CprE 381 – Computer Organization and Assembly-Level Programming

Proj-B 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-B instructions for the context of the following questions.

a. [Part 0] How are instruction and data memory initialized in the simulation? How does MARS interface with ModelSim?

   Instructions and data memory are initialized via MARS.
   MARS interfaces with ModelSim through Quartus and Python files/scripts that for our specific case were provided in the toolflow files.

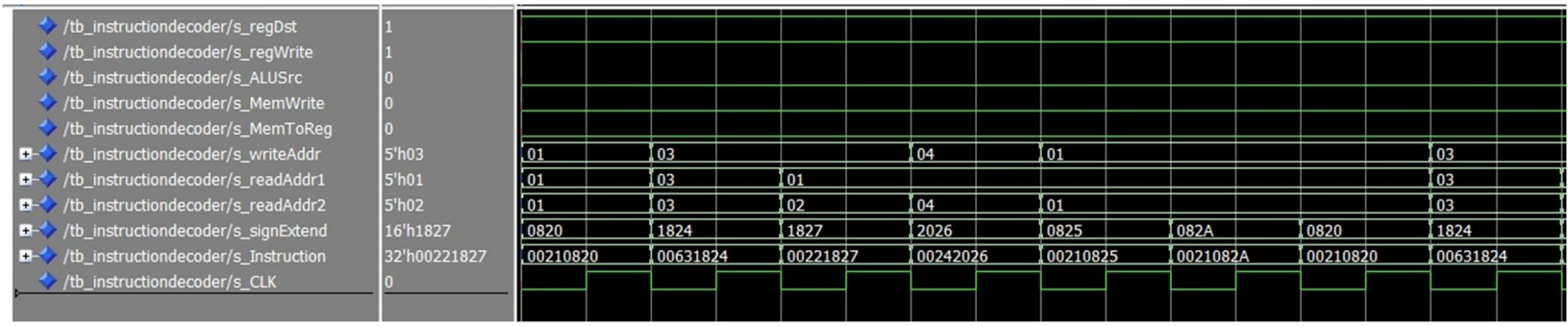
b. [Part 0] In the MIPS skeleton VHDL, how is a halting / termination condition detected? What MIPS assembly instruction does this correspond to?

   To give a specific instance in the provided VHDL would be in the "tb_SimplifiedMIPSProcessor.vhd" file where (unaltered) on line 108 alias halt is defined and given a signal of s_Halt that when in the instance it is equal to 1 later on it will notify of the halt and then use writeline with the values equivalent to dumping the line and will stop the process. This is an example of how it would go about detecting a condition of termination.
   This corresponds to the MIPS assembly instruction of jr.

c. [Part 1 (a)] Modify the provided spreadsheet to include the list of *M* instructions to be supported in this phase alongside their binary Instruction OPcodes and Funct fields, if applicable. Create a separate column for each binary bit. Inside this spreadsheet, create a new column for the *N* control signals needed in your single-cycle processor implementation. The end result should be an *M***N* table where each row corresponds to the output of the control logic module for a given instruction. *[You can attach the spreadsheet as a separate file, with a single spreadsheet for all Phase I and Phase II instructions.]*

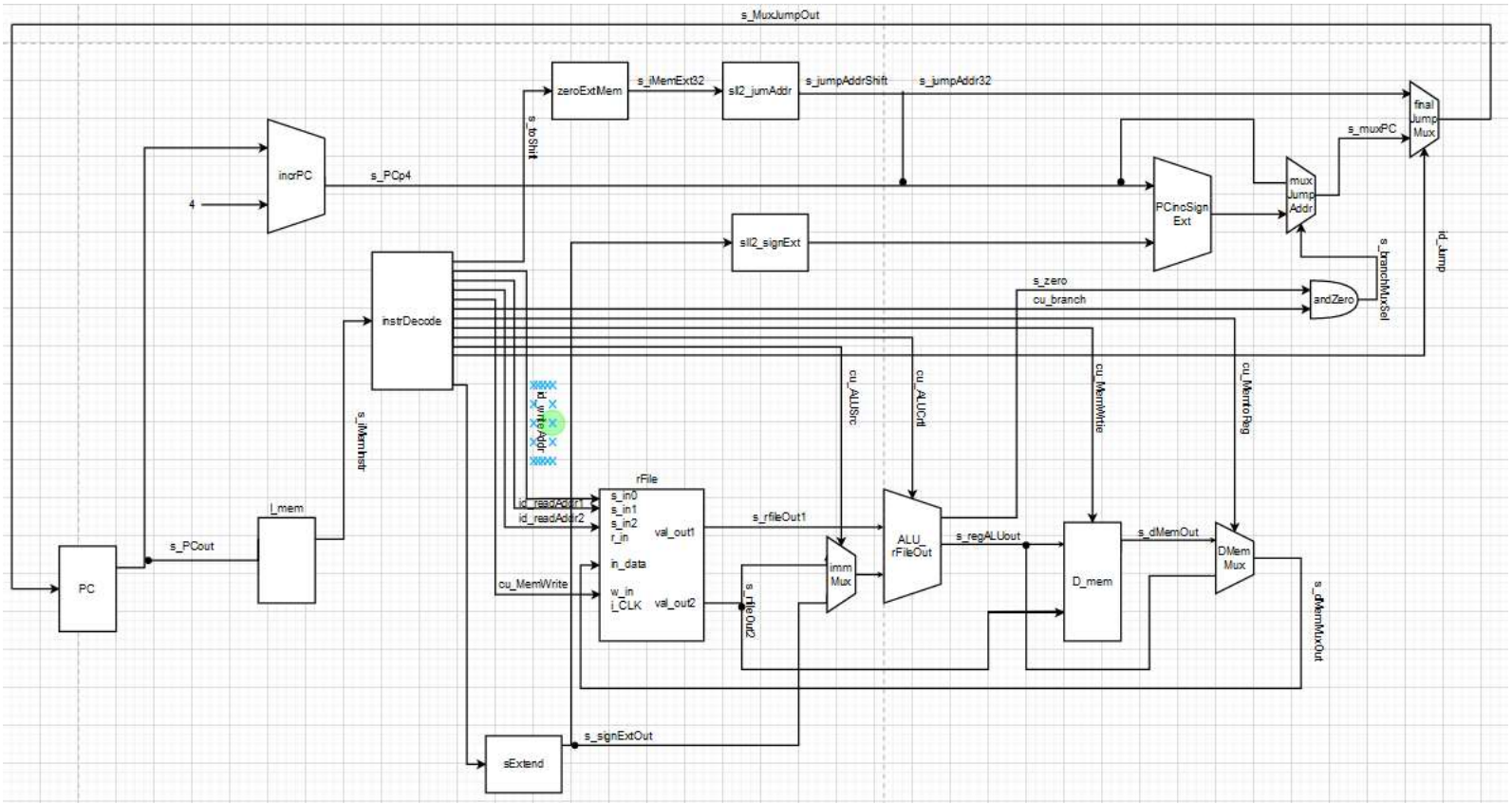
   See attached spreadsheet.

d. [Part 1 (b)] Implement the control logic module using whatever method and coding style you prefer. Create a testbench to test this module individually, and show that your output matches the expected control signals from part 1a). *[Please include waveforms and explanations.]*



Here is a basic test bench setup for the early stage control unit. What is occurring is that for every clock cycle another instruction it setup. It takes in a 32 bit instruction that varies in what instruction it is that is being called, so for example in the first cycle the instruction being called corresponds to the instruction with an opcode of 000000 and a function code of 100000 being add. Then corresponding to the add instruction it sets up the output select signals shown above that we believe to be correct based off of our original evaluation from the excel spreadsheet that is attached to this file.

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



The challenges that we faced while working through the design of the processor/schematic

for said processor was a variety of attributes including the time it took to ensure its correctness as well making sure it would work with the components that we already had and were already planning on creating. It also had to work with the control signal ideas that we setup in the excel sheet already.

f. [Part 4 (a)] Describe these possibilities as a function of the different control flow-related instructions.

As it was stated in the document that the Instruction Fetch Logic must be able to support any instructions that would modify the program control flow with non-trivial updates in order to support cases other than PC = PC + 4 would have a few possibilities. Some of which that would be among the different control flow-related instructions are those that maintain a different dataflow but then also require the PC to be incremented by a value other than 4. This would be an instruction like jump which requires an altered Datapath that will still need to access results from that of most the normal Datapath along with its additional route requiring a greater addition on the part of the PC and Instruction Fetch Logic.

g. [Part 4 (c)] Update your processor schematic for the instruction fetch logic and any other datapath modifications needed for control flow instructions. In your writeup, describe what additional control signals are needed.

Additional control signals that will be need to be implemented into the control unit for this stage would include a jump, and a branch control signal for the new portions of the processor. Other control signals would not be needed, but might be added in order to more easily get the correct Datapath processing for each of the potential MIPS instructions.

h. [Part 4 (d)] Include ModelSim waveforms, and describe how the execution of the control flow possibilities corresponds to the waveforms in your writeup.

Keeping the add instruction from the previous testbench testing the newer signals that may or may not be used at any given time inside of our processor. After the long break in tests in

when it tests some of the second phase stuff starting with BEQ, then J, and finally JR. As an inspection on the resulting testbench will show how they each have different impacts and that when it is a jump the jump select goes to true, and when it is a jr the jr select goes to true.

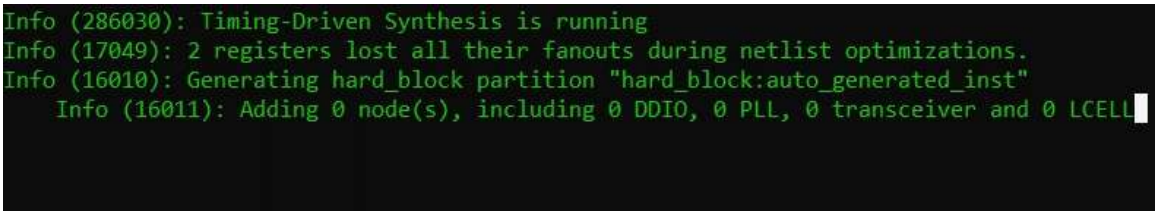
i. [Part 5 (a)] In your writeup, show the ModelSim output for the individual instruction tests, and provide a discussion of result correctness. It may be helpful to also annotate the waveforms directly.
Tests are also available in the file turn in.

These are the first few instructions of the test add immediate value 1 to register 1 and value 2 to register 2. Then adds register 1 and register 2 and put the contents into register 3 with the third value of 00000003.

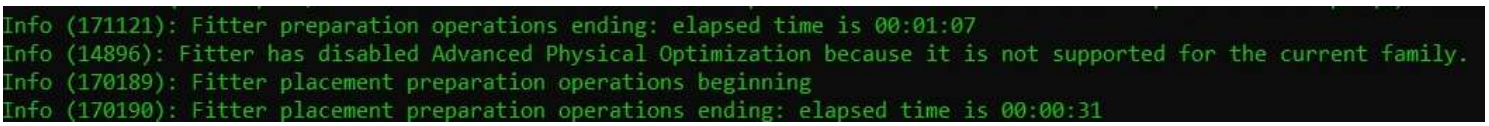
j. [Part 5 (b)] In your writeup, show the ModelSim output for the Bubblesort test, and provide a discussion of result correctness. It may be helpful to also annotate the waveforms directly.

k. [Part 5 (c) BONUS] In your writeup, show the ModelSim output for the Mergesort test, and provide a discussion of result correctness. It may be helpful to also annotate the waveforms directly.

l. [Part 7] 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?



Synthesis as it is running pictured above.



m. [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	0h	.5h		.5h	1 h	
Pencil/paper design	2h	.5h		2h	1h	
VHDL design	4h	3h		25h	11h	
Assembly coding	0h	0h		0h	2.5h	
Simulation	0h	.5h		3h	5.5h	
Debugging	0h	1h		7h	11h	
Report writing	0h	.5h		0h	2h	
Other: (Excel sheet)	2h	2h		0h	2h	
Total	8h	8h		37.5h	36h	

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

It was definitely a tough lab with a lot of potential errors that would then require fixing, but in terms of being able to get the information or resources that we needed there wasn't an instance of lacking. One potential change that might have made it better would be an easier way to ensure some of our less group specific control values early on as that could then become detrimental to rework later.

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

The most interesting part of the lab for us was getting all of the components to work together in the sense that they could then become a functional processor that when running the assembly files it could actually perform tasks and instructions. It was a really interesting way of seeing the results of what we had put together rather than a series of test benches because it made it seem more real and functional which was really fascinating.