CprE 381, Computer Organization and Assembly-Level Programming

Lab 2 Report

Student Name Trent Walraven

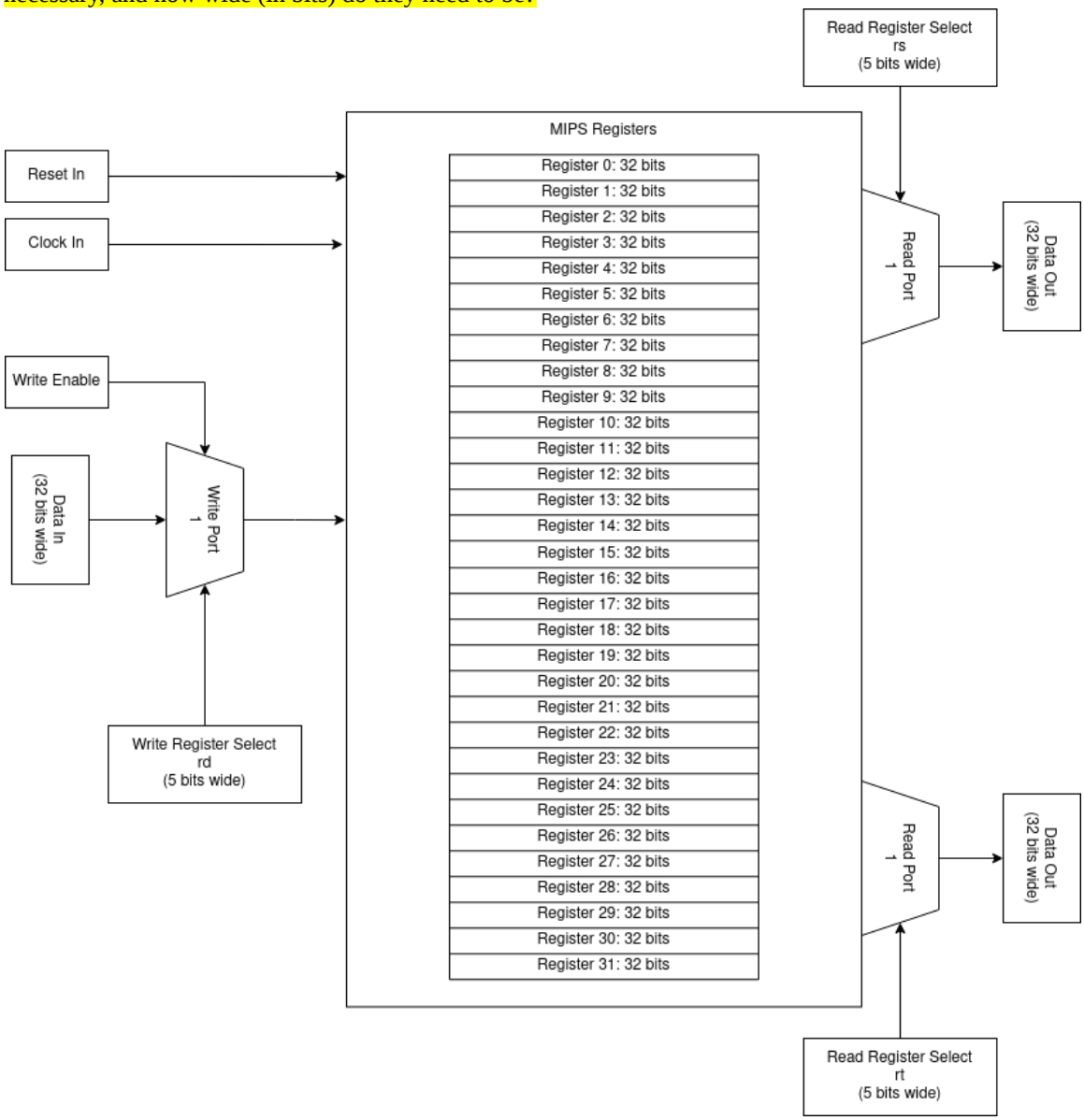
Submit a typeset pdf version of this on Canvas by the due date. Refer to the highlighted language in the lab document for the context of the following questions.

After talking with the substitute TA for my lab, he suggested I put this here, to show I did this since my main TA was not in the lab.

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

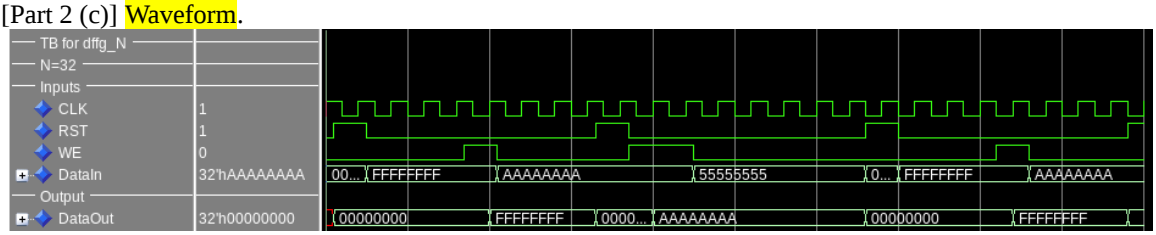
This flip flop looks has active high, where I high value is used to select when a new value is being loaded in, and when the value should be cleared. This has a rising edge sensitivity where the value will change based on the values on the rising edge of the clock. This makes it a synchronous since it will change with all other devices on the clock.

[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?



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

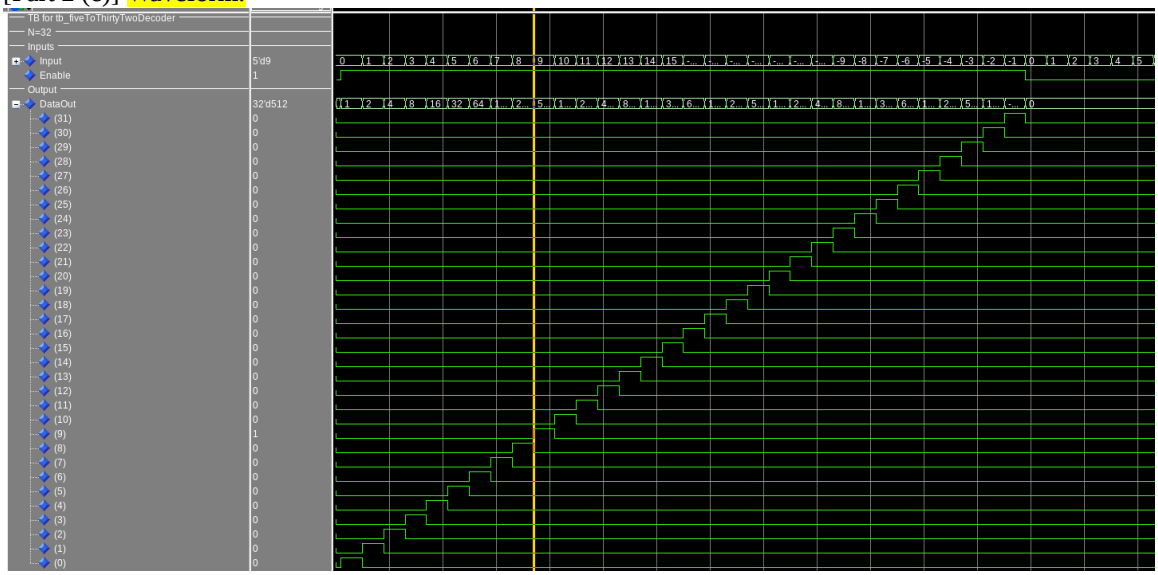
[Part 2 (c)] Waveform.



This waveform shows values being loaded into the output register on write-enable and positive clock edge, and resetting when reset is enabled.

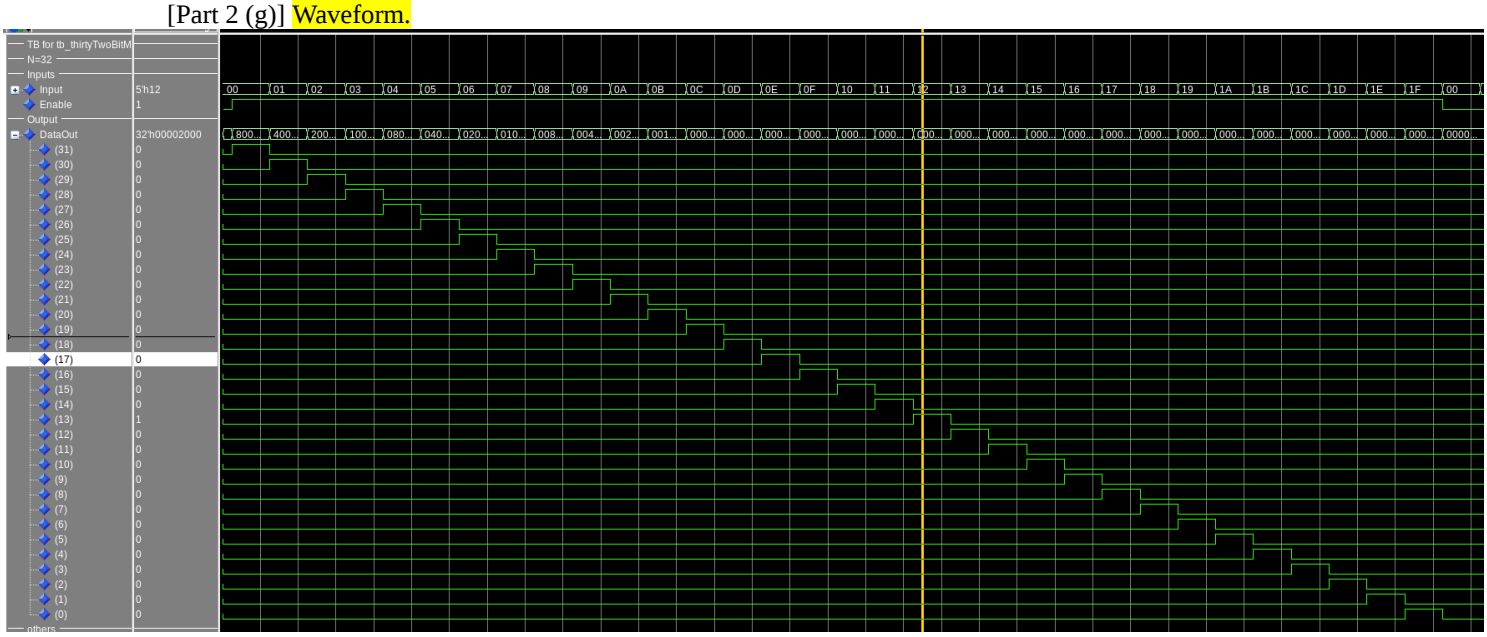
[Part 2 (d)] What type of decoder would be required by the MIPS register file and why?
The MIPS register file will need a 5:32 to decoder to properly set the individual write enables for a single register at a time.

[Part 2 (e)] Waveform.



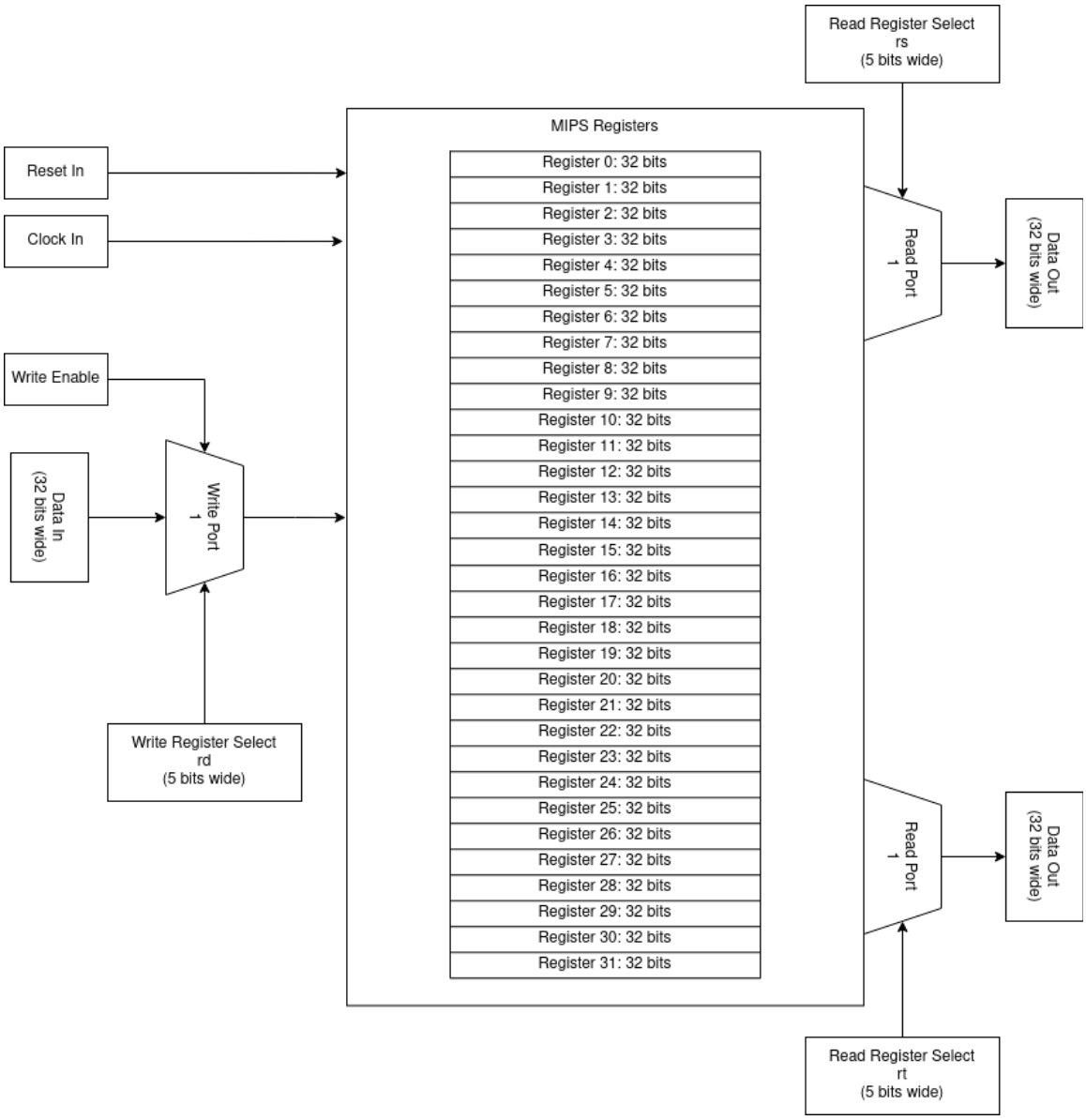
This shows the decoder going through all input values of the decoder with write enabled, and showing that no output is given when the enable is turned off.
[Part 2 (f)] In your write-up, describe and defend the design you intend on implementing for the next part.
I'm planning on using the dataflow method similar to the decoder, with a when statement based on the incoming select value will direct the correct 32 bit input to the output. This to me seems like the quickest method as it directly takes the input, and uses it for directing the output value with little extra logic needed. Making it the most likely to take the least amount of time.

[Part 2 (g)] Waveform.

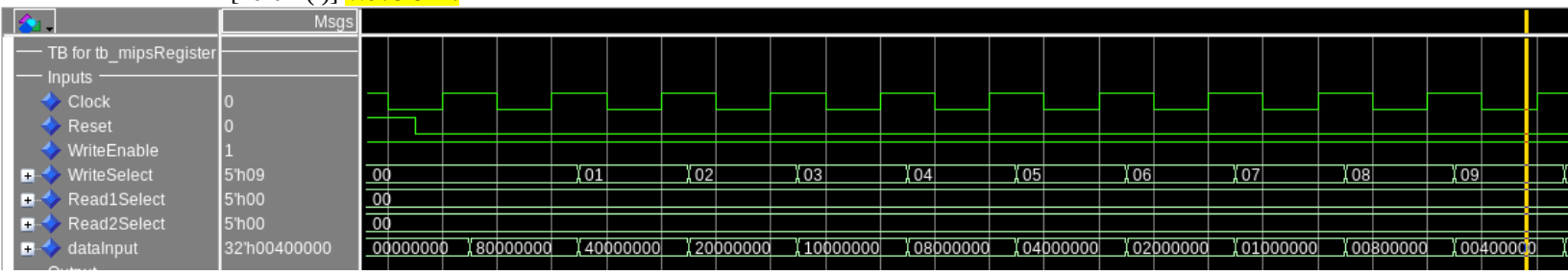


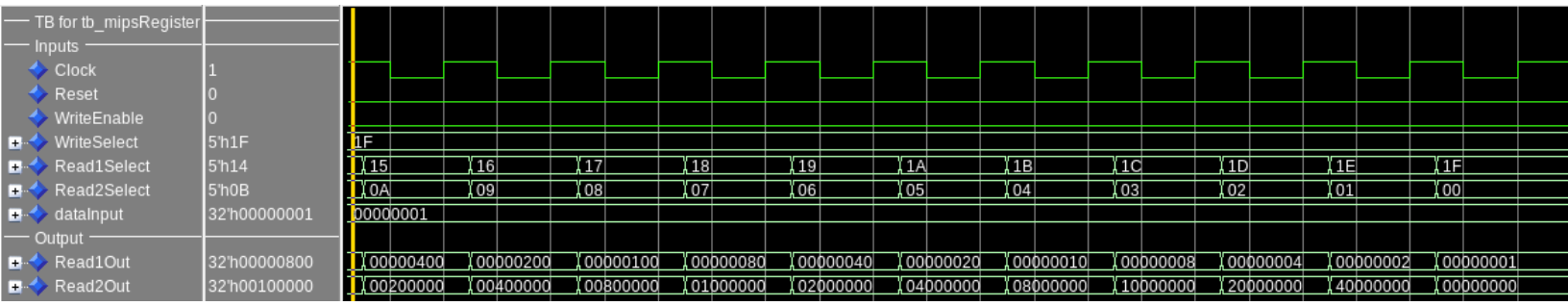
This waveform shows the 32 bits output from the selected input with data0 having bit 31 on, through data31 having bit 0 on. It also shows when enable is turned off that no output is given

[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.



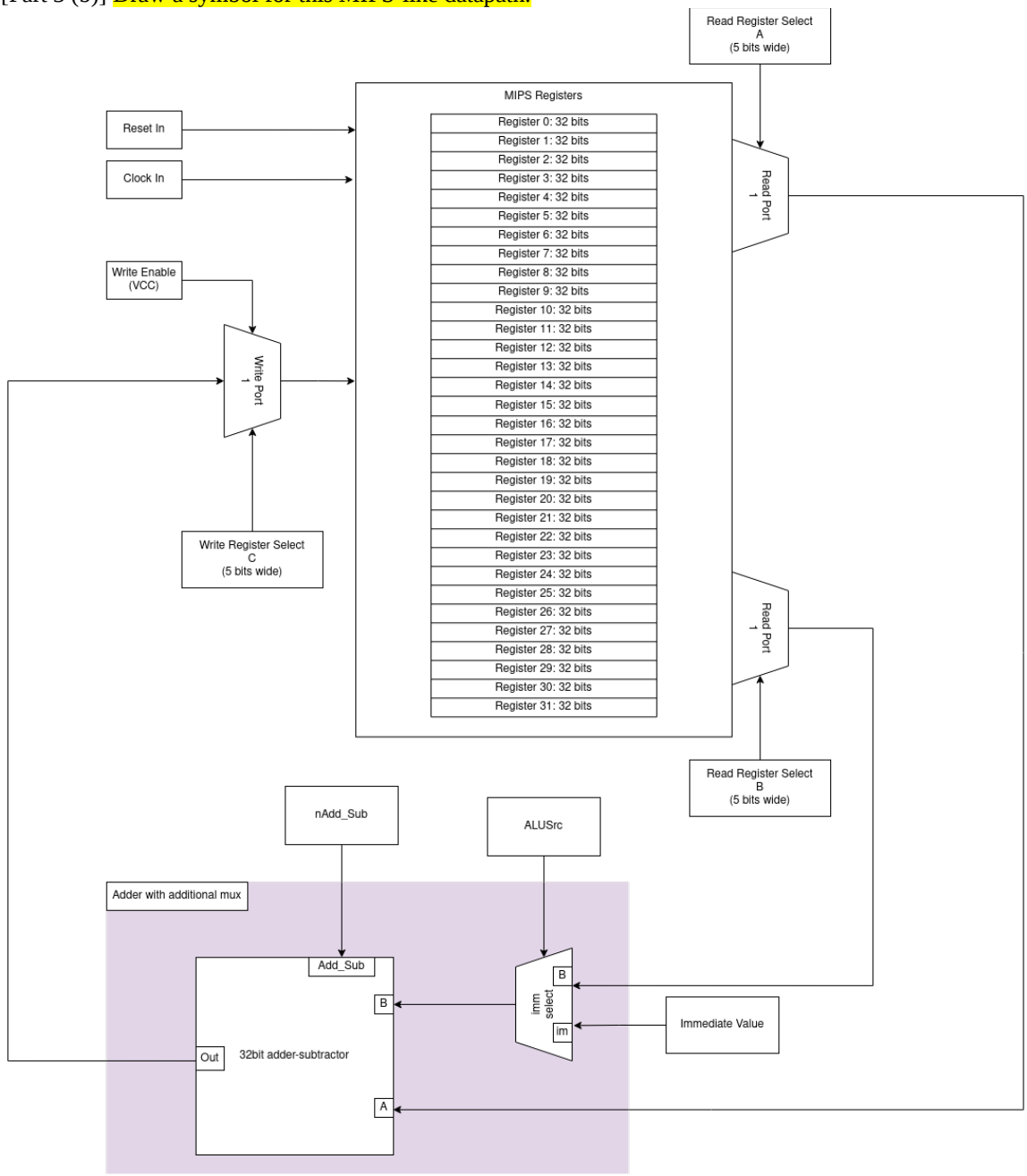
[Part 2 (i)] Waveform.



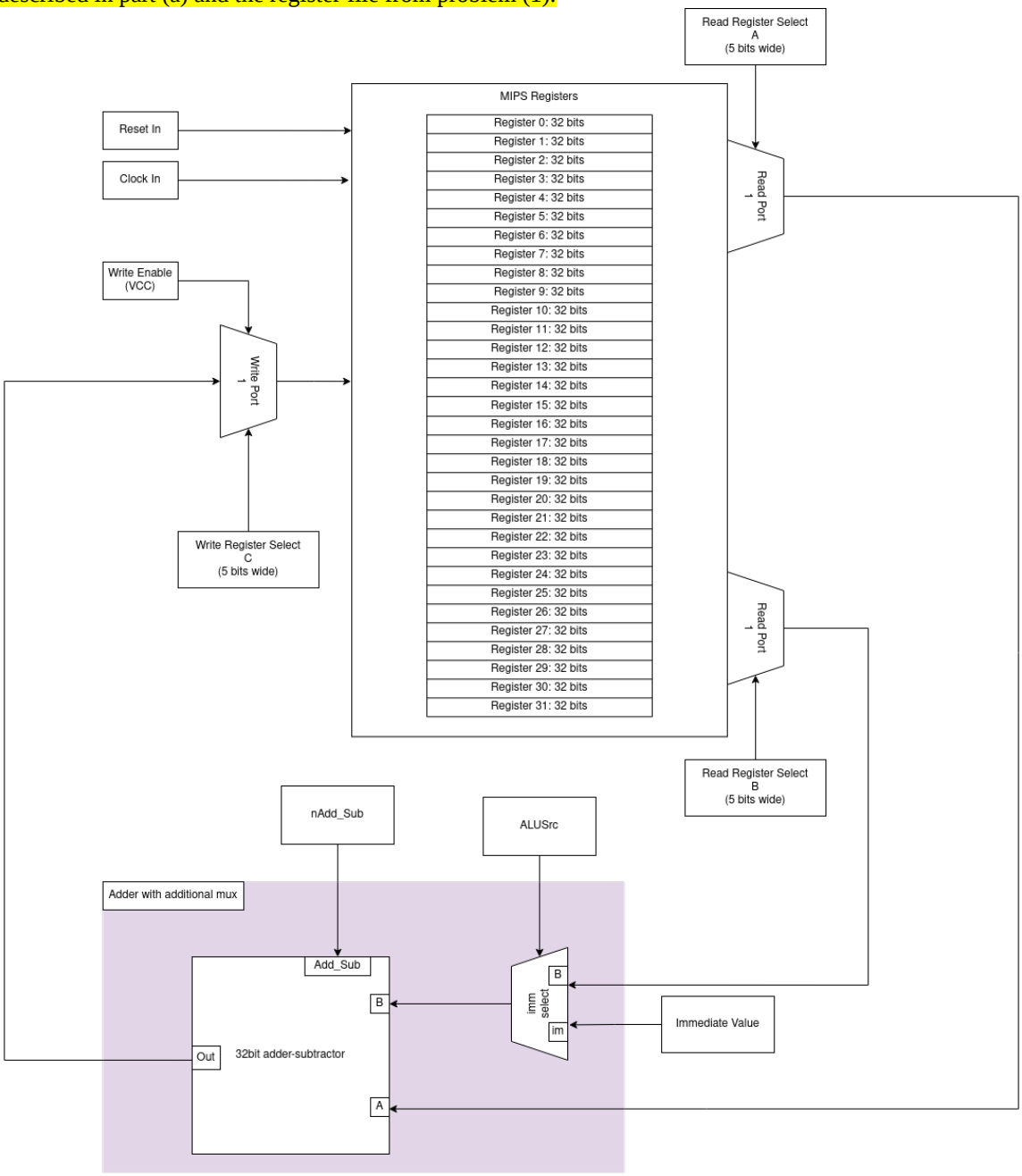


This first image shows how write enabled is on, and values are being written to registerX where X is the bit being written. Reading of these values can be seen in the next screenshot. This waveform shows readports 1, and 2 increasing and decreasing value the register they are reading from. With the registers being set to their corresponding bits set so register31 bit 31 is set, register30 bit 30 is set, and so on. This also shows how register0 always reads zero even though data has been written to it.

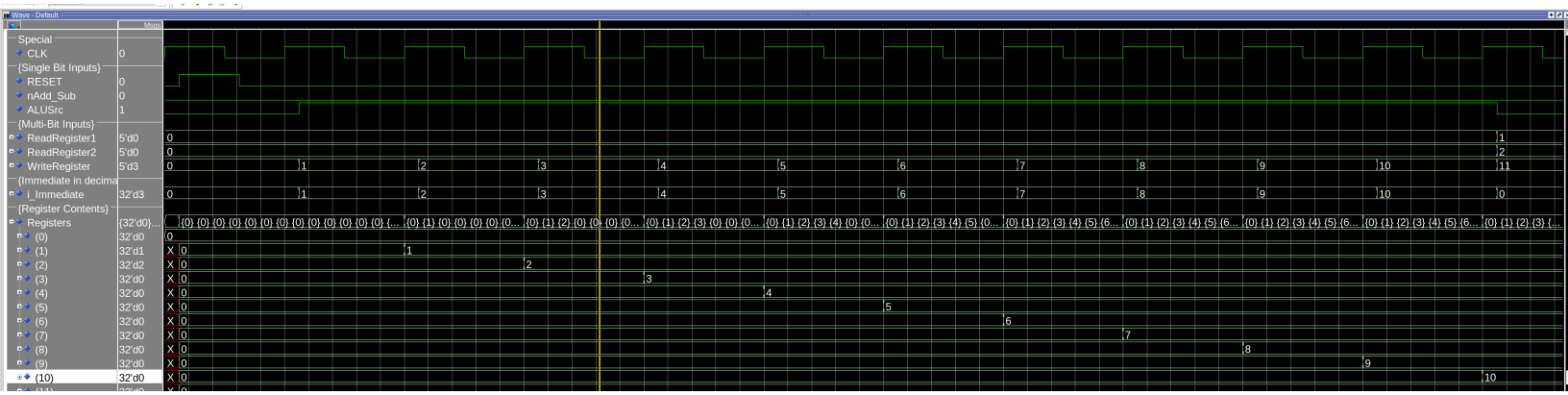
[Part 3 (b)] Draw a symbol for this MIPS-like datapath.



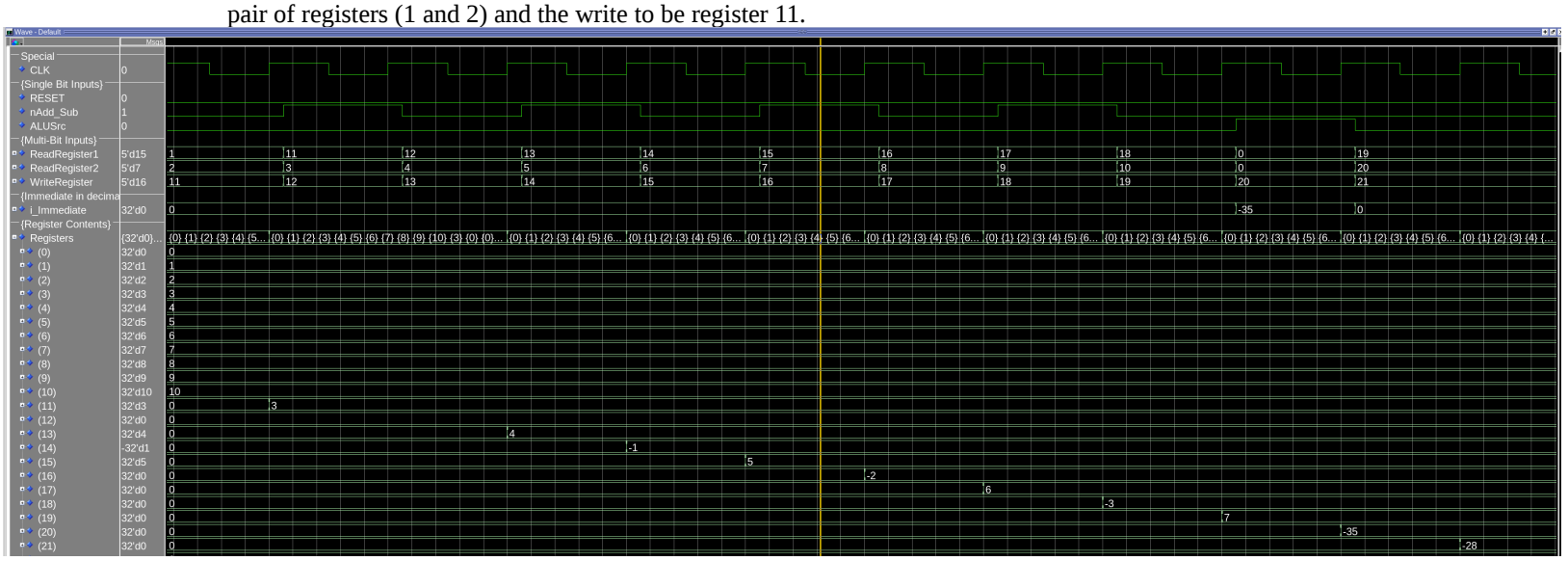
[Part 3 (c)] Draw a schematic of the simplified MIPS processor datapath consisting only of the component described in part (a) and the register file from problem (1).



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



This first image is showing the registers begin reset to 0, then the first 10 instruction being done, with all the value in decimal. It also shows how the shortly after the rising clock, a new instruction is loaded. This is most clear at the end where instead of adding an immediate value to a register, it changes ALUSrc to be a pair of registers (1 and 2) and the write to be register 11.



This images shows a continuation of the image above this. With the adding of registers 1 and 2 into 11 happening on the first rising edge shown. Here the second half of instructions can be seen executing with each value being correctly stored in the registers. The final register values are on the next page.

The final register state should be as follows:
#Regsiter Values after
#running the assembly
$0 = 0
$1 = 1
$2 = 2
$3 = 3
$4 = 4
$5 = 5
$6 = 6
$7 = 7
$8 = 8
$9 = 9
$10 = 10
$11 = 3
$12 = 0
$13 = 4
$14 = -1
$15 = 5
$16 = -2
$17 = 6
$18 = -3
$19 = 7
$20 = -35
$21 = -28
$22 = 0
$23 = 0
$24 = 0
$25 = 0
$26 = 0
$27 = 0
$28 = 0
$29 = 0
$30 = 0
$31 = 0

[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 port DATA_WIDTH is the is the amount of data stored within each address of the memory. This could be thought of as the word size. This defaults to a value of 32 bits meaning total memory available is 32 bits * number of memory addresses available.

The generic port is ADDR_WIDTH is the number of bits that can be used to address the memory. By default this is 10 which means there are 1024 possible address. This would also mean that there are 4096 bytes of memory available by leaving the default generics.

The input of clk is the clock of the memory. On the rising edge of the clock the data if new data is to be written, it will write the data, and read it back out. Otherwise, it will read the data at the given address.

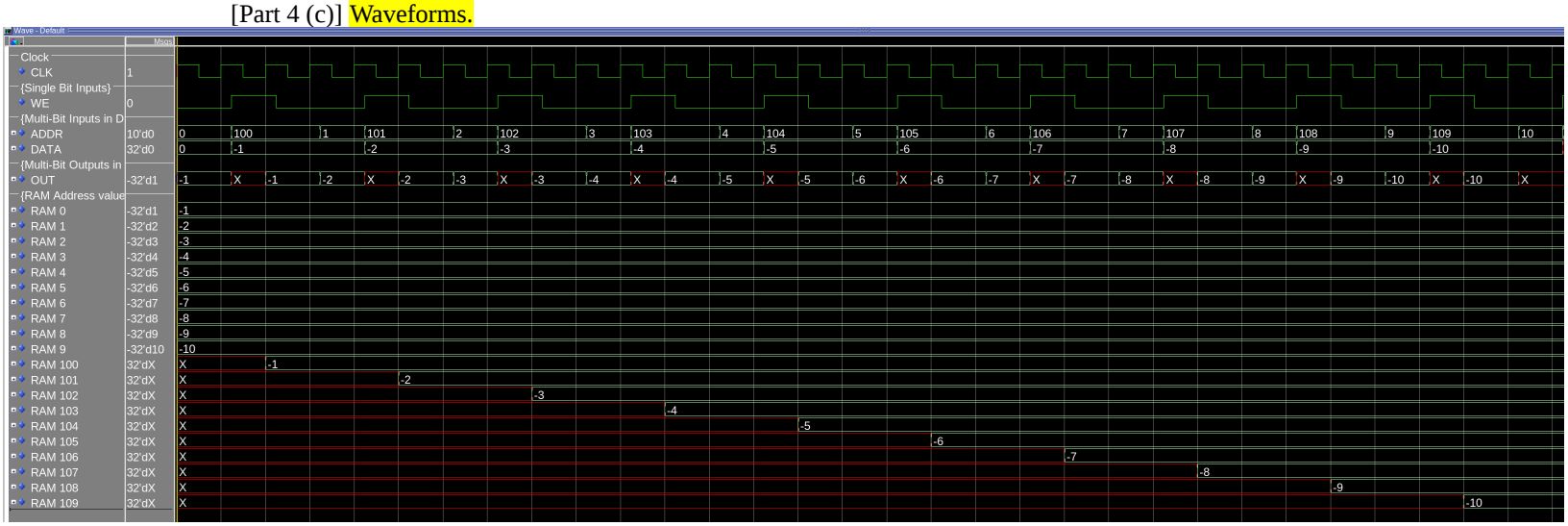
The input addr is the address that data will be pulled from or written to. Whatever this value is on the rising edge of the clock determines where data is being written or read from.

The input data is a the data that will be written to the address pointed to by addr. This will only write data on the rising edge of clock if the we input is high.

The input we is the write enable. If the is high during a rising clock edge, it will write the data on data into the memory.

The output q is where the data is being read from the memory. This will output the value that was also just written to memory if there was a write.

[Part 4 (c)] Waveforms.

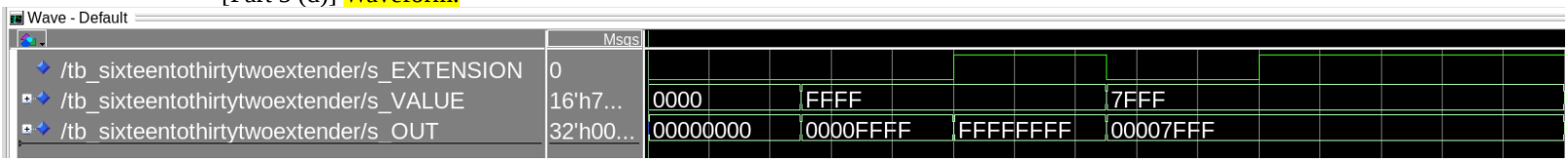


This is a waveform showing the reading of the value in the memory from 0-9, storing it temporarily. It then changes the ADDR value to 100-109 respectively for the value it just read from. Writes the temporarily stored value into the location, then reads it back to see it was written correctly.

[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?
Some instruction that require sign extension are addi, addiu, slti, sltiu, and all the load and store instructions. Some instructions that are zero extended are andi, and ori.

[Part 5 (b)] what are the different 16-bit to 32-bit “extender” components that would be required by a MIPS processor implementation?
We need to implement 1 for the pair of add instructions since they are in the ALU together. Another for the the both of the set less than instructions. You would then need to implement 2 more, one for load word, and another for store word. So there would be a total of 4 implemented in a MIPS Processor.

[Part 5 (d)] Waveform.

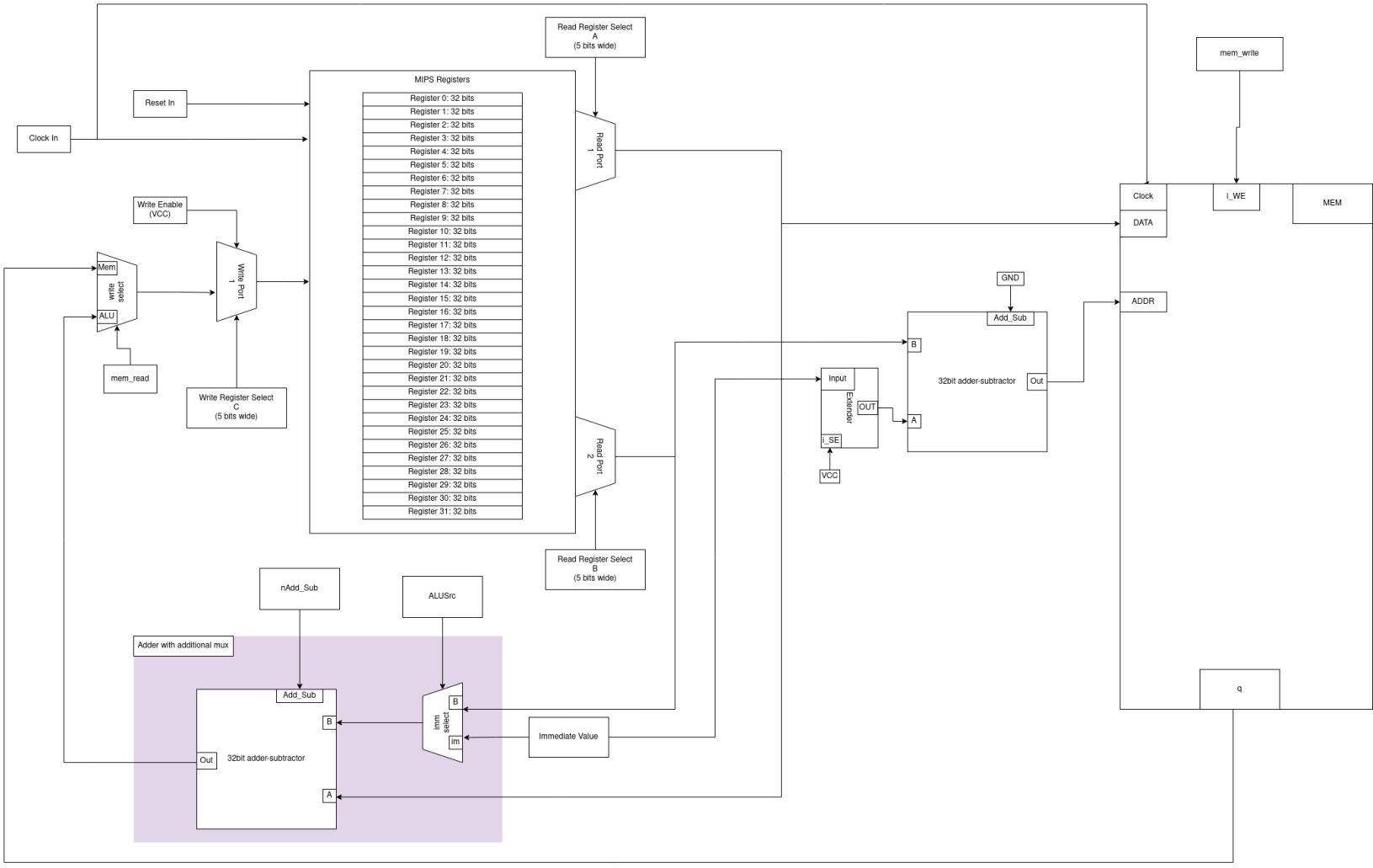


This waveform shows a value with 1 being the most significant bit being on the input. With sign extension off it fills with all 0s, and when it it is turned on it fills with all ones. Then a value with 0 as the most significant bit is loaded, and remains all 0 filled whether sign extension is on or off.

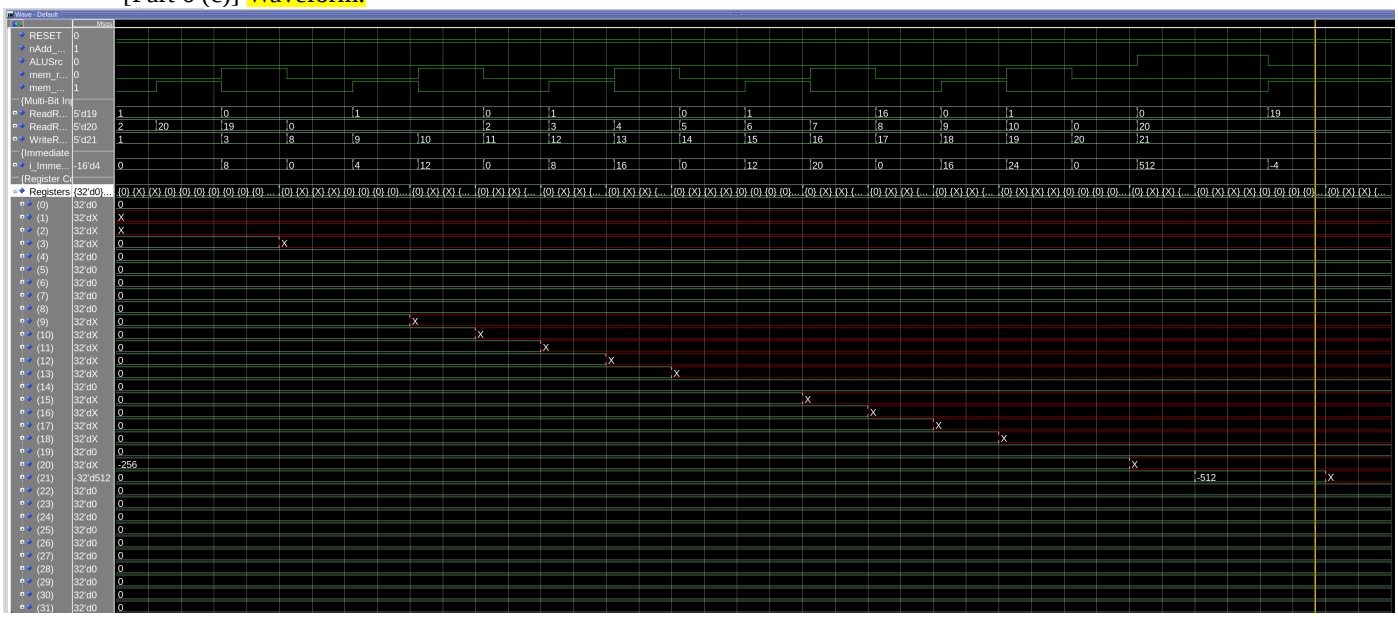
[Part 6 (a)] what control signals will need to be added to the simple processor from part 2? How do these control signals correspond to the ports on the mem.vhd component analyzed in part 3?
I will need to implement a memory write control signal. This will turn on the i_WE on the memory to write to the addr coming from readport2 on the registerfile, and adding the immediate value.

I will also need to implement a memory read control signal. This will toggle a write port of the register file to get it’s input from the mem output instead of the ALU output.

[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 waveform shows the registers taking values, from memory and putting them into registers, and storing values from registers into memory.