CprE 381, Computer Organization and Assembly-Level Programming

Lab 2 Report

Student Name Jackson Hafele

[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?

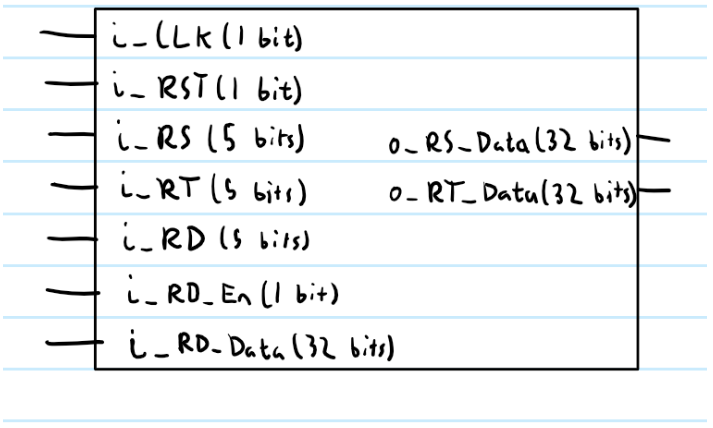


Figure 1. MIPS Register File Blackbox Diagram

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

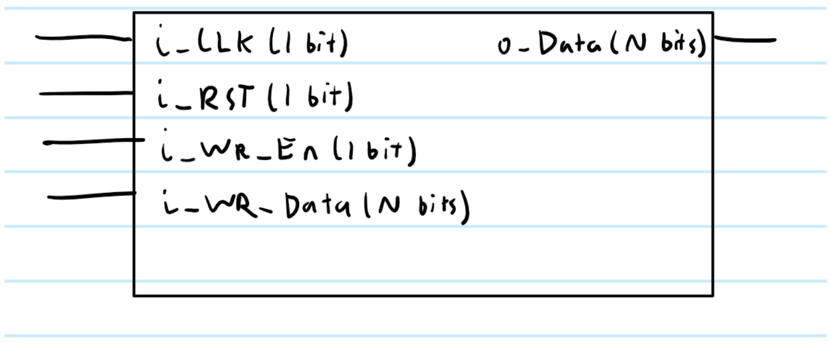


Figure 2. N-bit Register File Blackbox Diagram

[Part 2 (c)] Waveform.

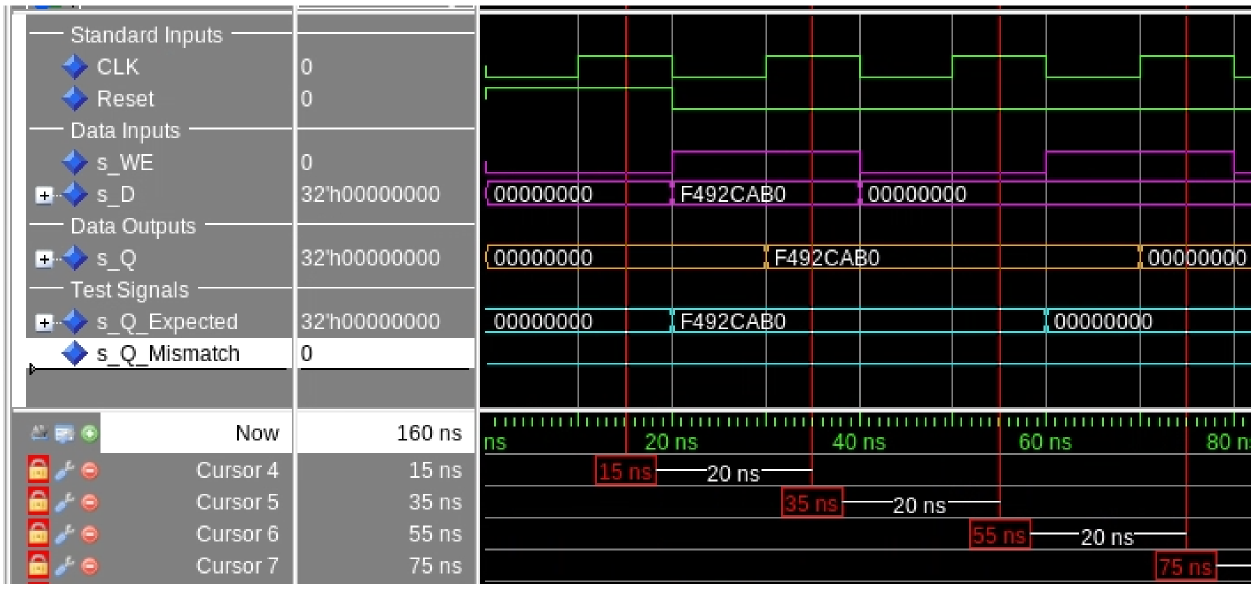


Figure 3. N-bit Register Waveform 1

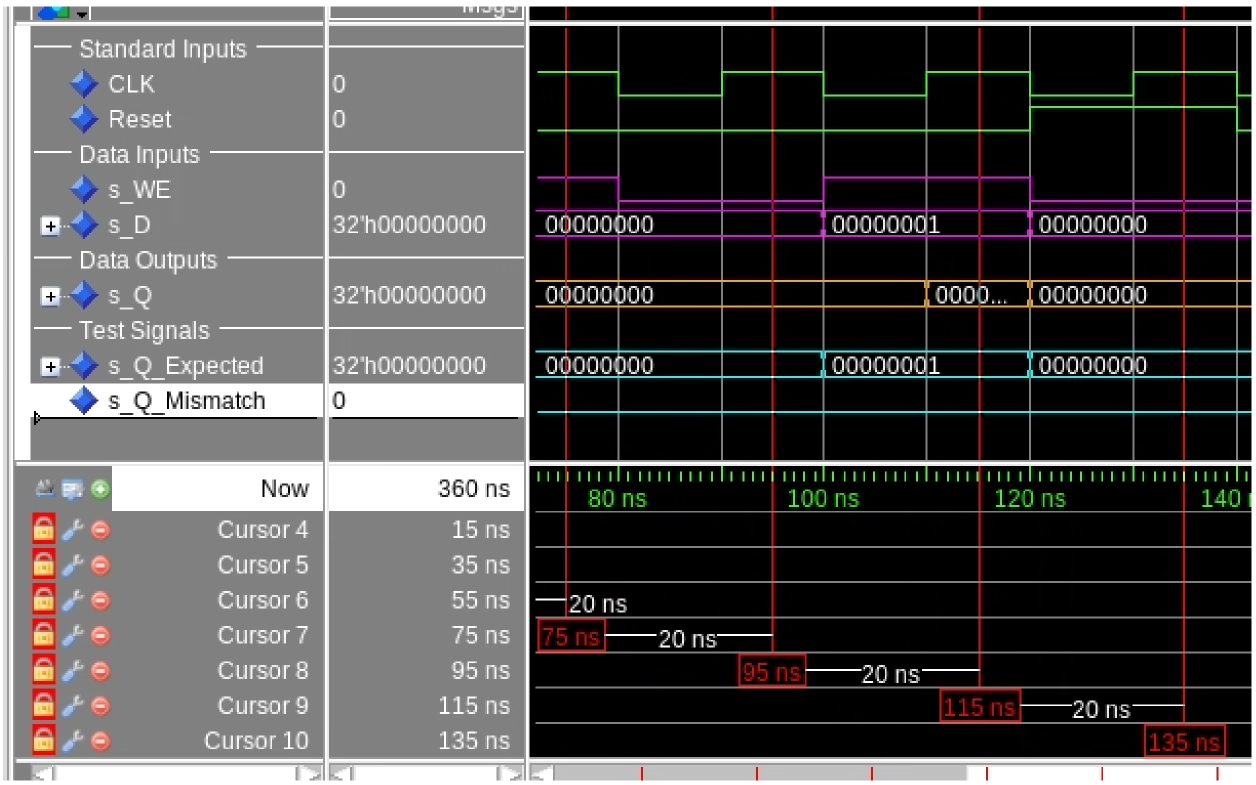


Figure 4. N-bit Register Waveform 1

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

The decoder required for a MIPS Register file would be a 5 to 32 decoder with an enable input. The decoder would need 5 bits to decode since the RS, RD, and RT portions of the ISA instructions are all 5 bits. These 5 bits can be decoded into a one-hot 32 bit value, defining which of the 32 registers should be selected with just 5 bits for each piece of the instruction. To make the write enable easier for each 32 bit register, I also decided to include the input i_WE to act as a write enable, which will output a one-hot decoded output based on the select line i_A, or 0 if i_WE is set to 0, since we may not want to always write to a register every clock cycle.

[Part 2 (e)] Waveform.

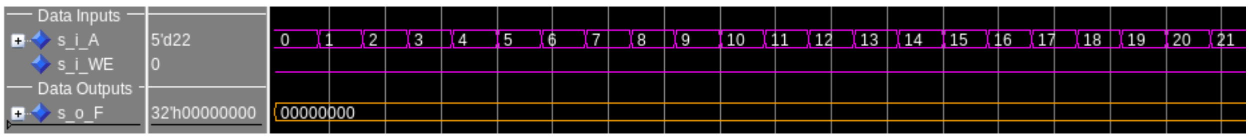


Figure 5. 5 to 32 Decoder Waveform 1 (i_WE = 0)

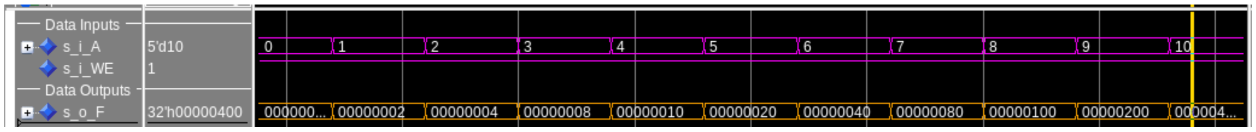


Figure 6. 5 to 32 Decoder Waveform 2 (i_WE = 1)

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

For the 32 bit 32-1 MUX, I will be creating an array type that is an array of 32 bit standard logic vectors. This would make it so I do not need to instantiate 32 1 bit 32-1 multiplexers, and can instead route the output of the top MUX module based on the index of the 32 wide array of logic vectors.

[Part 2 (g)]32 bit 32-1 MUX Waveform.

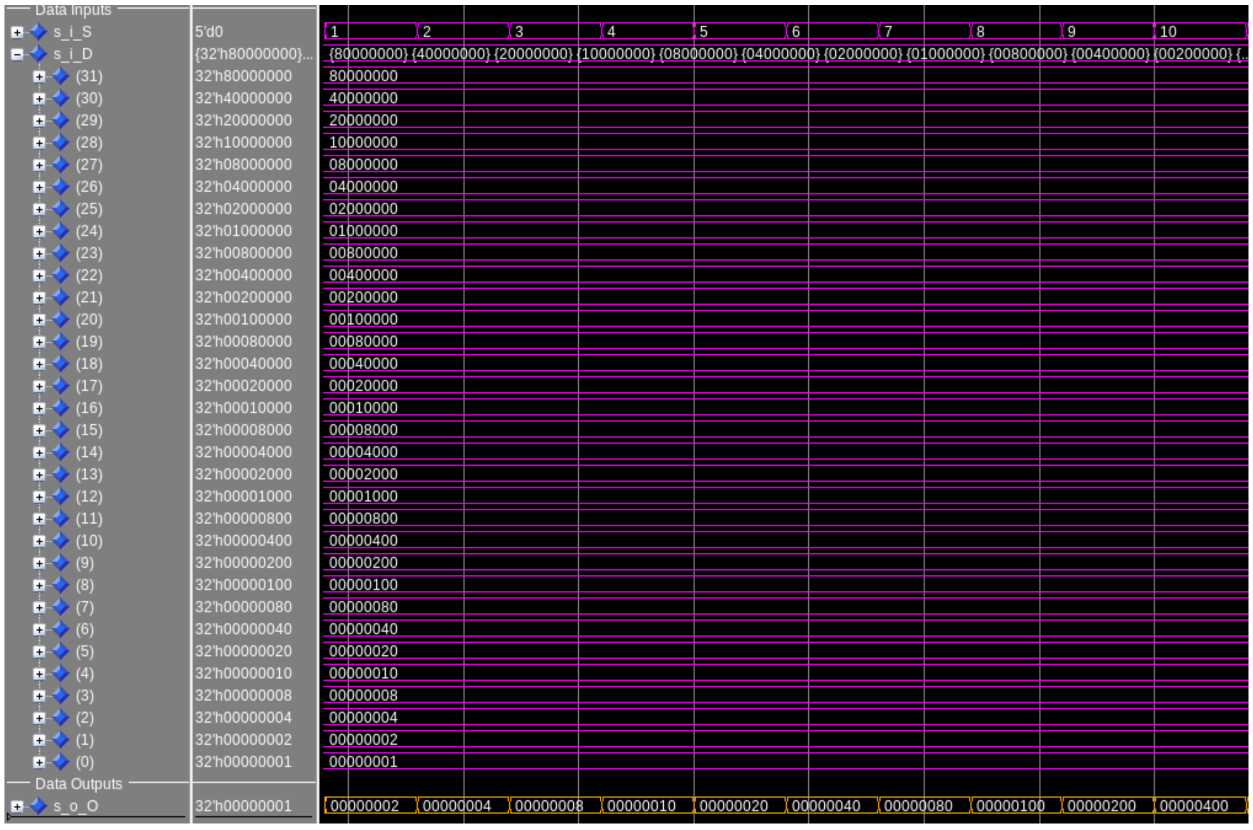

Data Inputs
s_i_S
5'd0
s_i_D
{32'h80000000}...
(31) 32'h80000000 80000000
(30) 32'h40000000 40000000
(29) 32'h20000000 20000000
(28) 32'h10000000 10000000
(27) 32'h08000000 08000000
(26) 32'h04000000 04000000
(25) 32'h02000000 02000000
(24) 32'h01000000 01000000
(23) 32'h00800000 00800000
(22) 32'h00400000 00400000
(21) 32'h00200000 00200000
(20) 32'h00100000 00100000
(19) 32'h00080000 00080000
(18) 32'h00040000 00040000
(17) 32'h00020000 00020000
(16) 32'h00010000 00010000
(15) 32'h00008000 00008000
(14) 32'h00004000 00004000
(13) 32'h00002000 00002000
(12) 32'h00001000 00001000
(11) 32'h00000800 00000800
(10) 32'h00000400 00000400
(9) 32'h00000200 00000200
(8) 32'h00000100 00000100
(7) 32'h00000080 00000080
(6) 32'h00000040 00000040
(5) 32'h00000020 00000020
(4) 32'h00000010 00000010
(3) 32'h00000008 00000008
(2) 32'h00000004 00000004
(1) 32'h00000002 00000002
(0) 32'h00000001 00000001
Data Outputs
s_o_O
32'h00000001
1 2 3 4 5 6 7 8 9 10
{80000000} {40000000} {20000000} {10000000} {08000000} {04000000} {02000000} {01000000} {00800000} {00400000} {00200000} {...
00000002 00000004 00000008 00000010 00000020 00000040 00000080 00000100 00000200 00000400

Figure 8. 32 bit 32-1 MUX Waveform

[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.

For my design, I will use 1 5-32 decoder, 32 32-bit registers, and 2 32-bit 32-1 Multiplexers. The schematic drawing for this design can be seen below in **Figure 9.** I planned to decode the 5 bit destination register input i_RD and the write enable input i_RD_WE through the decoder module. If the write enable input is 0, the decoder will output all 0's. If the write-enable input i_RD_WE is 1, the decoder will output a one-hot 32 bit output based on the value of i_RD.

The 32 bit output will then be split between the 32 32-bit Registers, with bit 0 aligning to the enable pin of register 0. The input clock i_CLK and input reset i_RST will each be routed to every 32-bit register to update the D Flip Flops on a positive clock edge and asynchronous reset. The input i_RD_Data from the top register file will be routed to each i_WR_Data input of the register to write the destination register.

Two 32 bit 32-1 multiplexers will be used to multiplex the correct operand outputs from the registers. The select lines will be the inputs i_RS and i_RT, each 5 bits wide. The outputs to theses multiplexers could be used for arithmetic after and then the output could come back through with the i_RD_Data after the arithmetic was performed.

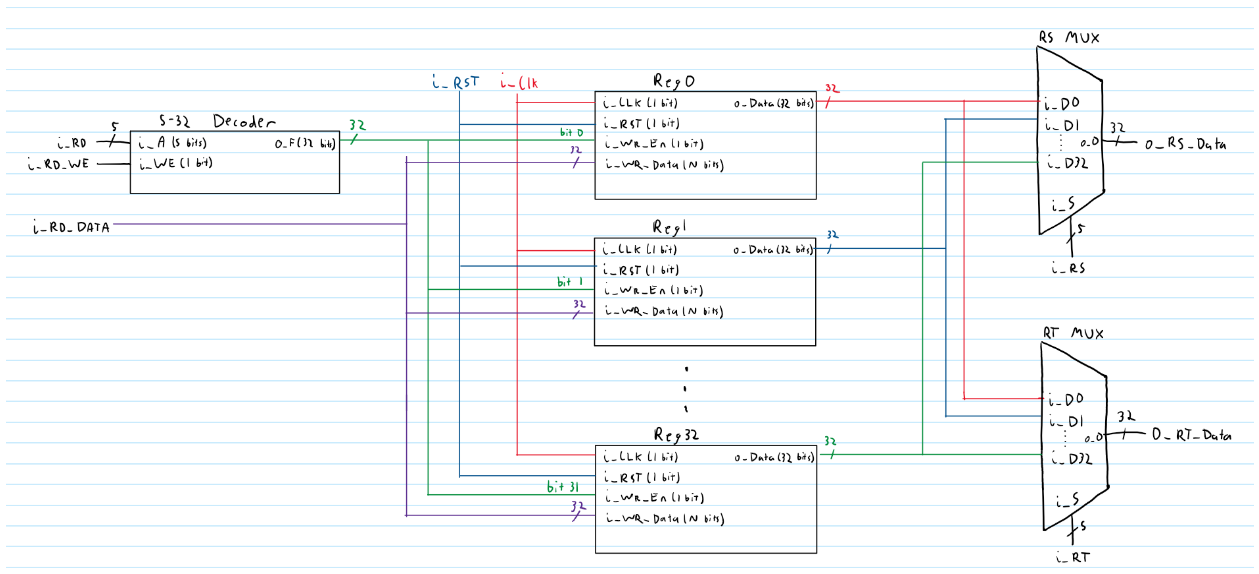


Figure 9. MIPS Register File Block Diagram

[Part 2 (i)] Waveform.

Test cases:

1. Set reset to active and verify read registers are 0
2. Ensure that register 0 can not be written to (always stays as 0)
3. Successfully write to a register
4. Read 32 bit written values to designed RS and RT registers
5. Read different data values after setting two different registers
6. Attempt a reset after two registers filled and confirm set to 0

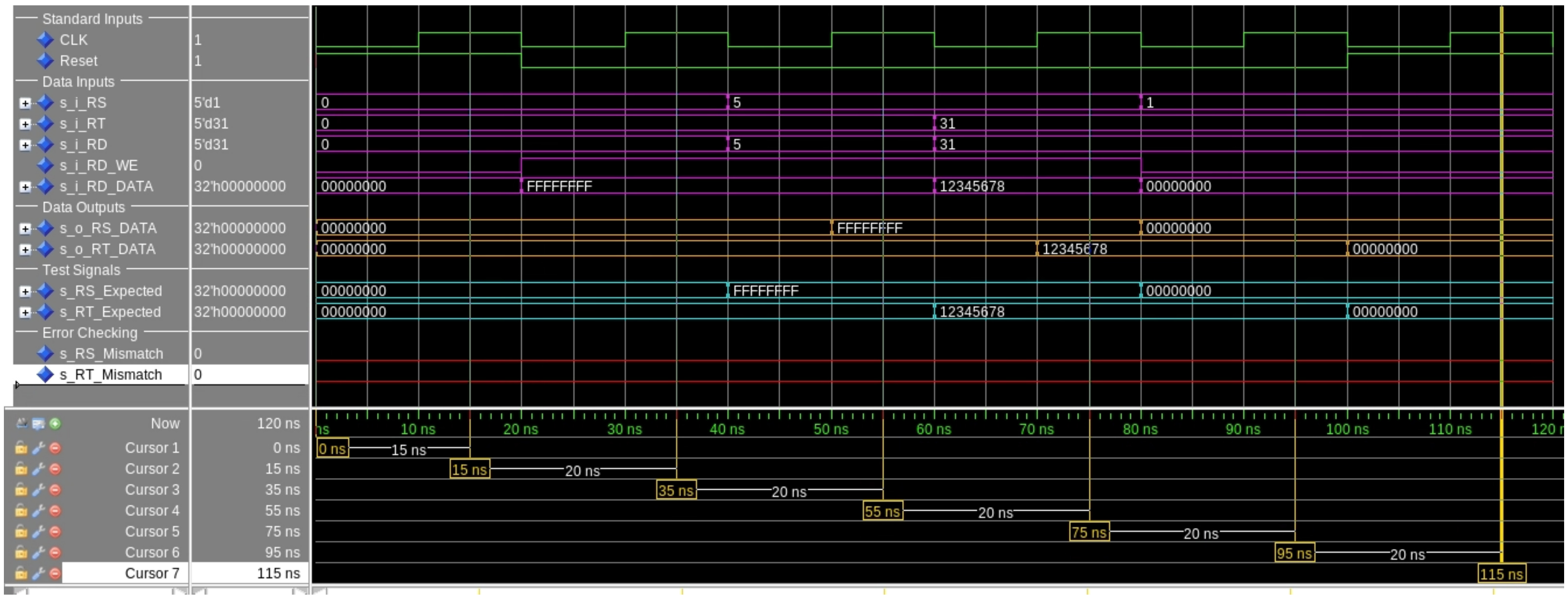


Figure 10. MIPS Register File Waveform

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

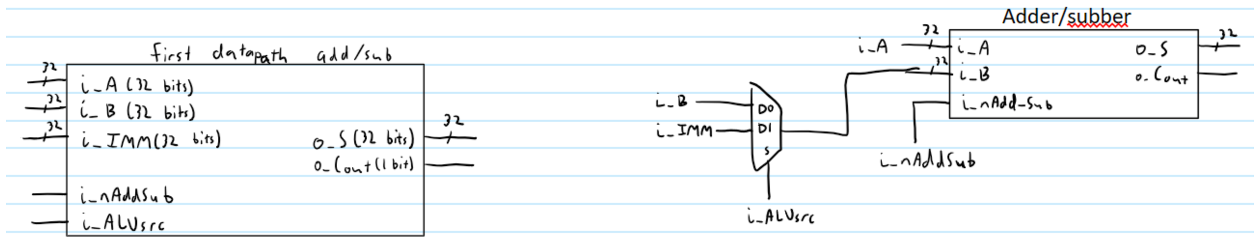


Figure 11. First MIPS Datapath Arithmetic Block

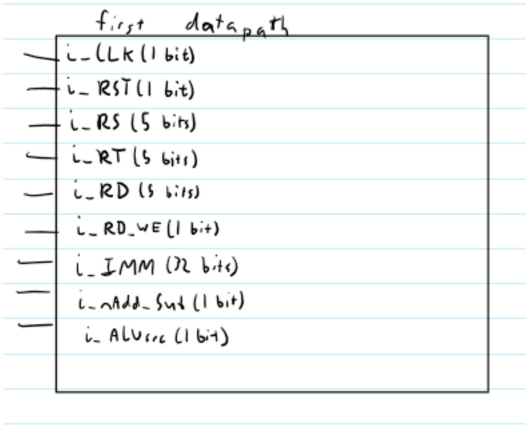


Figure 12. First MIPS Datapath Block

[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).

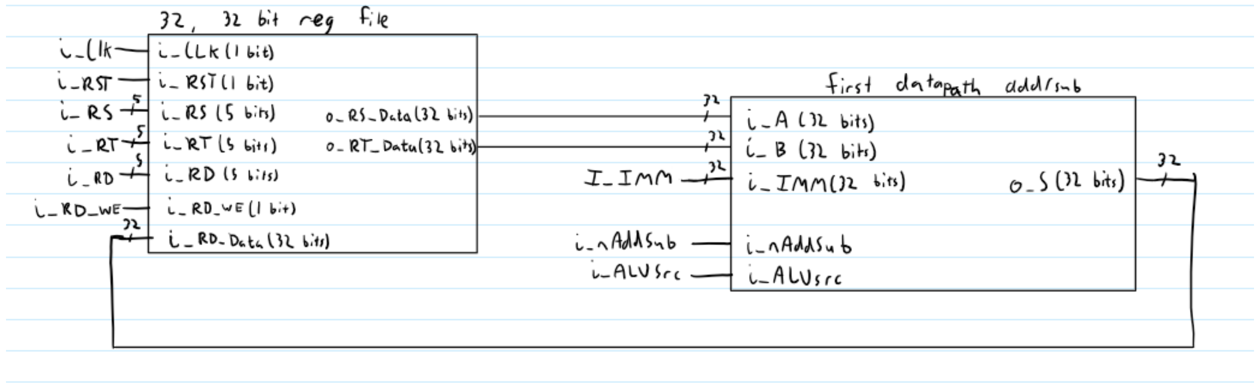


Figure 13. MIPS-like Datapath Circuit

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

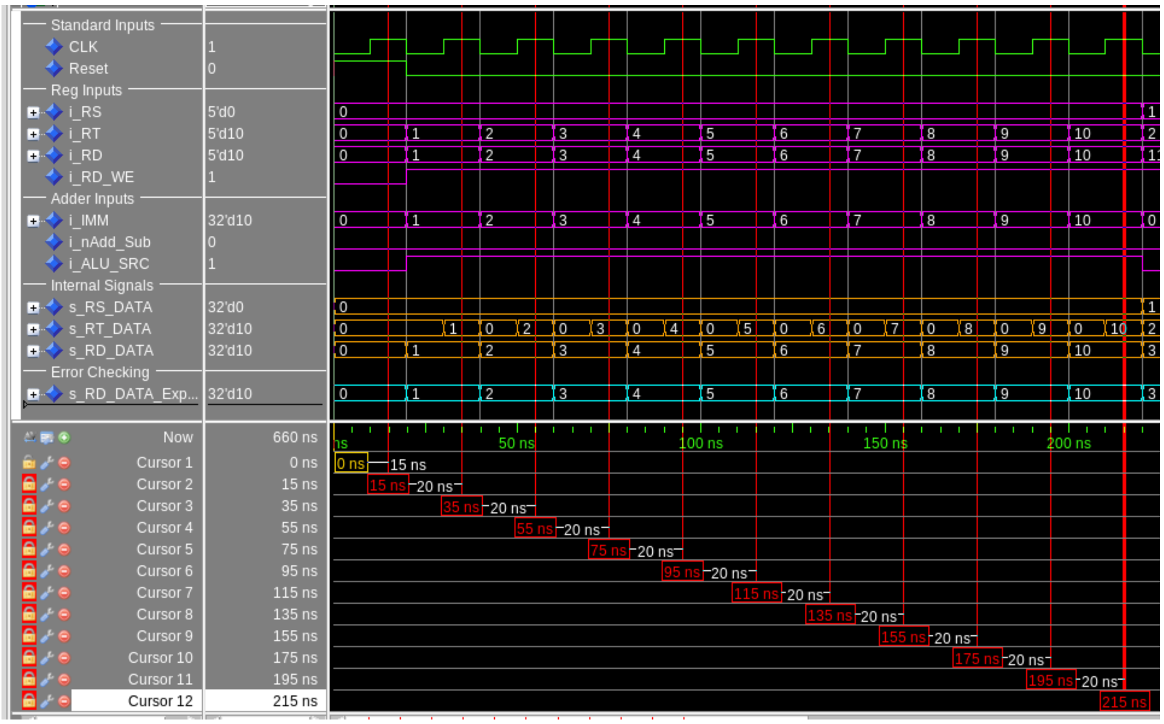


Figure 14. First MIPS Datapath Waveform 1

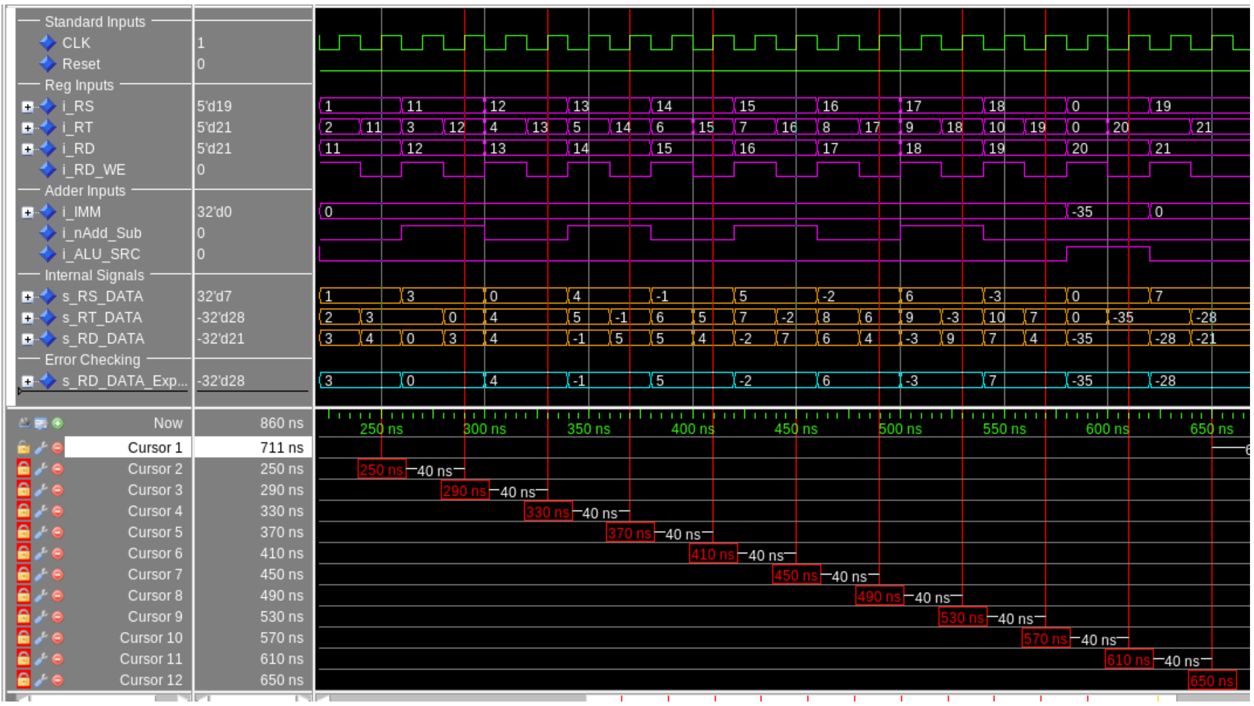


Figure 15. First MIPS Datapath Waveform 2

Register	Value	Register	Value
0	0	16	-2
1	1	17	6
2	2	18	-3
3	3	19	7
4	4	20	-35
5	5	21	-28
6	6	22	0
7	7	23	0
8	8	24	0
9	9	25	0
10	10	26	0
11	3	27	0
12	0	28	0
13	4	29	0
14	-1	30	0
15	5	31	0

Table 1. First Datapath Final Register State

[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 number of bits stored at each address. The memory file is word addressable, so this should stay at 32.

ADDR_WIDTH: The number of bits used to specify which address in memory should be accessed/written over. The address range goes from 0 to 2^(ADDR_WIDTH - 1)

clk: Clock to control when data can be written to memory. Positive edge triggered

addr: addr is used as an index to access a word in the ram signal, which is of type memory_t. Addr is (ADDR_WIDTH - 1) long.

data: The data that can be written into a word of memory. Data is DATA_WIDTH bits long.

we: This is the write enable signal that is 1 bit. On the rising edge of a clock, the data will get written to an address if we is 1. Otherwise, nothing will be written.

q: bit vector output that is DATA_WIDTH bits wide. The output q is selected based on the address input addr, which is used as an index for the ram signal to output a standard logic vector.

[Part 4 (c)] Waveforms.

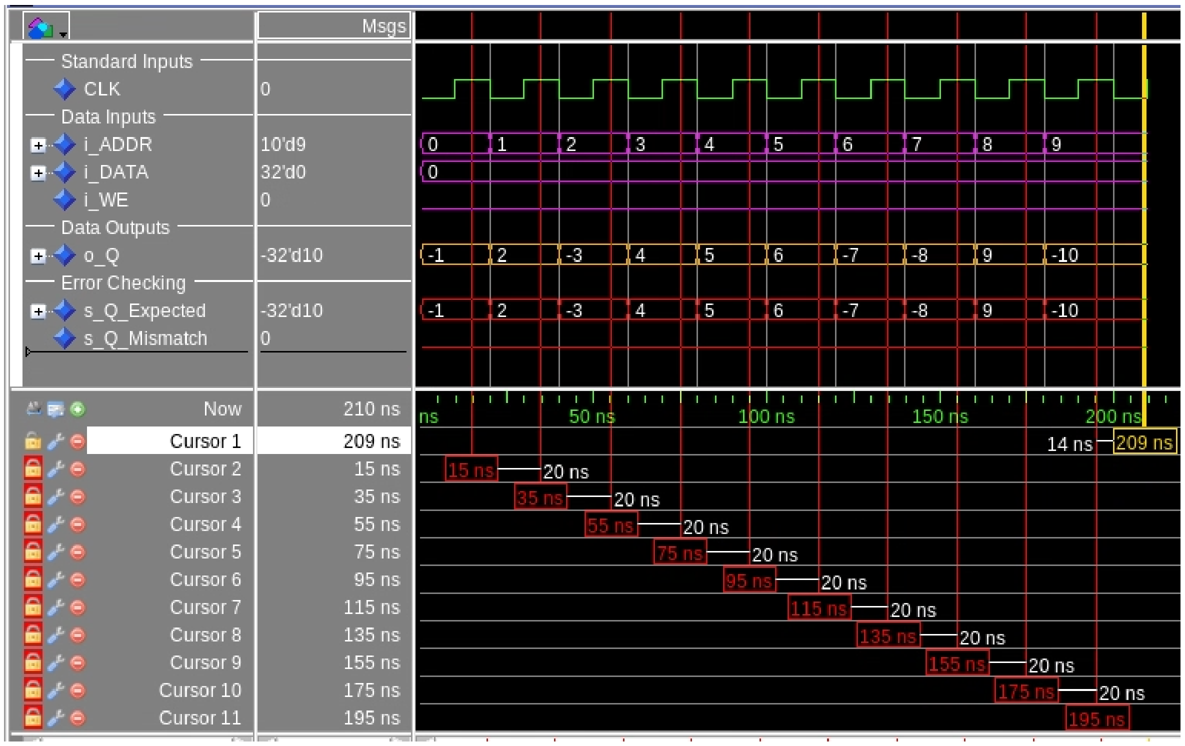


Figure 16. tb_dmem Waveform, Test Step ii

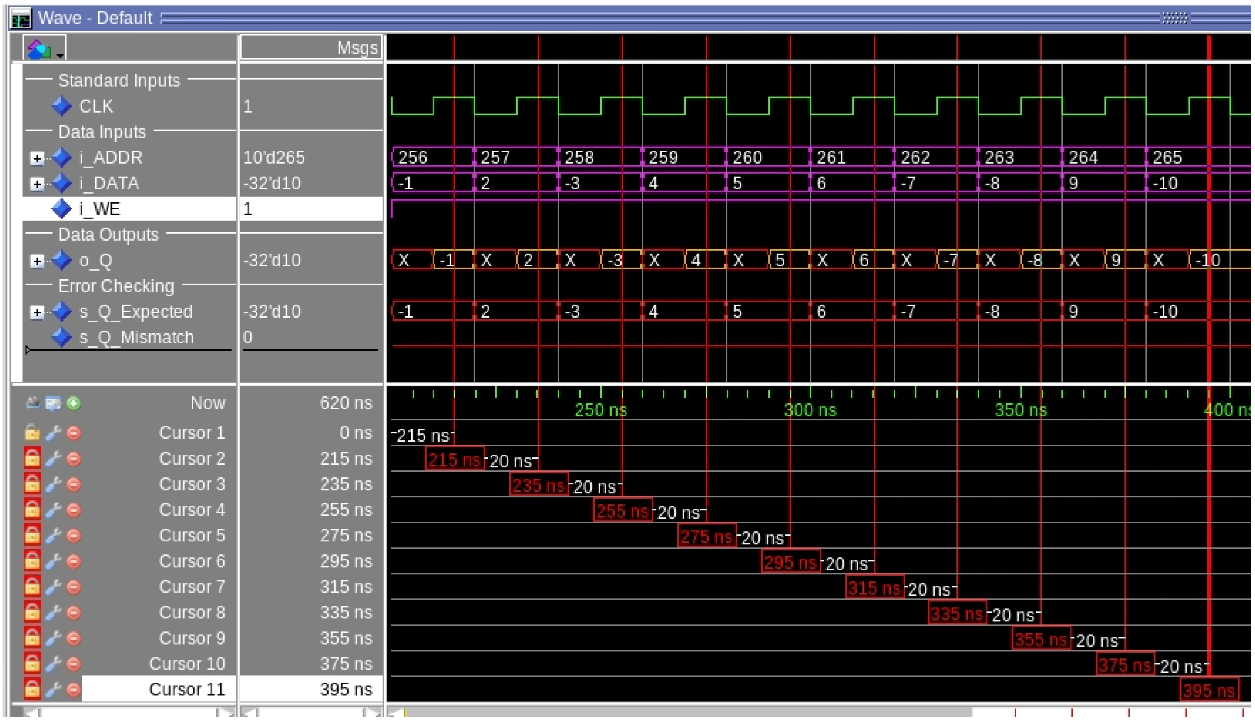


Figure 17. tb_dmem Waveform, Test Step iii

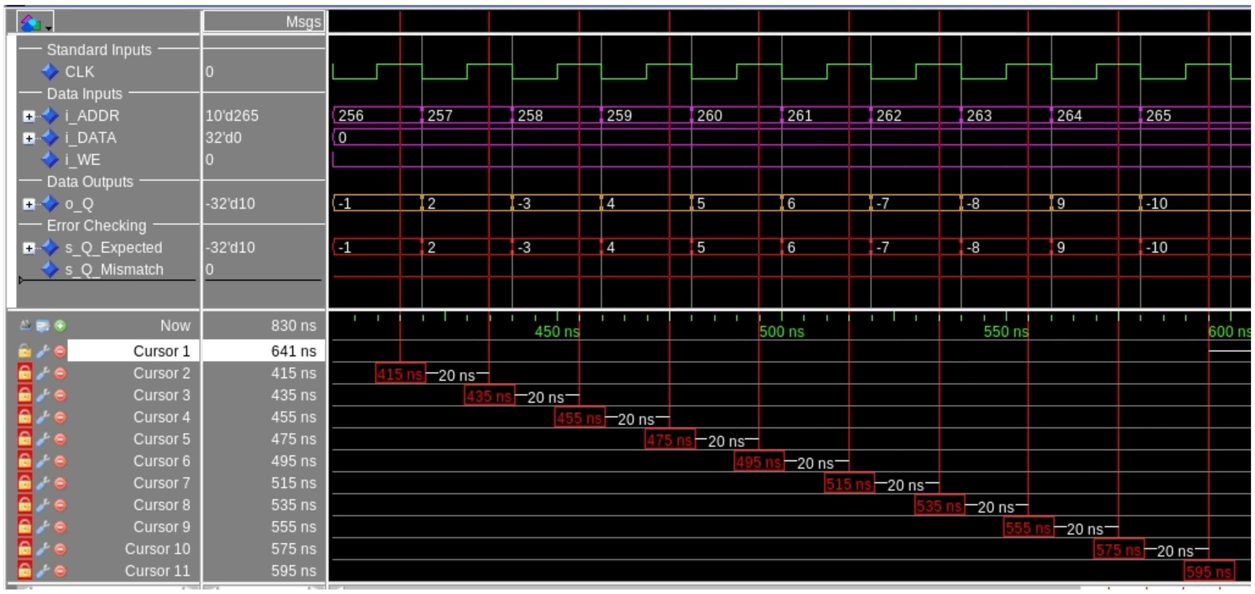


Figure 18. tb_dmem Waveform, Test Step iv

[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?

Instructions using an immediate operand will require a sign extended value. This is because an immediate value is 16 bits of the I-type instruction format. Some instructions using an immediate with this extension would be add immediate, add immediate unsigned, load word, and store word.

Some instructions that require a value to be zero extended include and immediate and or immediate.

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

An extender would be needed in an ALU to extend a signed 16 bit immediate value into a 32 bit signed value for arithmetic operations. A zero bit extender would also be needed for logic functions such as a bitwise immediate and/or operation. A signed extender would also be needed between the memory module of the processor to sign extend address bits.

[Part 5 (d)] Waveform.

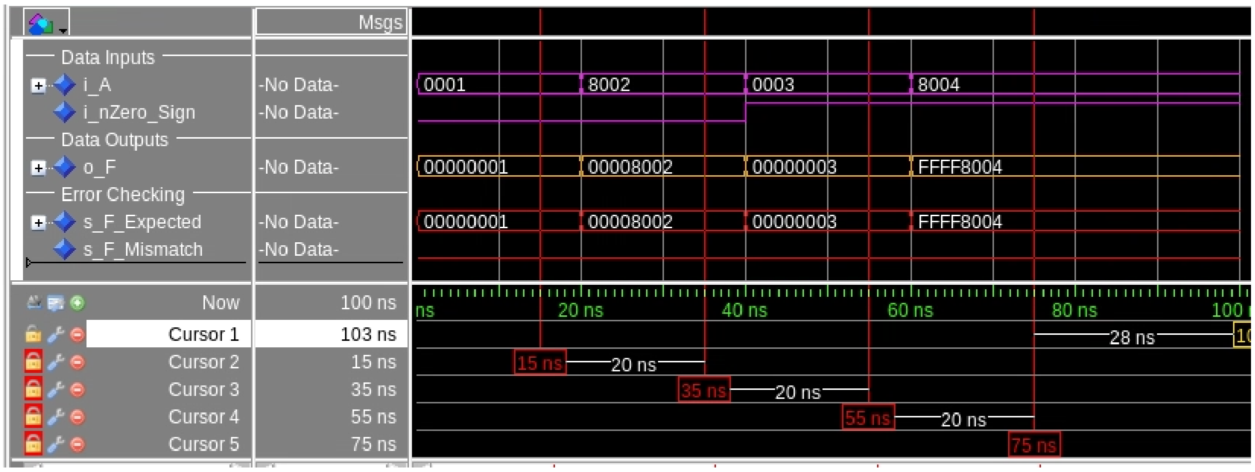


Figure 19. tb_extend_16t32 Waveform

[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 plan to add one control signal called i_SW to handle when data will be written to the memory module mem.vhd. The i_SW control signal will correspond to the write enable control on the memory module. When i_SW is active, the memory module will write the designated register value to memory at the memory's address. When i_SW is 0, no write to memory will occur.

I also plan to add another control signal, i_nADDER_MEM, which when 0 will supply the destination register in my register file with the 32 bit output from my adder/subtractor unit. When i_nADDER_MEM is 1, it will instead output the destination register data with the output from the memory module that is implemented for this second datapath.

[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.

One of the main takeaways from this design is that the immediate value needs to be extended from 16 bits to 32 bits with the sign extender module. Since every I type instruction in this datapath uses a signed 32 bit value, and not a 0 extended value, I set i_nZero_Sign to 1. This signed value can be used as an immediate for arithmetic outputs, or to be summed to find a specific address for the lw and sw instructions.

The memory module introduced the control signal i_SW mentioned above, and the data to be written on the i_SW instruction is given from the second readable register RT. As mentioned before, the address is added between the immediate and the contents inside the first readable register RS. The output will be given from the same specified address, and can be loaded back into the register file when the control bit i_nADDER_MEM is set to 1. Otherwise, this 32 bit 2-1 MUX will route the arithmetic output from the adder/subtractor when arithmetic instructions are performed, when i_nADDER_MEM is set to 0.

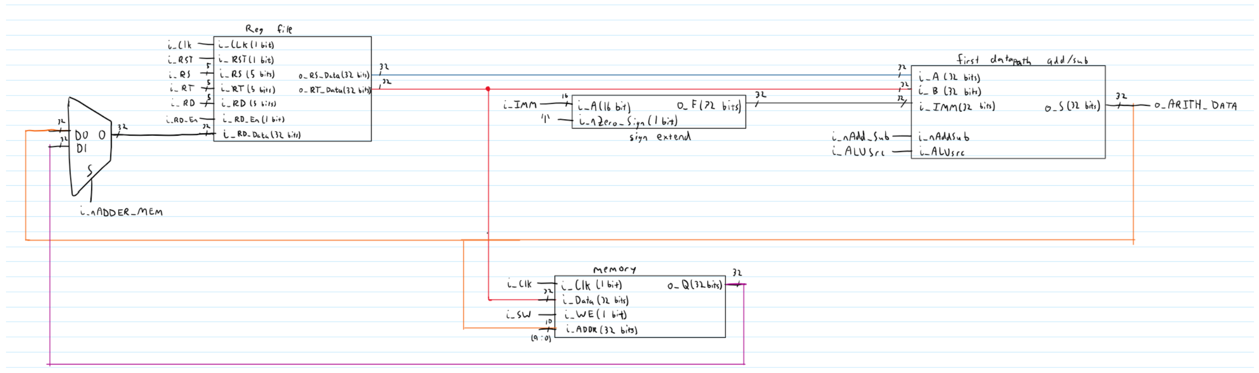


Figure 20. Second Datapath Schematic

[Part 6 (c)] Waveform.

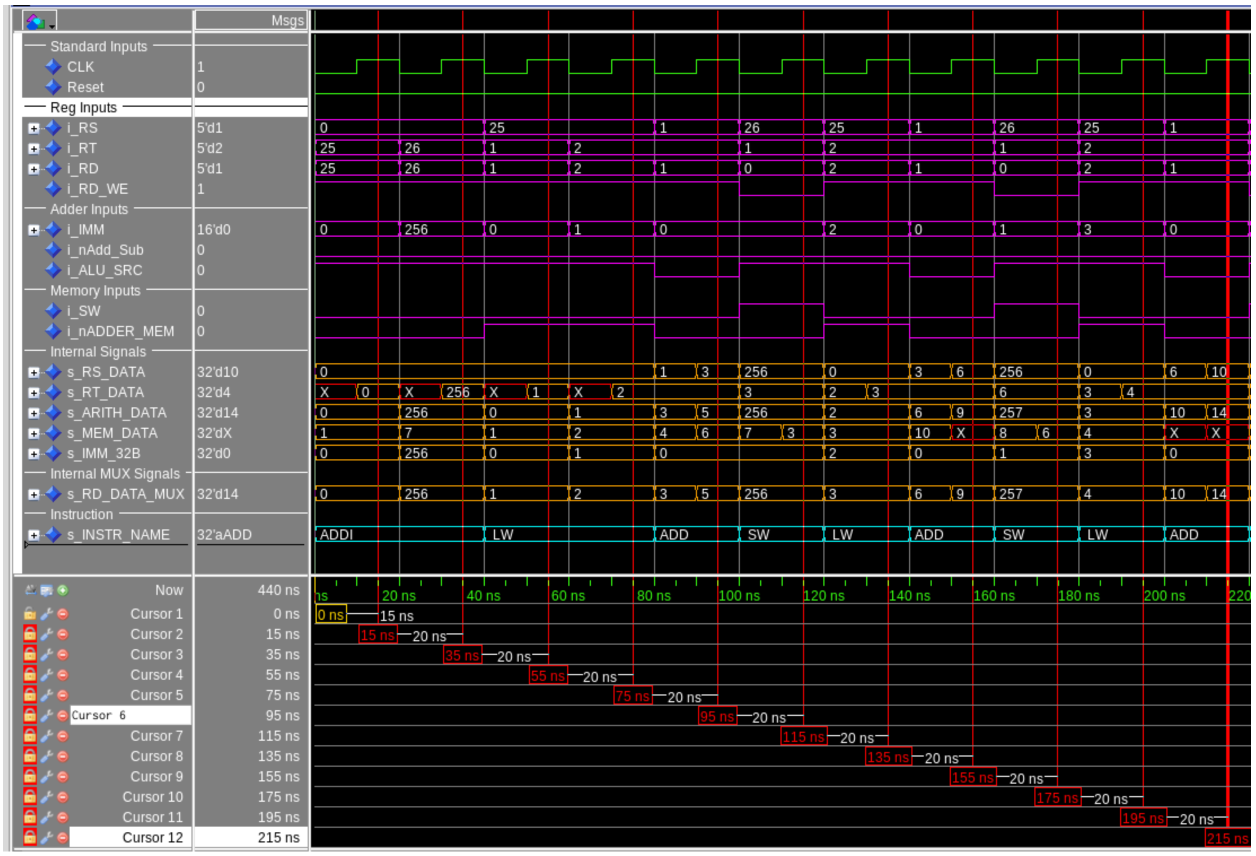
To check each instruction, proceed as follows:

- **ADDI**: Verify RS reg and immediate written correctly by viewing s_RT_DATA
- **ADD**: Verify written to RS destination reg by viewing s_RS_DATA
- **LW**: Verify word loaded into correct register by verifying s_MEMORY_DATA loaded to register RT, which is made same as destination register to view change.
- **SW**: Verify s_MEM_DATA changes after writing to specified address, given by a value in s_RT_DATA and the RT register address

Second Datapath Waveform 1 Expected Results							
Instruction	**$t1**	**$t2**	**$t25**	**$t26**	**$27**	**A**	**B**
addi $25, $0, 0	X	X	0	X	X	{1, 2, 3, 4, 5, 6, 10}	{7, 8, 9, 10, 11, … , 12, 13}
addi $26, $0, 256	X	X	0	256	X	{1, 2, 3, 4, 5, 6, 10}	{7, 8, 9, 10, 11, … , 12, 13}
lw $1, 0($25)	1	X	0	256	X	{1, 2, 3, 4, 5, 6, 10}	{7, 8, 9, 10, 11, … , 12, 13}
lw $1, 4($25)	1	2	0	256	X	{1, 2, 3, 4, 5, 6, 10}	{7, 8, 9, 10, 11, … , 12, 13}
add $1, $1, $2	3	2	0	256	X	{1, 2, 3, 4, 5, 6, 10}	{7, 8, 9, 10, 11, … , 12, 13}
sw $1, 0($26)	3	2	0	256	X	{1, 2, 3, 4, 5, 6, 10}	{3, 8, 9, 10, 11, … , 12, 13}
lw $2, 8($25)	3	3	0	256	X	{1, 2, 3, 4, 5, 6, 10}	{3, 8, 9, 10, 11, … , 12, 13}
add $1, $1, $2	6	3	0	256	X	{1, 2, 3, 4, 5, 6, 10}	{3, 8, 9, 10, 11, … , 12, 13}
sw $1, 4($26)	6	3	0	256	X	{1, 2, 3, 4, 5, 6, 10}	{3, 6, 9, 10, 11, … , 12, 13}
lw $2, 12($25)	6	4	0	256	X	{1, 2, 3, 4, 5, 6, 10}	{3, 6, 9, 10, 11, … , 12, 13}
add $1, $1, $2	10	4	0	256	X	{1, 2, 3, 4, 5, 6, 10}	{3, 6, 9, 10, 11, … , 12, 13}

Table 2. Second Datapath Waveform 1 Expected Outputs (Based on test instructions)

The waveform below in **Figure 21** is the first half of instructions, correlating to the instructions above for the second datapath test. In **Table 2** above, I mapped out the expected values for the changed temporary registers and the arrays A and B in memory.



	Msgs
Standard Inputs	
CLK	1
Reset	0
Reg Inputs	
i_RS	5'd1
i_RT	5'd2
i_RD	5'd1
i_RD_WE	1
Adder Inputs	
i_IMM	16'd0
i_nAdd_Sub	0
i_ALU_SRC	0
Memory Inputs	
i_SW	0
i_nADDER_MEM	0
Internal Signals	
s_RS_DATA	32'd10
s_RT_DATA	32'd4
s_ARITH_DATA	32'd14
s_MEM_DATA	32'dX
s_IMM_32B	32'd0
Internal MUX Signals	
s_RD_DATA_MUX	32'd14
Instruction	
s_INSTR_NAME	32'aADD

Now	440 ns
Cursor 1	0 ns
Cursor 2	15 ns
Cursor 3	35 ns
Cursor 4	55 ns
Cursor 5	75 ns
Cursor 6	95 ns
Cursor 7	115 ns
Cursor 8	135 ns
Cursor 9	155 ns
Cursor 10	175 ns
Cursor 11	195 ns
Cursor 12	215 ns

Figure 21. Second Datapath Waveform 1

Second Datapath Waveform 2 Expected Results							
Instruction	**$t1**	**$t2**	**$t25**	**$t26**	**$27**	**A**	**B**
sw $1, 8($26)	10	4	0	256	X	{1, 2, 3, 4, 5, 6, 10}	{3, 6, 10, 10, 11, … , 12, 13}
lw $2, 16($25)	10	5	0	256	X	{1, 2, 3, 4, 5, 6, 10}	{3, 6, 10, 10, 11, … , 12, 13}
add $1, $1, $2	15	5	0	256	X	{1, 2, 3, 4, 5, 6, 10}	{3, 6, 10, 10, 11, … , 12, 13}
sw $1, 12($26)	15	5	0	256	X	{1, 2, 3, 4, 5, 6, 10}	{3, 6, 10, 15, 11, … , 12, 13}
lw $2, 20($25)	15	6	0	256	X	{1, 2, 3, 4, 5, 6, 10}	{3, 6, 10, 15, 11, … , 12, 13}
add $1, $1, $2	21	6	0	256	X	{1, 2, 3, 4, 5, 6, 10}	{3, 6, 10, 15, 11, … , 12, 13}
sw $1, 16($26)	21	6	0	256	X	{1, 2, 3, 4, 5, 6, 10}	{3, 6, 10, 15, 21, … , 12, 13}
lw $2, 24($25)	21	10	0	256	X	{1, 2, 3, 4, 5, 6, 10}	{3, 6, 10, 15, 21, … , 12, 13}
add $1, $1, $2	31	10	0	256	X	{1, 2, 3, 4, 5, 6, 10}	{3, 6, 10, 15, 21, … , 12, 13}
Addi $27, $0, 512	31	10	0	256	512	{1, 2, 3, 4, 5, 6, 10}	{3, 6, 10, 15, 21, … , 12, 13}
sw $1, -4($27)	31	10	0	256	512	{1, 2, 3, 4, 5, 6, 10}	{3, 6, 10, 15, 21, … , 31, 13}

Table 3. Second Datapath Waveform 2 Expected Outputs (Based on test instructions)

The table above, **Table 3**, and the waveform below, **Figure 22**, correspond to the second half of the second datapath instructions given in the last section of that lab report.

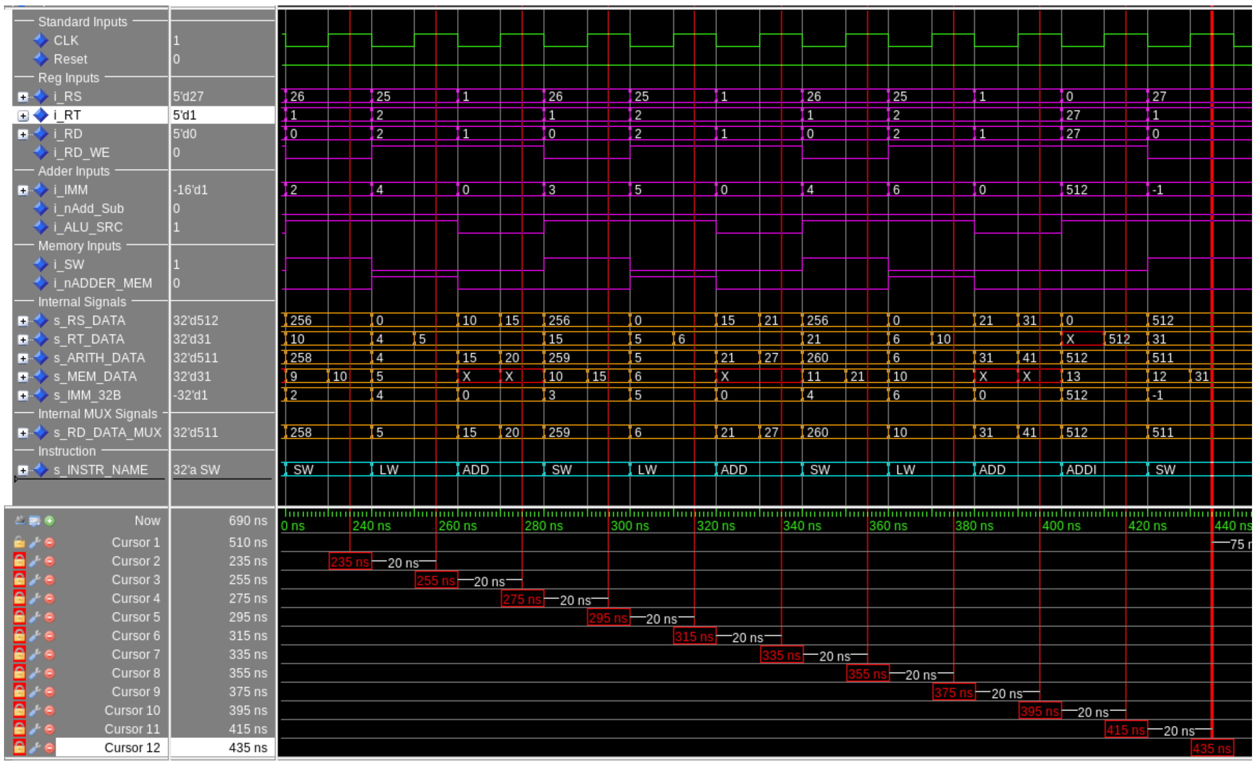
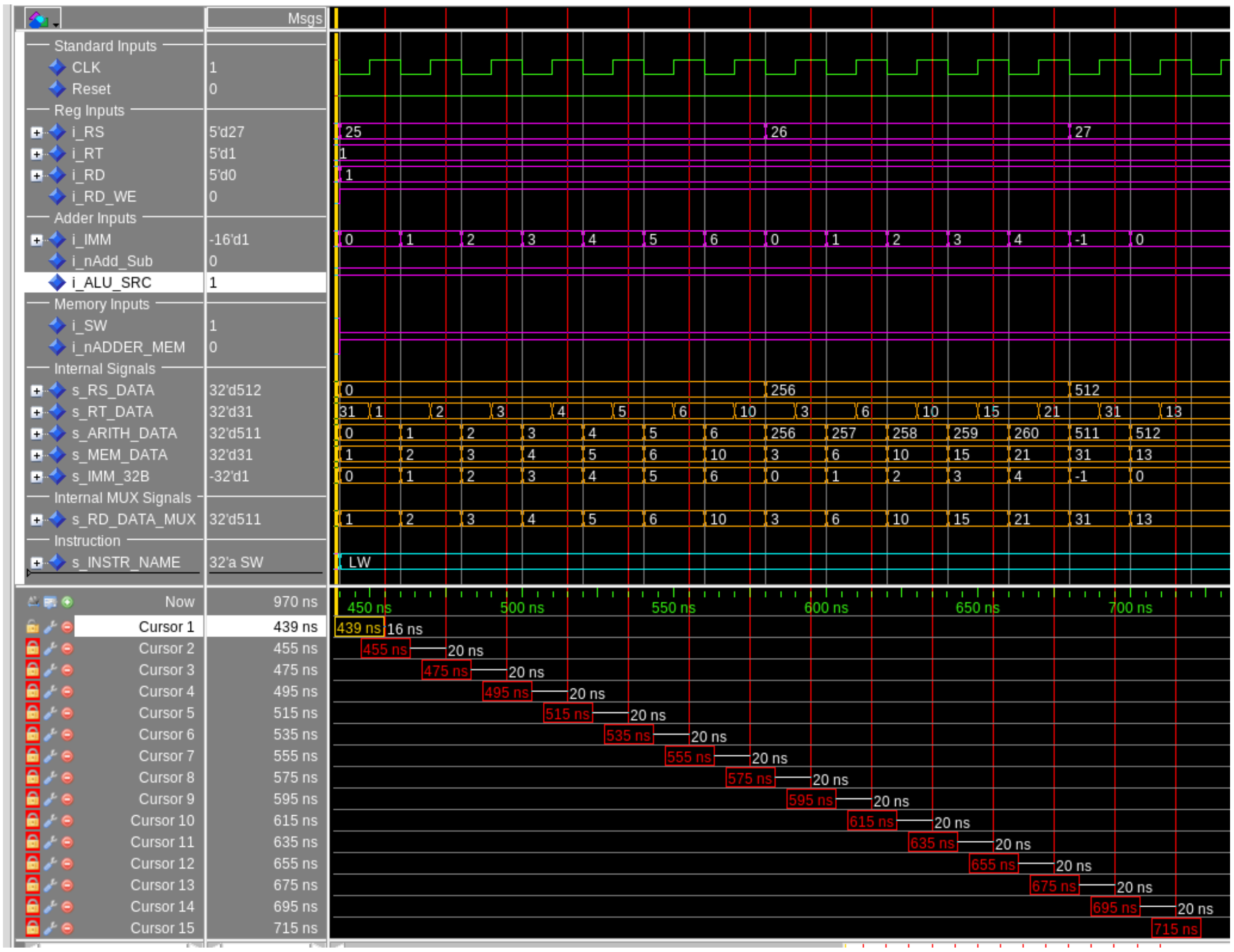


Figure 22. Second Datapath Waveform 2

Index	Array A	Array B
0	1	3
1	2	6
2	3	10
3	4	15
4	5	21
5	6	X
6	10	X
255	X	31
256	X	13

Table 4. Second Datapath Waveform 3 Expected Outputs (Based on test instructions)

The final screenshot given below, in Figure 23, shows the array's A and B as updated in the above instructions. I first indexed through array A, indices 0 to 6, to verify that they did not change based on my initial values loaded into the memory file. I then verified that all array indices updated in array B were changed as expected. The following final results were expected, like mentioned above.



```
                     Msgs
Standard Inputs
  CLK                1
  Reset              0
Reg Inputs
  i_RS               5'd27
  i_RT               5'd1
  i_RD               5'd0
  i_RD_WE            0
Adder Inputs
  i_IMM              -16'd1
  i_nAdd_Sub         0
  i_ALU_SRC          1
Memory Inputs
  i_SW               1
  i_nADDER_MEM       0
Internal Signals
  s_RS_DATA          32'd512
  s_RT_DATA          32'd31
  s_ARITH_DATA       32'd511
  s_MEM_DATA         32'd31
  s_IMM_32B          -32'd1
Internal MUX Signals
  s_RD_DATA_MUX      32'd511
Instruction
  s_INSTR_NAME       32'a SW

Now                  970 ns
Cursor 1             439 ns
Cursor 2             455 ns
Cursor 3             475 ns
Cursor 4             495 ns
Cursor 5             515 ns
Cursor 6             535 ns
Cursor 7             555 ns
Cursor 8             575 ns
Cursor 9             595 ns
Cursor 10            615 ns
Cursor 11            635 ns
Cursor 12            655 ns
Cursor 13            675 ns
Cursor 14            695 ns
Cursor 15            715 ns
```

Figure 23. Second Datapath Waveform 3