CprE 381, Computer Organization and Assembly-Level Programming

Lab 2 Report

Student Name _____Owen Jewell_____

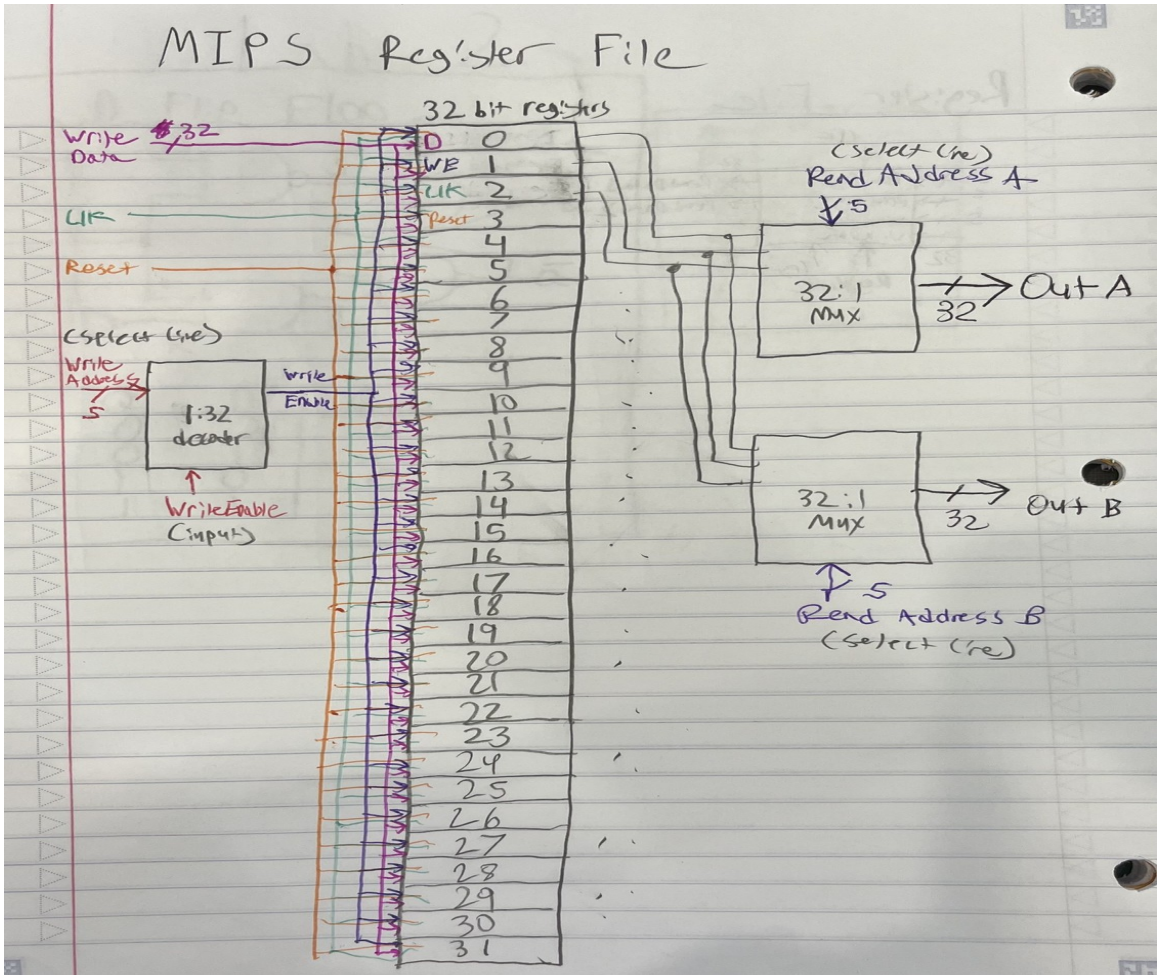
Submit a typeset pdf version of this on Canvas by the due date. Refer to the highlighted language in the lab document for the context of the following questions.

[Part 0] Describe the provided DFF in dffg.vhd in terms of edge sensitivity and reset type (active low/high and synchronous/asynchronous).

The D flip flop in dffg.vhd is asynchronous reset and synchronous write. The edge sensitivity is rising edge because when the clock is 1 is when the output signal is changed to the value of D (input). The reset is active high because when the reset value is 1 it resets the output value to 0.

[Part 2 (a)] Draw the interface description for the MIPS register file. Which ports do you think are necessary, and how wide (in bits) do they need to be?

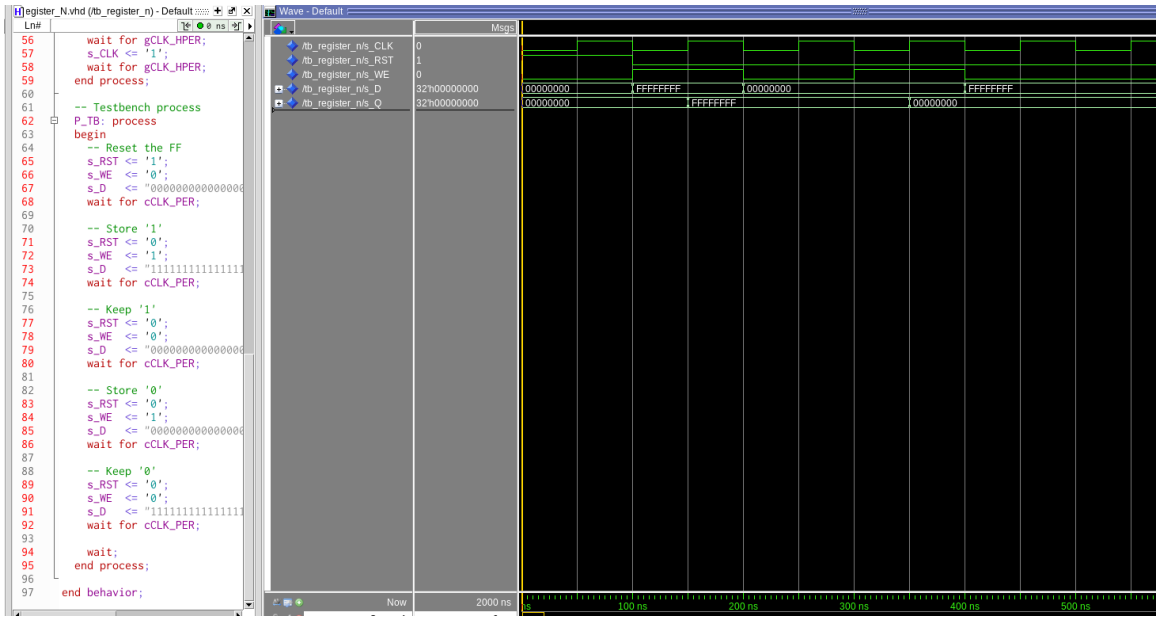
(Decoder should say 5:32, its a 5 bit address select)



[Part 2 (b)] Create an N-bit register using this flip-flop as your basis.

```
14
15  library IEEE;
16  use IEEE.std_logic_1164.all;
17
18  entity register_N is
19  generic(N : integer := 32);
20    port(i_CLK        : in std_logic;                          -- Clock input
21         i_RST        : in std_logic;                          -- Reset input
22         i_WE         : in std_logic;                          -- Write enable input
23         i_WD         : in std_logic_vector(N-1 downto 0);     -- Write Data value in
24         o_Q          : out std_logic_vector(N-1 downto 0));    -- Data value output
25  end register_N;
26
27  architecture structural of register_N is
28
29    component dffg is
30      port(i_CLK        : in std_logic;
31           i_RST        : in std_logic;
32           i_WE         : in std_logic;
33           i_D          : in std_logic;
34           o_Q          : out std_logic);
35    end component;
36
37  begin
38
39    -- Instantiate N dffg instances.
40    G_NBit_DFFG: for i in 0 to N-1 generate
41      DFFGI : dffg port map(
42                  i_CLK    => i_CLK,
43                  i_RST    => i_RST,
44                  i_WE     => i_WE,
45                  i_D      => i_WD(i),
46                  o_Q      => o_Q(i));
47    end generate G_NBit_DFFG;
48
49  end structural;
50
51
```

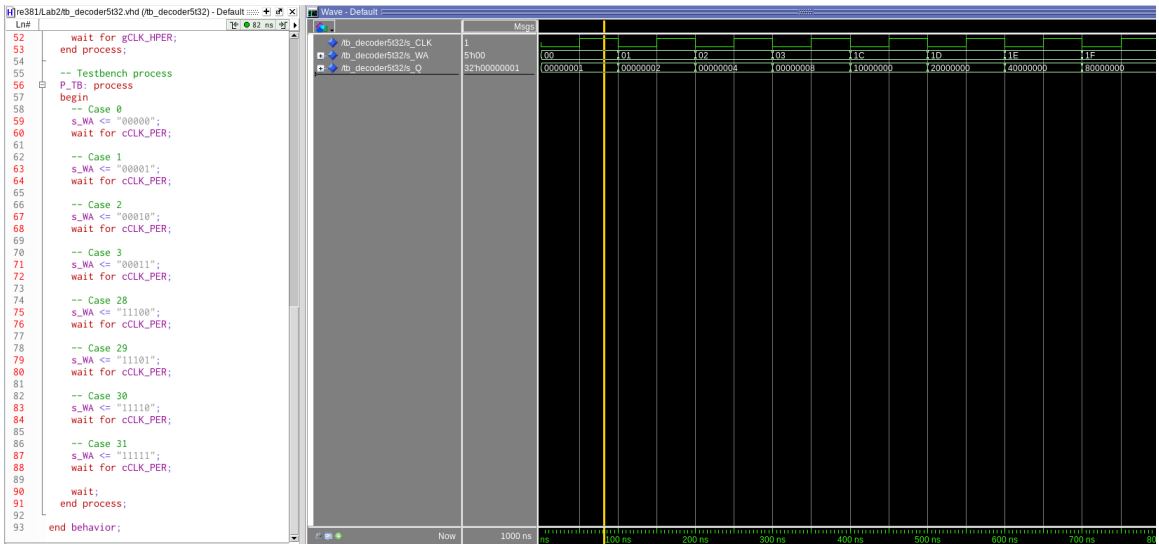
[Part 2 (c)] Waveform.



[Part 2 (d)] What type of decoder would be required by the MIPS register file and why?

The MIPS register file will need a 5:32 decoder because you need 5 bits to select registers 0-31.

[Part 2 (e)] Waveform.

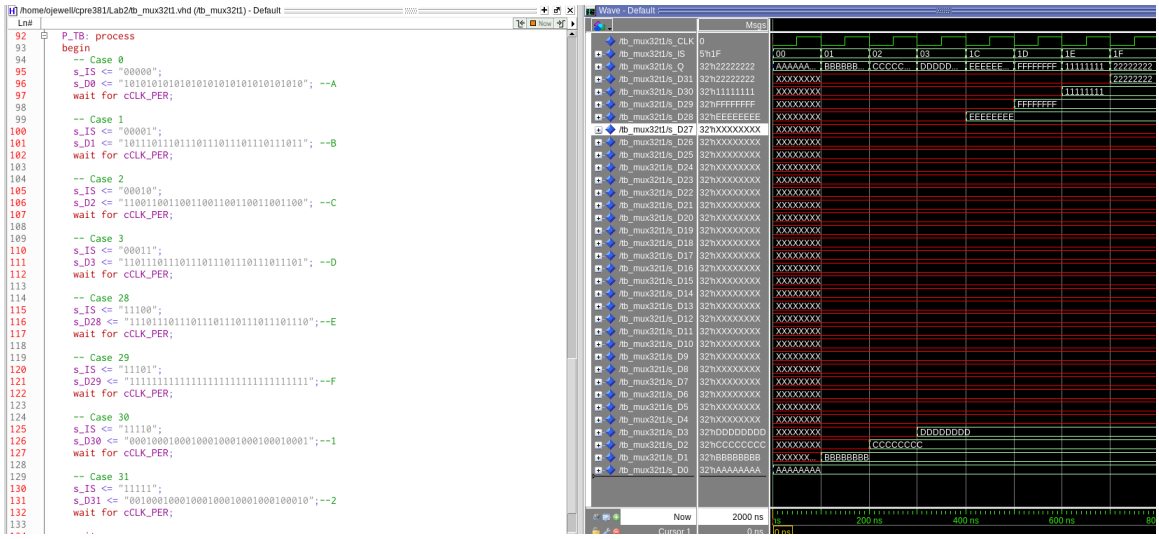


```
    wait for gCLK_HPER;
  end process;

  -- Testbench process
  P_TB: process
  begin
    -- Case 0
    s_WA <= "00000";
    wait for cCLK_PER;

    -- Case 1
    s_WA <= "00001";
    wait for cCLK_PER;

    -- Case 2
    s_WA <= "00010";
    wait for cCLK_PER;

    -- Case 3
    s_WA <= "00011";
    wait for cCLK_PER;

    -- Case 28
    s_WA <= "11100";
    wait for cCLK_PER;

    -- Case 29
    s_WA <= "11101";
    wait for cCLK_PER;

    -- Case 30
    s_WA <= "11110";
    wait for cCLK_PER;

    -- Case 31
    s_WA <= "11111";
    wait for cCLK_PER;

    wait;
  end process;

end behavior;
```

[Part 2 (f)] In your write-up, describe and defend the design you intend on implementing for the next part.

I am going to use the same strategy that I used for my 5:32 decoder for my 32:1 multiplexer. This strategy is one that I found in Free Range VDHL. The strategy is to use the dataflow method with the “with select then” statement. This statement allows me to assign a specific input vector to the output vector based on the 5 bit sleect.

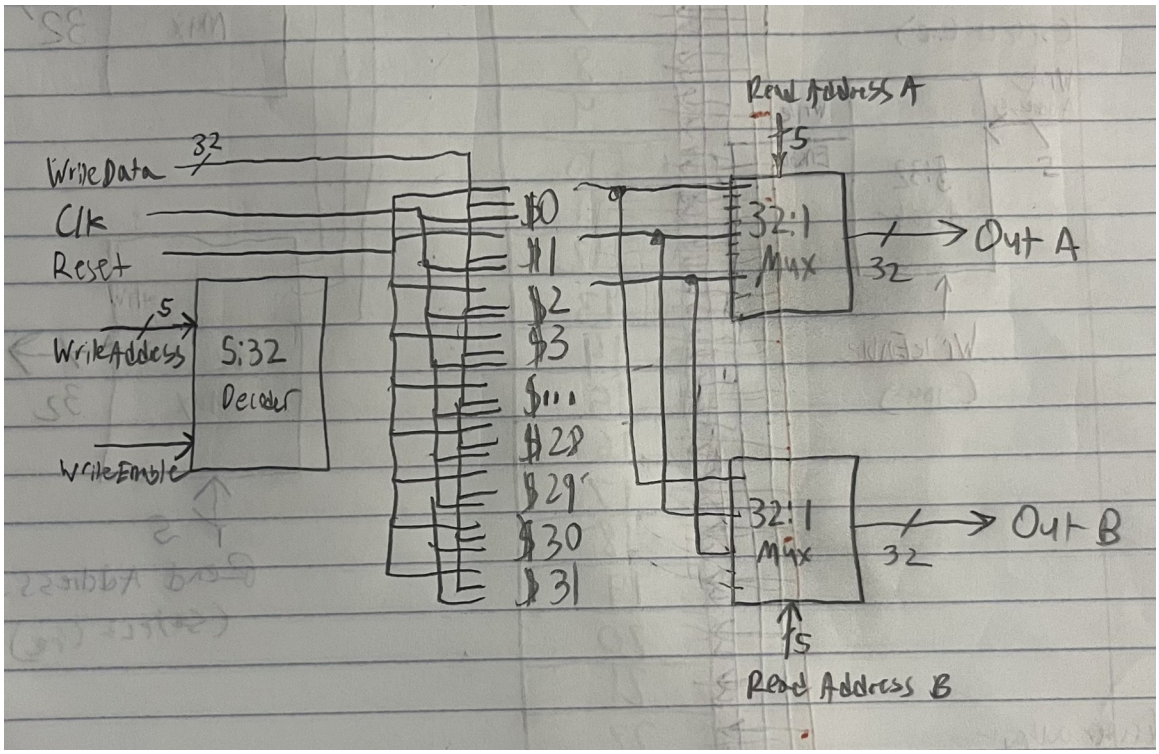
[Part 2 (g)] Waveform.



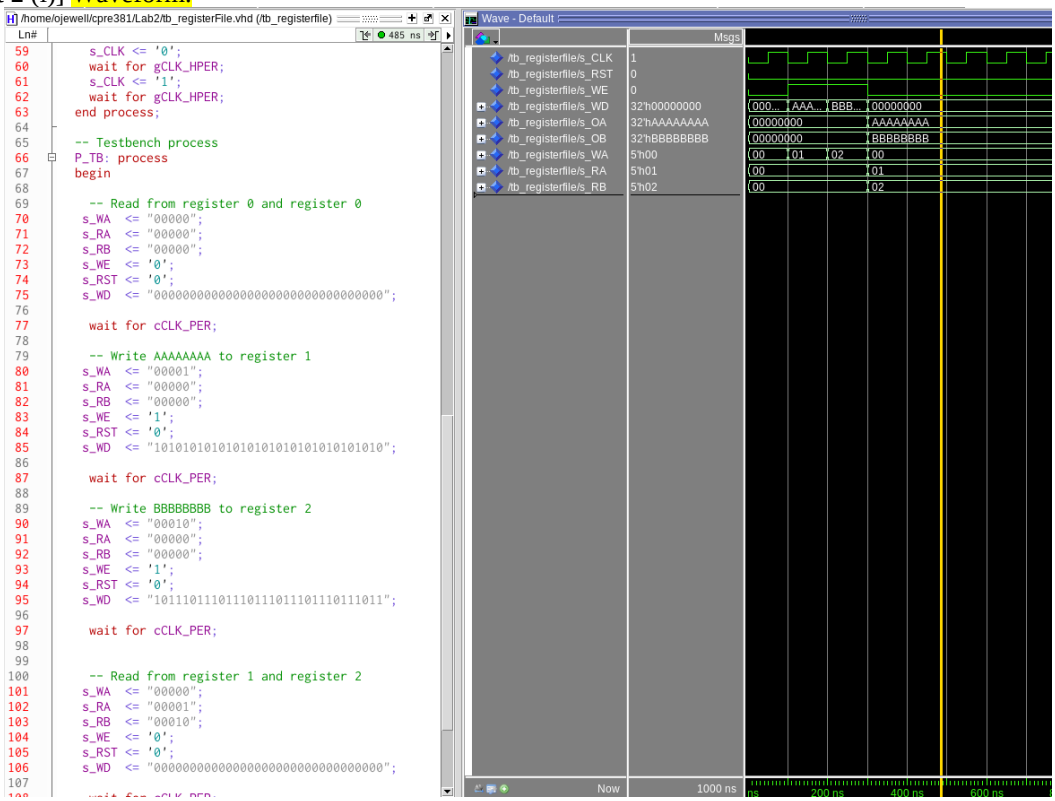
```
  P_TB: process
  begin
    -- Case 0
    s_IS <= "00000";
    s_D0 <= "10101010101010101010101010101010"; --A
    wait for cCLK_PER;

    -- Case 1
    s_IS <= "00001";
    s_D1 <= "10111011101110111011101110111011"; --B
    wait for cCLK_PER;

    -- Case 2
    s_IS <= "00010";
    s_D2 <= "11001100110011001100110011001100"; --C
    wait for cCLK_PER;

    -- Case 3
    s_IS <= "00011";
    s_D3 <= "11011101110111011101110111011101"; --D
    wait for cCLK_PER;

    -- Case 28
    s_IS <= "11100";
    s_D28 <= "11101110111011101110111011101110";--E
    wait for cCLK_PER;

    -- Case 29
    s_IS <= "11101";
    s_D29 <= "11111111111111111111111111111111";--F
    wait for cCLK_PER;

    -- Case 30
    s_IS <= "11110";
    s_D30 <= "00010001000100010001000100010001";--1
    wait for cCLK_PER;

    -- Case 31
    s_IS <= "11111";
    s_D31 <= "00100010001000100010001000100010";--2
    wait for cCLK_PER;
```

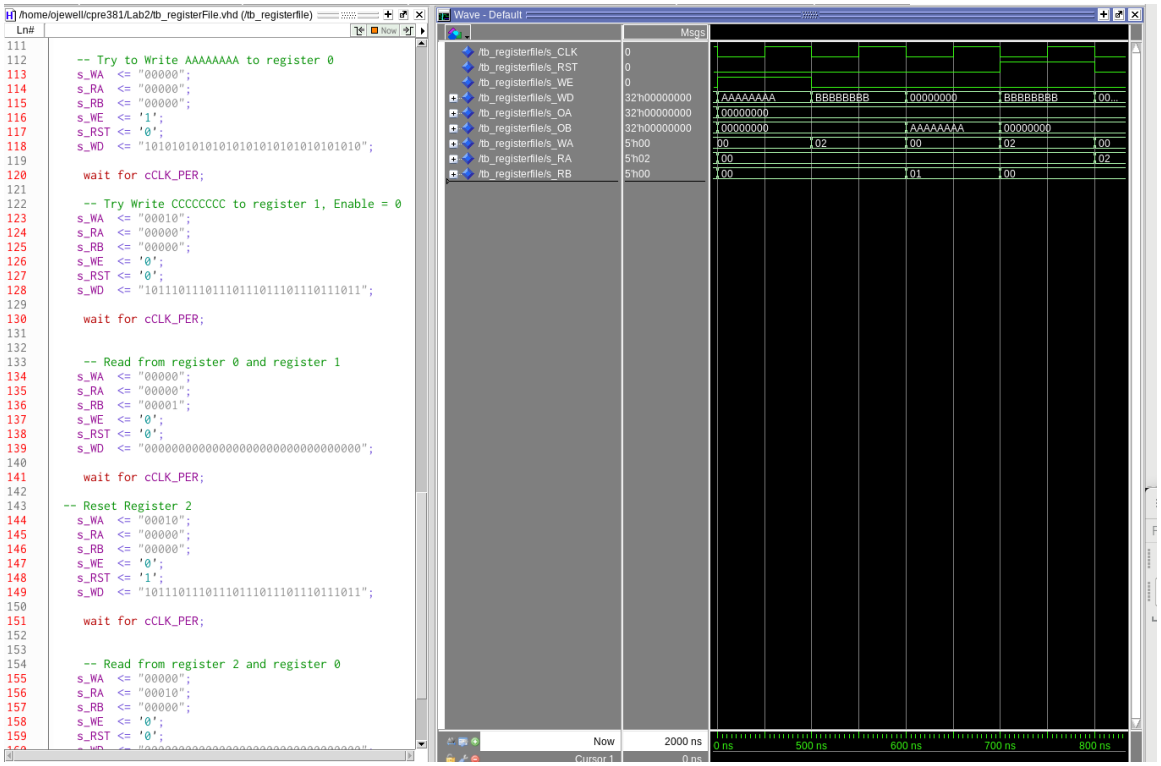
Although it may not look like it the waveform did work. I only assigned the first and last 4 in the test bench because they are in the statement so the middle ones will work if the ends do.

[Part 2 (h)] Draw a (simplified) schematic for the MIPS register file, using the same top-level interface ports as in your solution describe above and using only the register, decoder, and mux VHDL components you have created.

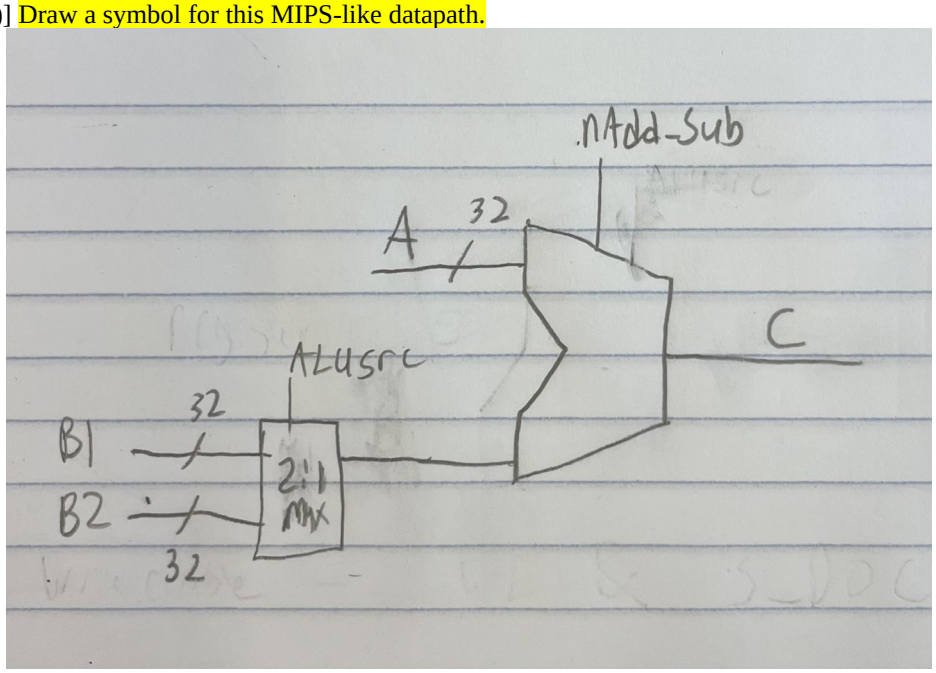


[Part 2 (i)] Waveform.

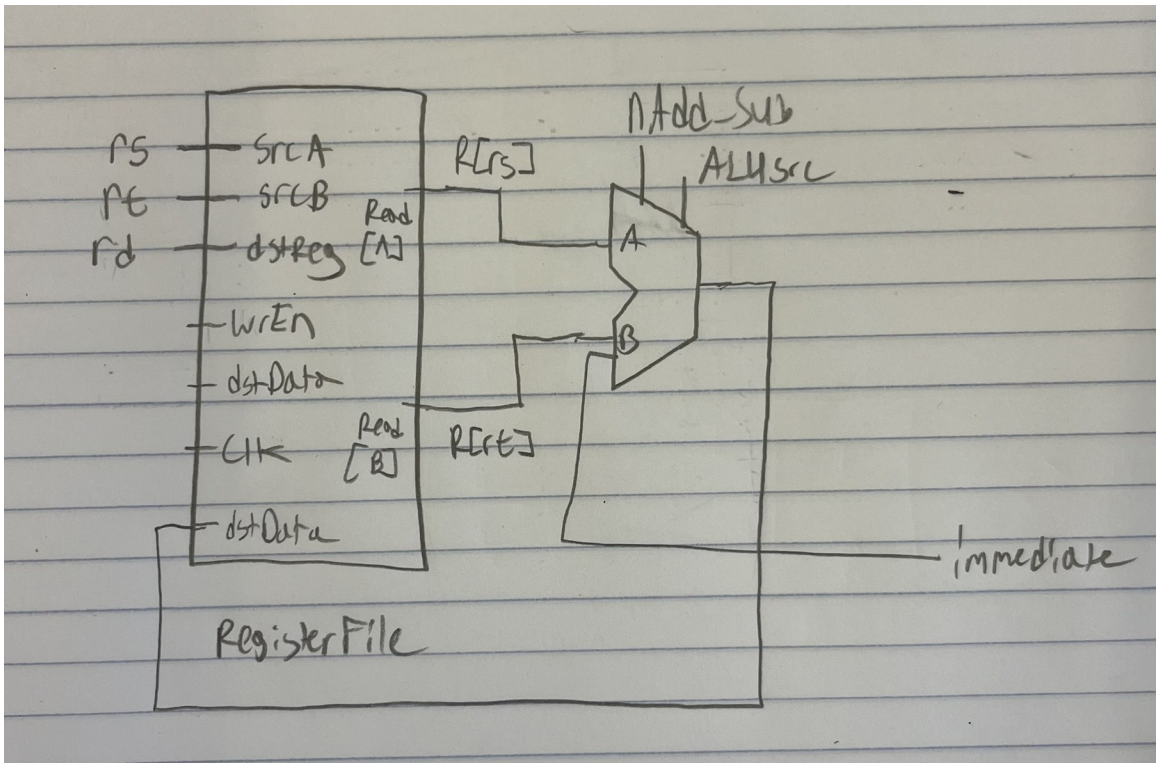




[Part 3 (b)] Draw a symbol for this MIPS-like datapath.

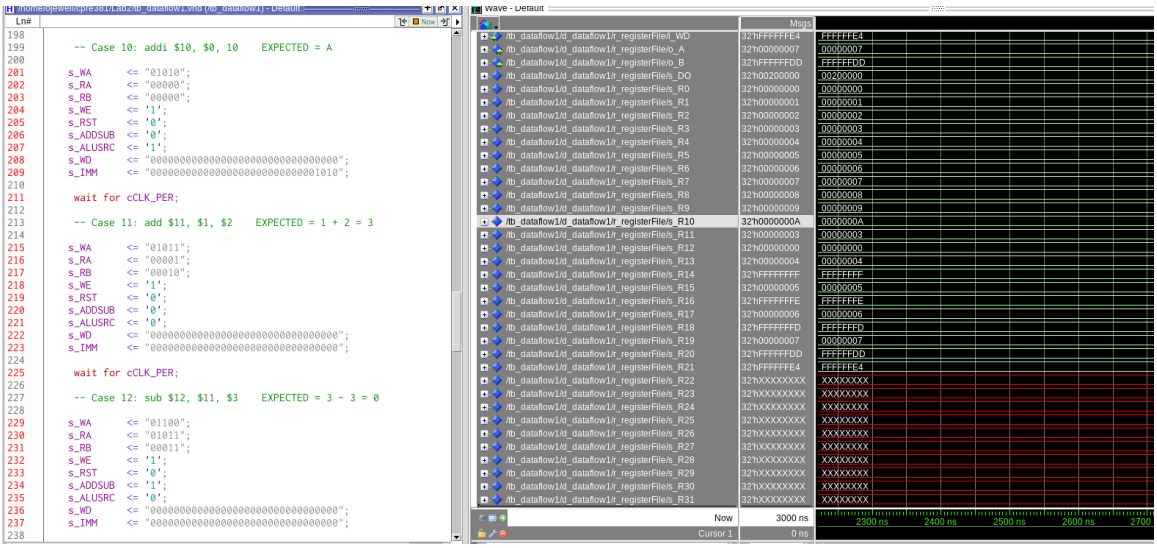


[Part 3 (c)] Draw a schematic of the simplified MIPS processor datapath consisting only of the component described in part (a) and the register file from problem (1).



[Part 3 (d)] Include in your report waveform screenshots that demonstrate your properly functioning design. Annotate what the final register file state should be.

Case 1: addi $1, $0, 1 EXPECTED = 1
Case 2: addi $2, $0, 2 EXPECTED = 2
Case 3: addi $3, $0, 3 EXPECTED = 3
Case 4: addi $4, $0, 4 EXPECTED = 4
Case 5: addi $5, $0, 5 EXPECTED = 5
Case 6: addi $6, $0, 6 EXPECTED = 6
Case 7: addi $7, $0, 7 EXPECTED = 7
Case 8: addi $8, $0, 8 EXPECTED = 8
Case 9: addi $9, $0, 9 EXPECTED = 9
Case 10: addi $10, $0, 10 EXPECTED = A
Case 11: add $11, $1, $2 EXPECTED = 1 + 2 = 3
Case 12: sub $12, $11, $3 EXPECTED = 3 - 3 = 0
Case 13: add $13, $12, $4 EXPECTED = 0 + 4 = 4
Case 14: sub $14, $13, $5 EXPECTED = 4 - 5 = -1 (FFFFFFFF)
Case 15: add $15, $14, $6 EXPECTED = -1 + 6 = 5
Case 16: sub $16, $15, $7 EXPECTED = 5 - 7 = -2 (FFFFFFFE)
Case 17: add $17, $16, $8 EXPECTED = -2 + 8 = 6
Case 18: sub $18, $17, $9 EXPECTED = 6 - 9 = -3 (FFFFFFFD)
Case 19: add $19, $18, $10 EXPECTED = -3 + 10 = 7
Case 20: addi $20, $0, -35 EXPECTED = -35 (FFFFFFDD)
Case 21: add $21, $19, $20 EXPECTED = 7 + -35 = -22 (FFFFFFE4)



As can be seen in the screenshot all of the actual register values match the expected

[Part 4 (a)] Read through the mem.vhd file, and based on your understanding of the VHDL implementation, provide a 2-3 sentence description of each of the individual ports (both generic and regular).

The generic section of the memory entity contains the definitions for two different size vectors. The first, data_width is of size 32 and will be used to represent the actual words being stored in memory. The second, addr_width, is of size 10 and is the width of the address in memory.

The port section contains 4 in ports and 5 out ports. The clk port is an in port that will represent the clock cycle. Data will be stored in memory on the positive edge of the clock. This is handled by the rising edge clock cycle. The second port is the addr port which is also an in port. The addr value is an unsigned value used to access a specific address in memory. The data port is in the third in port. This 32 bit vector will be used to store words in memory. The last in port is the we port. This is the write enable port. If this std_logic value is a 1 then data can be stored in memory, but if it is a 0 then data cannot be written. The last port is the q port which is the output port. This 32 bit vector will be used when data is being read from memory.

[Part 4 (c)] Waveforms.

[Part 5 (a)] What are the MIPS instructions that require some value to be sign extended? What are the MIPS instructions that require some value to be zero extended?

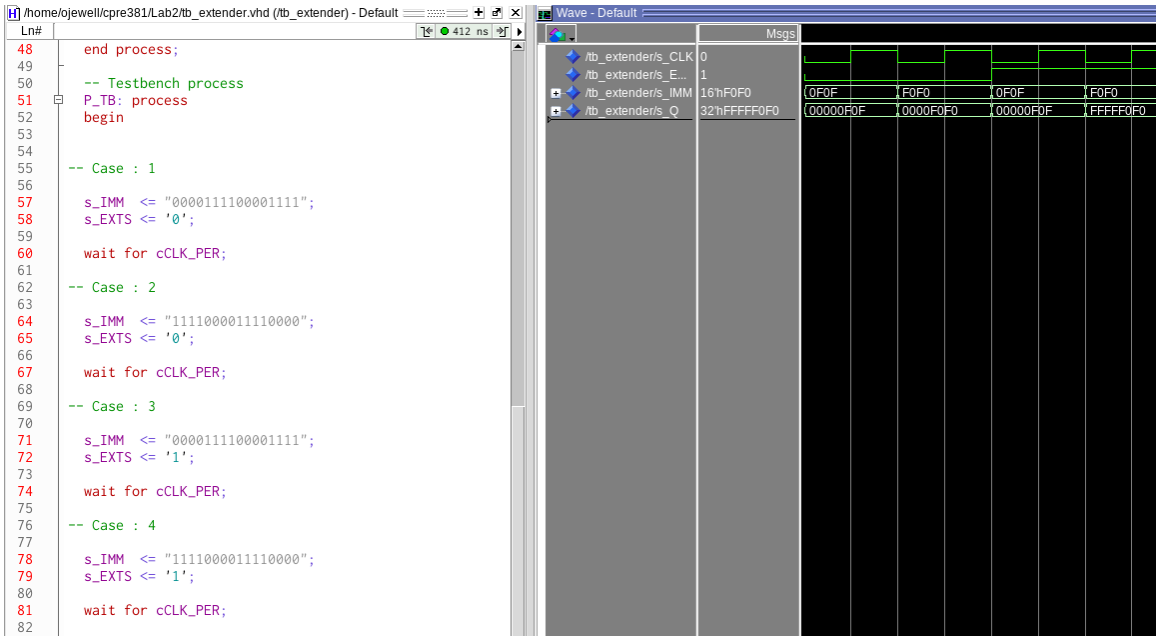
Addi,addiu, lbu, lhu, ll, lw, slti, sltiu, sb, sc, sh, sw all require a value to be sign extended.

Andi, ori, require a value to be zero extended.

[Part 5 (b)] what are the different 16-bit to 32-bit “extender” components that would be required by a MIPS processor implementation?

First the component needs to check the 1 bit select that indicates either a 0 extend or a sign extend. If it os a 0 extend it is pretty simple to iterate through bits 16-31 and add 0’s. Else if it is a sign extend you iterate though bits 16-31 and set them to input bit 15).

[Part 5 (d)] Waveform.

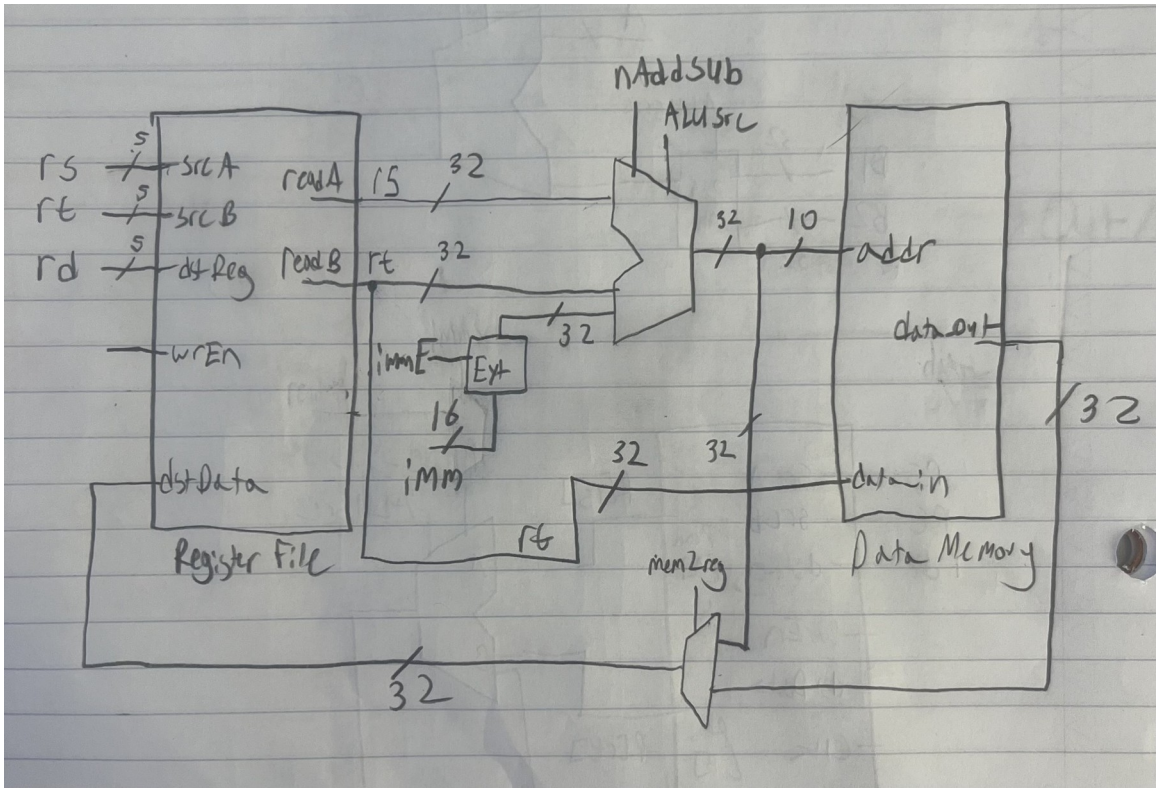


As can be seen in the screenshot the first two test cases 0 extend a number with a 0 and then a 1 in the most significant bit position. Then in the next two cases I sign extend a number with a 0 and a number with a 1 in the most significant bit position.

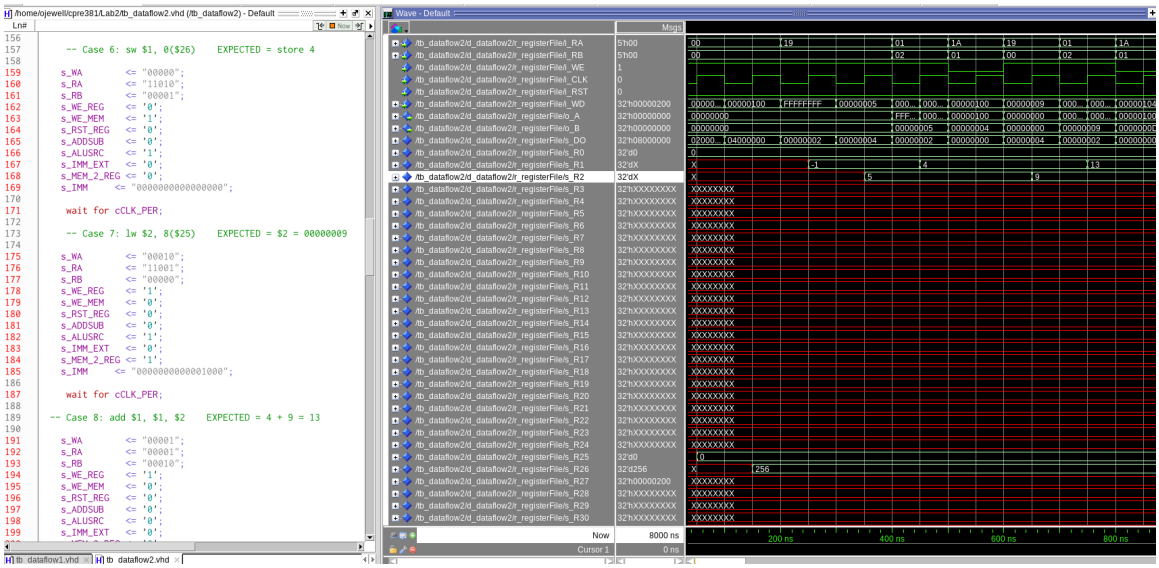
[Part 6 (a)] what control signals will need to be added to the simple processor from part 2? How do these control signals correspond to the ports on the mem.vhd component analyzed in part 3?

I will need to add a new signal that is the write enable for data memory. This directly corresponds to the write enable of mem.vhd. I will also need the immediate input and the extension type selector for the immediate extender. I also need to take the output of the mini alu and the the output of the memory module and mux these into the register data in port to load from memory or operation. Lastly there will need to be a signal from R[rt} output of register file to data_in of mem.vhd in order to store into memory.

[Part 6 (b)] Draw a schematic of a simplified MIPS processor consisting only of the base components used in part 2, the extender component described in part 4, and the data memory from part 3.



[Part 6 (c)] Waveform.



This output is using the actual offset of 4 as specified in the instruction. The next pictures will use offsets of 0,1,2,3,4,5,6 instead of 0,4,8,12,16,20,24.