Due Date: March 30, 2021

(Worth about 30% of the Project)

Goal: You need to implement Register unit, MUX, Sign-Extension and ALU. Finally, Write a test bench to test the complete design.

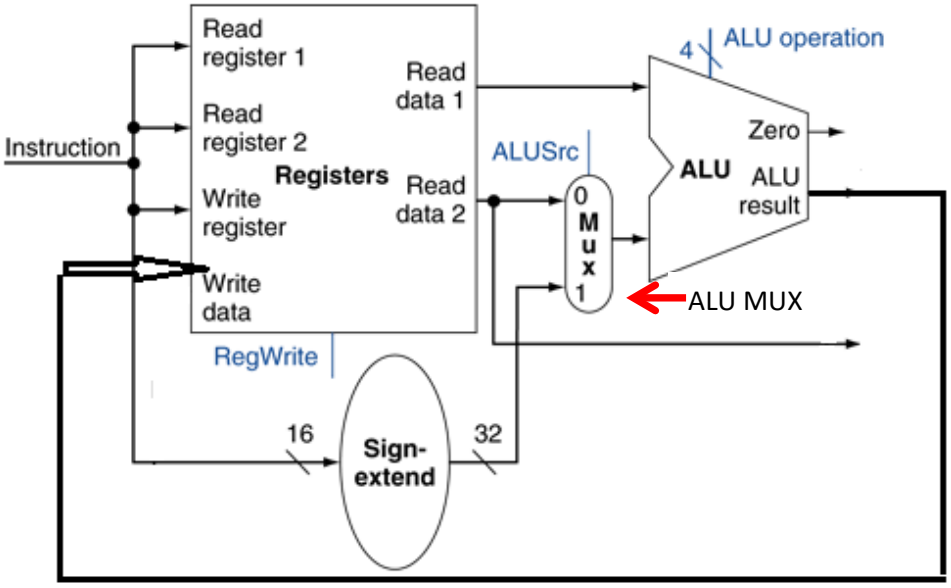

Figure 1.

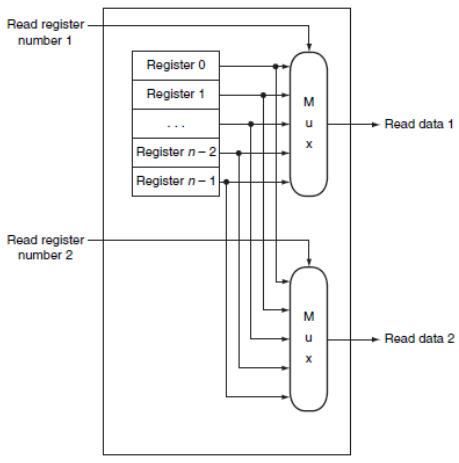

Figure 2. Read Unit

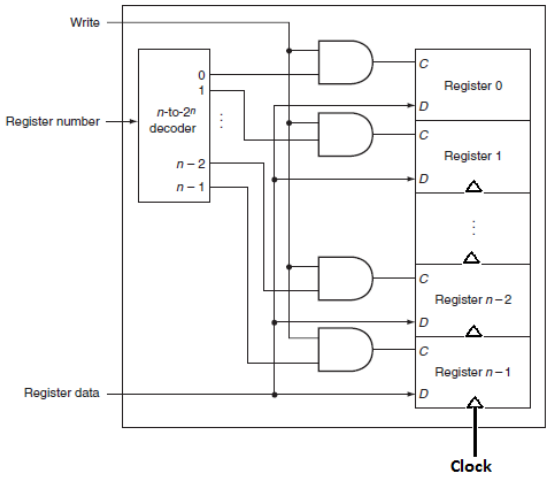

Figure 3. Write Unit

High level Description of each unit:

Registers: It should have 5 bits read/write register inputs (compliant to MIPS), a 32 bit Write data port and two **32 bit Read data outputs**.
The block diagram of ***read unit*** of register is provided in figure 2. This unit shows 32 registers of 32 bit each (Register 0 to Register (n-1), with n = 32). The select signal to the mux is the operand from the instruction memory (which you implemented in phase 1 of the project). This signal is same as Read register 1 and Read register 2 in Figure 1.
The block diagram of the ***write unit*** of the register file is provided in Figure 3. Write unit has four input signals. Clock, Write (which is same as RegWrite in Figure 1), Register Number, which is same as input connected to the write register in Figure1 and Register data, which needs to be connected to the Write data port of Figure 1. Also notice that each Register has three inputs, C, D and Clock. The C input acts as an enabling input of the register (enable the write port of the specific register), and write at the positive edge of the next Clock signal. The clock signal is the master clock at which the register writes the data. D is 32 bit input data coming from the data memory (Register Data). And n to 2^n decoder shows 5 to 32 decoder.
ALU: This should be a combinational block. Must have two 32 bit inputs (operands), 4 bit input control signals (ALU operation), 32 bit output and the result port and one bit output at zero port. ALU

SignExtend: It replicates the 16^{th} bit 16 times to make a 32 bit input to ALU-Mux. If required you may use ieee.numeric_std.all library. The input and output of the sign extension unit should be std_logic_vector

ALU-Mux: Two 32 bit inputs (std_logic_vector) and one 32 bit output (std_logic_vector) with select signal, which will eventually come from control unit (but not in this phase of the project)

Constraints:

1. Cannot use megafunctions or IPs for ANY block.
2. Modify the general ALU code (pre-tested) that is provided in appendix.
3. Register file unit should be implemented as shown in the Figure 1 and Figure 2 (distinctly provide the codes for decoder, multiplexer and registers).
4. For testing purposes, follow the test bench requirements provided below.

Test Bench Requirements:

1. First load the values to the register, t0, t1, t2, s0, s1, and s2 (while making sure that s1 and s2 are getting the same values) with non-zero values. Make sure that you know the register numbers, which are provided in your book.
2. Generate proper control signals and perform execution of following set of MIPS instructions:
   - **a.** add $t3, $t0, $t1
   - **b.** sub $t4, $s0, $s1
   - **c.** and $ t5, t3, t4 (the values calculated in add and sub instruction are used here)

Phase 2 of the Project – ECE 4120/5120 (Spring 2021)

If you want, you can integrate this phase of the project with your first phase, but I'll grade it independently. Last phase will require integration of all the phases.

Deliverables:

I. A pdf file containing following:

1. Modified block diagram, showing the bits and additional connections (**no hand drawings and or copy and paste of any figure from this document**)
2. What are the objectives of this phase
3. Elaboration on VHDL implementation of each unit
4. Use the same device/board for synthesis that you used in ECE 4110 and show the snapshot of that device selection.
5. Flow summary, RTL view and Technology map view
6. **Elaboration on test bench:** e.g. *how you loaded the registers*, and how you confirmed the execution of each of these instructions
7. **Elaboration on the waveforms** and snapshot of the waveforms should also be included. Show using separate waveforms to elaborate on **loading** the registers, and **writing** the result for **each** instruction. Elaboration should state whether the waveform show the successful implementation or not, with explanation on how you verified it
8. From the time quest timing analyzer find following information **and make a discussion**:
   a. Fmax
   b. Timing report of the setup time – make sure that it is showing positive slack.
9. Conclusions

II. A zipped VHDL Implementation of the Project: A folder containing all the files generated by Quartus II, including all the .vhd, should be submitted to the dropbox via ilearn. Name the submitted folder as your lastname_S21_phase2.

Appendix ALU Code:

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;
ENTITY alu_1 IS
        PORT (          s               : IN     STD_LOGIC_VECTOR(2 DOWNTO 0) ;
                        A, B    : IN     STD_LOGIC_VECTOR(31 DOWNTO 0) ;
                        F               : Buffer         STD_LOGIC_VECTOR(31 DOWNTO 0)  ;
                        Compare: OUT STD_logic ) ;
END alu_1 ;

ARCHITECTURE Behavior OF alu_1 IS
BEGIN
        PROCESS ( s, A, B )
        BEGIN
                CASE s IS
                        WHEN "000" =>
                                F <= x"00000000" ;
                        WHEN "001" =>
                                F <= B - A ;
                        WHEN "010" =>
                                F <= A - B ;
                        WHEN "011" =>
                                F <= A + B ;
                        WHEN "100" =>
                                F <= A XOR B ;
                        WHEN "101" =>
                                F <= A OR B ;
                        WHEN "110" =>
                                F <= A AND B ;
                        WHEN OTHERS =>
                                F <= x"11111111" ;
                                --Compare <= '0';
                END CASE ;
        END PROCESS ;

                        process(F,s)
                         begin
                          if (F = x"00000000" and s = "100") then
                                        Compare <= '1';
                                                else
                                                Compare <= '0';
                                        end if;
            end process;
END B    ehavior ;
```