CprE 381, Computer Organization and Assembly-Level Programming

Lab 2 Report

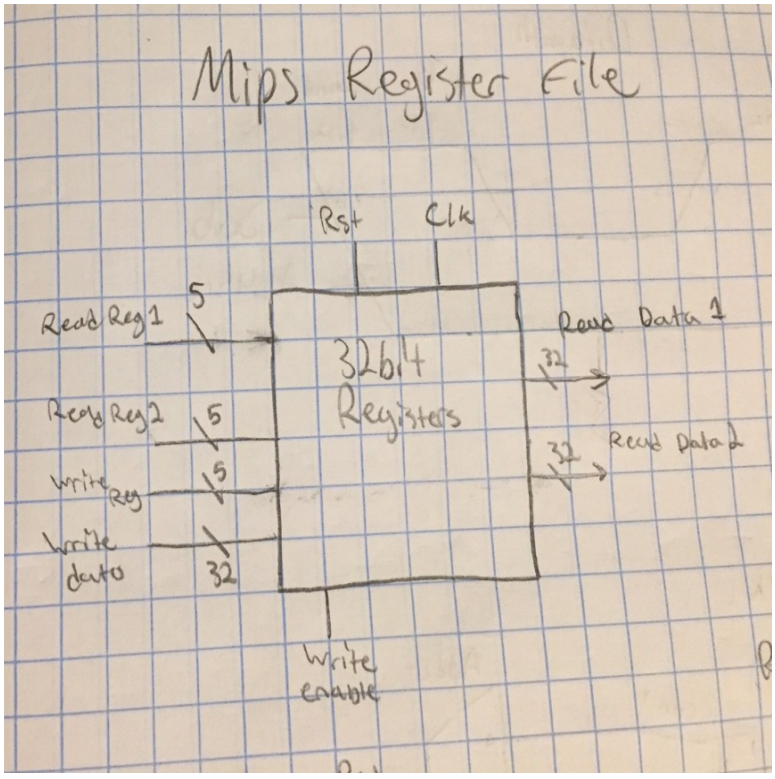
Student Name _____Jacob Boicken_____

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.vhdin terms of edge sensitivity and reset type (active low/high and synchronous/asynchronous).

The D Flip flop works on activates on the rising edge of the clk and resets asynchronously to 0.

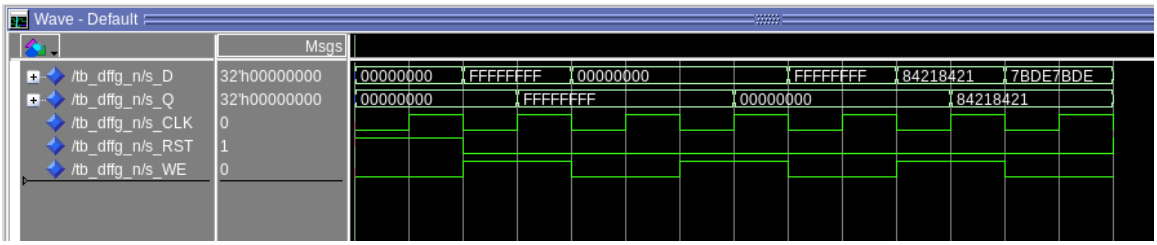
[Part 2 (a)] Draw the interface description for the MIPS register file. Which ports do you think are necessary, and how wide (in bits) do they need to be?



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

In vhdl submitted. Nothing needed here.

[Part 2 (c)] Waveform.

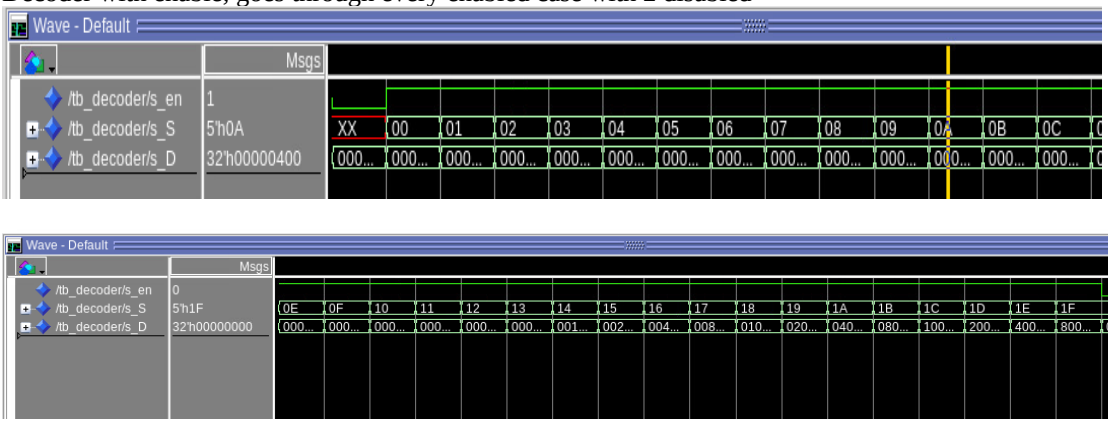


[Part 2 (d)] What type of decoder would be required by the MIPS register file and why?

5:32 decoder as the register file needs to take in an input of 5 bits to select which of the 32 registers to read/write from

[Part 2 (e)] Waveform.

Decoder with enable, goes through every enabled case with 2 disabled

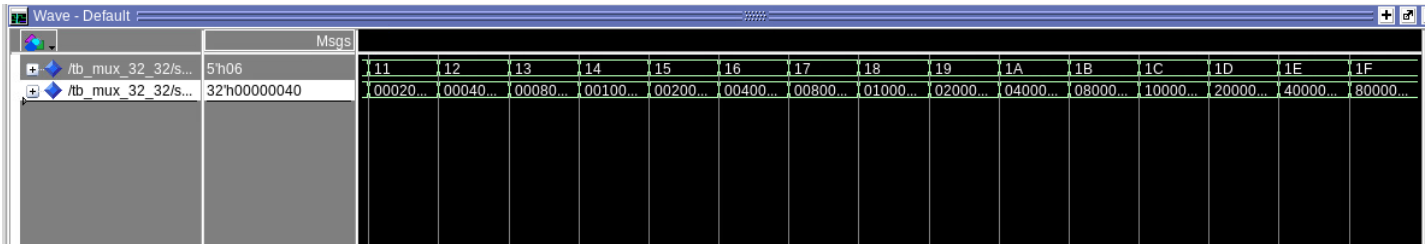


[Part 2 (f)] In your write-up, describe and defend the design you intend on implementing for the next part.

I chose to do a dataflow designed for a 32 bit, 32:1 MUX. This is because is should be easy to do by mapping each 32bit input to an array then converting the 5bit select input to an int to index the array.

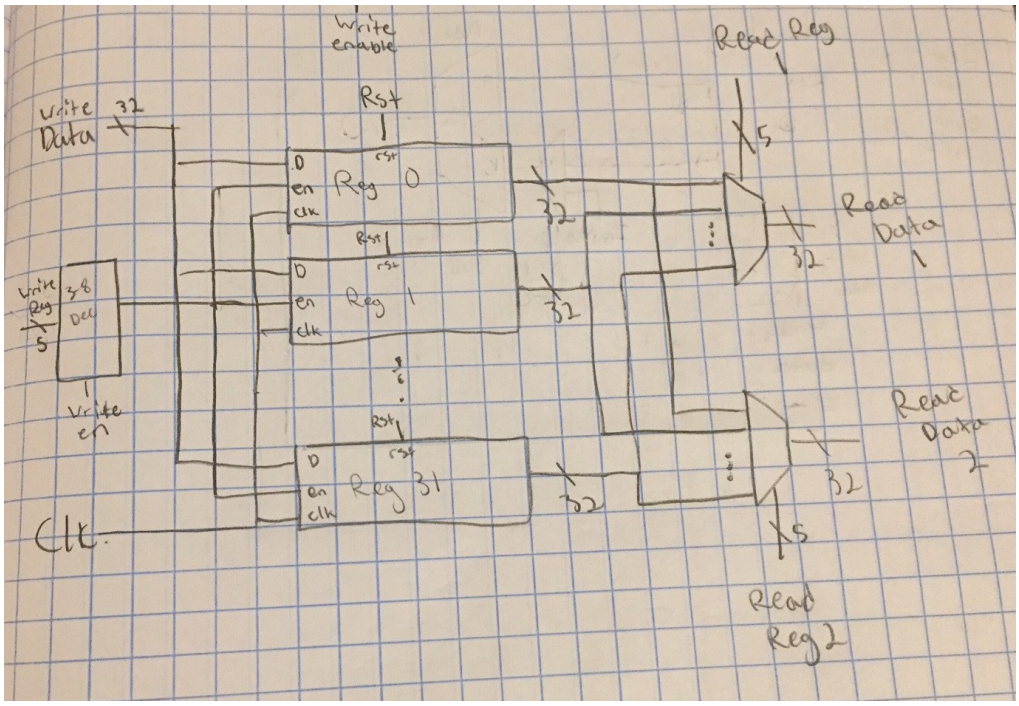
[Part 2 (g)] Waveform.

I made an 32 length array of 32 bit std logic vectors, and loaded each up with different values.



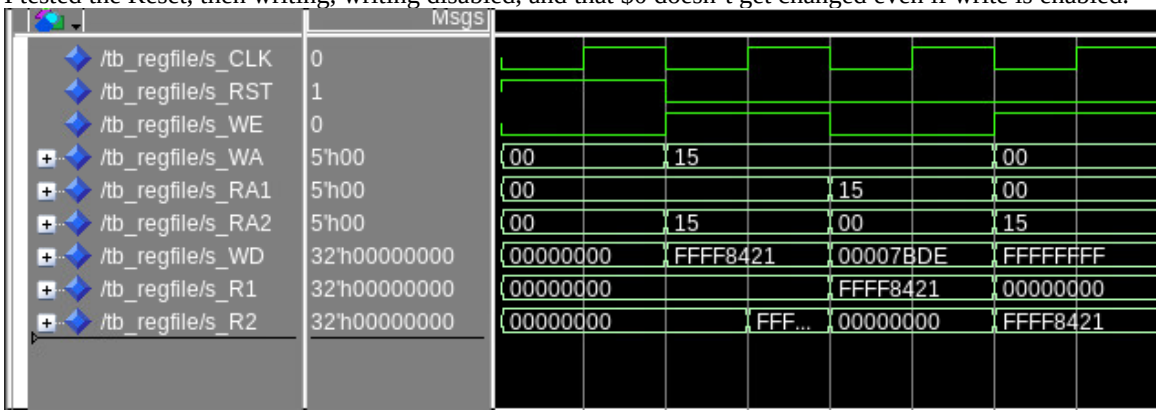
[Part 2 (h)] Draw a (simplified) schematic for the MIPS register file, using the same top-level interface ports as in your solution describe above and using only the register, decoder, and mux VHDL components you have created.

(Write Enable from previous picture)

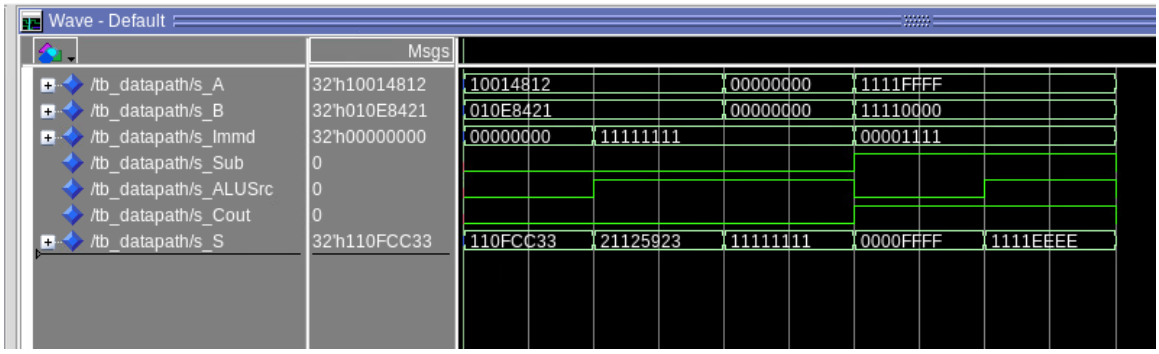


[Part 2 (i)] Waveform.

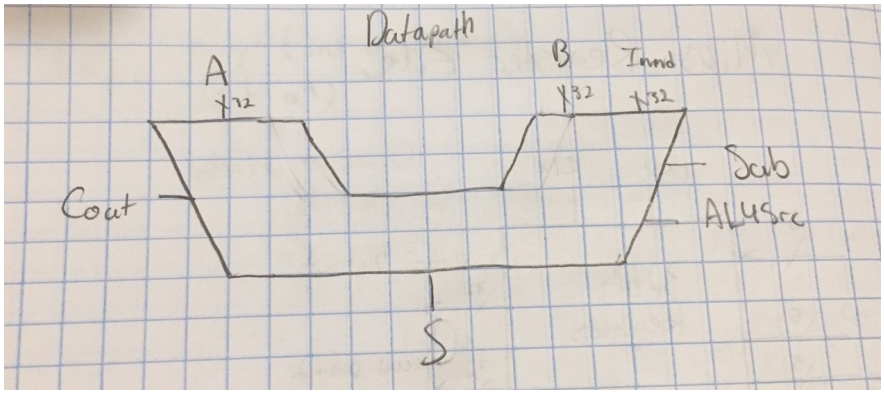
I tested the Reset, then writing, writing disabled, and that $0 doesn't get changed even if write is enabled.



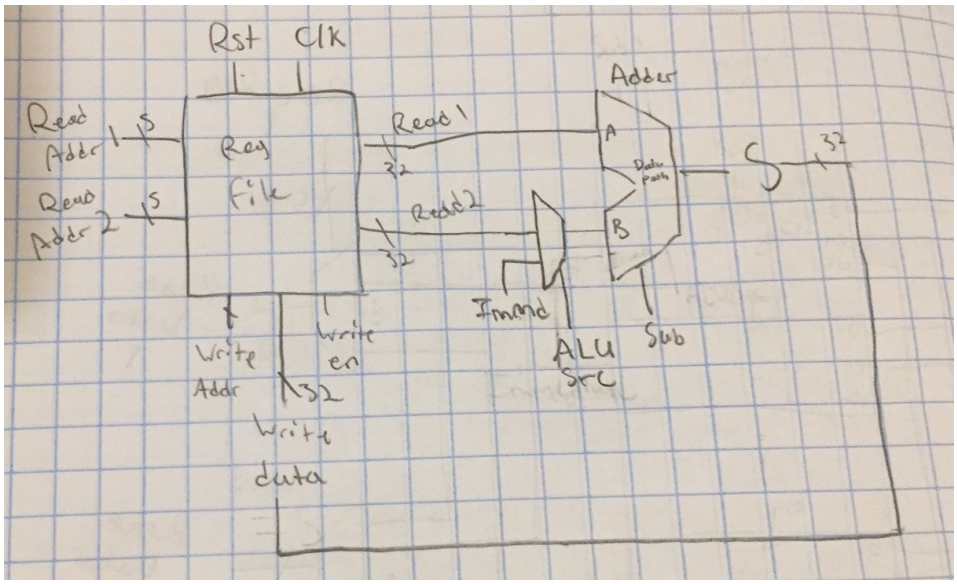
[Part 3 (a)] Waveform.
I tested adding A + B, A+Immd, Loading Immd, A-B, and A - Immd



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

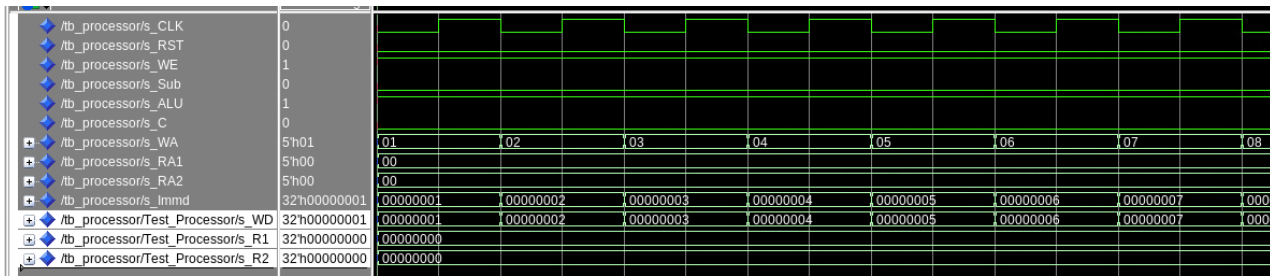


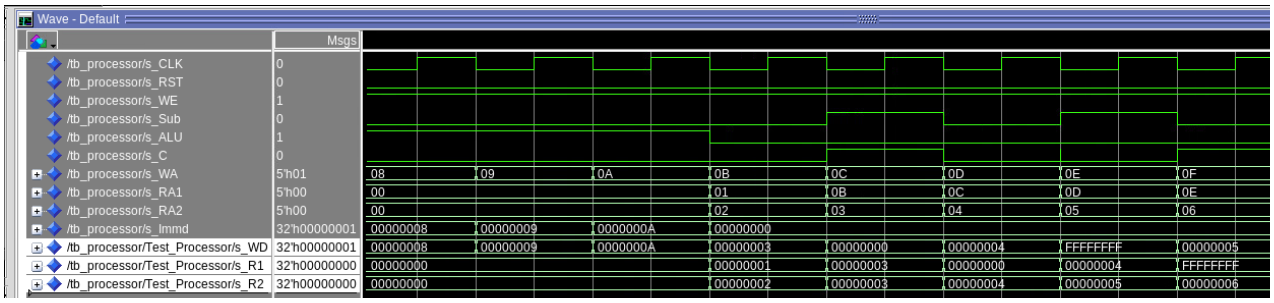
[Part 3 (c)] Draw a schematic of the simplified MIPS processor datapath consisting only of the component described in part (a) and the register file from problem (1).

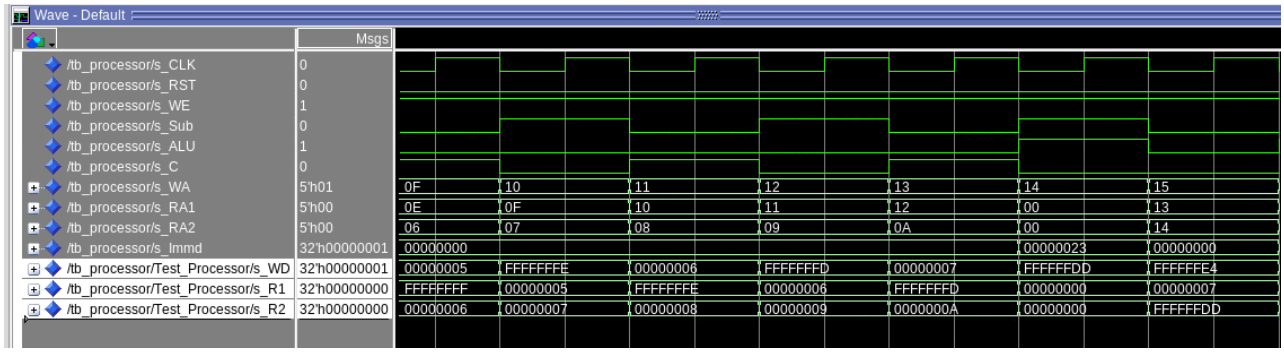


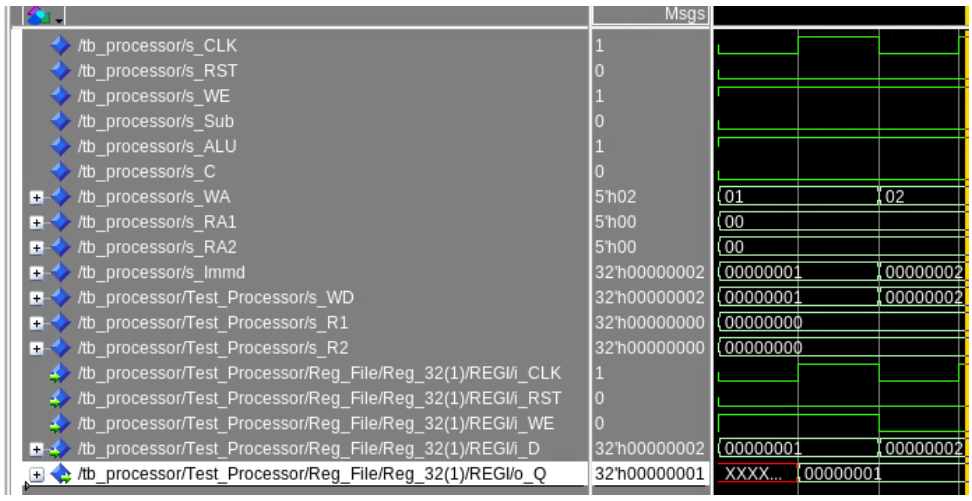
[Part 3 (d)] Include in your report waveform screenshots that demonstrate your properly functioning design. Annotate what the final register file state should be.

First two pictures show the loading of immediate values. Second and third show arithmetic between registers and loading negative value. Final image shows the first instruction being loaded into the actual register.
Final register is 7 – 35 = -28, which is shown in 3rd picture.









[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).

data_width – the size of memory at each address which is default 32 bit / 4bytes
addr_width – the amount of memory available regarding 2 ^ addr_width

clk – The rising edge clock which activates the loading of data into memory.

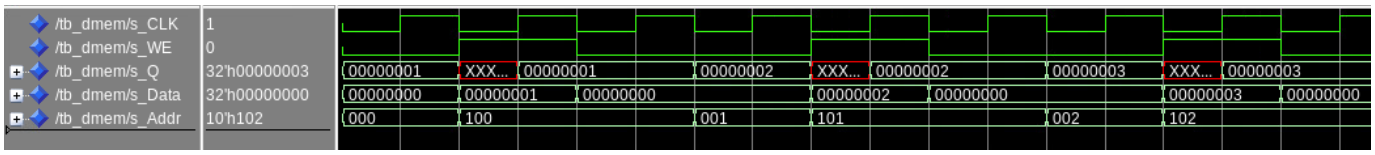
Addr – The address of memory can be any value 1023 to 0 in form of std logic vector. Is used to index an array ram of 32bit words.

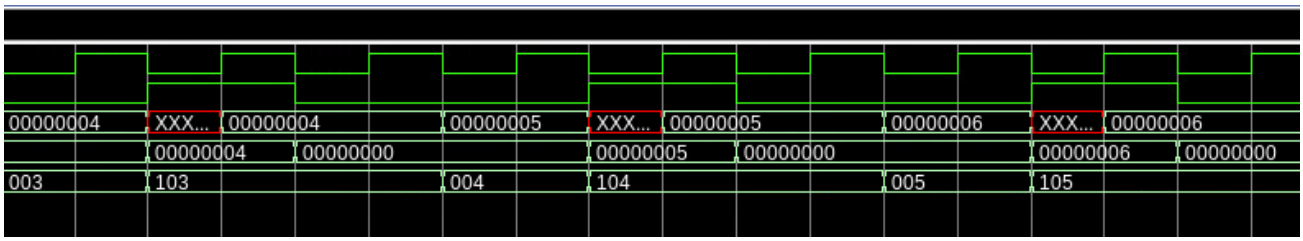
Data – The data to be stored into the array ram. Is an std logic vector of 32 bits in length.

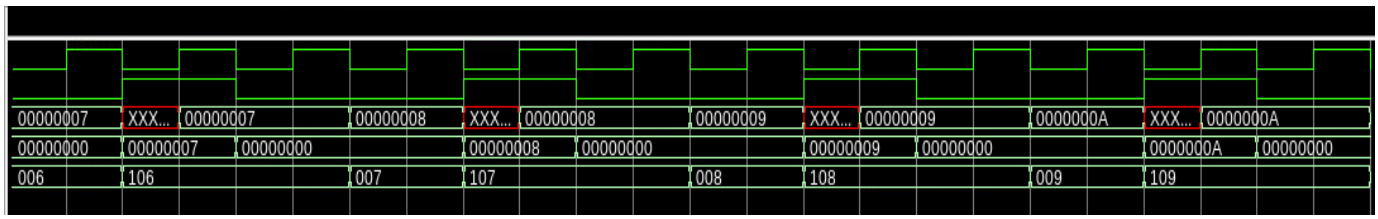
We – The write enable bit. If it is high, 1, then data can be stored into ram at addr.
Q – The return value. If the clk is rising then, we get back the value that was stored into ram. Otherwise we get what was already in ram at index Addr.

[Part 4 (c)] Waveforms.
I do each 3 steps at once for each memory location. So 0x0, 0x100, 0x1, 0x101, and so on.

I did not use the modified dmem that I created. Was not sure which I needed to use. So values are 1, 2, … and not negative.







[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?

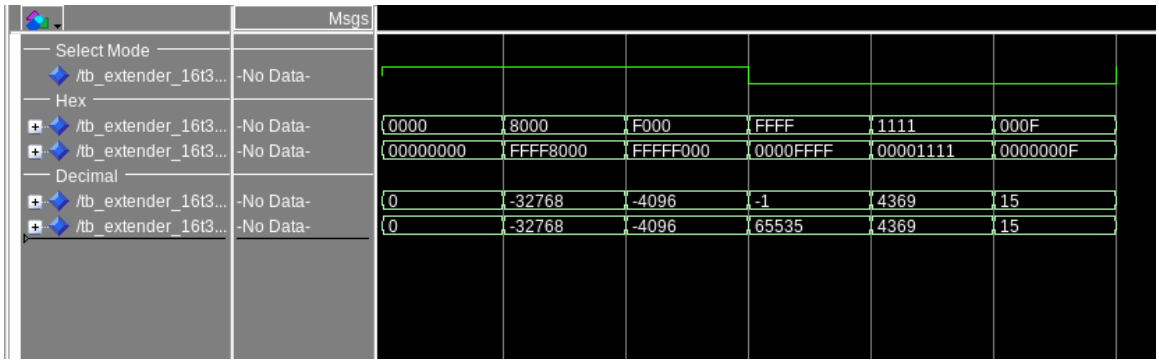
Zero: Andi, Ori
Sign: Addi, Addiu, load and stores (ll, lw, sb, etc), load and store floating point, Slti, Sltiu

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

I think this would just need an add gate. All it does is calculate the result of input ExtendType (high for signed and low for zero) and 15th bit of the data ANDed together. This result is then set down the outputs top 16 bits and the input 16 bits are set to the outputs bottom 16 bits.

[Part 5 (d)] Waveform.

Top bit is extend type
Top 16 bits is the input and bottom 32 bits is output.
Then, there are the values in decimal.

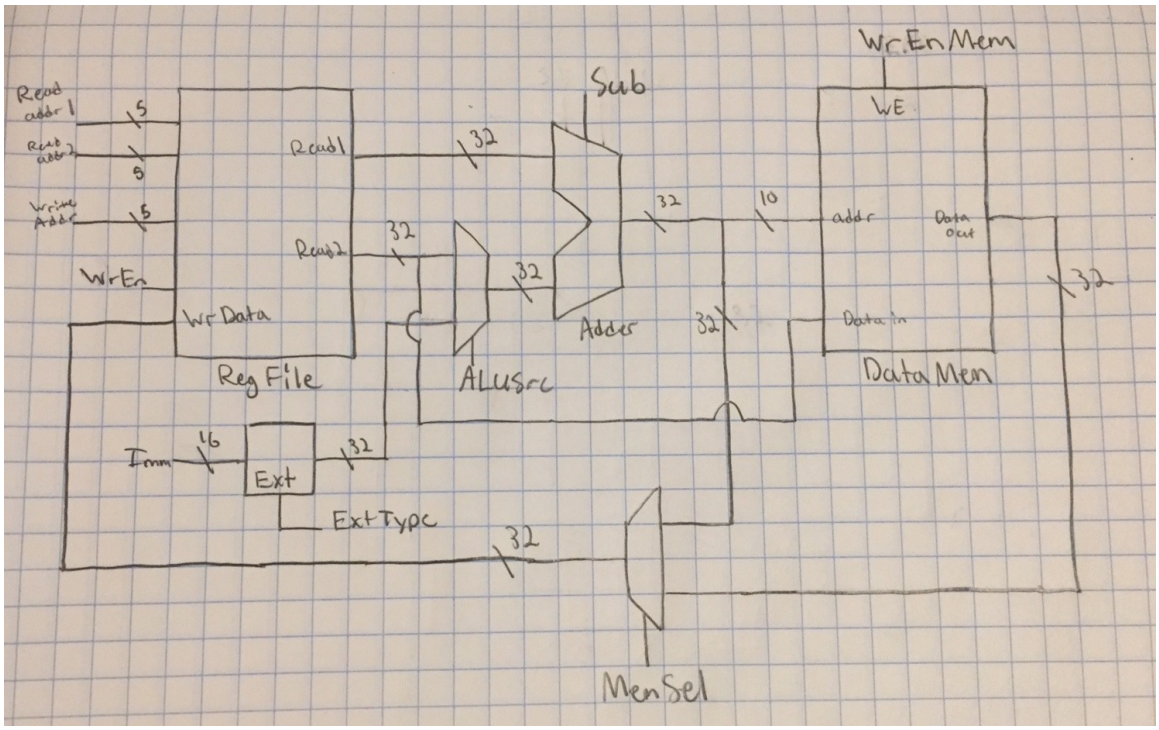


[Part 6 (a)] what control signals will need to be added to the simple processor from part 2? How do these control signals correspond to the ports on the mem.vhd component analyzed in part 3?

Write enable for memory
Select Mem or ALU for register write data
Zero or One extend select

We need an additional write enable memory bit. This is to control the stores to memory when using a sw instruction (or similar). Additionally, we need to add a control signal for a MUX to select if we are loading a value from memory.

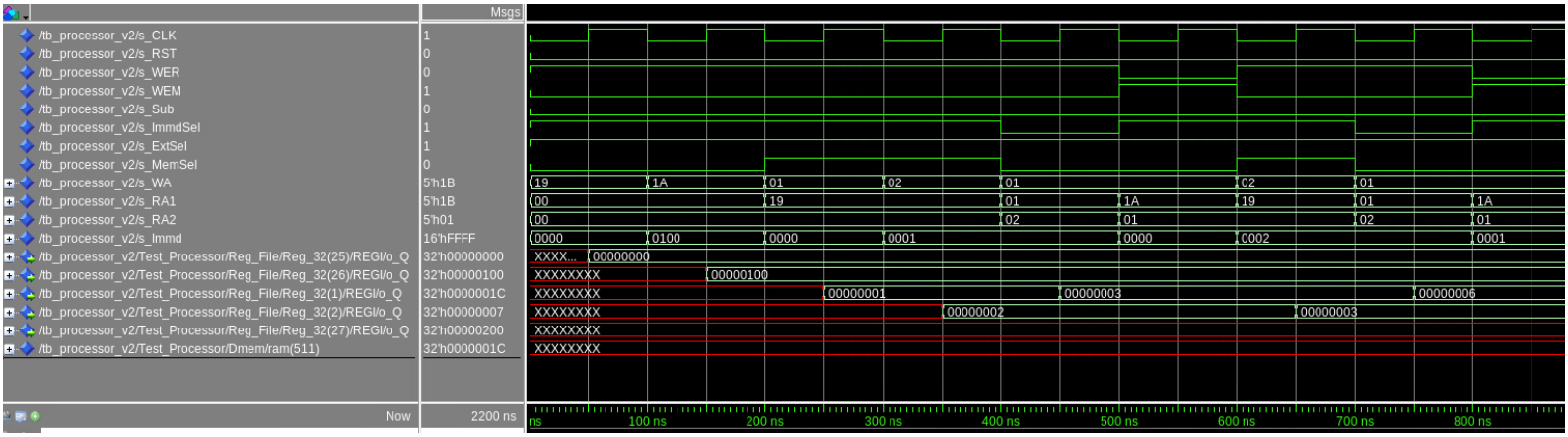
[Part 6 (b)] Draw a schematic of a simplified MIPS processor consisting only of the base components used in part 2, the extender component described in part 4, and exit

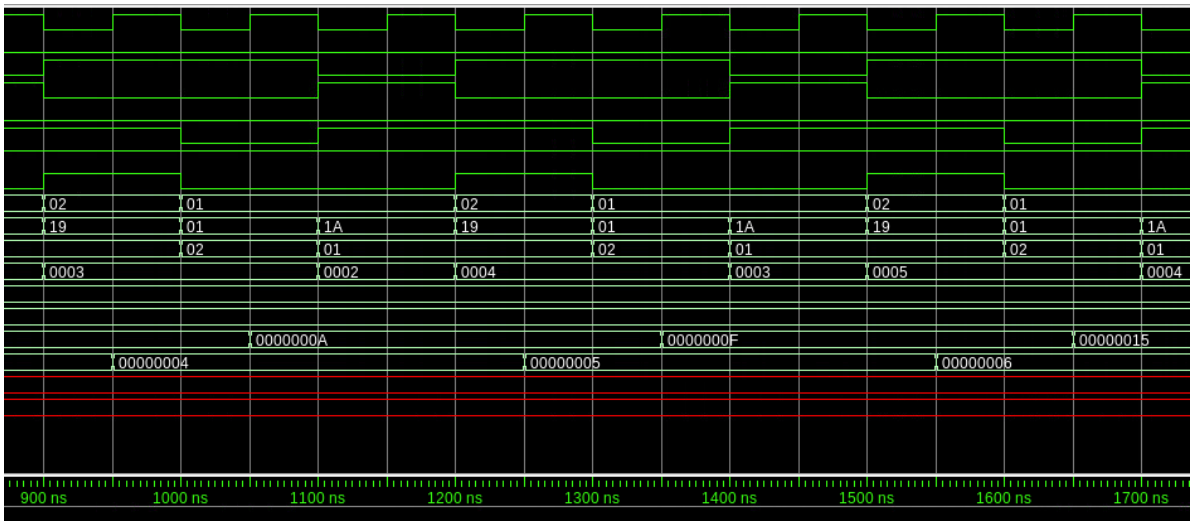


[Part 6 (c)] Waveform.

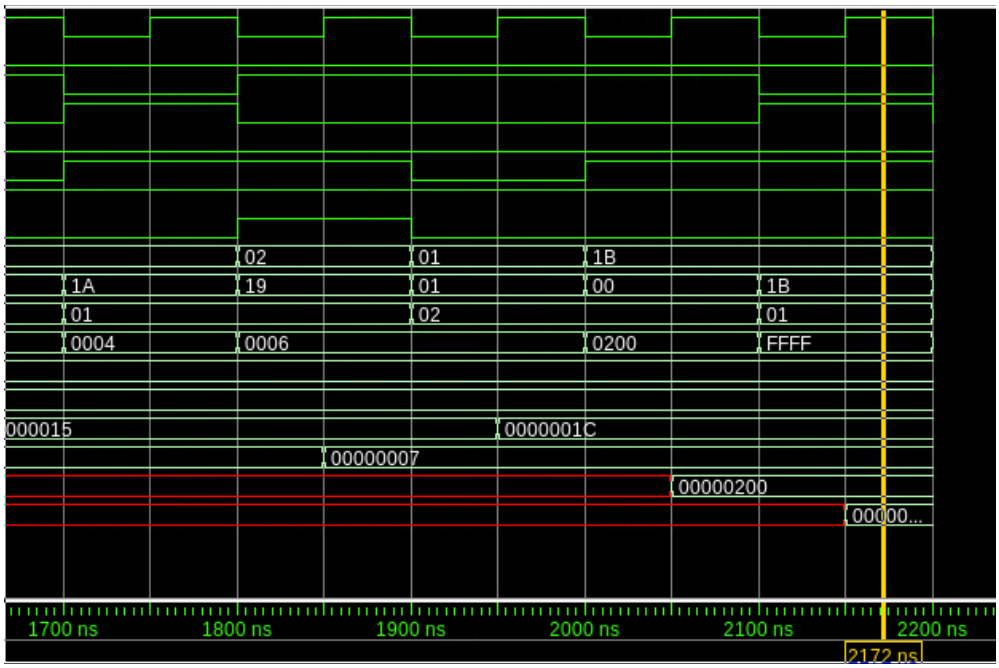
Fourth picture shows final value loaded in address 511 in memory.

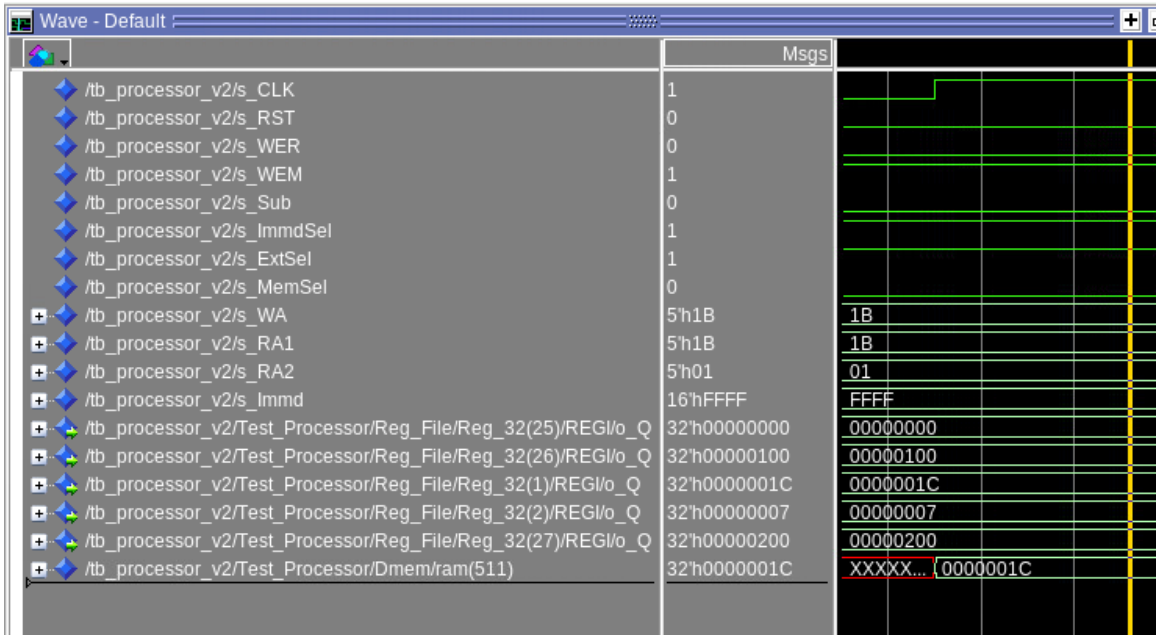
I did not use the modified dmem that I created. Was not sure which I needed to use. So values are 1, 2, … and not negative. This test loaded up values of 1, 2 … until 7 and adds them all to together. So the last value stored is 0x1C or 28.





(More on next page.)


02
01
1B
1A
19
01
00
1B
01
02
01
0004
0006
0200
FFFF
000015
0000001C
00000007
00000200
00000...
1700 ns
1800 ns
1900 ns
2000 ns
2100 ns
2200 ns
2172 ns


Wave - Default
Msgs
/tb_processor_v2/s_CLK
/tb_processor_v2/s_RST
/tb_processor_v2/s_WER
/tb_processor_v2/s_WEM
/tb_processor_v2/s_Sub
/tb_processor_v2/s_ImmdSel
/tb_processor_v2/s_ExtSel
/tb_processor_v2/s_MemSel
/tb_processor_v2/s_WA
/tb_processor_v2/s_RA1
/tb_processor_v2/s_RA2
/tb_processor_v2/s_Immd
/tb_processor_v2/Test_Processor/Reg_File/Reg_32(25)/REGI/o_Q
/tb_processor_v2/Test_Processor/Reg_File/Reg_32(26)/REGI/o_Q
/tb_processor_v2/Test_Processor/Reg_File/Reg_32(1)/REGI/o_Q
/tb_processor_v2/Test_Processor/Reg_File/Reg_32(2)/REGI/o_Q
/tb_processor_v2/Test_Processor/Reg_File/Reg_32(27)/REGI/o_Q
/tb_processor_v2/Test_Processor/Dmem/ram(511)
1
0
0
1
0
1
1
0
5'h1B
5'h1B
5'h01
16'hFFFF
32'h00000000
32'h00000100
32'h0000001C
32'h00000007
32'h00000200
32'h0000001C
1B
1B
01
FFFF
00000000
00000100
0000001C
00000007
00000200
XXXXX...
0000001C