CprE 381, Computer Organization and Assembly-Level Programming

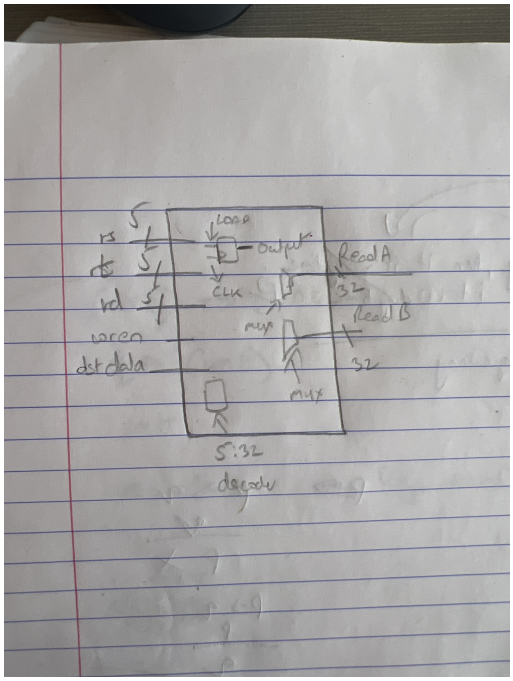
Lab 2 Report

Student Name Varun Advani

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.

PreLab: The D Flip-Flop given in dffg_vhd is a rising edge sybchronous D Flip-Flop, which means that is has synchronous write and asynchronous reset

[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?



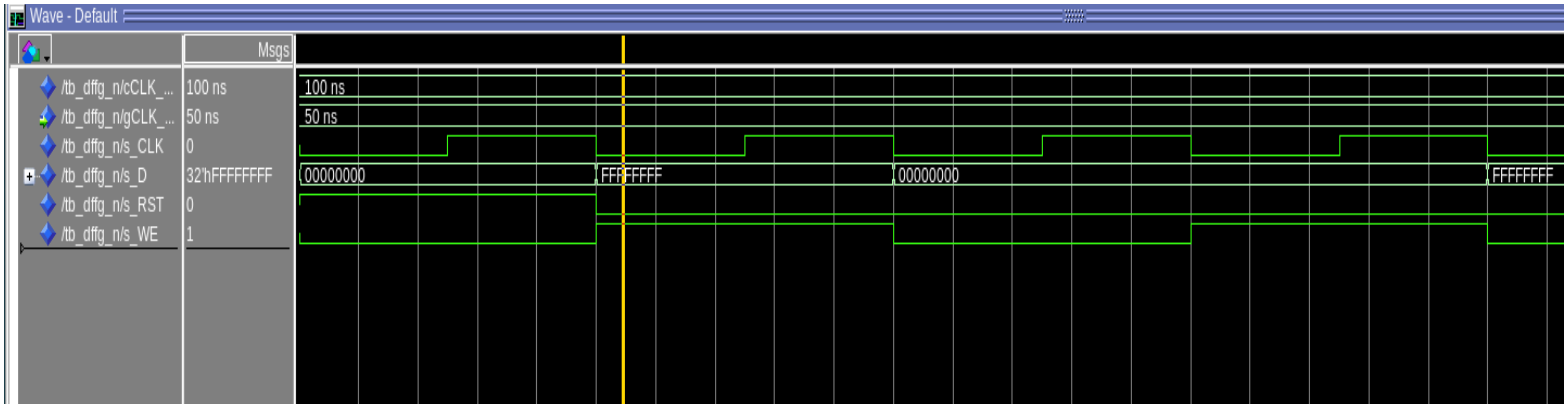
The ports required for a 32-bit MIPS register file would be 1 write port with a 5-bit write address, 1 32-bit Write Data port, A clock and a reset both of width 1 bit, and 2 32-bit read ports to output the data written into the register file.

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

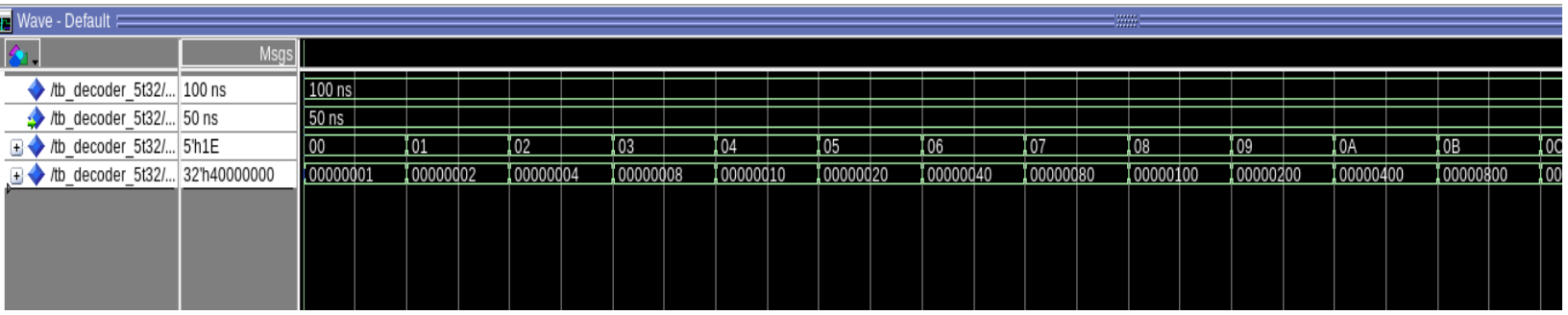
dffg_N.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;


entity dffg_N is
  generic(N : integer := 32);
  port(i_CLK        : in std_logic;
       i_RST        : in std_logic;
       i_WE         : in std_logic;
       i_D          : in std_logic_vector(N-1 downto 0);
       o_Q          : out std_logic_vector(N-1 downto 0));
end dffg_N;


architecture structural of dffg_N is

  component dffg is
  port(i_CLK        : in std_logic;
       i_RST        : in std_logic;
       i_WE         : in std_logic;
       i_D          : in std_logic;
       o_Q          : out std_logic);
  end component;

begin


  G_NBit_DFFG: for i in 0 to N-1 generate
    ONESCOMPI: dffg port map(
               i_CLK     => i_CLK,
               i_RST     => i_RST,
               i_WE      => i_WE,
               i_D       => i_D(i),
               o_Q       => o_Q(i));
  end generate G_NBit_DFFG;

end structural;
```

[Part 2 (c)] Waveform.



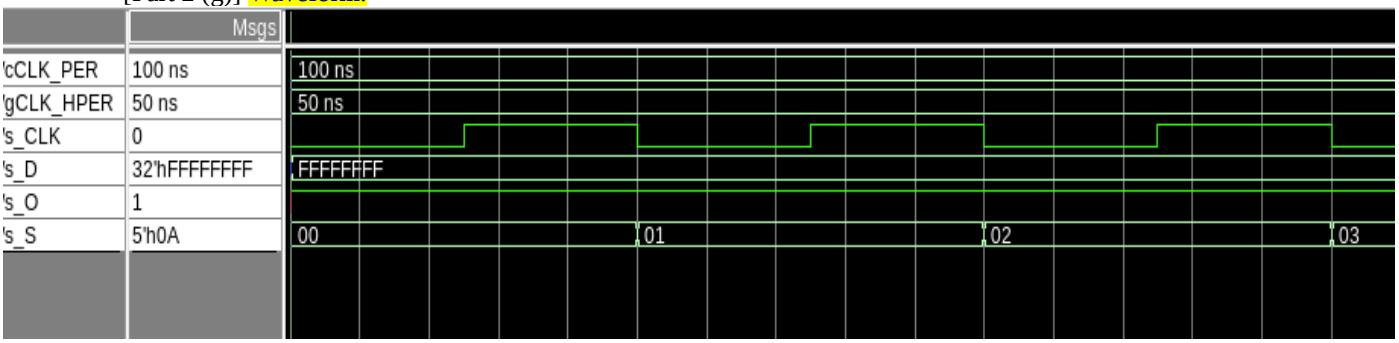
[Part 2 (d)] What type of decoder would be required by the MIPS register file and why?
A 5 : 32 decoder would be required for a MIPS register file because a a decoder is one hot encoded and hence each of the 32 outputs would act as select lines for the 32 registers. This is faciliatated with a 5 bit Write Address which is the input of the decoder.
[Part 2 (e)] Waveform.

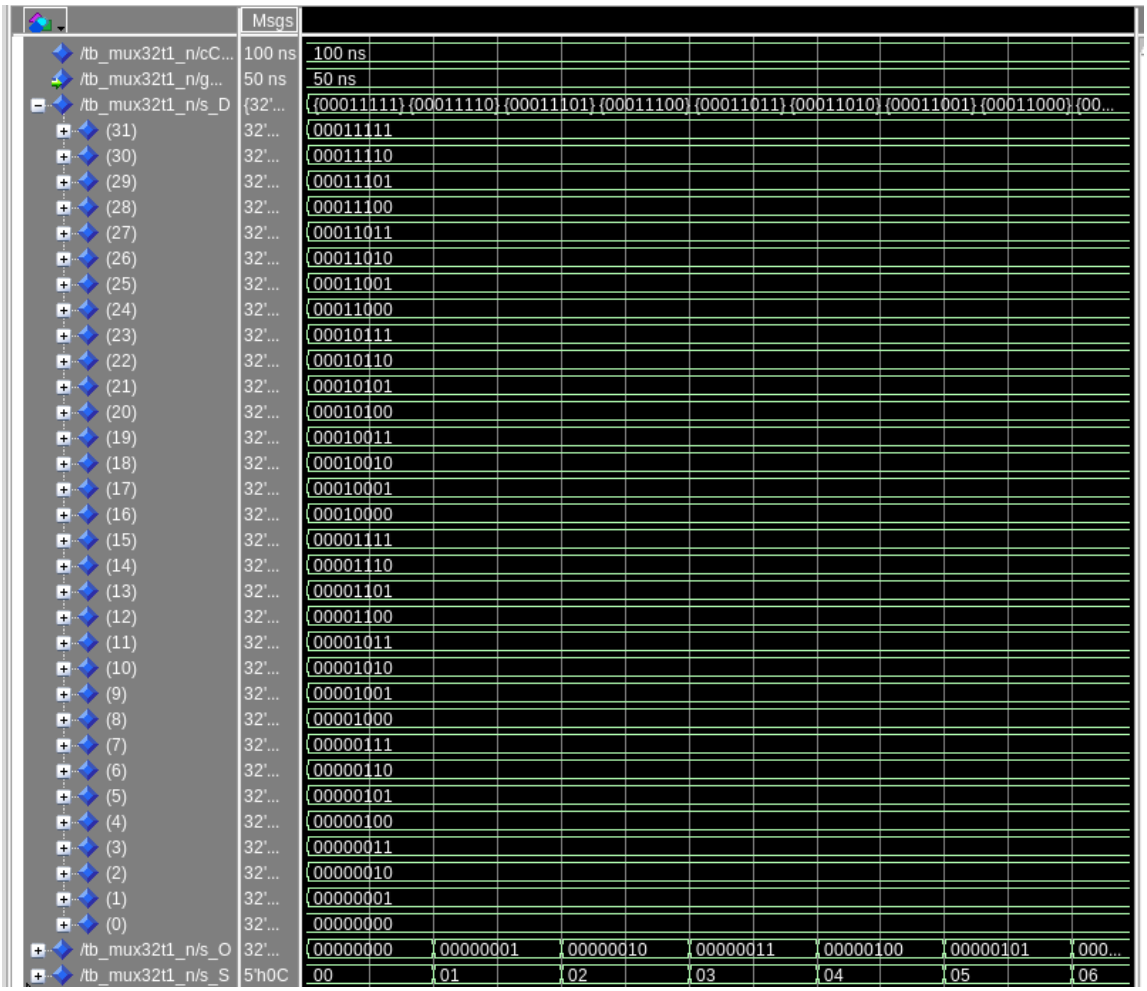


[Part 2 (f)] In your write-up, describe and defend the design you intend on implementing for the next part.

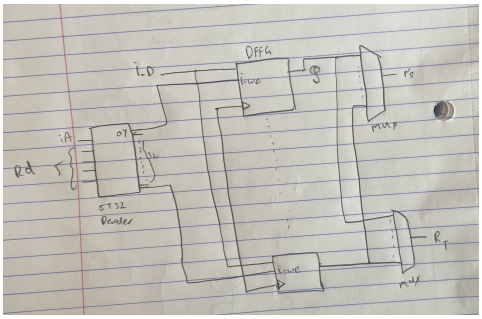
I created a new package called std_logic_matrix and created an array of 32 values for my 32-bit 32 to 1 MUX. I then used a loop to assign the output based on the 5-bit select line. I believe this design is a compact an efficient way in terms of code reusabilities when it comes to mapping components in the high level design file of the MIPS register file.

[Part 2 (g)] Waveform.

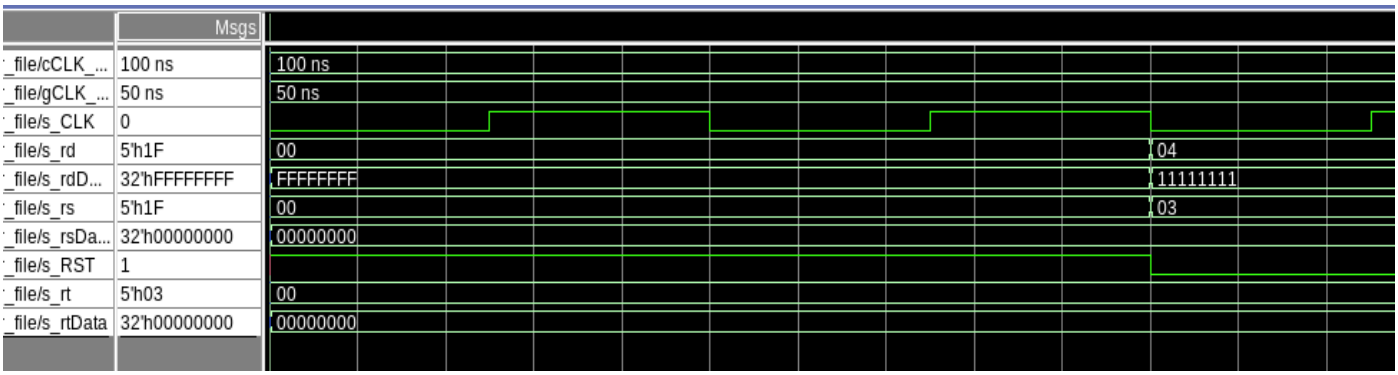


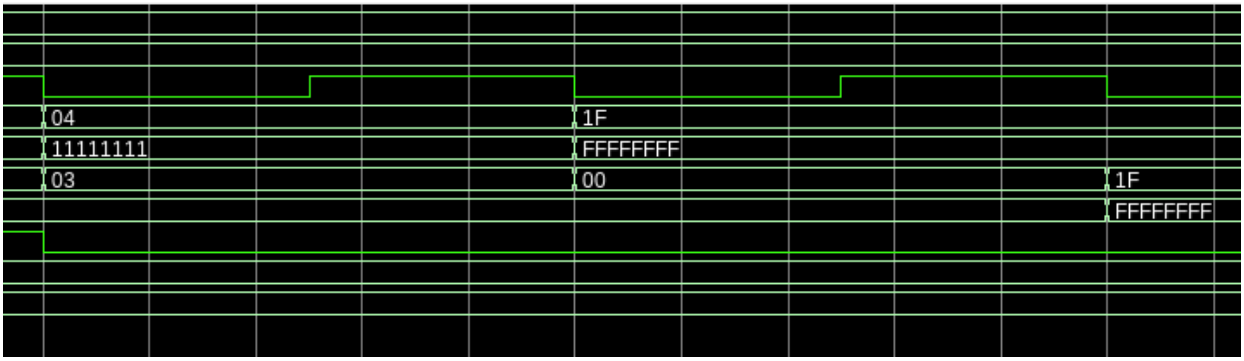


[Part 2 (h)] Draw a (simplified) schematic for the MIPS register file, using the same top-level interface ports as in your solution describe above and using only the register, decoder, and mux VHDL components you have created.

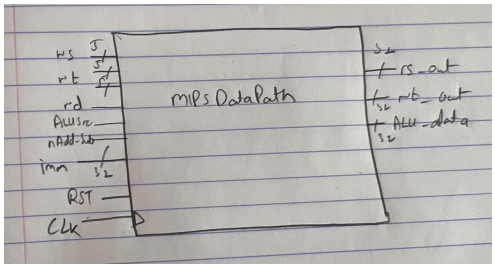


[Part 2 (i)] Waveform.

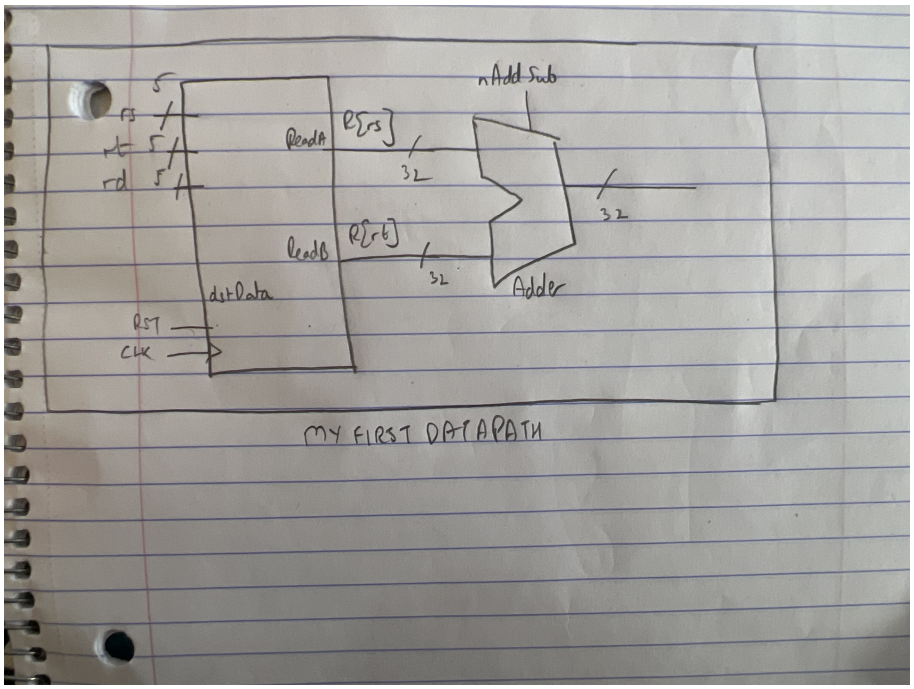




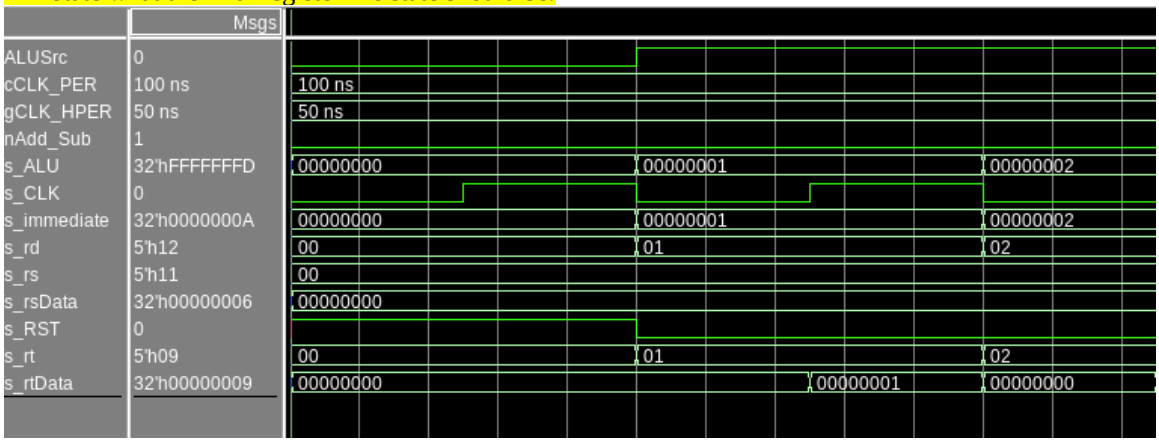
[Part 3 (b)] Draw a symbol for this MIPS-like datapath.

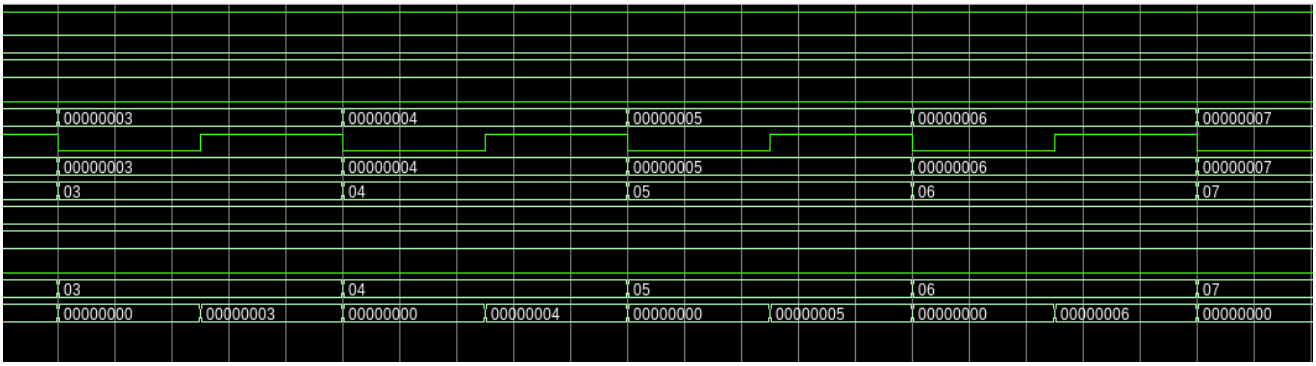


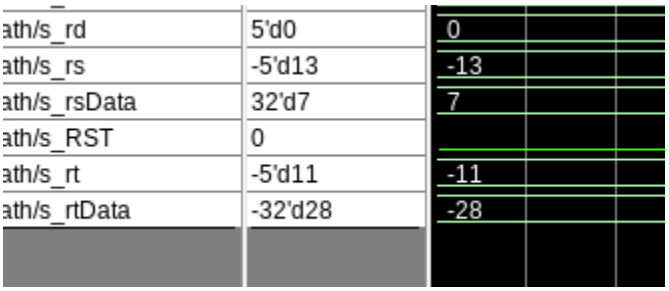
[Part 3 (c)] Draw a schematic of the simplified MIPS processor datapath consisting only of the component described in part (a) and the register file from problem (1).



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



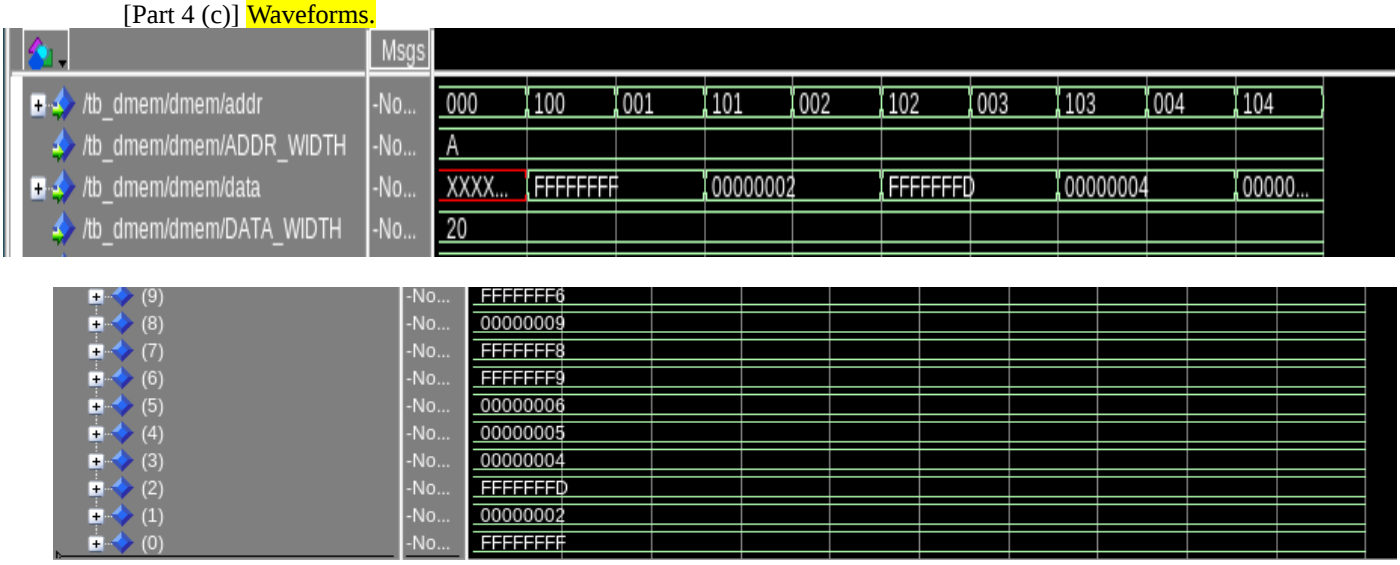




At the final stage, RtData would have the value of -28 or FFFFFFE4 in hex

[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 generics in mem.vhd setup the storage of some 32-bit data into a 10 bit hex memory location based on the select bit 'we' which would store data in memory if it is 1. The port q is the output from the memory module, that is used mapped to Rd from either the ALU or the destination data on the register file. The memory module defines are array of memory locations, and a ram signal to maps the values to their memory locations.

[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?

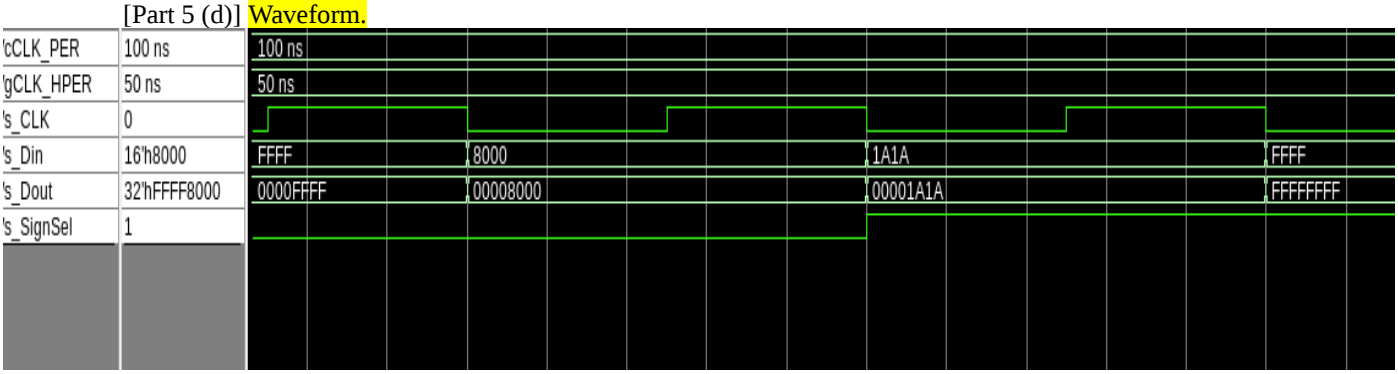
The MIPS instructions that require some value to be sign extended at load, store, and add immediate (Signed and unsigned). Some instructions that need to be zero extended are the logical operations such as and, or, and xor.

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

The 2 possible scenarios where the MIPS processor would use a sign extender would be to either extend the bits 16 – 31 with 0's, or to extend the bits 16 – 31 with the bit representation of the 15th bit (Either 0 or 1). The sign extender would therefore take in a 16 bit input, and based on a select decide to whether extend the

bits with 0’s or by the sign of the most significant bit of the 16 bit input. The output of the sign extender poses as the alternative input for R[t].

[Part 5 (d)] Waveform.



[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?

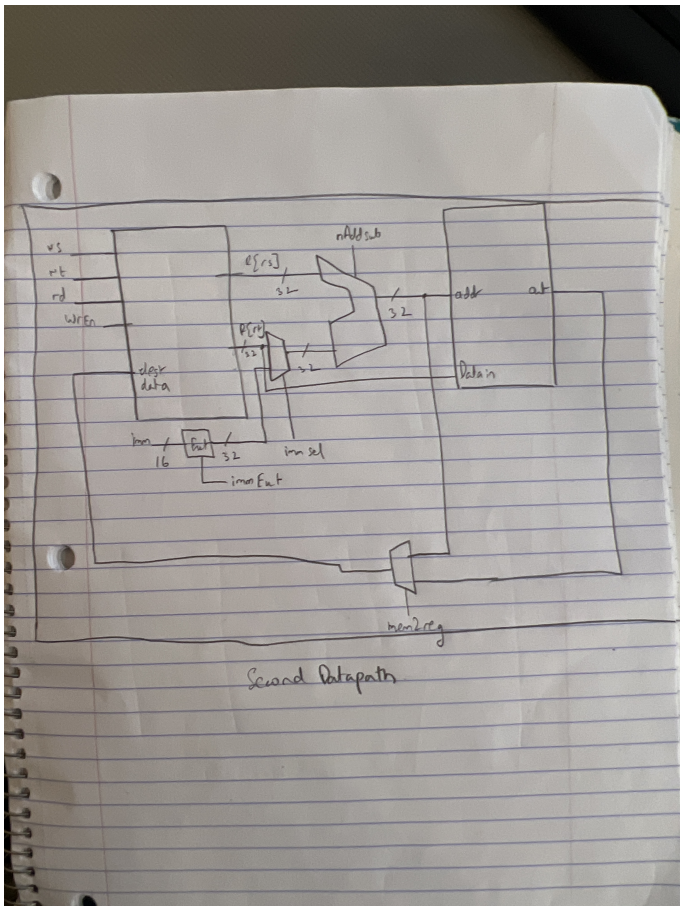
The control signals that would need to be added would be:
1. A write enable for the memory, which would carry out operations such as storing values to memory. It is mapped as the port ‘we’ in mem.vhd.

2. rdDataSel, which controls the output from a 2t1 MUX that selects whether to load values from memory, or from the ALU

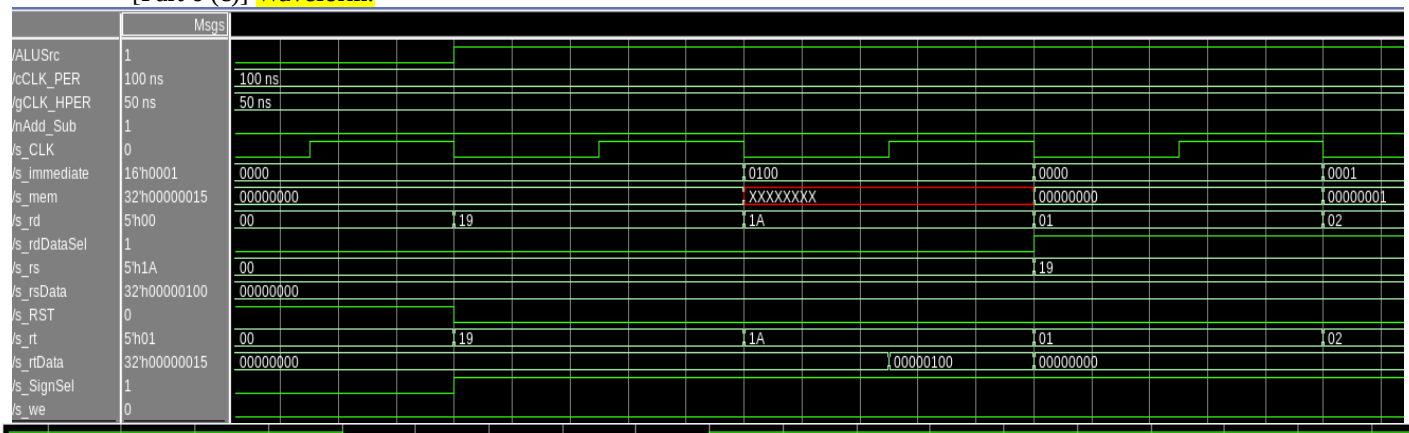
3. A control signal which carries the output of the 16t32 bit extender, that decides whether the 16 bit input would be zero extended or sign extended, depending on the MIPS operation.

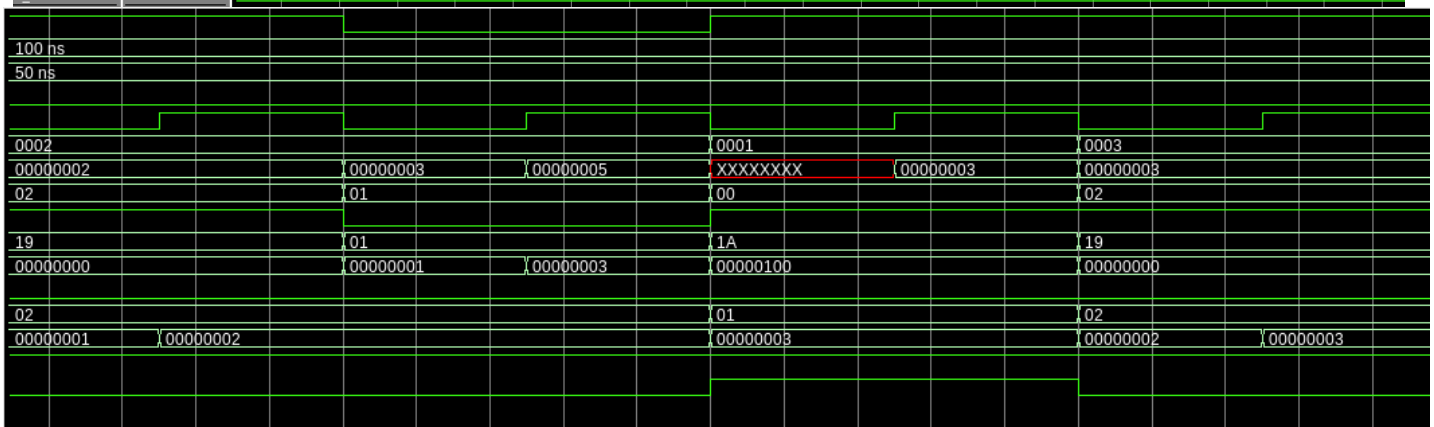
4. ImmSel, which is a control signal for the MUX that decides whether the output would be loaded onto the ALU from either the extender or the register file, depending on what MIPS instruction needs to be executed.

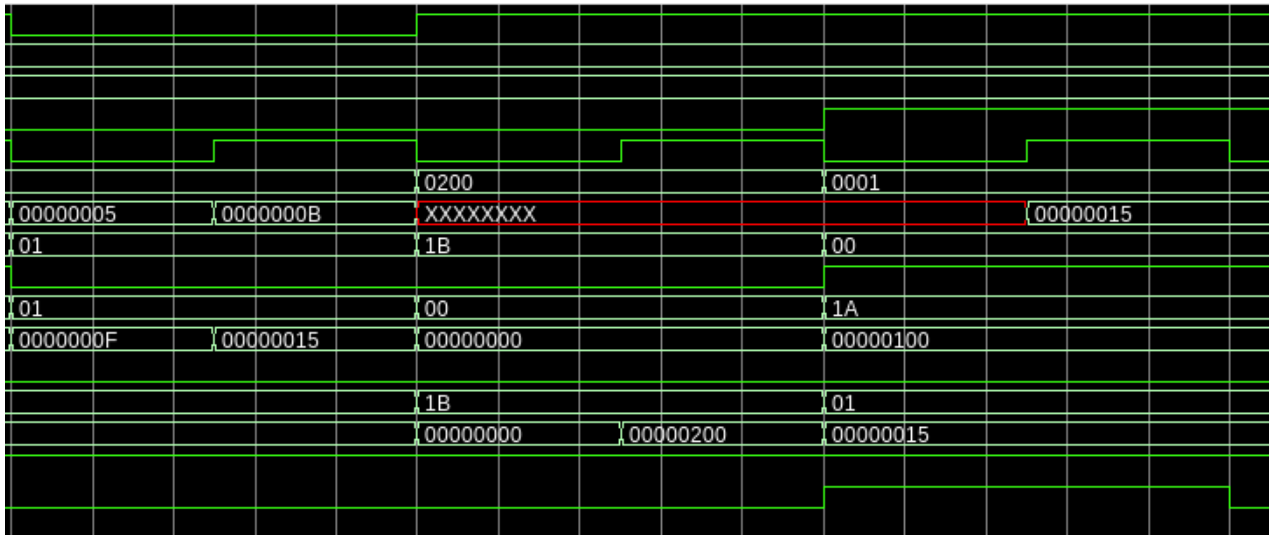
[Part 6 (b)] Draw a schematic of a simplified MIPS processor consisting only of the base components used in part 2, the extender component described in part 4, and the data memory from part 3.



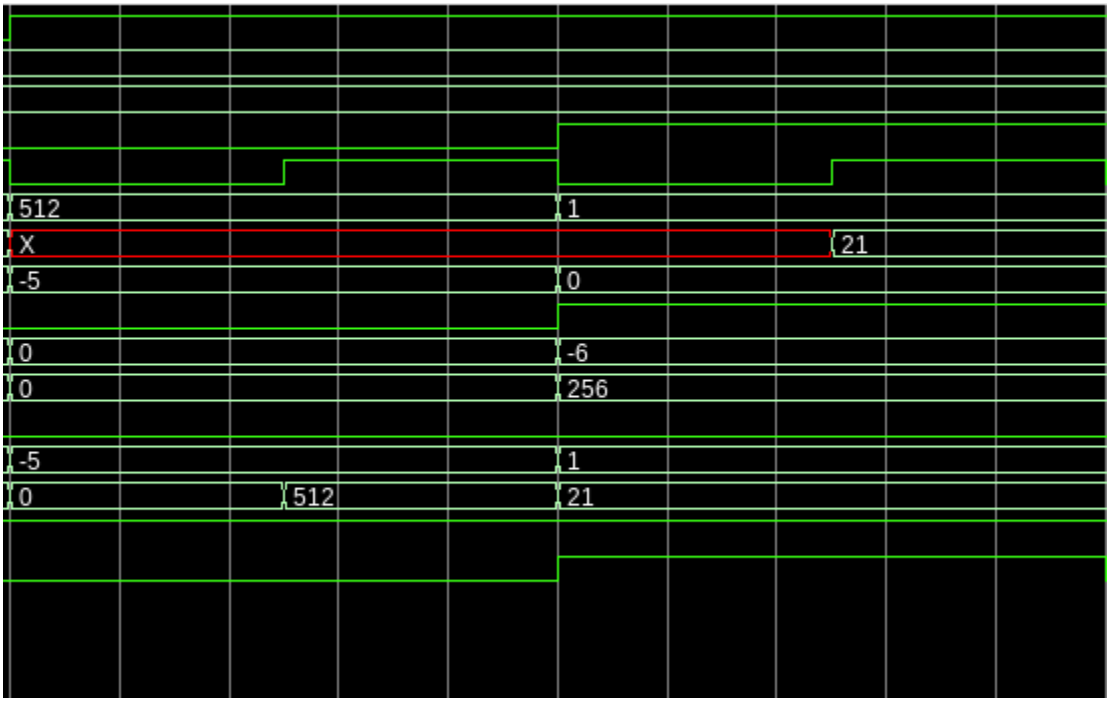
[Part 6 (c)] Waveform.







Final Register state in HEX



Final Register state in decimal