CprE 381 – Computer Organization and Assembly-Level Programming

Proj-C Report

Lab Partners _____Matthew G. Hoskins_____

_____Cameron Isbell_____

_____

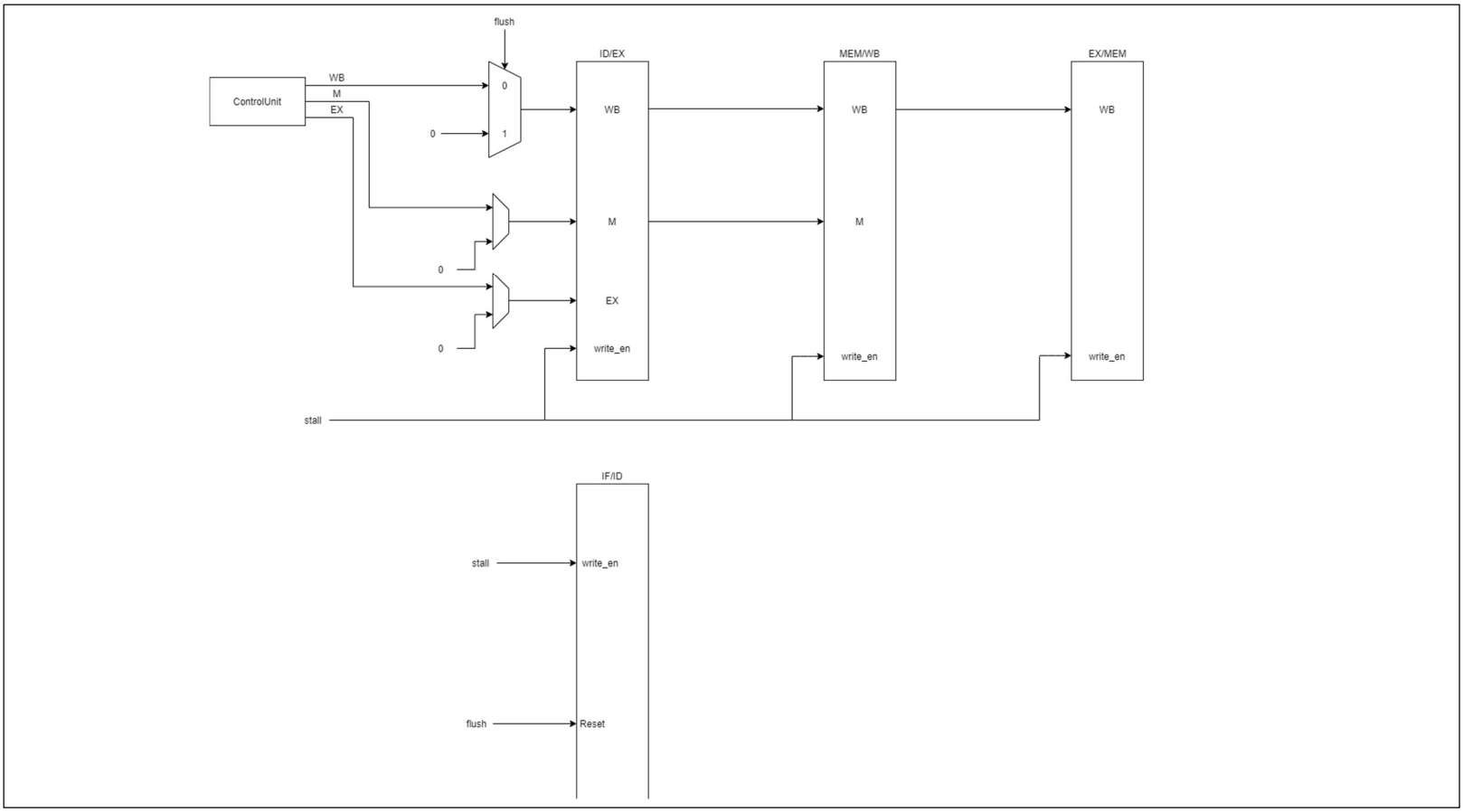
Section / Lab Time __ Section 6 / Th: 6:10pm – 8:00pm ___

Submit a typeset pdf version of this on Canvas by the due date. Refer to the highlighted language in the Proj-C instructions for the context of the following questions.

a. [Part 0] Come up with a global list of the datapath values and control signals that are required during each pipeline stage.

   All of the control signals that were used in the Single Cycle CPU will be required during each pipeline stage. To refrain from listing them all we will instead recognize that we will need to be able track all of the control signals that we have coming from our control unit.
   Other datapath values that will be needed for control purposes is four more signals for the state registers control values in order to correctly operate the pipeline stage.

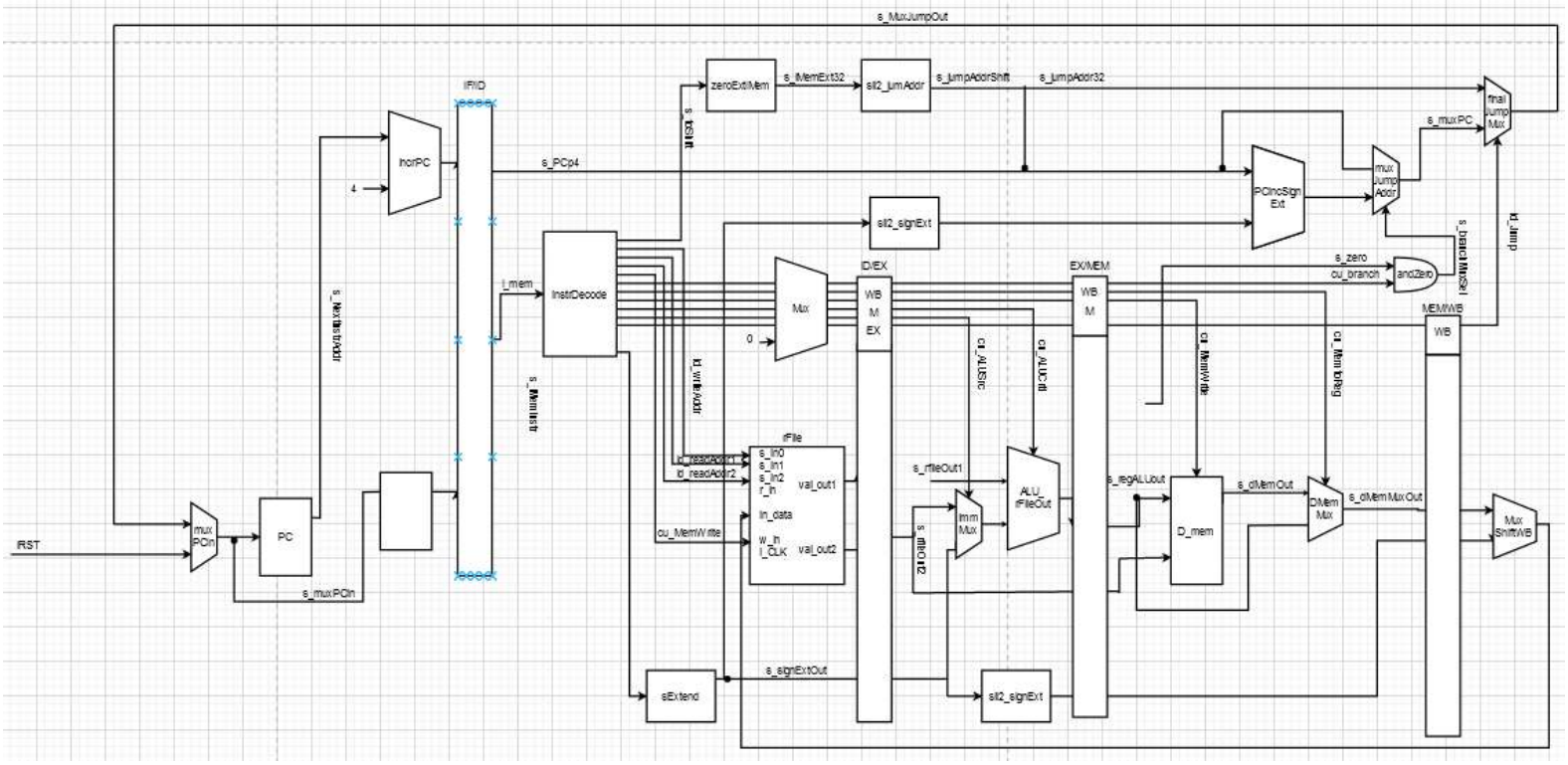
b. [Part 1 (a)] Draw a simple schematic showing how you could implement stalling and flushing operations given an ideal N-bit register.



c. [Part 1 (b)] Show that values that are stored in the initial IF/ID register are available as expected four cycles later, and that new values can be inserted into the pipeline every single cycle. *[Please include waveforms and explanations.]*

This is our bad, but we redesigned the registers and placed them into the processor at this point and unfortunately have not left us with time to go back and undue it, sorry. Are images of the testbench have been gone for a while. We did test it out though before where it was able to pass the values with every clock cycle taking in a new set at every cycle though.

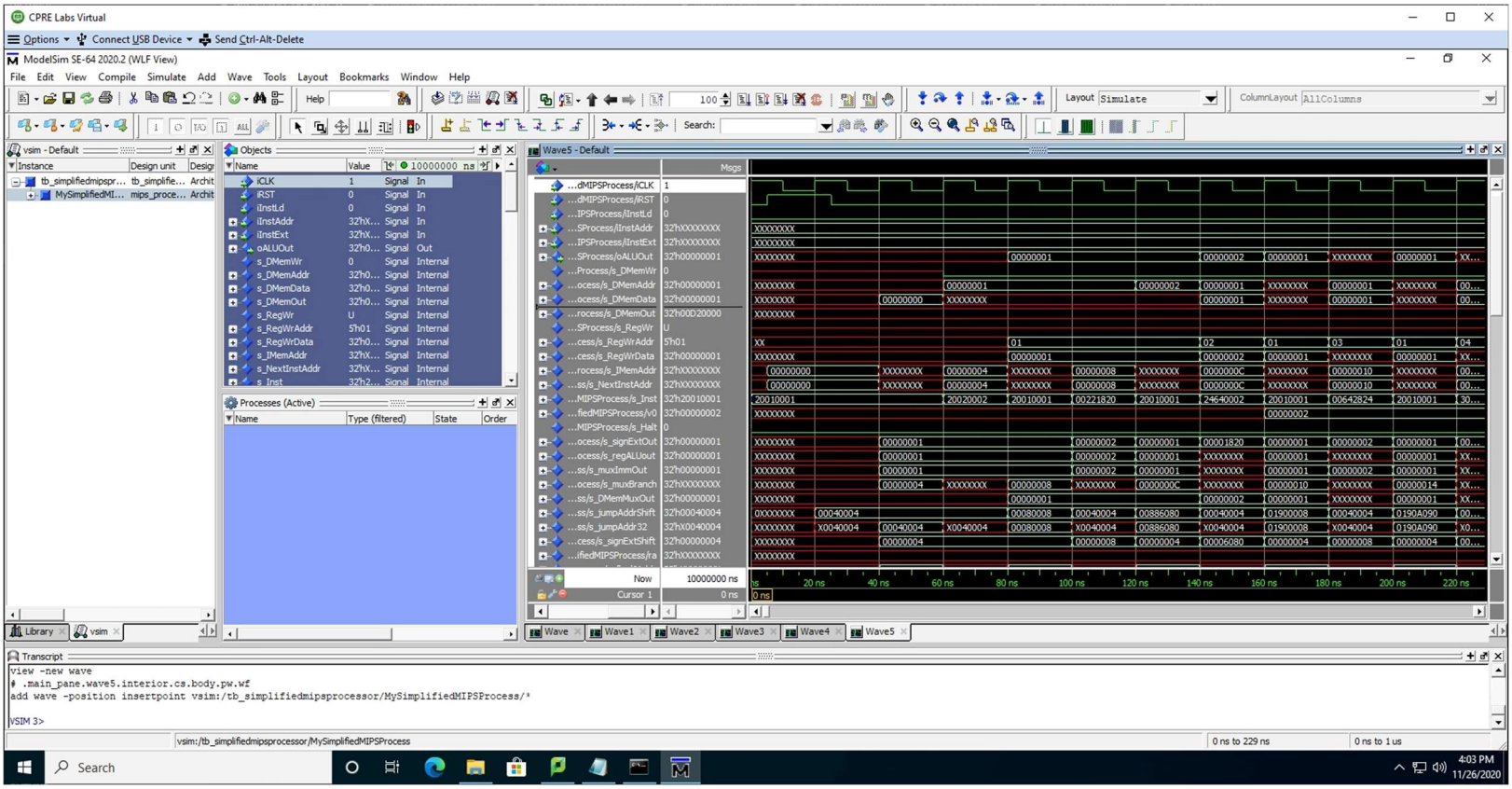
d. [Part 2] In your writeup, provide your schematic for this part, describe what challenges (if any) you faced in implementing this module.



While it is definitely impossible to accurately state that we did not incur any challenges whilst implementing this module for our processor; we do believe that of the challenges that we happened to encounter in this phase of design were relatively standard for that of any early implementation stage of a new aspect of the processor. One such example, would be that of having to ensure that what we added would correctly continue through the processor and remain consistent as the new additions would have to have connections to a variety of different components and files that were already developed and working.

e. [Part 3 (a)] In your writeup, show the ModelSim output for the individual instruction tests, and provide a discussion of result correctness. It may be helpful to also annotate the waveforms directly.

Here lies a ModeSsim output in the early stages (0ns ~ 220ns to be precise) of a testing of the



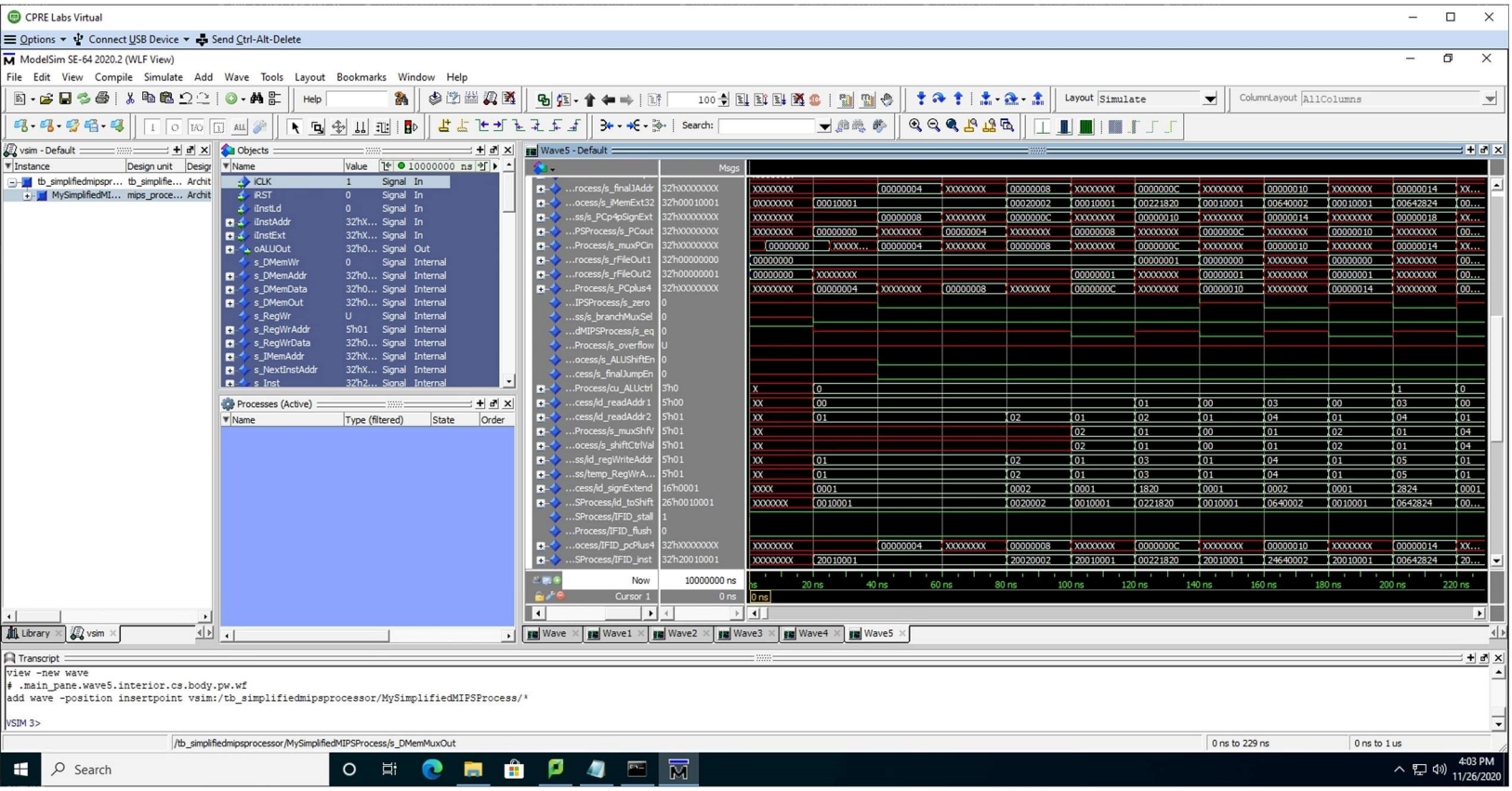
newly implemented processor with the new registers being used as data storage. It should be noted that this image does not in fact include all of the values but rather a third of them. The rest will follow further in the report. This image section includes a variety of values including, but not limited to: the clock, the instruction address, the ALU out value, the DMemWr signal, the DMemAddr, the DMemData, the DMemOut, the RegWr, the RegWrAddr, the RegWrData, the IMEMAddr, as well as other out values, and jumps and shift values at the bottom.

By looking at the IMemAddr (13 from the top) it show the address in which the IMem is pointing to and how it alternates between the actual use of its value as it is not constantly in use in this specific processor.

Notice as the halt signal (17 from the top) is at a constant 0 in this stage as it is prior to initialization to allow the processor to detect instances in which it should, itself, change this value from a hazard detection unit, but it is setup to take one if need be.

What is occurring during this sequence of the program is an initial setup of some registers using the addi instructions. It then moves on to test out the add in the form of a noop instruction, and then the add as a typical add instruction of storing previously setup register values into a new register. The program then continues through (utilizing noop instructions when need be) performing more of the supported instructions of the processor including: add, addi, add as noop, addiu, and, andi, lui, sw, lw, or, nor, ori, xor, xori, slt, slti, sll, srl,

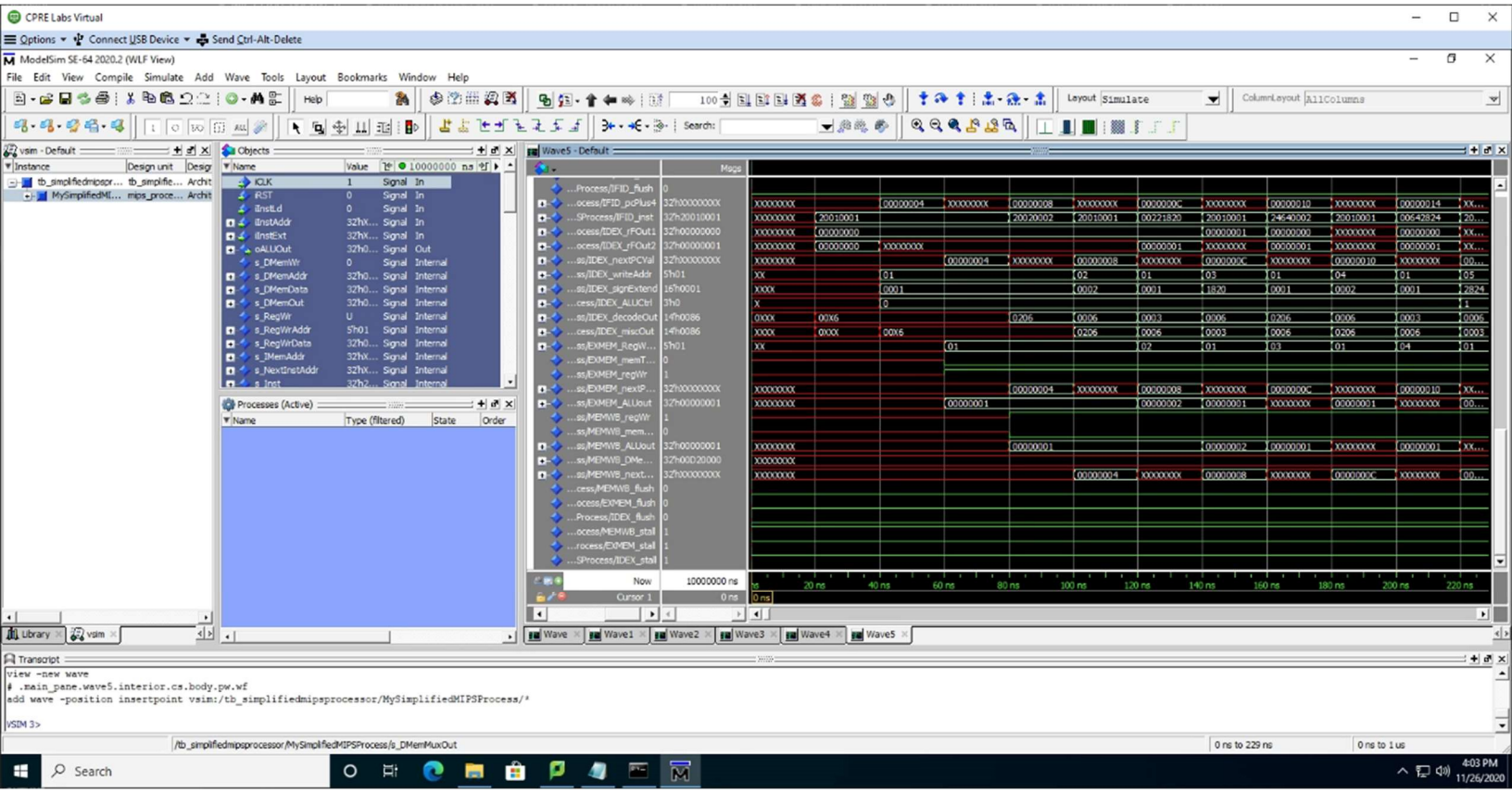
srav, srlv, beq, bne, j (not in this order though).



This is a continuation of a mapping of the signals that exist in the processor which includes a variety of signals including the values that are essentially temporary inside of the processor as they are generated and then passed into the various respective components that they correspond to.

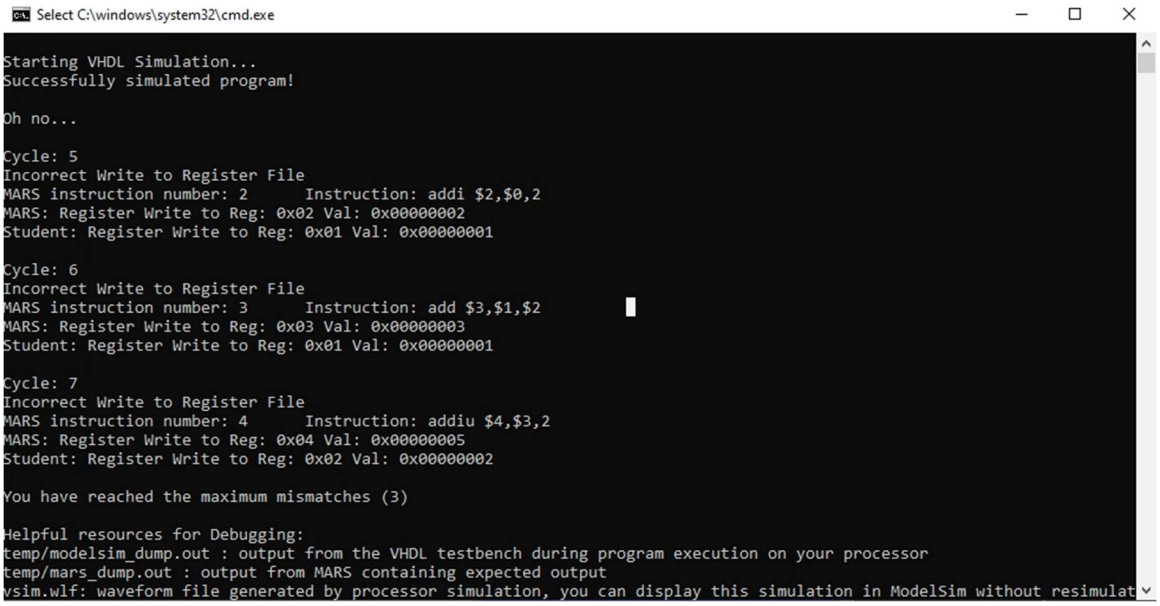
An example of this would be the values of the readAddr1 and readAddr2 (located 15 & 16 from the top correspondingly). These instructions are generated inside of the processor and are passed into the next component being the register file system after coming from the instruction decoder. With the new pipeline system the instruction decoder component gets its instruction value from the IF/ID pipeline register with the correct values when the processor determines this to be. As shown by the waveform the reading address between the two is changed at the appropriate times given the testing being run.

Another new feature of the pipeline components that can be seen with this section of the testing waveform is some of the values at the bottom indicating functions like the flush and stall of the IF/ID register, in this specific instance. The test purposely avoids situations where the registers would require a flush as a hazard unit is still to be added to the register, and this is why the IF/Id flush value is a 0 across the board in this test and because of this that register will never be flushed of its contents.



This is another continuation of the ModelSim waveform over the same time period of the testing process. One of the most noticeable of the features on this waveform is that of the variables at the bottom that is across the board of the same value. These are variables corresponding to the newer functionalities of the processor where, for example, the flush values of corresponding pipeline registers are always at the value of 0 as during this test section in which the image is derived never requires for the flush action to be taken on the registers. Not to mention that this is prior to the setup of a hazard detection unit set, so even if the test was in need of a flush there would be no way for it to occur.

Another notion that shows the progress of the test is the ALU control signal value that is being passed through the ID/EX pipeline register and it can be seen as starting at X as it does not begin simulation with a value and then it gains the value of 0 at 40 ns and then becomes a 1 at roughly 220 ns. This is because after the first assignment of the value is when the test program is going through the loading of variables and then testing of the add function and add function as an effective noop instruction. By the time that it is then passed a value of 1, from the ID/EX pipeline register, is because at this time the test is then using the and function which in this particular processor corresponds to the ALU control value of 1.



However an uncaught issue where everything was working other than from which location the data addressing was gathered from did occur.

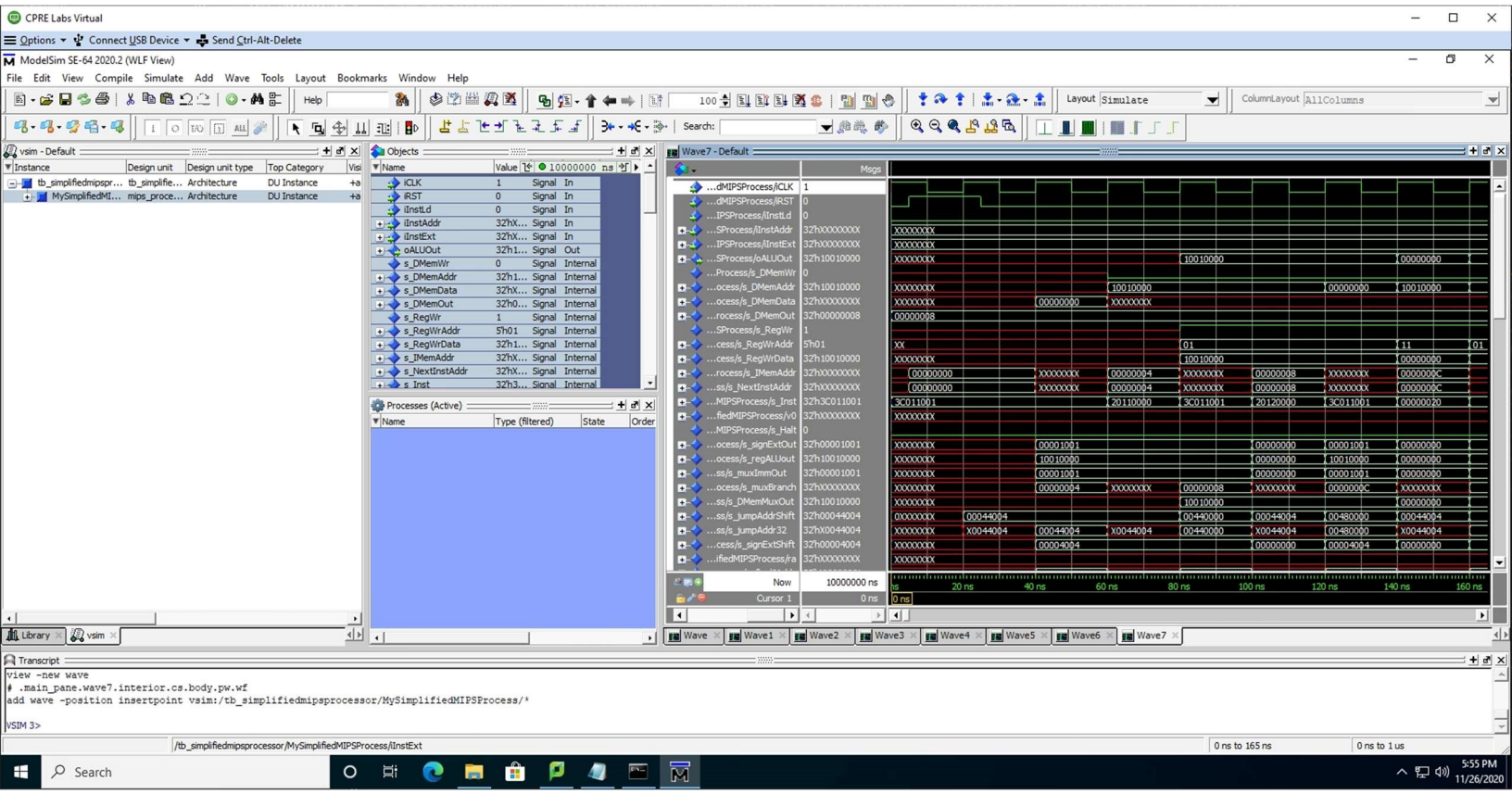
EDIT: As we went through last minute fixes we were able to get it to where everything was correct, but off by a cycle.

f. [Part 3 (b)] In your writeup, show the ModelSim output for the modified Bubblesort test, and provide a discussion of result correctness. It may be helpful to also annotate the waveforms directly.

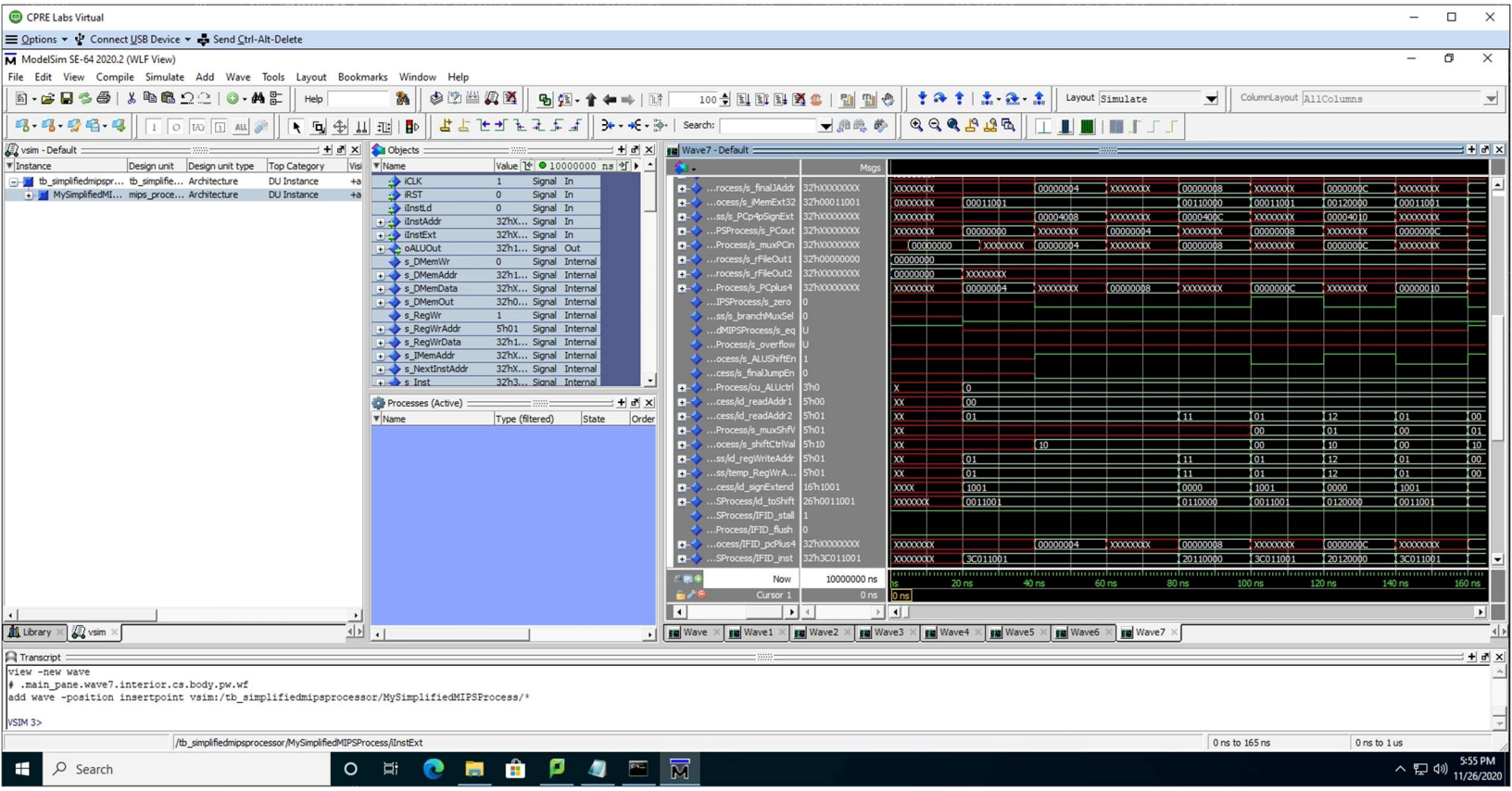
Similar to that of the test above we run into an error that we were not able to come up with a fix, but narrowed down the cause to be exclusively the connection between where the correct outputs of various operations are stored causing incorrect gathering of data when such registers or memory locations need to be accessed.

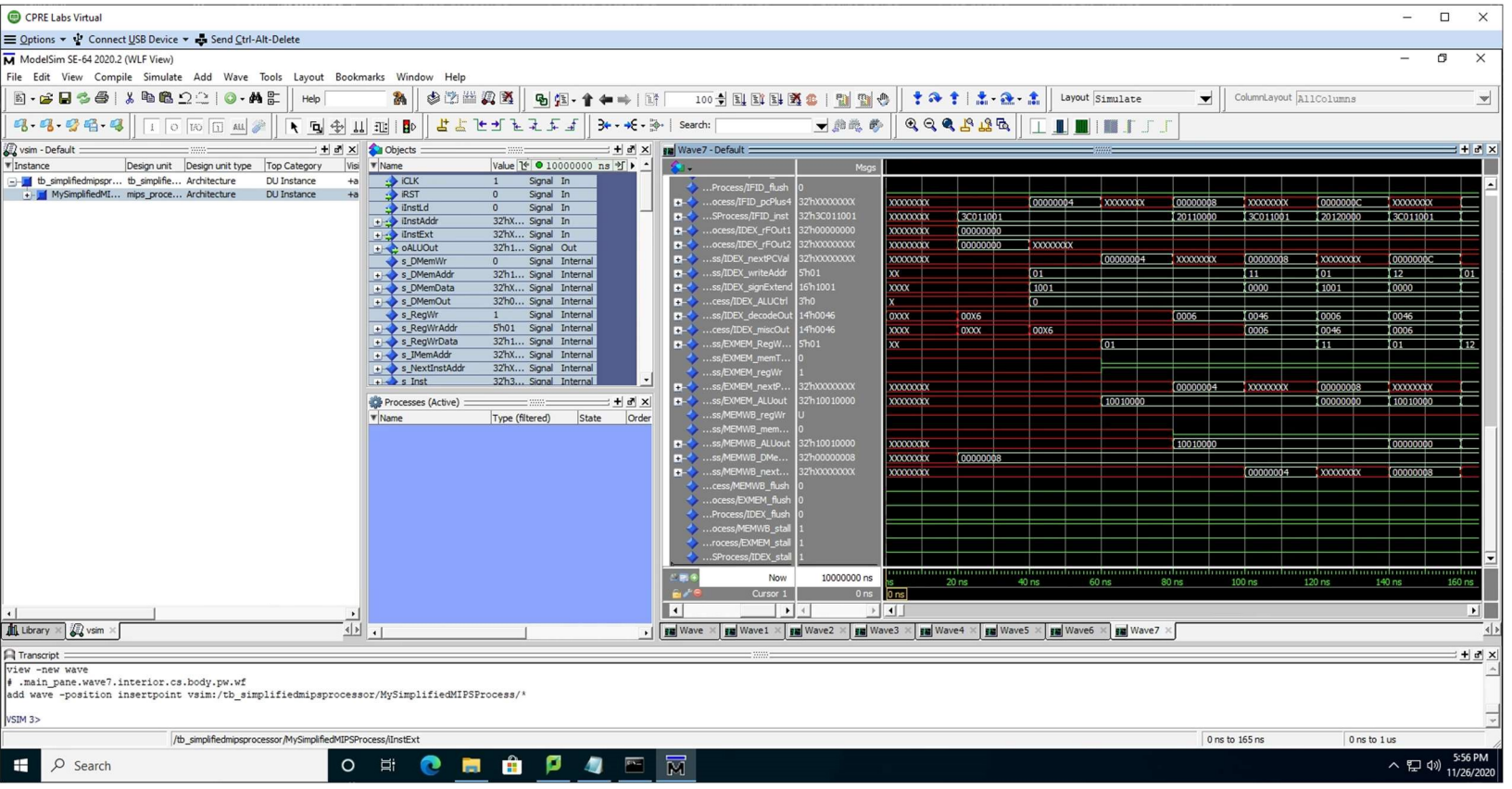
EDIT: As we went through last minute fixes we were able to get it to where everything was correct, but off by a cycle.

Waveform below show this:



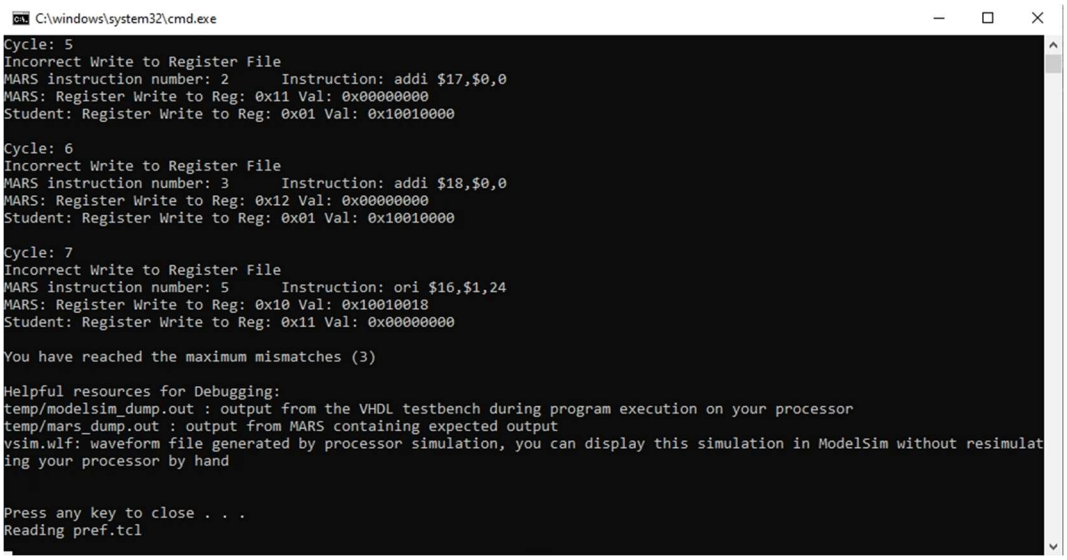
This like the other waveforms in this report show the first of three sets of the signals, but as they pertain to this test with being just a cycle off.


CPRE Labs Virtual
Options
Connect USB Device
Send Ctrl-Alt-Delete
ModelSim SE-64 2020.2 (WLF View)
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
vsim - Default
Objects
Wave7 - Default
Processes (Active)
Transcript
view -new wave
.main_pane.wave7.interior.cs.body.pw.wf
add wave -position insertpoint vsim:/tb_simplifiedmipsprocessor/MySimplifiedMIPSProcess/*
VSIM 3>
/tb_simplifiedmipsprocessor/MySimplifiedMIPSProcess/iInstExt
0 ns to 165 ns
0 ns to 1 us
Search
5:55 PM
11/26/2020


CPRE Labs Virtual
Options
Connect USB Device
Send Ctrl-Alt-Delete
ModelSim SE-64 2020.2 (WLF View)
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
vsim - Default
Objects
Wave7 - Default
Processes (Active)
Transcript
view -new wave
.main_pane.wave7.interior.cs.body.pw.wf
add wave -position insertpoint vsim:/tb_simplifiedmipsprocessor/MySimplifiedMIPSProcess/*
VSIM 3>
/tb_simplifiedmipsprocessor/MySimplifiedMIPSProcess/iInstExt
0 ns to 165 ns
0 ns to 1 us
Search
5:56 PM
11/26/2020

The two waveforms above are the next series of the signals as it pertains to the bubblesort test and being just one cycle off.

Assembly file for this can be found with the following datapath:
Proj-C -> cpre381-toolflow-release -> MARsWork -> Proj-C_test2 -> Proj-C_test2.asm



g. [Part 4] Report the maximum frequency your software-scheduled pipelined processor can run at and determine what your critical path is (specify each module/entity/component that this path goes through).

```
Info (115030): Assembler is generating device programming files
Info: Quartus Prime Assembler was successful. 0 errors, 1 warning
    Info: Peak virtual memory: 4735 megabytes
    Info: Processing ended: Thu Nov 26 15:49:33 2020
    Info: Elapsed time: 00:00:06
    Info: Total CPU time (on all processors): 00:00:04

Timing generation complete!
completed in 0:02:16.739031
Press any key to close . . .
```

```
#
# CprE 381 toolflow Timing dump
#

FMax: 29.76mhz Clk Constraint: 20.00ns

The path is given below

 ====================================================================
 From Node    : reg_nbit:ID_EX2|dffTrue:\G1:0:reg_n|s_Q
 To Node      : reg_nbit:EXMEM_ALUOutReg|dffTrue:\G1:31:reg_n|s_Q
 Launch Clock : iCLK
 Latch Clock  : iCLK
 Data Arrival Path:
```

More information on the synthesized processor is provided in the following file path:
Proj-C -> cpre381-toolflow-release -> temp -> timing.txt

h. [Part 5 (a)] Of the MIPS instructions supported for Project Part B, list which instructions produce values, and what signals in the pipeline these correspond to.

As this is essentially asking which instructions need to read from the data memory or the register file, the following instructions would fall into that categorization:

Instructions	Writes to Register File	Writes to Data Memory
Add	Yes	No
Addi	Yes	No
Addiu	Yes	No
And	Yes	No
Andi	Yes	No
Lui	Yes	No
Lw	No	Yes
Or	Yes	No
Ori	Yes	No
Sll	Yes	No
Sllv	Yes	No
Slt	Yes	No
Slti	Yes	No
Sra	Yes	No
Srlv	Yes	No
Sub	Yes	No
Sw	No	Yes
Xor	Yes	No
Xori	Yes	No

i. [Part 5 (b)] List which of these same instructions consume values, and what signals in the pipeline these correspond to.

As this is essentially asking which instructions both read and write values to either the register file or the data memory, the following instructions would fall into that categorization:

Instructions	Read from Register File	Read from Data Memory
Add	Yes	No
Addi	Yes	No
Addiu	Yes	No
And	Yes	No
Andi	Yes	No
Beq	Yes	No
Bne	Yes	No
J	No	Yes
Jal	No	Yes
Jr	Yes	No
Lw	No	Yes
Or	Yes	No
Ori	Yes	No
Sll	Yes	No
Sllv	Yes	No
Slt	Yes	No
Slti	Yes	No
Sra	Yes	No
Srlv	Yes	No
Sub	Yes	No
Sw	No	Yes
Xor	Yes	No
Xori	Yes	No

j. [Part 5 (c)] Come up with a generalized list of potential data dependencies. From this generalized list, select those dependencies that will require forwarding (write down the corresponding pipeline stages that will be forwarding and receiving the data), and those dependencies that will require hazard stalls.
As data dependencies are purely dependent on that of the program that is being run to make a generalized list can't get into specifics other than stating the following:
For any instruction that will require a reading from a register file then that register file must have gone through beyond the data memory prior to being accessed/utilized again. Similarly if an instruction requires a register file when writing it will need that register file not be actively written to for the sake of data consistency in the running program. The same logic and restrictions apply for reading and writing to data memory address. Therefore, forwarding would be valid for any instruction that is utilizing either data or a register file but not writing to it, and then it can be forwarded to an instruction at the ALU stage of operation. Otherwise it will need to be stalled.

k. [Part 6] Write a more generalized series of data forwarding and hazard detection logic equations based on the result from part 5).

l. [Part 7] Provide a high-level schematic drawing of the interconnection between components for the MIPS Hardware-scheduled pipelined Processor.

m. [Part 8 (a)] In your writeup, show the ModelSim output for the individual instruction tests and Bubblesort test (the original from Project Part B), and provide a discussion of result correctness. It may be helpful to also annotate the waveforms directly.

n. [Part 8 (b)] In your writeup, show the ModelSim output an application that attempts to exhaustively test the forwarding and detection logic in your pipeline, and provide a discussion of result correctness. It may be helpful to also annotate the waveforms directly.

o. [Part 10] Report the maximum frequency your processor can run at and determine what your critical path is. Draw this critical path on top of your top-level schematic. In your writeup, briefly discuss your critical path results. What components would you focus on to improve the frequency?

p. [Feedback] You must complete this section for your lab to be graded. Please complete each column **separately** for each team member; I expect it to take roughly 10 minutes (do not take more than 20 minutes).

i. How many hours did you spend on this lab?

Task	**During lab time**			**Outside of lab time**		
Team Initials	CI	MH		CI	MH	
Reading lab	.5h	1h		1h	1h	
Pencil/paper design	2h	3h		2h	3.5h	
VHDL design	1.5h	0h		7h	2h	
Assembly coding	0h	0h		0h	10h	
Simulation	1h	.5h		3h	1h	
Debugging	1h	.5h		6h	1.5h	
Report writing	0h	1h		.5h	3h	
Other:	0h	0h		0h	1h	
Total	6h	6h		19.5h	23h	

ii. If you could change one thing about the lab experience, what would it be? Why?

If we could change one thing about this particular lab experience would be to have had more of the time to go over the various aspects and concepts that this lab goes through with the TA's that would be able to offer their own insight into what we formulated. (Difficult due to the end of the year being busy for both us and the TA).

iii. What was the most interesting part of the lab?

The extensive process that we went through planning out the eventual additions to the processor that we had created thus far, and how those changes would

ultimately have an effect of the functionality. Going with that, the time spent discussing the aspects that we were each a little less informed on as we completed the planning portions of the lab like writing out the various dependency lists.