CprE 381, Computer Organization and Assembly-Level Programming

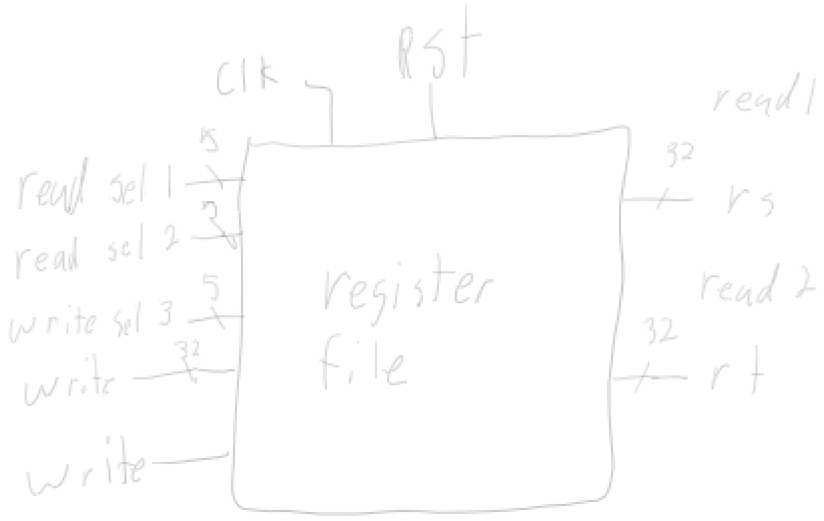
Lab 2 Report

Student Name _Thomas Gaul___

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.

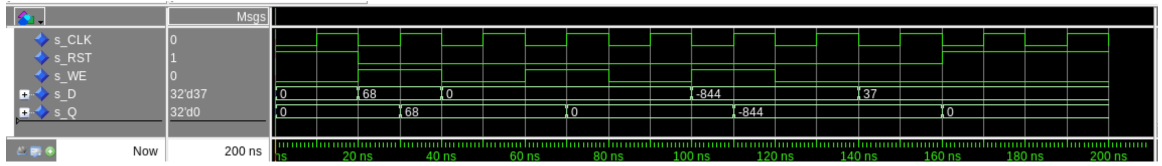
Prelab Describe the provided DFF in dffg.vhd in terms of edge sensitivity and reset type (active low/high and synchronous/asynchronous).
It is an active high asynchronous reset with rising edge sensitivity synchronous write.

[Part 2 (a)] Draw the interface description (i.e., the "symbol" or high-level blackbox) 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.

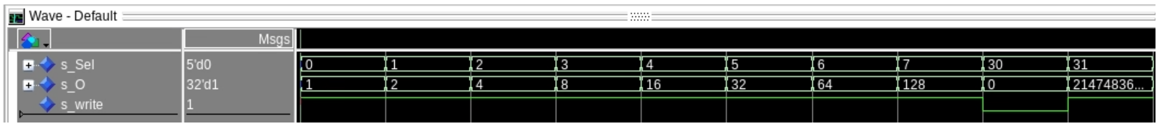
[Part 2 (c)] Waveform.



N bit register test bench ensuring values can be read saved and overwritten and the reset pin resets all them to 0 and keeps them 0 when attempted to be overwritten.

[Part 2 (d)] What type of decoder would be required by the MIPS register file and why?
We would need a 5 to 32 decoder to write a portion of the register file. We only want to write to one at a time and the decoder will only output to one of the 32 registers write enable lines at a time. Additionally my decoder has an enable line which when dropped to 0 outputs all 0's to handles no writing

[Part 2 (e)] Waveform.

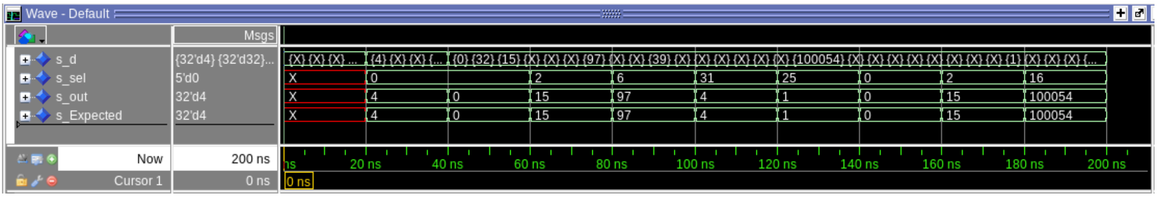


5 to 32 decoder implemented making sure different inputs output the correct one location as on. Additionally there is a write signal that allows it to output 0s everywhere to handle write enable disable in the future register file

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

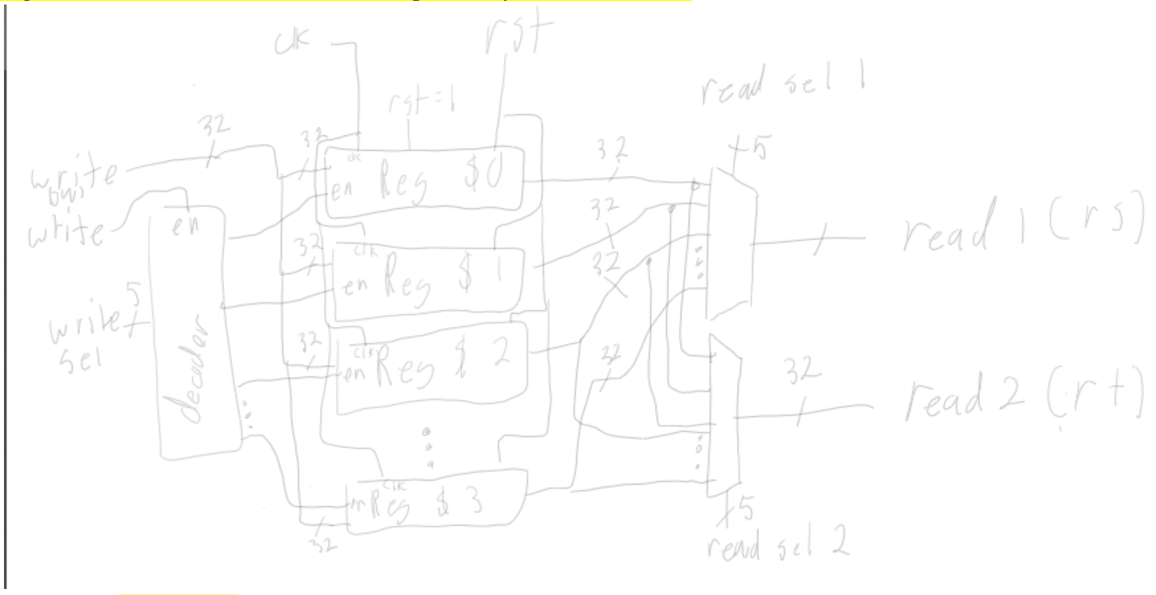
I designed my 32 bit 32 to one multiplexer with an array of 32 bit vector files. I chose to implement it this way to avoid having many different signal names and makes it easier to map future aspects as many of them connect 32 32 bits sections together.

[Part 2 (g)] Waveform.

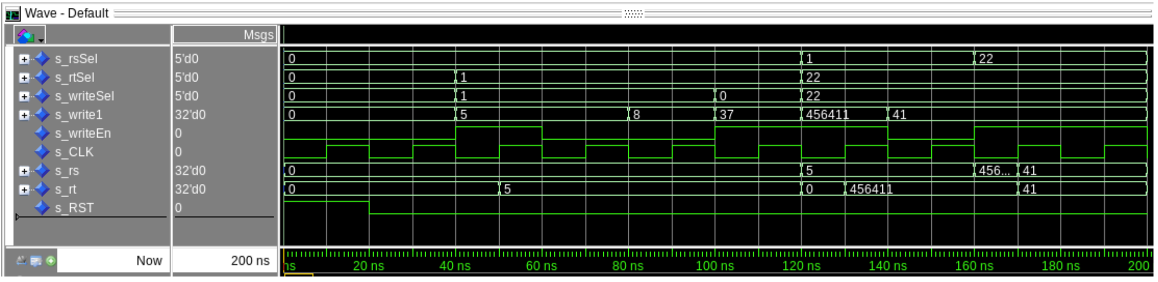

Wave - Default
Msgs
s_d {32'd4} {32'd32}...
s_sel 5'd0
s_out 32'd4
s_Expected 32'd4
{X} {X} {X} ... {4} {X} {X} {... {0} {32} {15} {X} {X} {X} {97} {X} {X} {39} {X} {X} {X} {X} {X} {X} {100054} {X} {X} {X} {X} {X} {X} {X} {X} {1} {X} {X} {X} {...
X 0 2 6 31 25 0 2 16
X 4 0 15 97 4 1 0 15 100054
X 4 0 15 97 4 1 0 15 100054
Now 200 ns
Cursor 1 0 ns
ns 20 ns 40 ns 60 ns 80 ns 100 ns 120 ns 140 ns 160 ns 180 ns 200 ns
0 ns

32 bit 32 to 1 multiplexer implemented with arrays test bench.

[Part 2 (h)] Draw a (simplified) schematic (i.e., components within the high-level blackbox) 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.

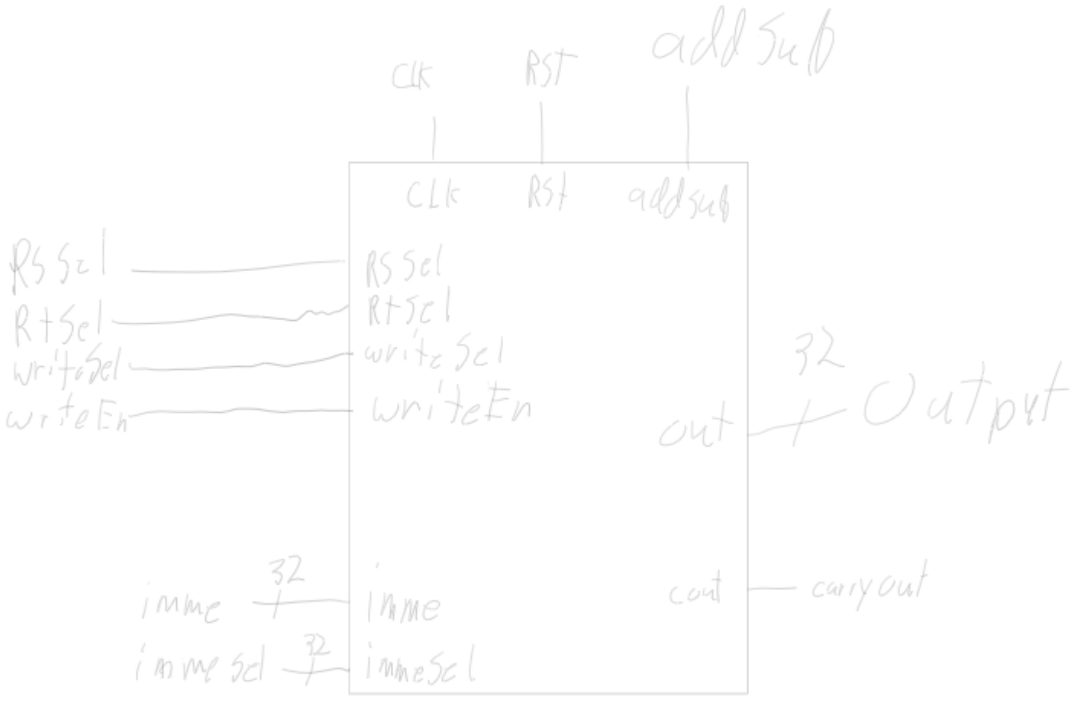


[Part 2 (i)] Waveform.

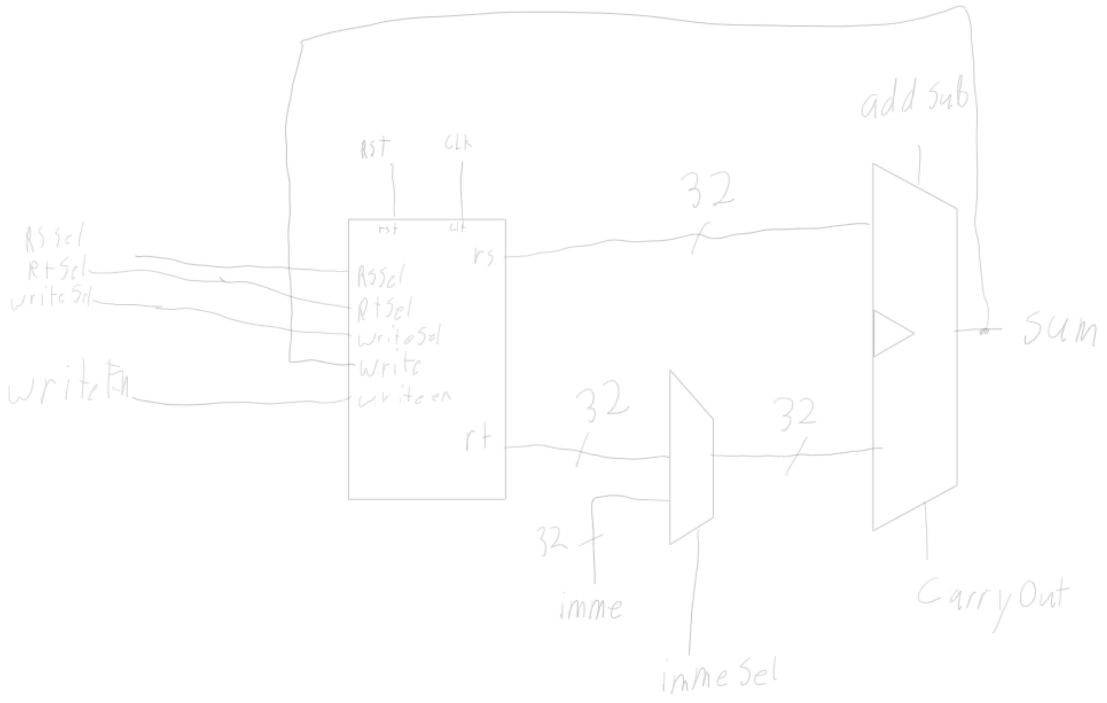


Register file test bench ensure register can be written to and read. Also made sure register 0 is always going to hold 0 even if over written

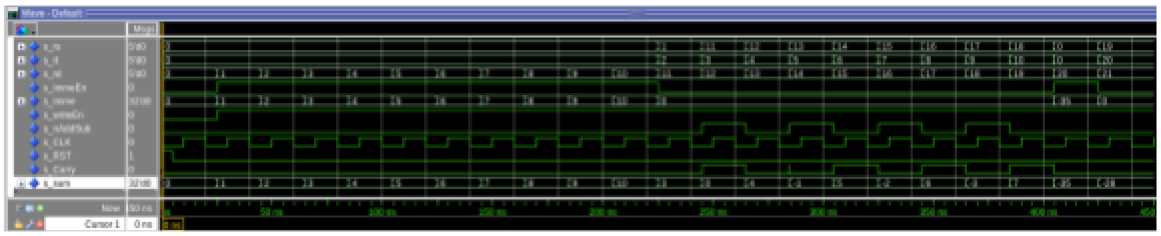
[Part 3 (b)] Draw a symbol for this MIPS-like datapath.



[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.

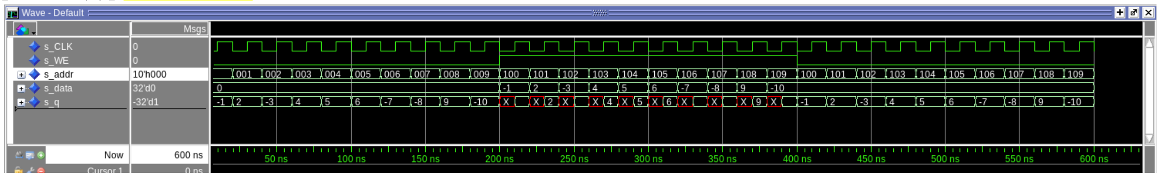


The final register state should be reading $21 should be -28 from all the additions and subtractions compounding.

[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).

Generics: DATA_WIDTH is the generic number of bits to be stored and ADD_WIDTH is the number of bits to select an address location. These two generics are applied in with the addr and the data ports, addr selects the address to read or write where as data pins are used for writing a specific value to the selected address. The clk port is the clock for the memory only writing on rising edge. we selects writing or reading with 1 being write and 0 being read. And finally q is the data stored in a selected location.

[Part 4 (c)] Waveforms.



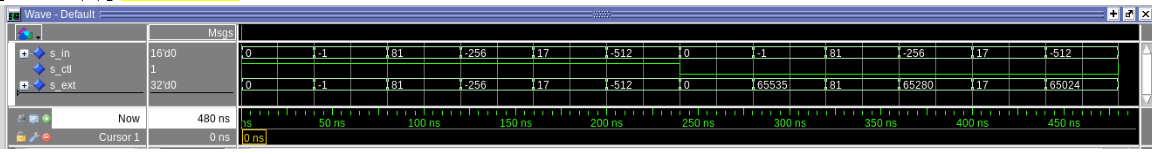
[Part 5 (a)] What are the MIPS instructions that require some value to be sign extended? What are the MIPS instructions that require some value to be zero extended?

Some mips instructions that would require sign extensions would be addi and subi with a negative number where as a zero extension would be needed for andi or ori. When addition to the program counter due to branching that would need a bit extender.

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

Immediate would need extender components going into the ALU, load halfwords would need extenders going into the register file. When adding to the program counter due to branching that would need a bit extender.

[Part 5 (d)] Waveform.

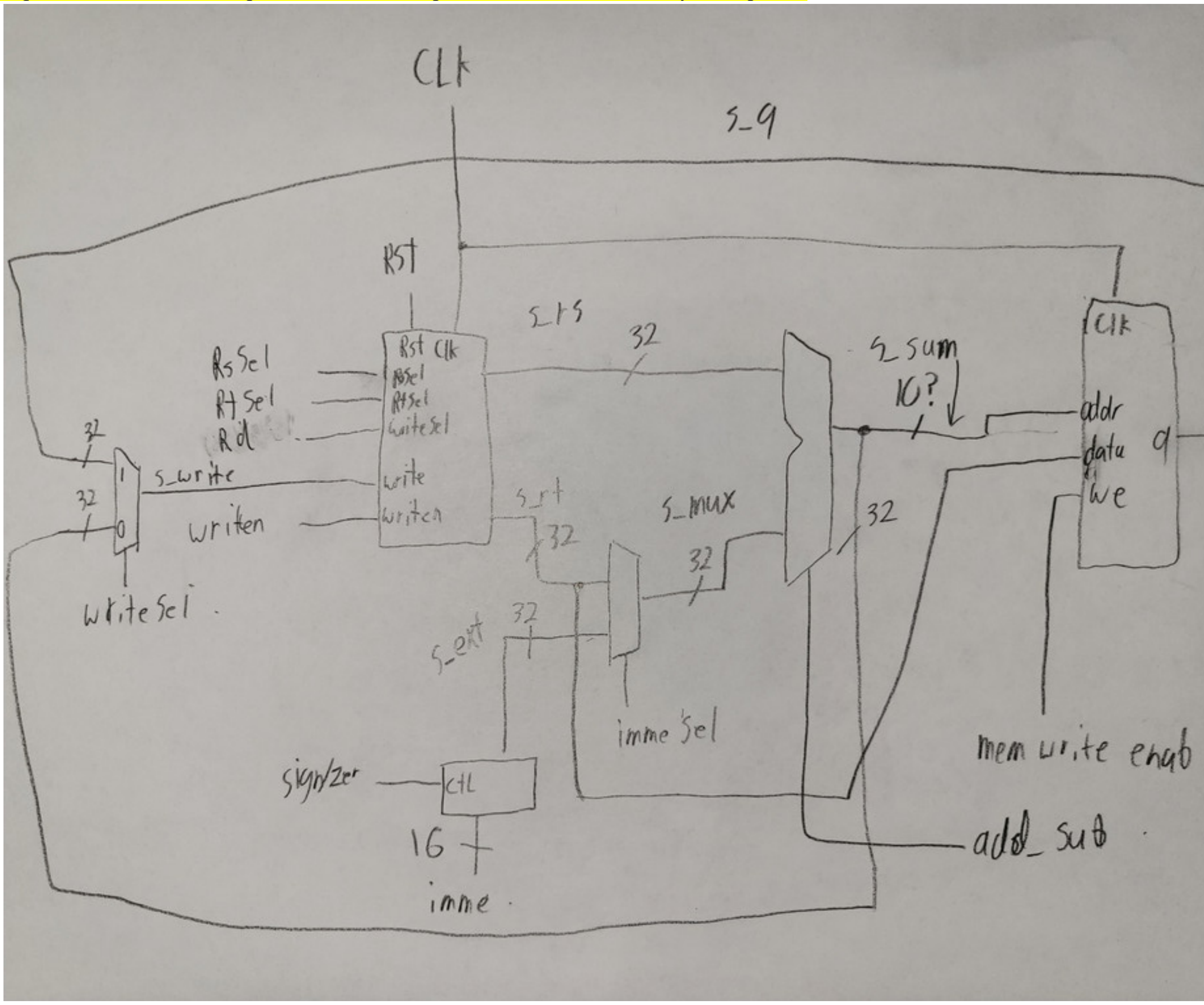


Test bench for the sign extender with an with the ability to select zero extend or sign extend implemented with for generate loops and a when then statement

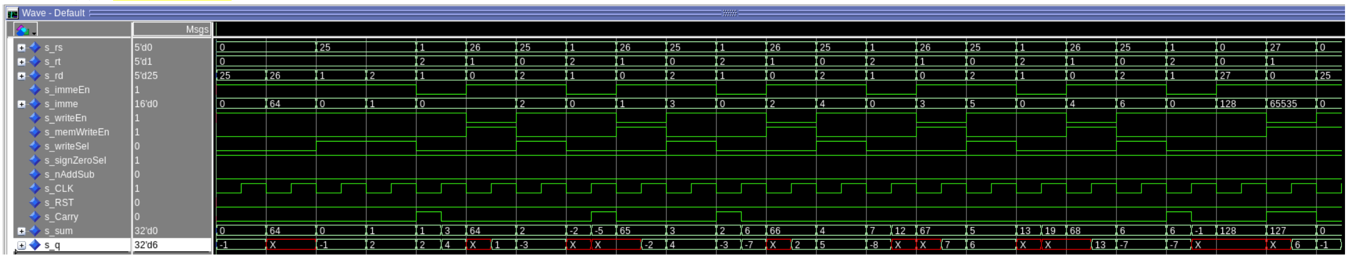
[Part 6 (a)] what control signals will need to be added to the simple processor from part 2? How do these control signals correspond to the ports on the mem.vhd component analyzed in part 3?

We need to add a control signal for deciding zero or sign bit extension for the bit extender, we also need a 32 bit 2 to 1 multiplexer to take the q and the alu sum to go to the write of the register file. Additionally there is a memory write control bit that goes into the WE port of the memory module.

[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.



Above is the wave form of the test bench "running" the assembly code of for the final part of the lab. I had the memory final initially loaded with the values we had to alter for the hex file in the memory part of the lab. Additionally all values involving the memory were divided by 4 because the assembly we work off of is byte addressable whereas the memory we used here is word addressable so when having offsets for load and store and addi for putting in address I divided them all by 4.