CprE 381, Computer Organization and Assembly-Level Programming

Lab 2 Report

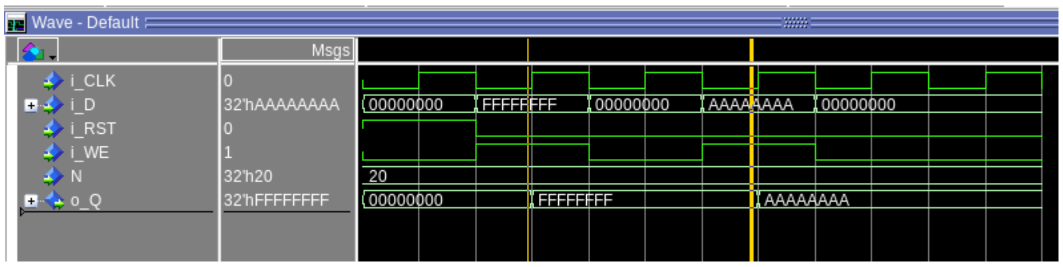
Student Name Shivansh Patel

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 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.

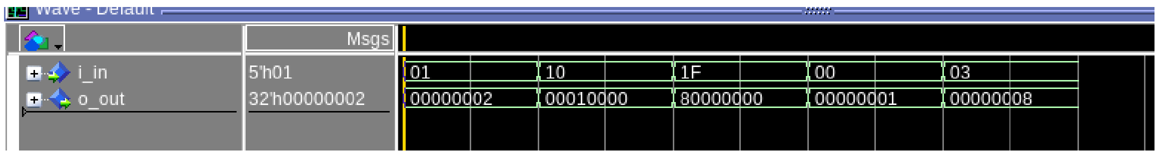


We can see that when the write enable is 1and reset is off at the first cursor, then the output changes to match the data entry. However, when the write enable is 0 between the cursors, the output does not change.

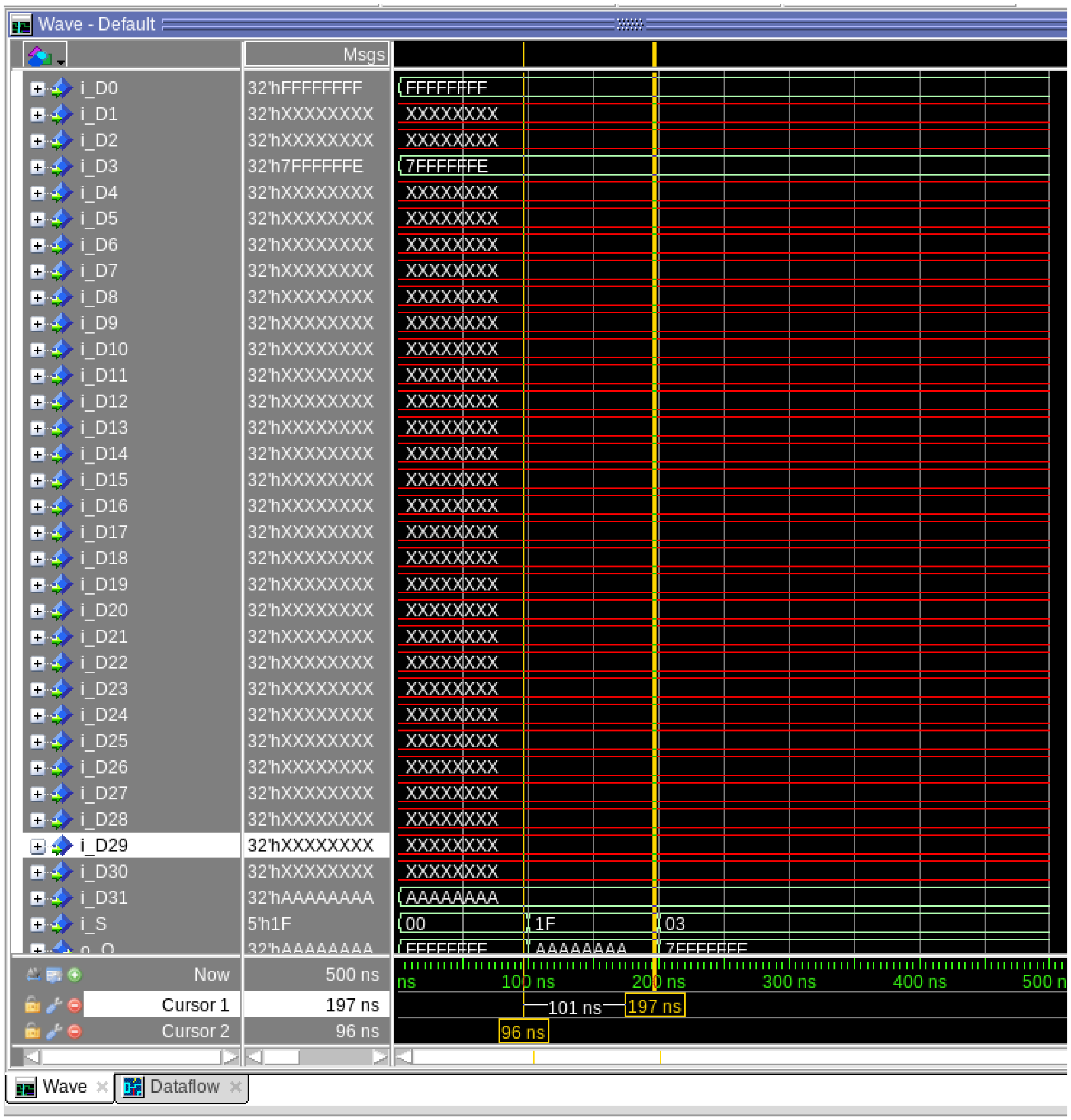
[Part 2 (d)] What type of decoder would be required by the MIPS register file and why?

We would need a 5:32 bit demux in order to determine which register in the mips file we want to write to.

[Part 2 (e)] Waveform.



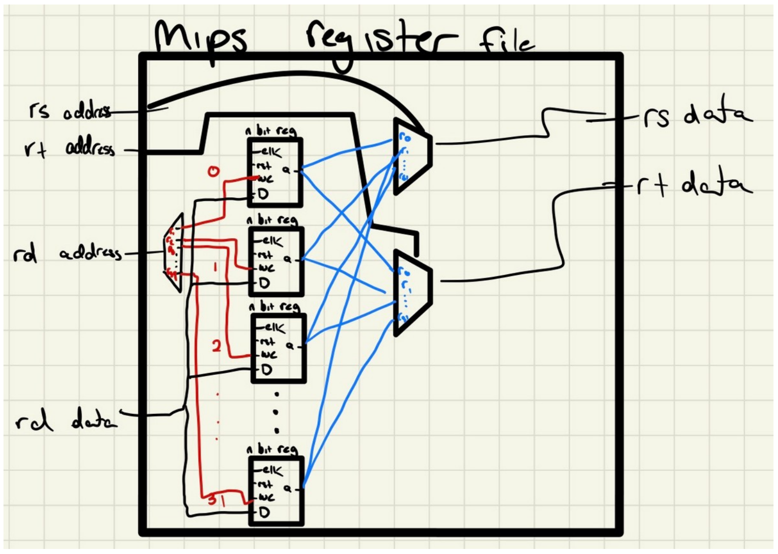
[Part 2 (f)] In your write-up, describe and defend the design you intend on implementing for the next part.
I would want to do the 32:1 mux through a dataflow architecture approach, I think this would be more simple than doing an approach of generating a series of 2:1 muxes and connecting them for the same functionality. We would need to have access to each of the 32 bits and connect them to the registers to select our output.
[Part 2 (g)] Waveform.


Wave - Default

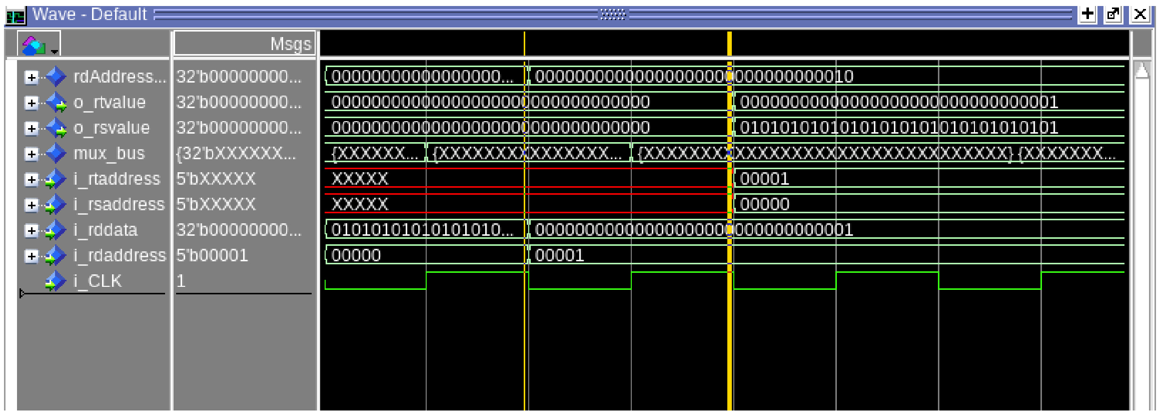
	Msgs
i_D0	32'hFFFFFFFF
i_D1	32'hXXXXXXXX
i_D2	32'hXXXXXXXX
i_D3	32'h7FFFFFFE
i_D4	32'hXXXXXXXX
i_D5	32'hXXXXXXXX
i_D6	32'hXXXXXXXX
i_D7	32'hXXXXXXXX
i_D8	32'hXXXXXXXX
i_D9	32'hXXXXXXXX
i_D10	32'hXXXXXXXX
i_D11	32'hXXXXXXXX
i_D12	32'hXXXXXXXX
i_D13	32'hXXXXXXXX
i_D14	32'hXXXXXXXX
i_D15	32'hXXXXXXXX
i_D16	32'hXXXXXXXX
i_D17	32'hXXXXXXXX
i_D18	32'hXXXXXXXX
i_D19	32'hXXXXXXXX
i_D20	32'hXXXXXXXX
i_D21	32'hXXXXXXXX
i_D22	32'hXXXXXXXX
i_D23	32'hXXXXXXXX
i_D24	32'hXXXXXXXX
i_D25	32'hXXXXXXXX
i_D26	32'hXXXXXXXX
i_D27	32'hXXXXXXXX
i_D28	32'hXXXXXXXX
i_D29	32'hXXXXXXXX
i_D30	32'hXXXXXXXX
i_D31	32'hAAAAAAAA
i_S	5'h1F
o_O	32'hAAAAAAAA
Now	500 ns
Cursor 1	197 ns
Cursor 2	96 ns

In the first cursor, I am reading out the value of the D0 input and the output represents that. In the second cursor segment I am reading the value of D31, and in the 3rd cursor I am reading the value of the D3 input and returning 7FFFFFFE in my output.

[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.



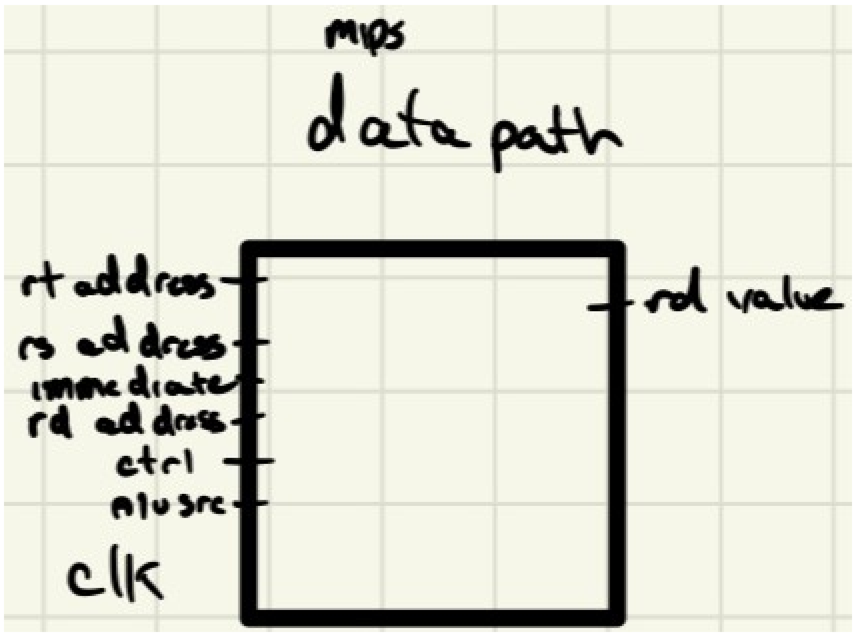
[Part 2 (i)] Waveform.



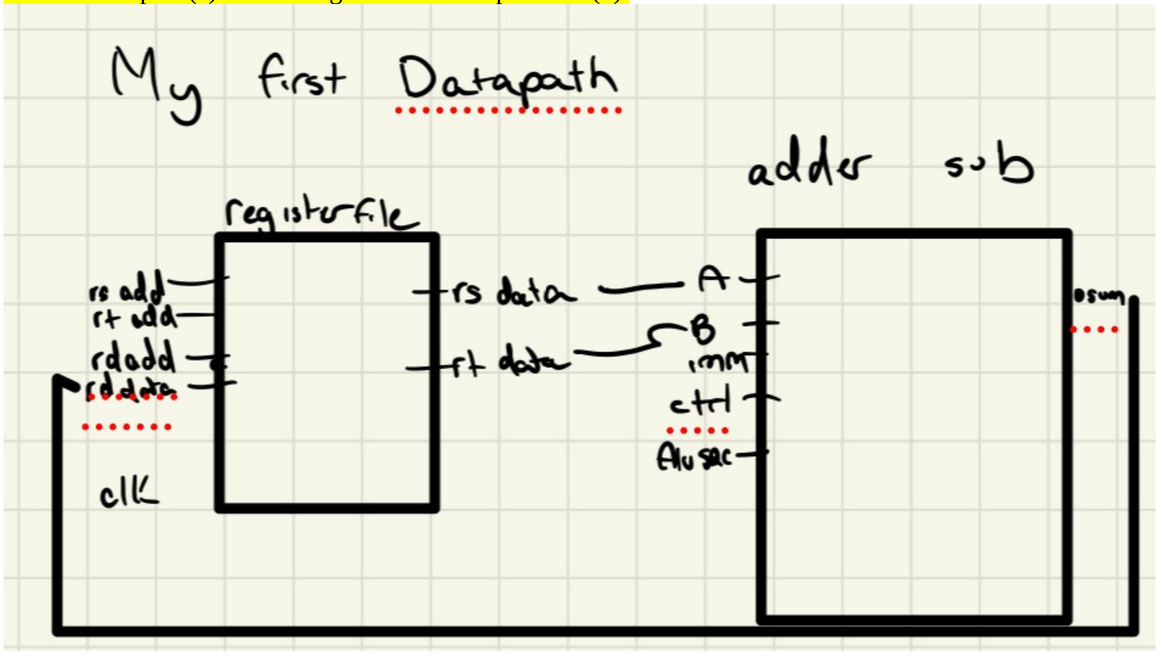
```
rdAddress...  32'b00000000...  00000000000000000...  00000000000000000000000000000010
o_rtvalue     32'b00000000...  00000000000000000000000000000000  00000000000000000000000000000001
o_rsvalue     32'b00000000...  00000000000000000000000000000000  01010101010101010101010101010101
mux_bus       {32'bXXXXXX...  {XXXXXX...  {XXXXXXXXXXXXXXX...  {XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX}  {XXXXXXX...
i_rtaddress   5'bXXXXX        XXXXX  00001
i_rsaddress   5'bXXXXX        XXXXX  00000
i_rddata      32'b00000000...  01010101010101010...  00000000000000000000000000000001
i_rdaddress   5'b00001        00000  00001
i_CLK         1
```

In the first cursor, I am setting rd address to the 0 register and setting the 0 register's value to 01010… in the second cursor, I am setting register 1's value to 000001. In the third cursor I am reading rt and rs and reading out the values which I set in the first and second cursor.

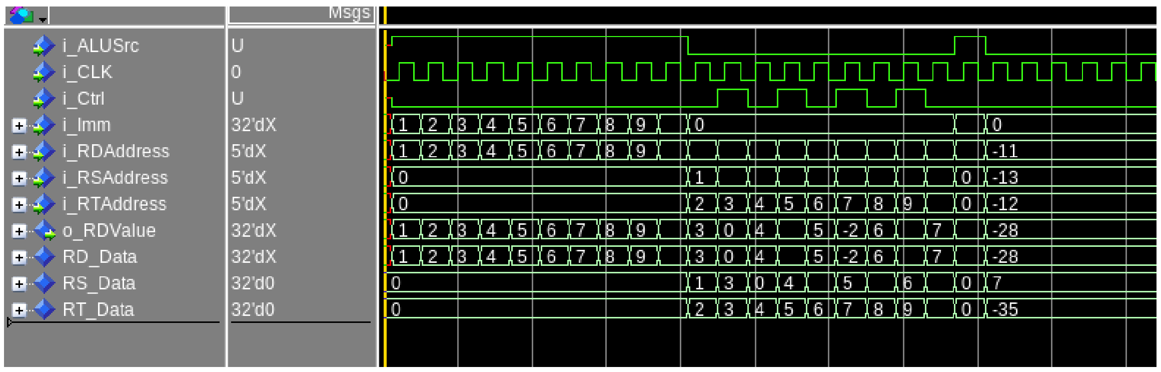
[Part 3 (b)] Draw a symbol for this MIPS-like datapath.

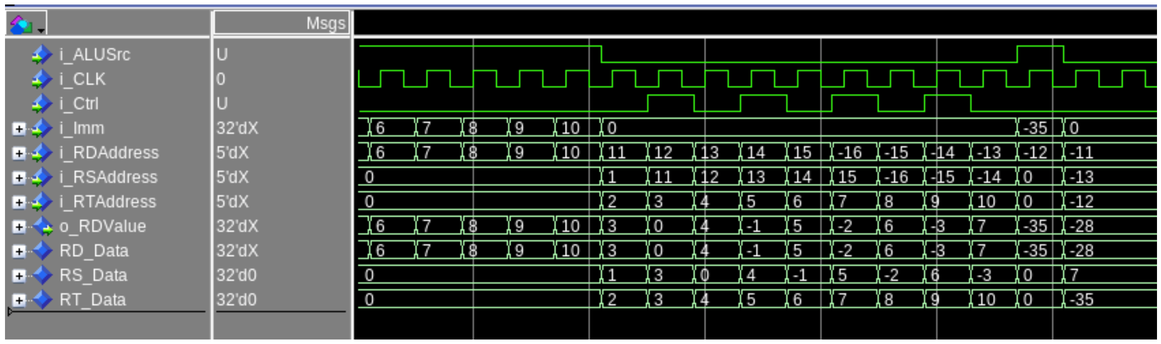


[Part 3 (c)] Draw a schematic of the simplified MIPS processor datapath consisting only of the component described in part (a) and the register file from problem (1).



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



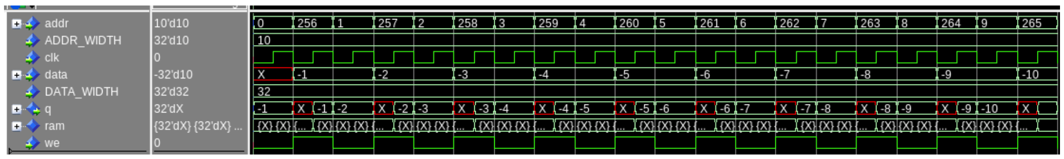


We see that for the first few clock cycles from 1 – 10, our immediate value is 1-10, and the register address we are storing it in (rd) corresponds to 1-10. This means that the function begins by working as the output value also corresponds. Next we go on to add the value from registers 1 and 2 into register 11. We see that the functionality is correct because rs and rt's address is 1 and 2 and the data inside is 1 and 2 respectively. We get an output value of 3 because we are adding them. In register 12, we test the subtraction function and see that the output value is 0 because we have our value of 3 and subtract a value of 3. This tests all of the functionality of the datapath.

[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 memory file gives us a 10 by 32 2d array for memory. If our write enable is 1 then we set the address 1-10 of the memory to equal our data value a 32 bit number. We then return that modified or unmodified value of the memory at the address.

[Part 4 (c)] Waveforms.



The memory is initially loaded with the values of -1 through -10 from ram values of 0 through 10. Then within each segment, the address is starting out by setting the output to the initial -1 where addr = 0 and the write enable is disabled putting the value of -1 into our q and data value for the next cycle, then the addr is moved up to 256 and the data is written when write enable is set to 1. This is repeated 10 times incramentally.

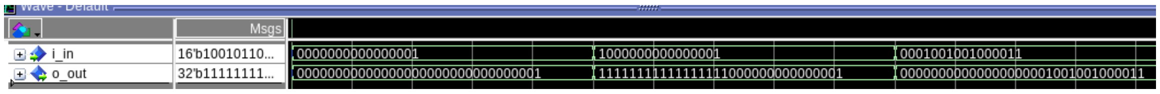
[Part 5 (a)] What are the MIPS instructions that require some value to be sign extended? What are the MIPS instructions that require some value to be zero extended?

A mips instruction that requires sign extension would be something like SRA vs SRL where with SRA you would need to keep track of the signed bit to extend it. Load word load byte etc will also require a sign extension if you are loading in a signed byte into a register then you want to keep its value so you will need the extender.

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

> You would need a separate extender component to go through before you store a word into a register if it is not 32 bits already.

[Part 5 (d)] Waveform.



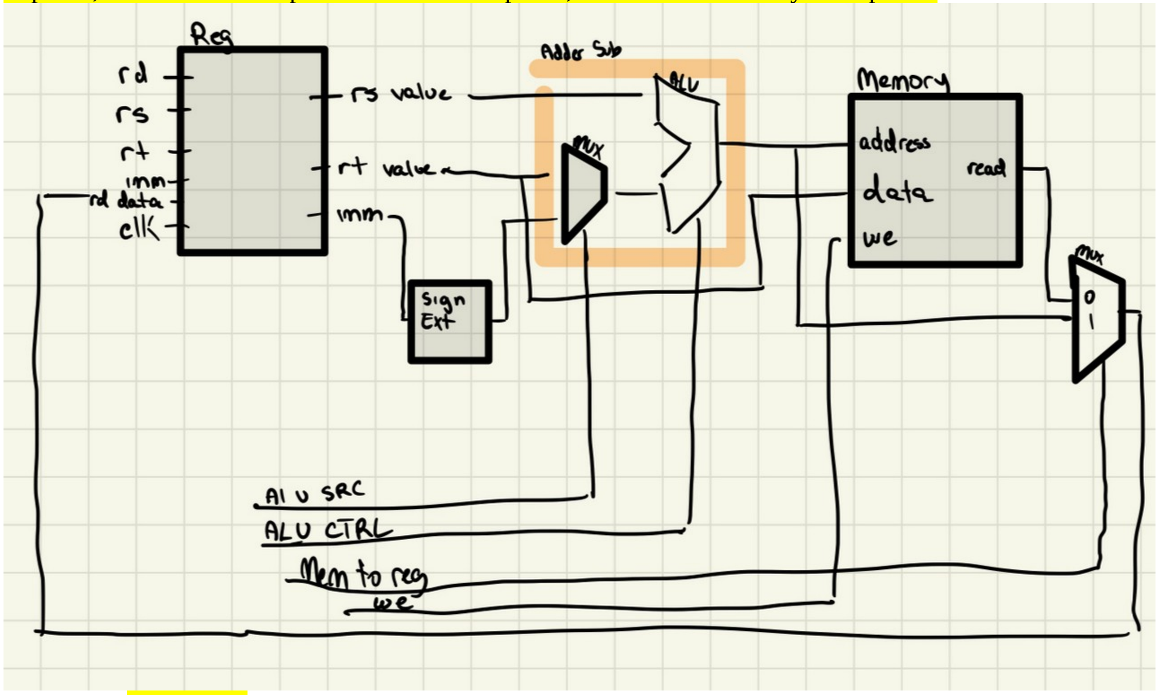
```
Wave - Default
          Msgs
i_in      16'b10010110...   0000000000000001                  1000000000000001                  0001001001000011
o_out     32'b11111111...   00000000000000000000000000000001  11111111111111111000000000000001  00000000000000000001001001000011
```

In this waveform, the input is 16 bits and the upper 16 bits of the output corresponds to the top bit of the input.

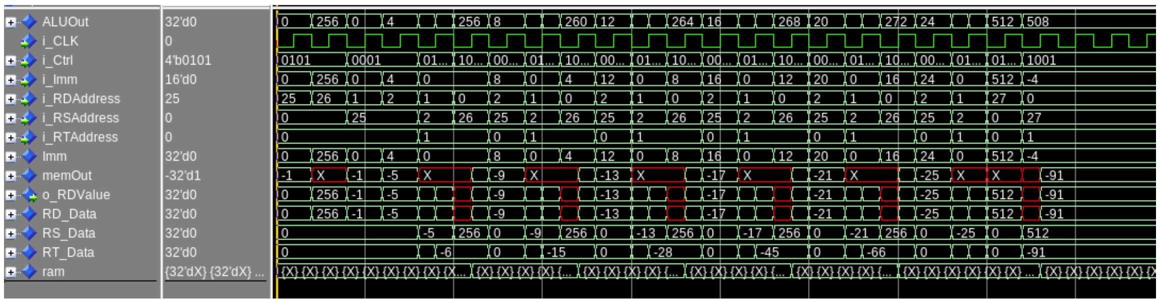
[Part 6 (a)] what control signals will need to be added to the simple processor from part 2? How do these control signals correspond to the ports on the mem.vhd component analyzed in part 3?

In order to get the memory functionality in the mips processor, we will still need our alu control for add/sub, we need an alu src for immediate vs register value, we will need to add a control value to tell if we want to take the output of the register as our write value or the output of the alu as our write value. We also need to add a control value for if we want to write to the memory or not.

[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.



ALUOut	32'd0
i_CLK	0
i_Ctrl	4'b0101
i_Imm	16'd0
i_RDAddress	25
i_RSAddress	0
i_RTAddress	0
Imm	32'd0
memOut	-32'd1
o_RDValue	32'd0
RD_Data	32'd0
RS_Data	32'd0
RT_Data	32'd0
ram	{32'dX} {32'dX} ...

For this waveform, I modified dmem.hex to continue placing values from memory address 0 to mem address 32 as values of -1 - -32. In the waveform, I can check that all of the functions are working correctly because we initially have our value loaded in from memory 0+4 as -5 and we add that to -1 and get a value of -6. We store this value of -6 into the memory of array B. This then continues until we store our last value into the memory address at memory 508, or the last value of B. We can certify that this should be -91 as a result of adding all of the previous components of RT_Data which represents the sequence of doing add $1, $1, $2 after each lw.