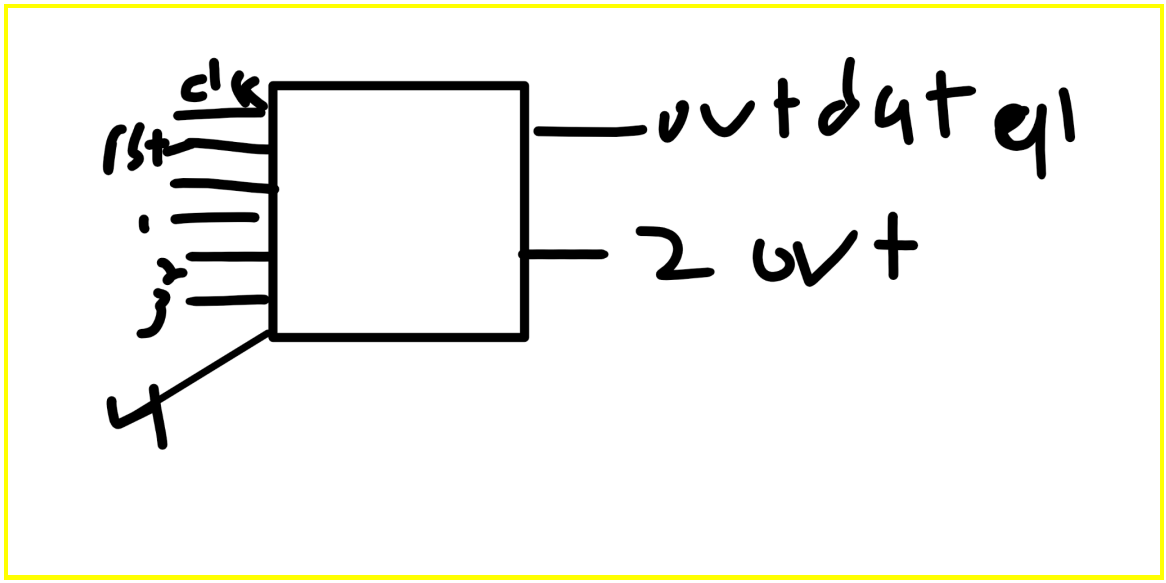
CprE 381, Computer Organization and Assembly-Level Programming

Lab 2 Report

Student Name ____Michael Mota_____

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.

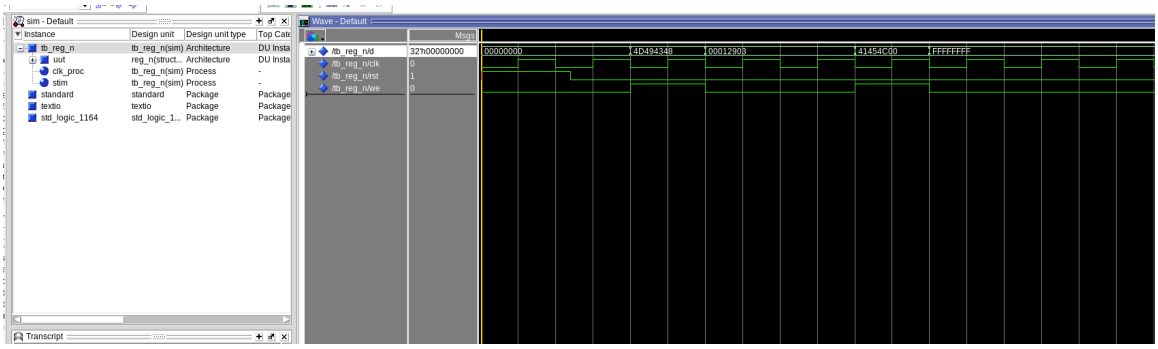
[Part 3 (a)] Draw the interface description (i.e., the "symbol" or high-level blackbox) for the RISC-V register file. Which ports do you think are necessary, and how wide (in bits) do they need to be?



We need a clock input which is 1 bit because writes happen on a clock edge, plus a write-enable (1 bit) to tell the register file when to actually write. We also need three 5-bit address inputs—rs1, rs2, and rd—since 5 bits selects one of 32 registers, and a 32-bit wdata input for what you're writing. Finally, we need two 32-bit outputs, one for the rs1 read data and one for the rs2 read data.

[Part 3 (b)] Create an N-bit register using this flip-flop as your basis.

[Part 3 (c)] Waveform.



reate a testbench to test your register design to make sure it is working as expected, and include a waveform screenshot in your report PDF. A sample testbench that also incorporates a clock generator can be found in tb_dff.vhd. [You will need to modify this slightly for your N-bit version.]

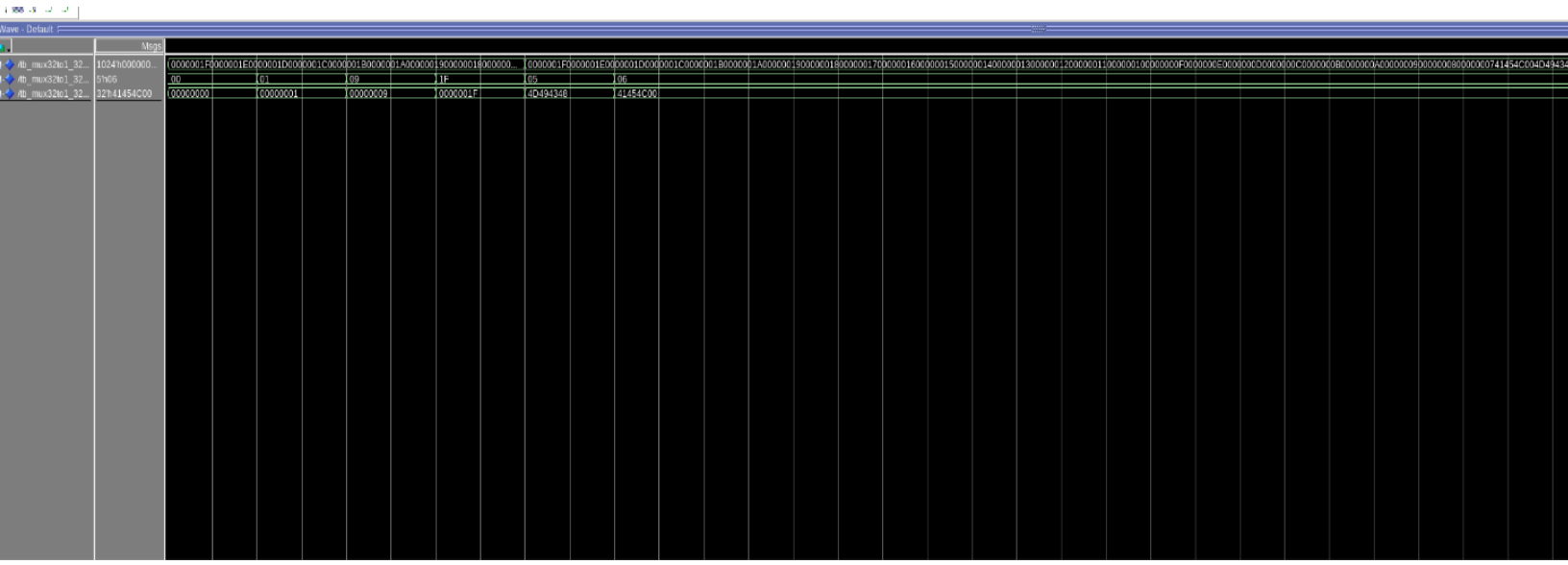
[Part 3 (d)] What type of decoder would be required by the RISC-V register file and why?
For a riscv register file you need a 5-to-32 (5:32) decoder because the RISC-V register file has 32 registers, and a register index like rd is 5 bits (since 2^5 = 32). The decoder turns that 5-bit rd value into a one-hot 32-bit write-enable bus so only the selected register gets written.

[Part 3 (e)] Waveform.

This waveform is showing your 5-to-32 decoder doing the one-hot thing: when the 5-bit input changes (like 00, 01, 1F), the 32-bit output jumps so exactly one bit is '1' (ex: 00 -> 00000001, 01 -> 00000002, 1F -> 80000000).

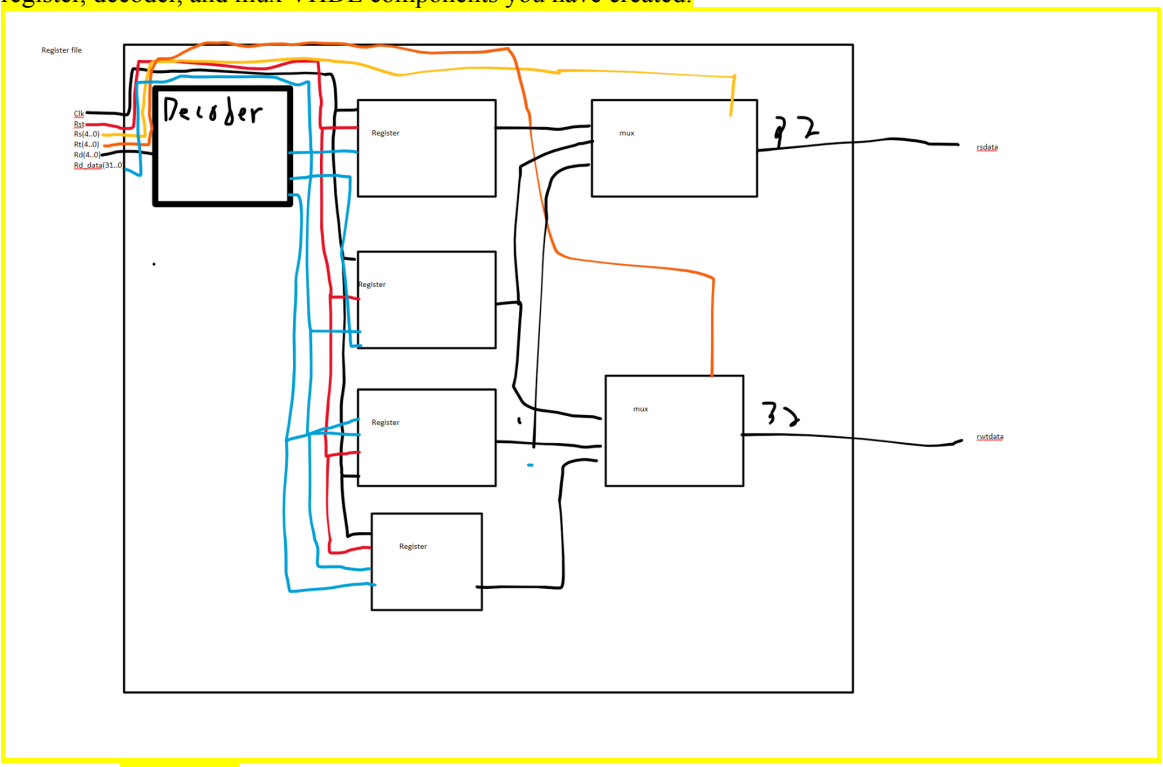
[Part 3 (f)] In your write-up, describe and defend the design you intend on implementing for the next part.
For the register-file read path, i will implement the 32-bit 32:1 mux as a dataflow component using a single combinational selection statement (e.g., with i_S select o_Y <= ... or an equivalent case in a combinational process). This approach is compact, easy to verify in simulation, and synthesizes efficiently into the same kind of mux structure a structural "tree of 2:1 muxes" would produce, without the coding overhead. Since reads must be available immediately (no clocked behavior), a purely combinational dataflow mux is the most appropriate choice for selecting one 32-bit register output from the 32 parallel register outputs.

[Part 3 (g)] Waveform.

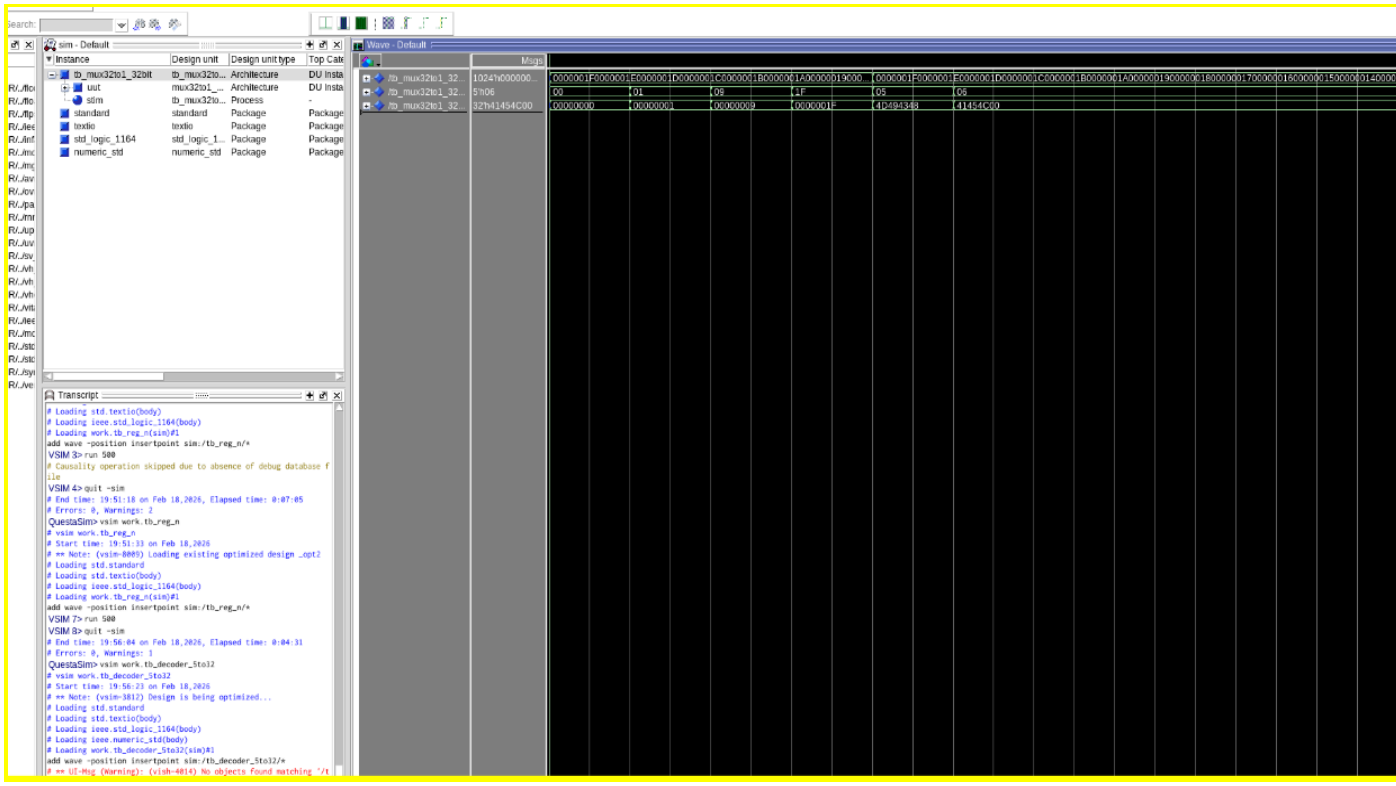


In this waveform, we verify the behavior of our 32-bit 32:1 multiplexer. The wide upper signal represents the concatenated set of 32 possible 32-bit inputs, the middle signal is the 5-bit select value, and the lower signal is the 32-bit output. As we vary the select (e.g., 0, 1, 9, 31), the output updates immediately to the corresponding 32-bit slice, demonstrating the expected purely combinational selection used for register-file reads.

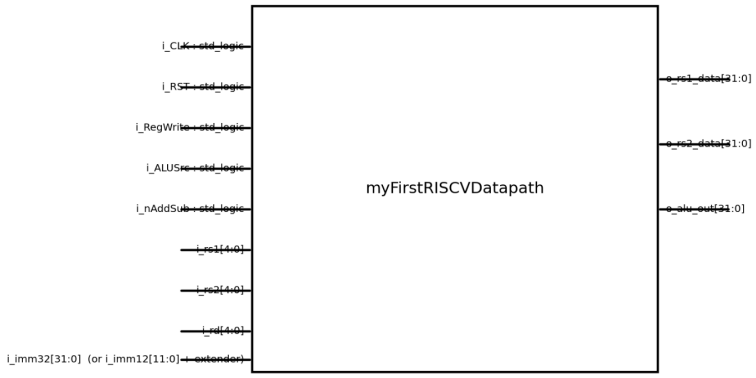
[Part 3 (h)] Draw a (simplified) schematic (i.e., components within the high-level blackbox) for the RISC-V register file, using the same top-level interface ports as in your solution describe above and using only the register, decoder, and mux VHDL components you have created.



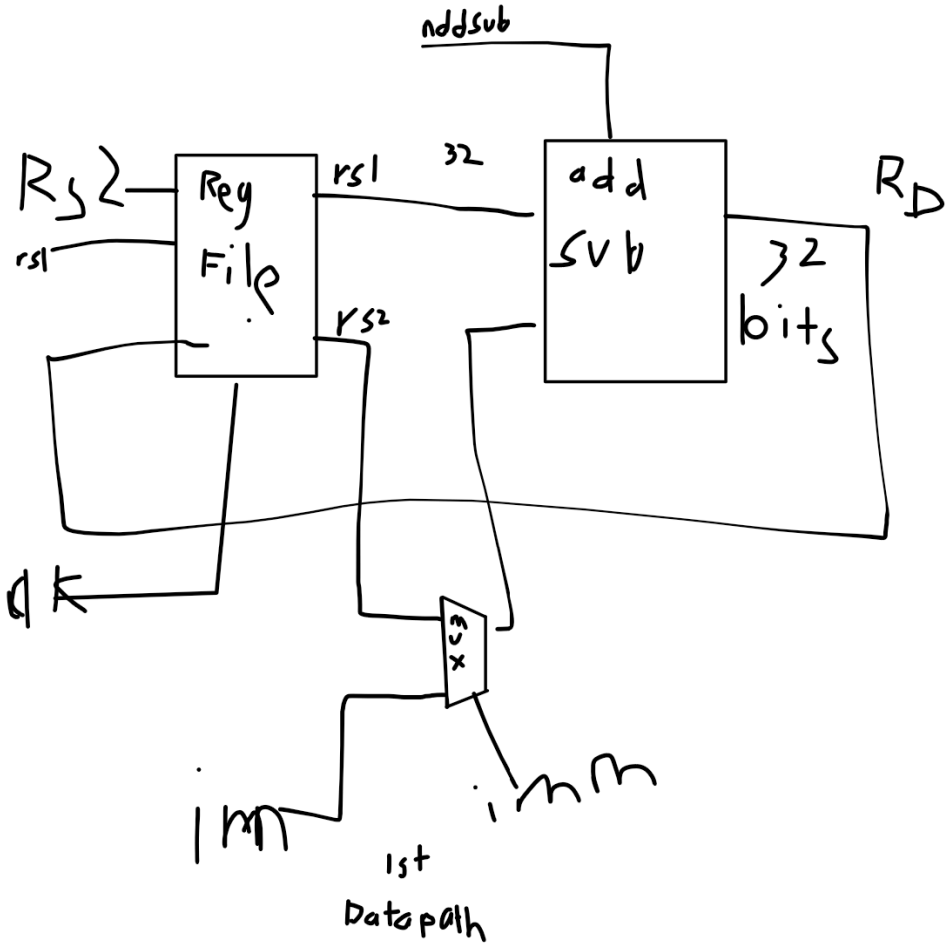
[Part 3 (i)] Waveform.



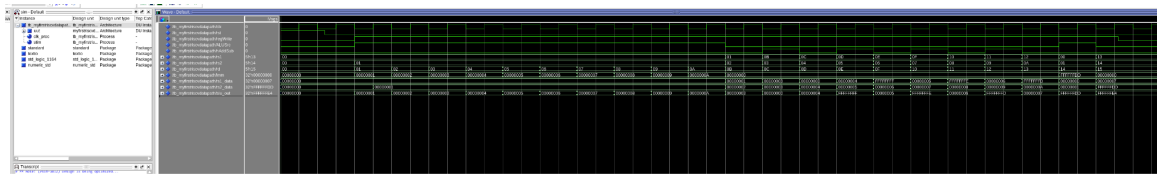
[Part 4 (b)] Draw a symbol for this RISC-V-like datapath.



[Part 4 (c)] Draw a schematic of the simplified RISC-V processor datapath consisting only of the component described in part (a) and the register file from problem (1).

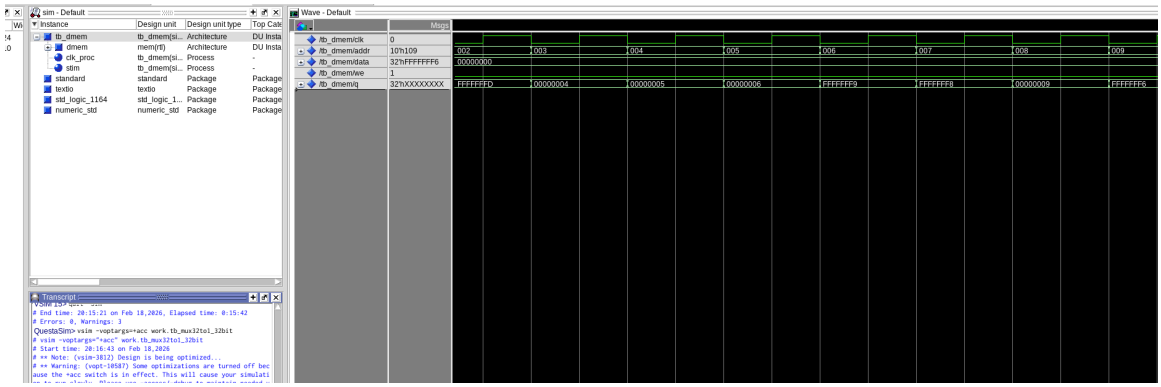


[Part 4 (d)] Include in your report waveform screenshots that demonstrate your properly functioning design. Annotate what the final state of the register file should be and provide a mapping of the registers used in your drawing to the register names.



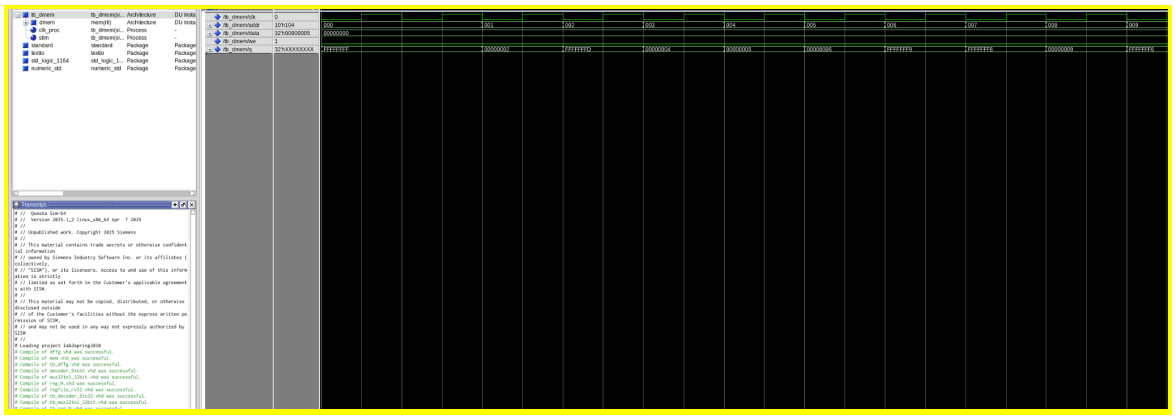
[Part 5 (a)] Read through the mem.vhd file, and based on your understanding of the VHDL implementation, provide a 2-3 sentence description of each of the individual ports (both generic and regular).
DATA_WIDTH (generic): This generic sets the number of bits stored in each memory location (the word size). With the default value of 32, each address holds one 32-bit word, matching RV32 data width. ADDR_WIDTH (generic): This generic sets the width of the address input and therefore the number of word locations in memory. The memory depth is 2 $ADDR\_WIDTH$ words, and in this lab the addresses index words, not bytes. clk (port): The clock controls when writes occur in the RAM. When we='1', the memory writes data into the selected address on the rising edge of clk. addr (port): This input selects which word of memory is being accessed. The output q reflects the contents at addr, so changing addr changes which word is read. data (port): This is the value that will be written into memory during a store operation. It is only committed to memory on a rising clock edge when we='1'. we (port): This write-enable signal determines whether a write happens on the next rising edge of clk. If we='0', memory contents are not modified. q (port): This is the read-data output that reports the contents of the currently addressed word. It becomes valid once addr is stable (and after a write, it updates right after the clock edge that performs the write).

[Part 5 (c)] Waveforms and hex file.

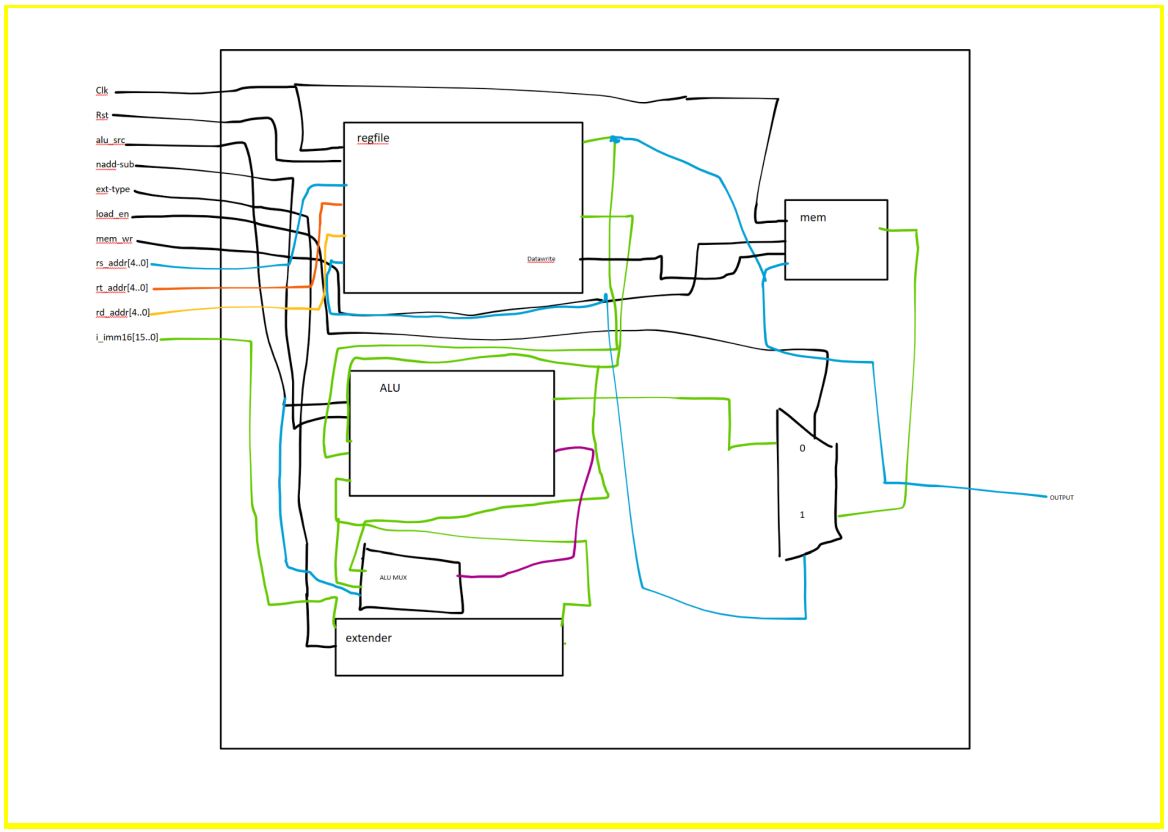


[Part 6 (a)] What are the RISC-V instructions that require some value to be sign extended? What are the RISC-V instructions that require some value to be zero extended?
In risc v insturtations the 12-bit immediates used by I-type and S-type instructions are treated as signed values, so they must be sign-extended to 32 bits before being used (this includes addi, slti, xori, ori, andi, jalr, and the address offsets for loads/stores like lw/sw). For loads, the signed variants lb and lh also require sign extension of the loaded 8-bit or 16-bit value up to 32 bits. Zero extension is required when the loaded value is explicitly unsigned. The base instructions that do this are lbu and lhu, which zero-extend the 8-bit or 16-bit value to 32 bits.
[Part 6 (b)] what are the different 12-bit to 32-bit “extender” components that would be required by a RISC-V processor implementation?
A RISC-V processor needs extender hardware to expand 12-bit immediates into 32-bit values used by the datapath. At minimum, this means a 12→32 sign extender (replicating bit 11 into bits 31..12) for normal immediates and address offsets, and a 12→32 zero extender (filling bits 31..12 with zeros) for cases where the value is treated as unsigned. In practice, many designs implement this as a single extender module with a control bit that selects between sign-extend and zero-extend.

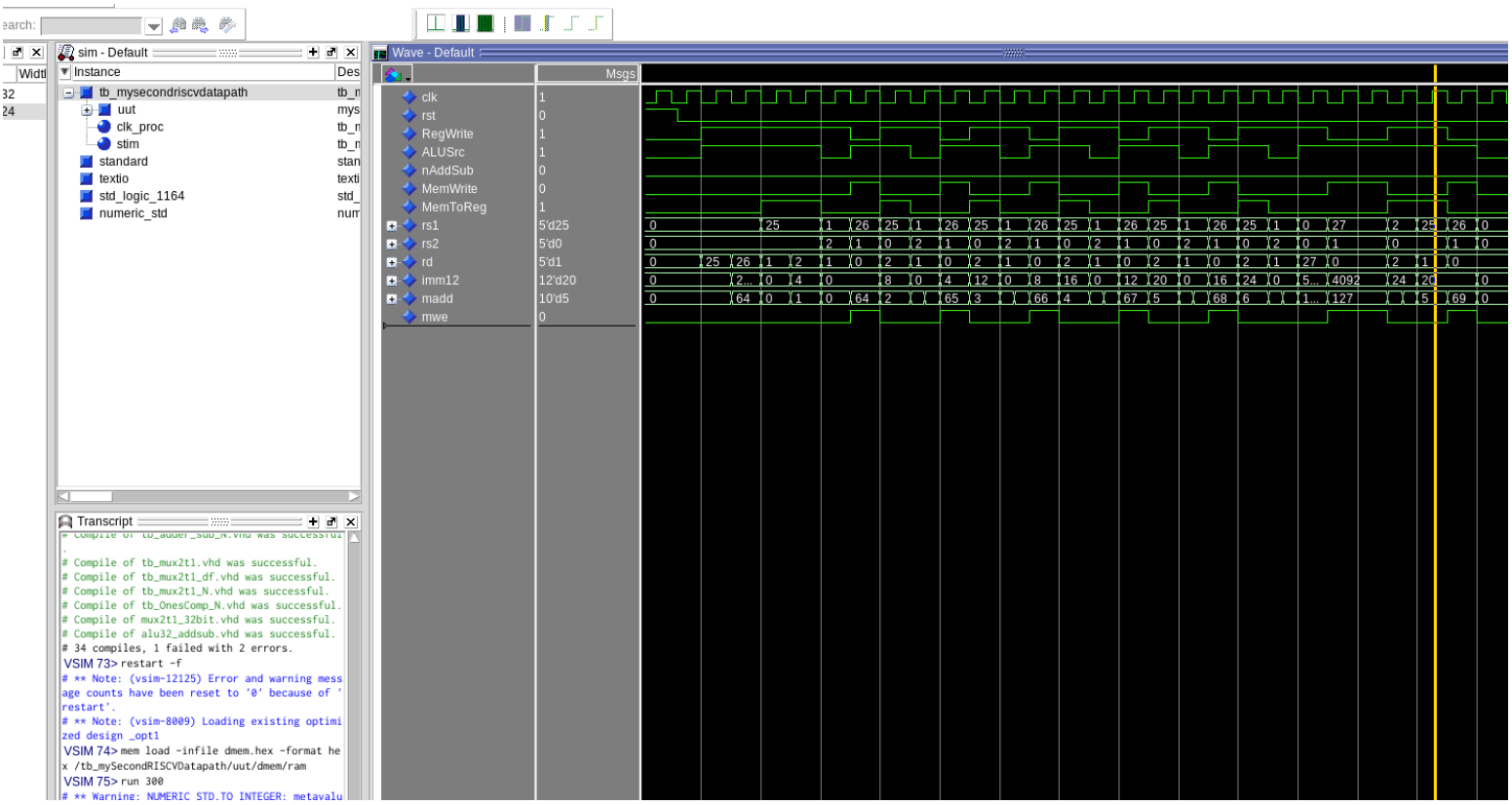
[Part 6 (d)] Waveform.



[Part 7 (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?
To add load and store support to the simple datapath from part 2, we need control signals that decide when memory is written and what value gets written back to the register file. Concretely, we add MemWrite to control whether a store occurs, and MemToReg to select the register write-back source (ALU result for arithmetic vs memory output for loads). In mem.vhd, MemWrite maps directly to the memory port we, the ALU-generated address drives addr, store data comes from the register file into data, and the loaded value is read from q and then selected by MemToReg for register write-back.
[Part 7 (b)] Draw a schematic of a simplified RISC-V 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 7 (c)] Waveform.


sim - Default
Instance
tb_mysecondriscvdatapath
uut
clk_proc
stim
standard
textio
std_logic_1164
numeric_std
Transcript
Compile of tb_mux2t1.vhd was successful.
Compile of tb_mux2t1_df.vhd was successful.
Compile of tb_mux2t1_N.vhd was successful.
Compile of tb_OnesComp_N.vhd was successful.
Compile of mux2t1_32bit.vhd was successful.
Compile of alu32_addsub.vhd was successful.
34 compiles, 1 failed with 2 errors.
VSIM 73> restart -f
** Note: (vsim-12125) Error and warning message counts have been reset to '0' because of 'restart'.
** Note: (vsim-8009) Loading existing optimized design _opt1
VSIM 74> mem load -infile dmem.hex -format hex /tb_mySecondRISCVDatapath/uut/dmem/ram
VSIM 75> run 300
** Warning: NUMERIC STD.TO INTEGER: metavalu
Wave - Default
Msgs
clk
rst
RegWrite
ALUSrc
nAddSub
MemWrite
MemToReg
rs1
rs2
rd
imm12
madd
mwe