CprE 381, Computer Organization and Assembly-Level Programming

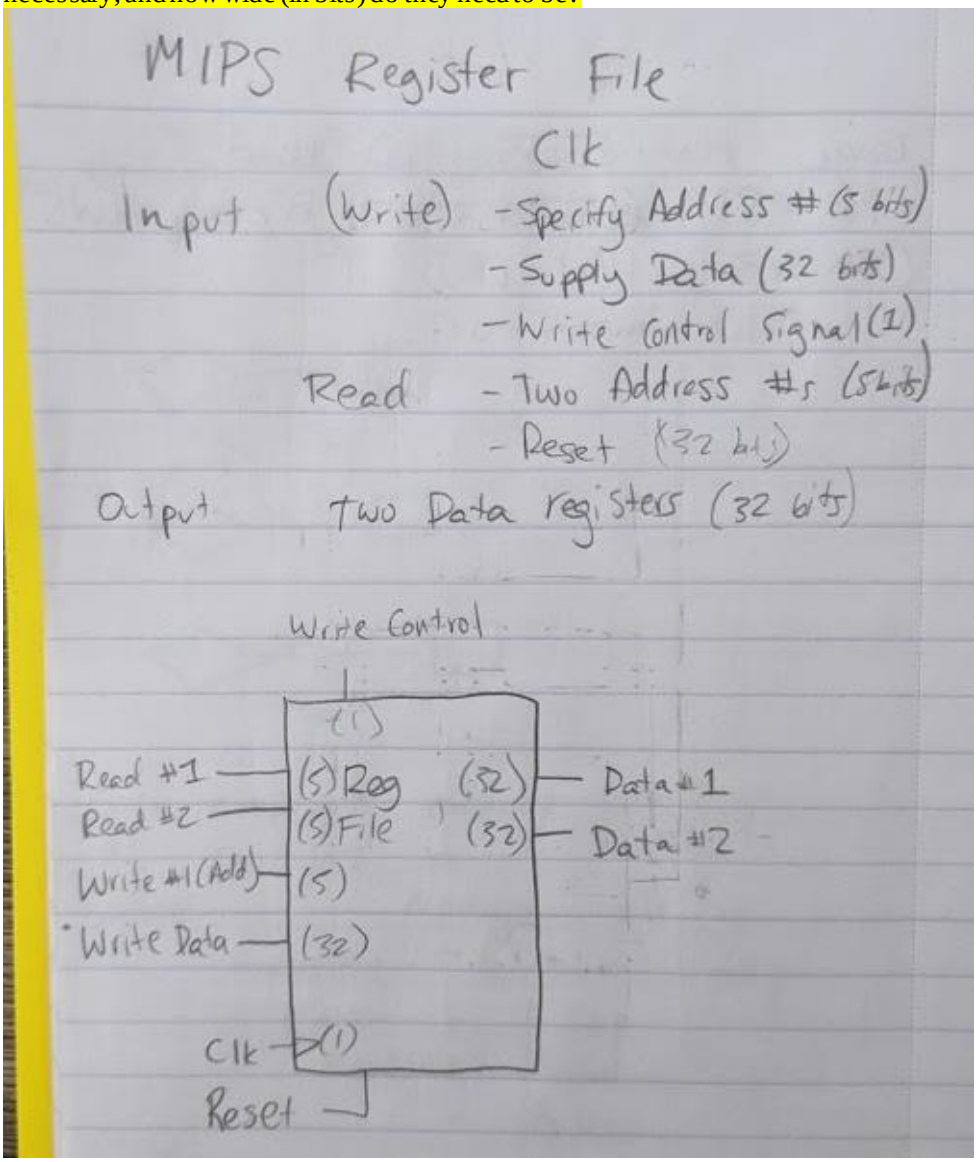
Lab 2 Report

Student Name _____Andrew Deick_____

Prelab: The reset type is active high, because the register is reset when reset goes high. The reset is asynchronous because the reset is checked before the clock. The edge sensitivity of the register is rising edge.

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.

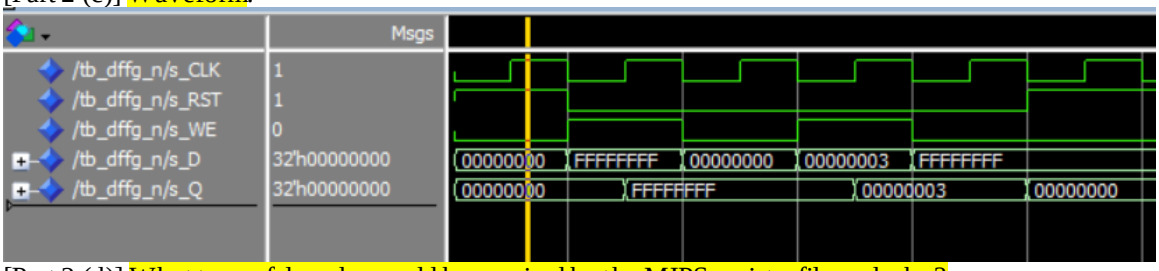
[Part 2 (a)] Draw the interface description for the MIPS register file. Which ports do you think are necessary, and how wide (in bits) do they need to be?



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

I created the register and named it dffg_n.vhd

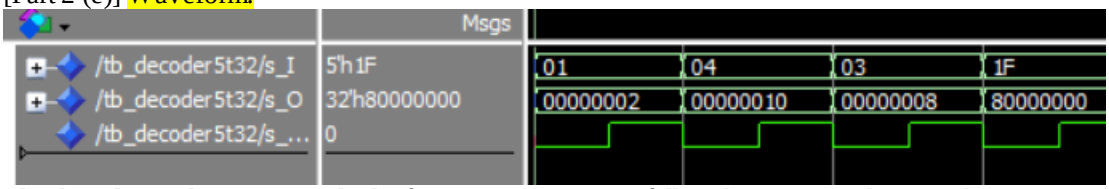
[Part 2 (c)] Waveform.



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

Because there are 32 registers in the MIPS register file, we need a 5:32 decoder to determine which register to write to.

[Part 2 (e)] Waveform.

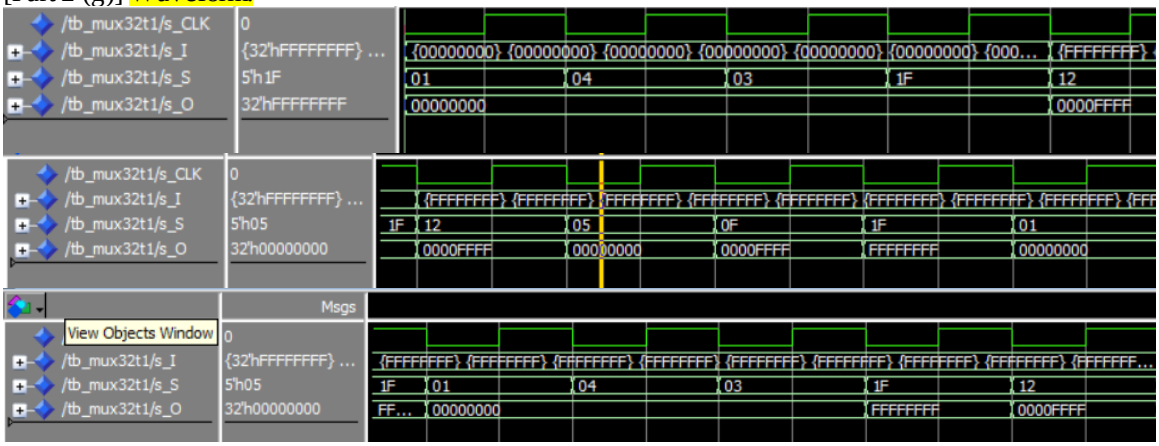


The decoder works as expected. The five inputs are successfully selecting a single output bit.

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

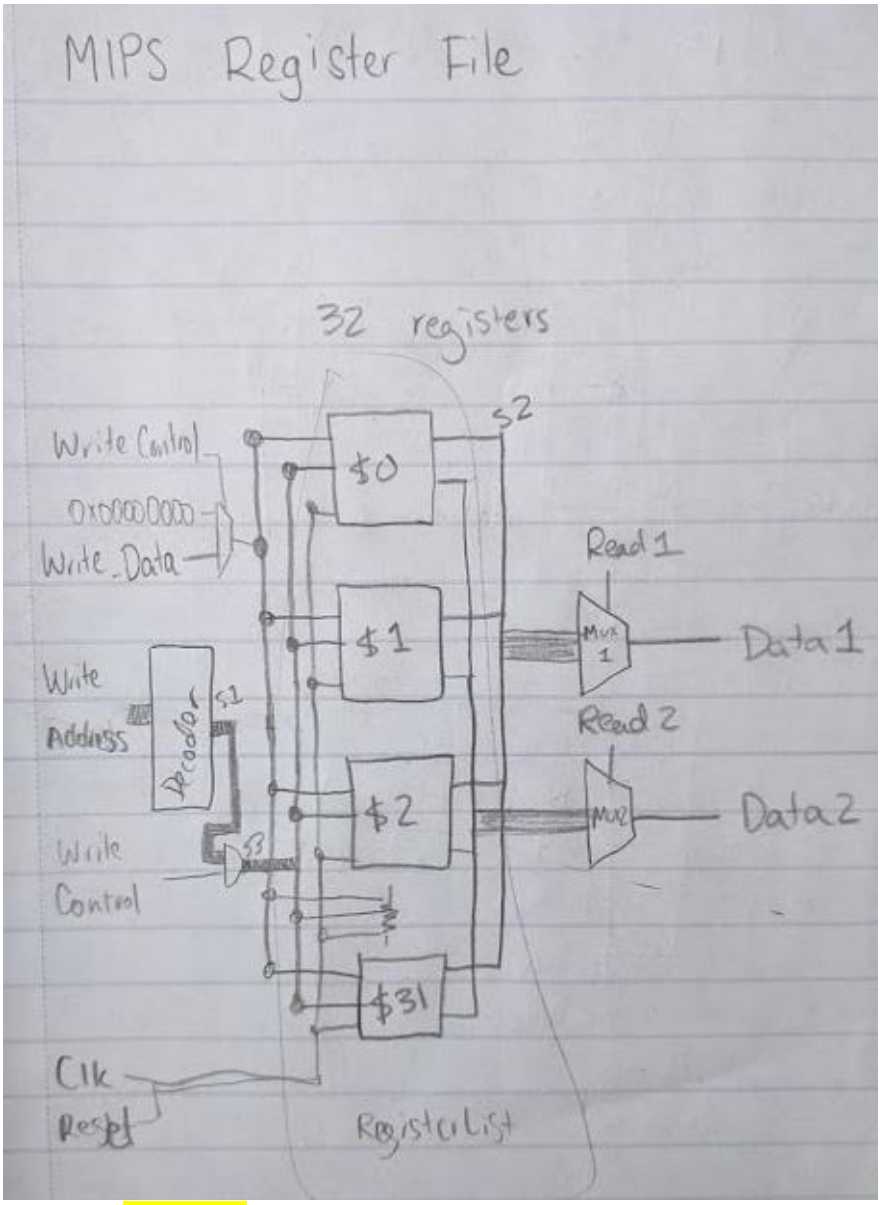
I am going to create a two dimensional array that is 32x32. The 5 select bits will choose which of the 32 inputs to load and to transfer the single dimension array of 32 bits through to the output.

[Part 2 (g)] Waveform.

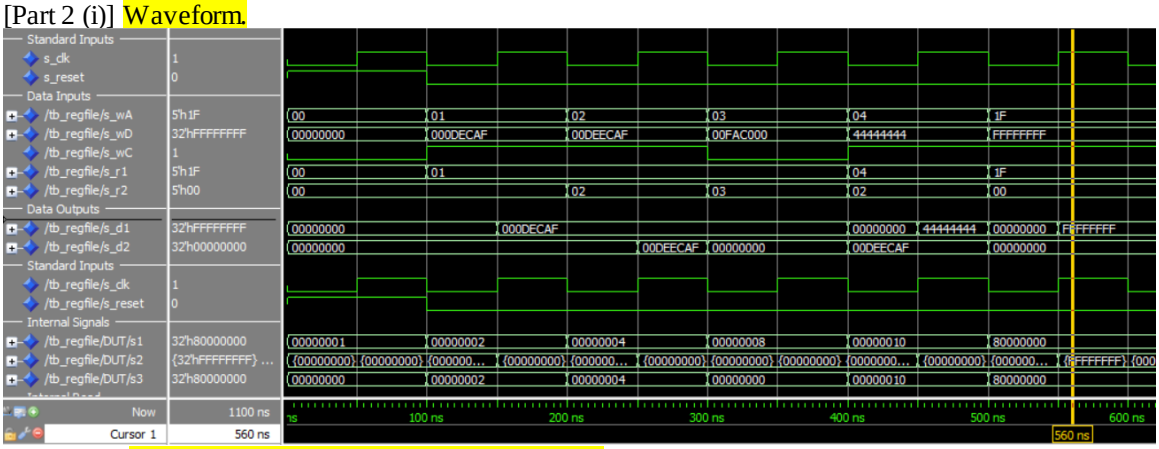


I have a total of 12 test cases. For the first four test cases, all inputs are set to '0x00000000'. For the remainder of the test cases, registers 0-10 are 0x00000000, 11-20 are 0x0000FFFF, and 21-31 are 0xFFFFFFFF. Then, we reuse the first four switch values at the end (because input values are different).

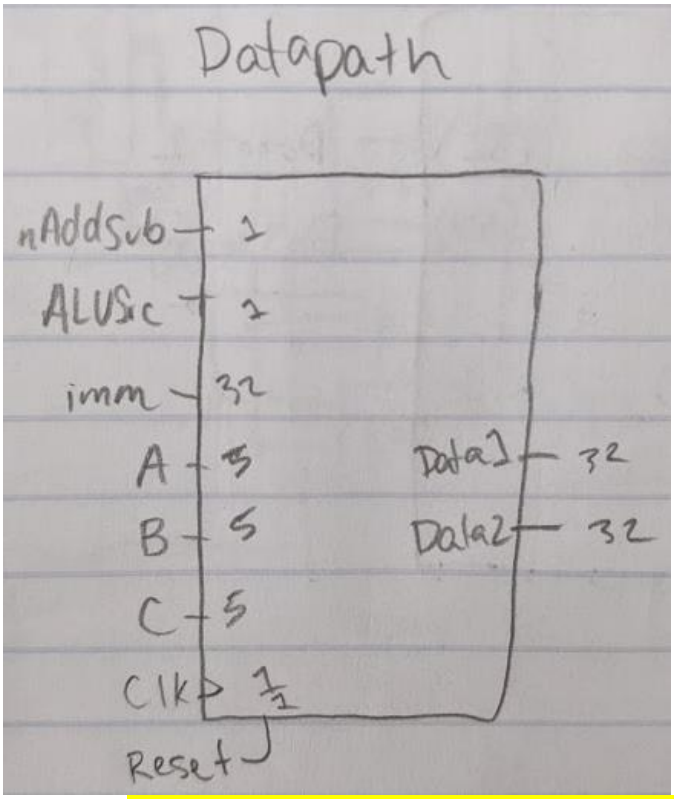
[Part 2 (h)] Draw a (simplified) schematic for the MIPS register file, using the same top-level interface ports as in your solution describe above and using only the register, decoder, and mux VHDL components you have created.



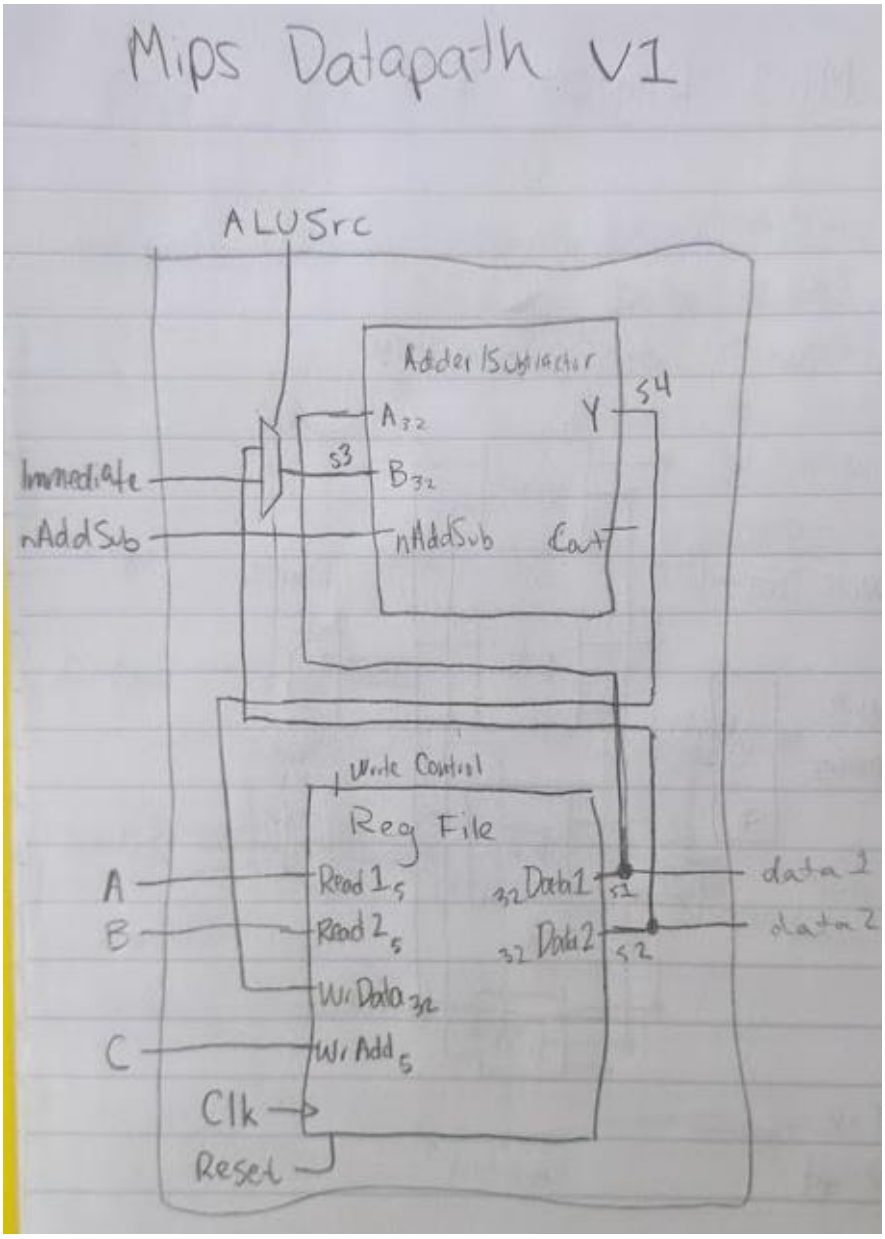
[Part 2 (i)] Waveform.



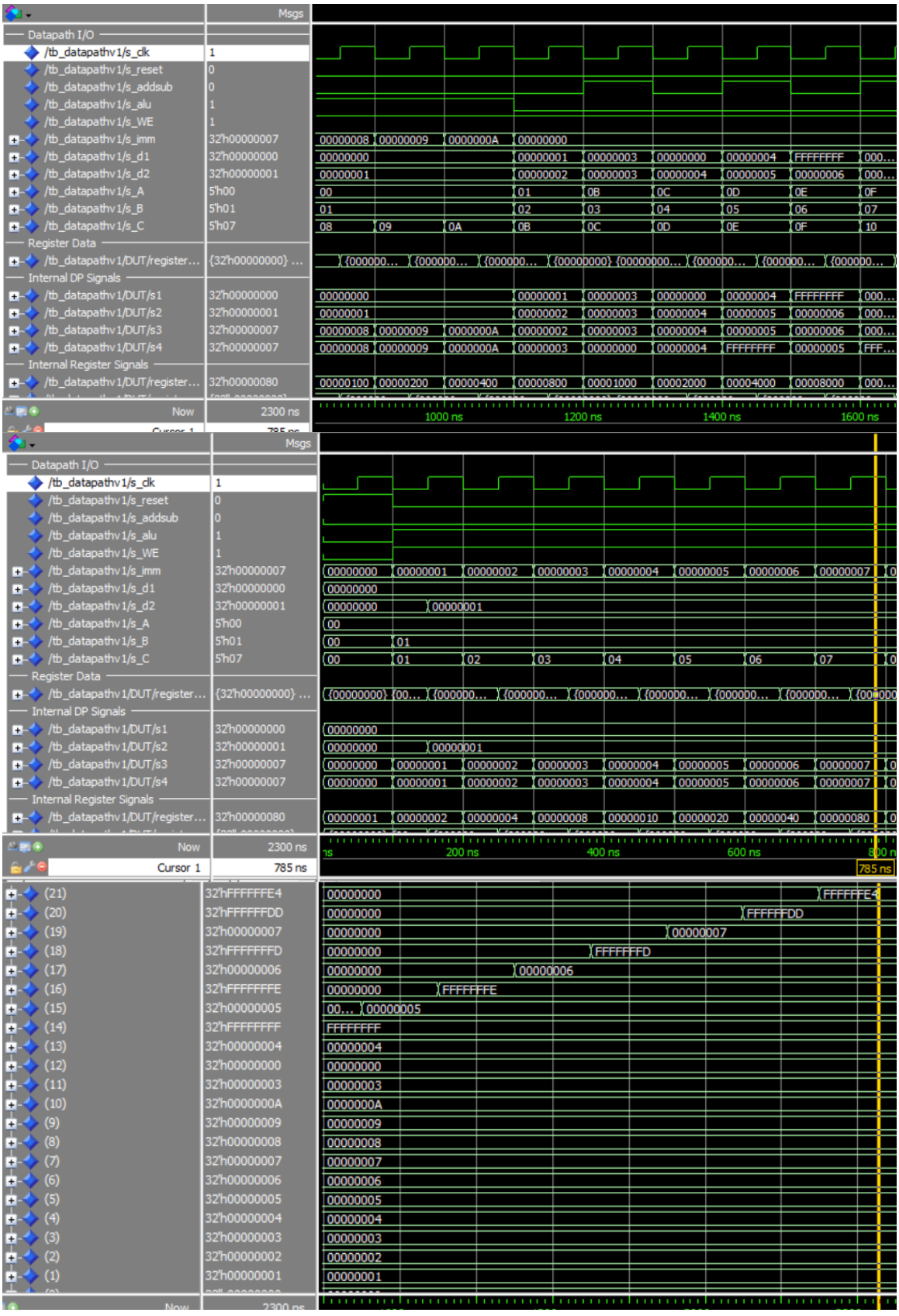
[Part 3 (b)] Draw a symbol for this MIPS-like datapath.



[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 waveforms screenshots that demonstrate your properly functioning design. Annotate what the final register file state should be.


Msgs
Datapath I/O
/tb_datapathv1/s_clk
/tb_datapathv1/s_reset
/tb_datapathv1/s_addsub
/tb_datapathv1/s_alu
/tb_datapathv1/s_WE
/tb_datapathv1/s_imm
/tb_datapathv1/s_d1
/tb_datapathv1/s_d2
/tb_datapathv1/s_A
/tb_datapathv1/s_B
/tb_datapathv1/s_C
Register Data
/tb_datapathv1/DUT/register...
Internal DP Signals
/tb_datapathv1/DUT/s1
/tb_datapathv1/DUT/s2
/tb_datapathv1/DUT/s3
/tb_datapathv1/DUT/s4
Internal Register Signals
/tb_datapathv1/DUT/register...
Now
2300 ns
1000 ns
1200 ns
1400 ns
1600 ns
Msgs
Datapath I/O
/tb_datapathv1/s_clk
/tb_datapathv1/s_reset
/tb_datapathv1/s_addsub
/tb_datapathv1/s_alu
/tb_datapathv1/s_WE
/tb_datapathv1/s_imm
/tb_datapathv1/s_d1
/tb_datapathv1/s_d2
/tb_datapathv1/s_A
/tb_datapathv1/s_B
/tb_datapathv1/s_C
Register Data
/tb_datapathv1/DUT/register...
Internal DP Signals
/tb_datapathv1/DUT/s1
/tb_datapathv1/DUT/s2
/tb_datapathv1/DUT/s3
/tb_datapathv1/DUT/s4
Internal Register Signals
/tb_datapathv1/DUT/register...
Now
2300 ns
Cursor 1
785 ns
200 ns
400 ns
600 ns
785 ns
(21)
(20)
(19)
(18)
(17)
(16)
(15)
(14)
(13)
(12)
(11)
(10)
(9)
(8)
(7)
(6)
(5)
(4)
(3)
(2)
(1)

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000001
$v0	2	0x00000002
$v1	3	0x00000003
$a0	4	0x00000004
$a1	5	0x00000005
$a2	6	0x00000006
$a3	7	0x00000007
$t0	8	0x00000008
$t1	9	0x00000009
$t2	10	0x0000000a
$t3	11	0x00000003
$t4	12	0x00000000
$t5	13	0x00000004
$t6	14	0xffffffff
$t7	15	0x00000005
$s0	16	0xfffffffe
$s1	17	0x00000006
$s2	18	0xfffffffd
$s3	19	0x00000007
$s4	20	0xffffffdd
$s5	21	0xffffffe4

The expected results are from MARS, and we can verify that we have received the correct result.

[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 is a generic that describes the size of the memory module. Like the registers, it is sized at 32 bits.

Address Width is a generic that describes how many bits are needed to specify a module, and it therefore also tells us how many modules there are. 2^10 = 1024, so it looks like there is going to be anywhere from 512 to 1024 modules in order for the address size to be reasonable. Because the number of modules is often based on the address width, it is a good assumption to make that there is 1024 memory modules.

Clk is straightforwardly, a clock that allows the module to function on edges.

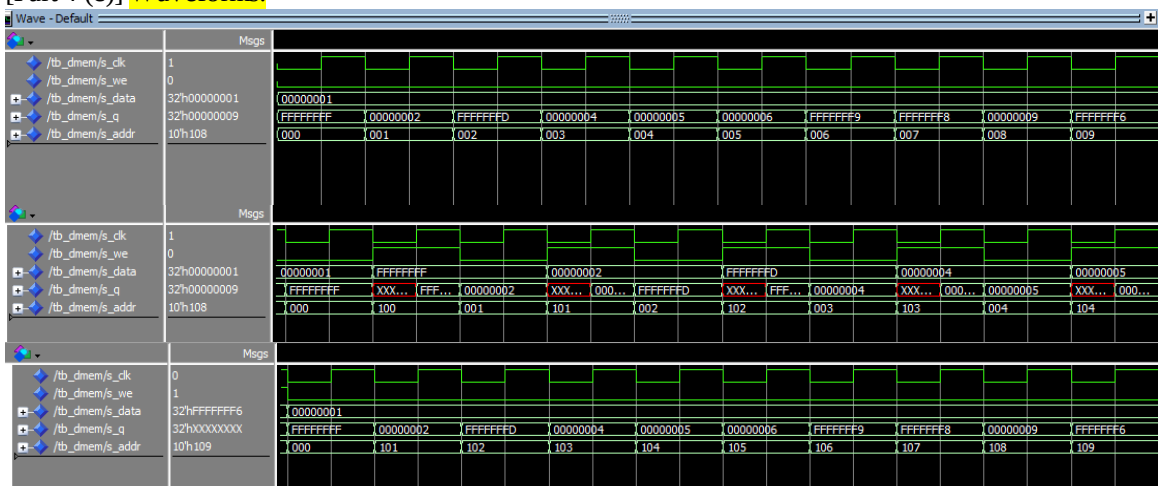
Addr is the memory address that is being written to and that is outputting data.

Data describes the input data to the memory module. It should be written in tandem with WE.

WE describes "Write Enable". There will always be data input to the register, but it should only overwrite the current contents when WE is high.

Q describes the output of memory. It returns 'data width'. Internally, it is plugged back into the input in case 'WE' is low, so that the module can 'remember'. We can expect it to be undefined before anything is written to it.

[Part 4 (c)] Waveforms.



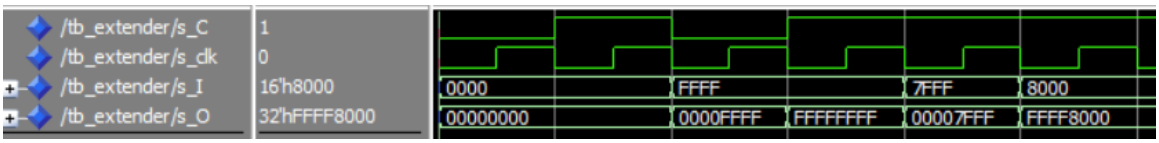
[Part 5 (a)] What are the MIPS instructions that require some value to be sign extended? What are the MIPS instructions that require some value to be zero extended?

Some MIPS instructions, like addiu deal in unsigned numbers. Therefore, we would want to zero extend the immediate. Other instructions, like addi, store, and load give signed immediates because sometimes their values are to be interpreted as a negative. Therefore, those should be zero extended. Logic instructions, that we may implement later on, like AND, OR, or XOR, will also be zero extended.

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

> The MIPS processor would require a sign extender and a zero extender. I implemented both of these, with a control switch for MIPS to choose which one. The first 16 bits will be copied always, and the last 16 bits are set to the 15th (sign bit) AND the control signal. Therefore, if the control signal is zero, then the bits will be zero. If the control bit is 1, then the bits (specifically the last 16) will be set to the value of the sign bit.

[Part 5 (d)] Waveform.

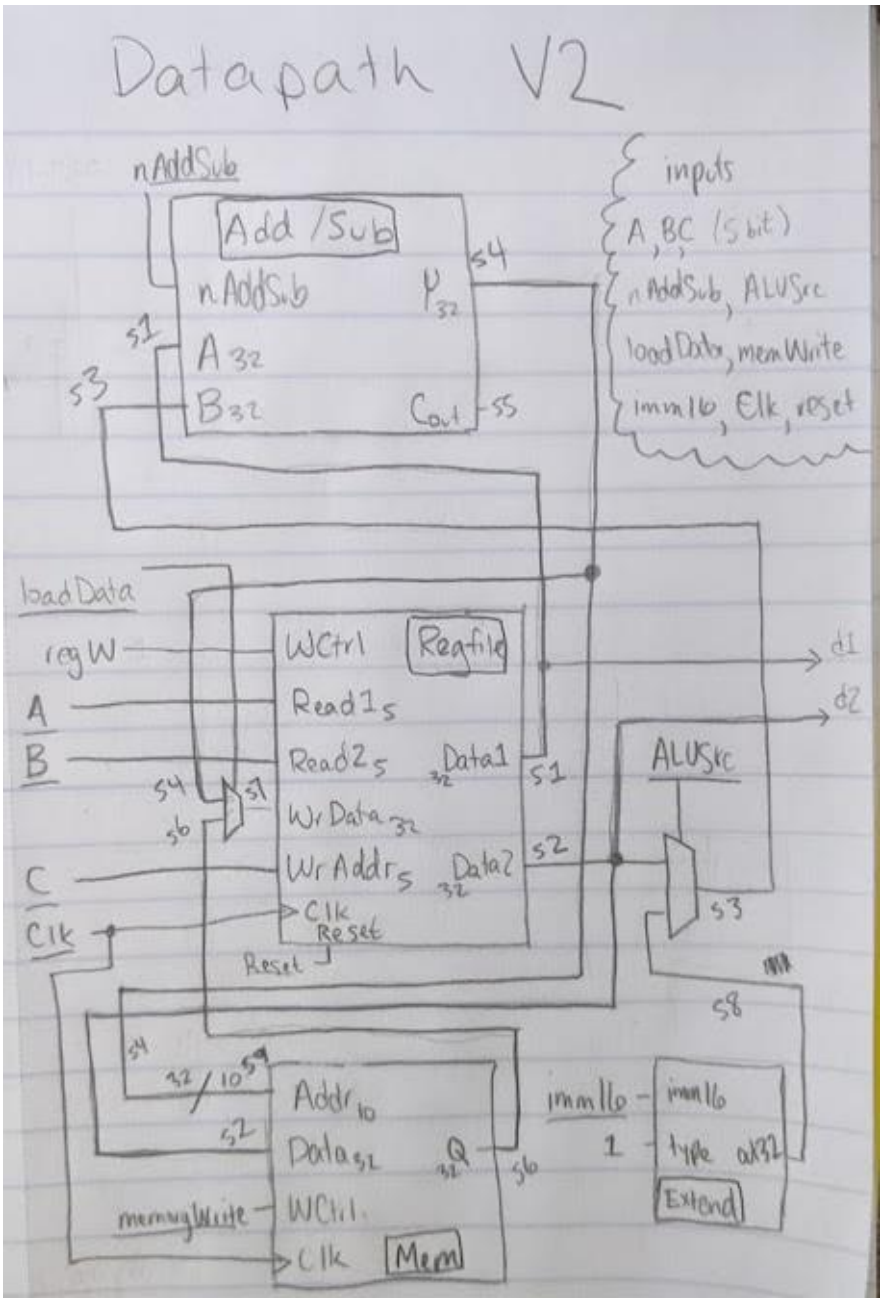


All bits have expected values.

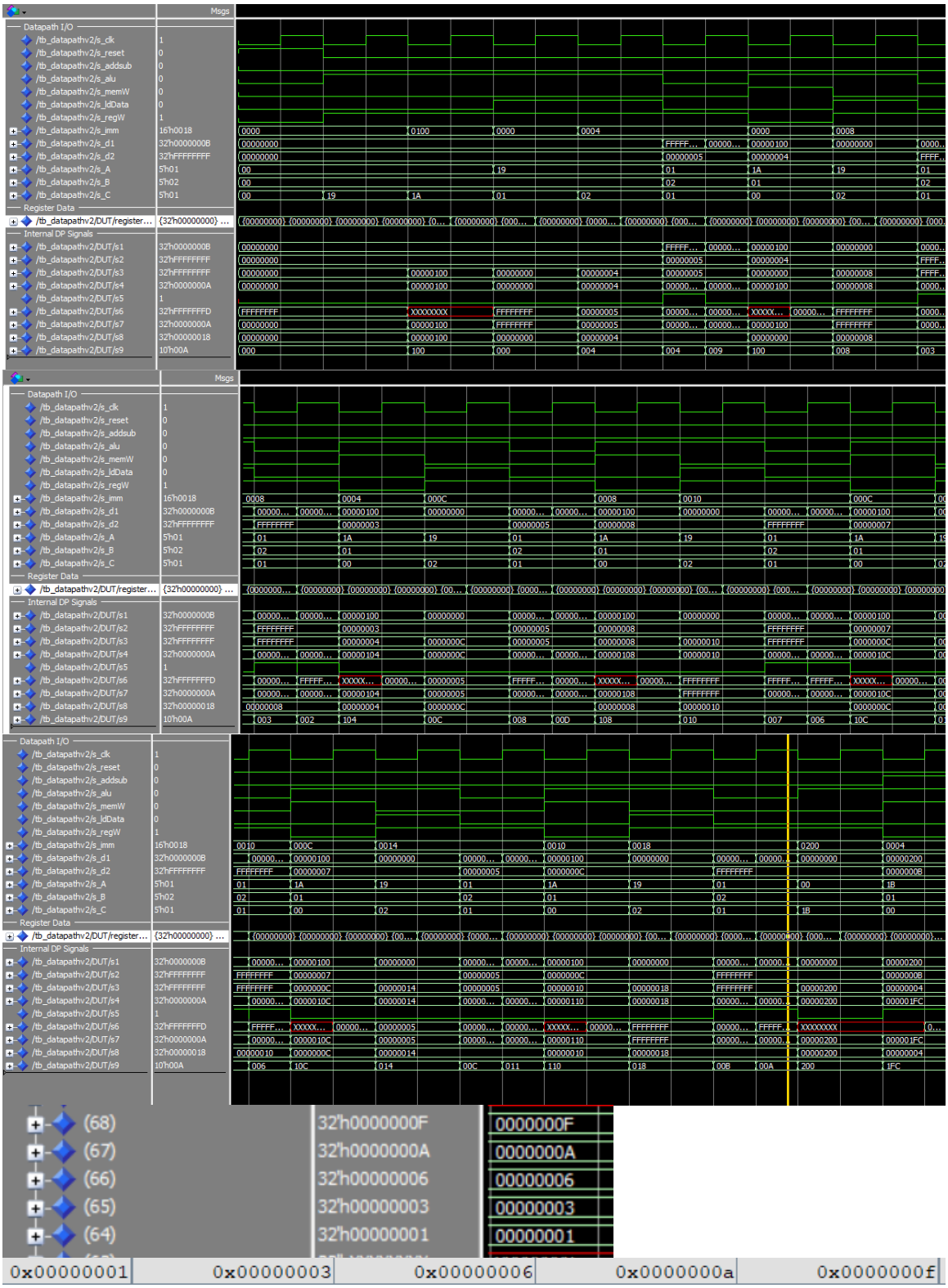
[Part 6 (a)] what control signals will need to be added to the simple processor from part 2? How do these control signals correspond to the ports on the mem.vhd component analyzed in part 3?

I changed the name of "WriteControl" to "Register Write" to more accurately reflect its' function, because I've since added two new controls. The first is "Memory Write", to determine when the memory should change to reflect its' input (specifically on a **sw** instruction) and I've also added "Load Data" which is 0 if data should be accessed from the ALU, but is 1 if data should be loaded from memory (specifically in a **lw** instruction). I also elected not to add a control signal for the extend type, as all of the instructions currently require sign extension, so I've written a '1' to the input. As more instructions are added, it may be necessary to zero extend, which will add another control signal.

[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.



There are some red undefined values, but this is to be expected because the memory values that I am reading are undefined. This is not a problem, because proper MUX configurations ensure that an undefined value is never read as an input. The memory's end state matches the memory of MARS, and everything runs smoothly.