000");
begin

   read_Process: process(RE, ADDR)
      variable raddr1 : integer range 0 to 4096;
      variable tempdata : ramword;
   begin
      -- convert address to integer
  IF ( RE = '1' ) THEN
      raddr1 := conv_Integer(ADDR);
      raddr1 := raddr1/4;
      tempdata  := (ram(raddr1));
      DOUT  <= to_stdlogicvector(tempdata);
  END IF;
   end process read_Process;

   write_Process: process(WE, CLK)
      variable waddr : integer range 0 to 4096;
   begin
      if ( WE = '1' AND CLK'EVENT AND CLK = '1' ) then
    -- convert address to integer
    waddr := conv_Integer(ADDR);
        waddr := waddr/4;
    ram(waddr) <= to_bitvector(DIN);
      end if;
   end process write_Process;
end rtl;

```