CprE 381 – Computer Organization and Assembly Level Programming

Project Part 3

[Note: This is the final and summative component for your term project. The purpose of this assignment is to put the other components of your project into context and allow you to relate the detailed implementations you have done to the higher-level concepts we have interacted with in other aspects of the course. As such, ***this report is expected to be of higher quality than your previous reports, in particular with regard to its clarity, analysis, and readability****. Please make an effort to provide context for all figures and some flow through the paper (i.e., don't just copy the report template and respond directly). You have three working processor designs: Congratulations for getting to this step!*

Disclaimer: Due to this project being due during ~~Dead~~Prep Week, there will be no extensions granted. Please turn in whatever work you have completed by the due date.]

0. **Prelab.** Review your notes, evaluations, and feedback regarding your single-cycle, software scheduled pipeline, and hardware-scheduled pipeline. Make sure you have the three designs ready to evaluate. In particular, ensure that you have your synthesis results handy.

1. **Introduction.** Write a one paragraph summary/introduction of your term project.
   I made three designs of processor, single-cycle design processor, software pipeline processor, and hardware pipeline processor. The processors are allowed to excute add, addu, and, nor, xor, or, slt, sll, srl, sra, sub, subu, jr, addi, addiu, andi, lui, lw, ori, xori, slti, sw, beq, bne, j, jal, bgez, bgezal, bgtz, blez, bltz, btzal, and halt instruction. According to the timing log file, the single-cycle design processor has the lowest frequency compared to pipeline design processor. The software pipeline design processor has the highest frequency. The critical path for the pipeline design process is at the execution stage due to high CPI. And the critical path for single-cycle design unit will be PCReg => IMem => P_branchInst => controlLogicMux => ALUMux => ALU => DMem => MemMux => g_RegFile. Based on the information, I got from the timing file. The data needs 42.354 ns to write back to register file in the case of critical path in single-cycle design processor. For the hardware pipeline design processor, the data need 23.001 ns for data arrive from Pipeline ID/EX register to Pipeline EX/MEM register, which is the critical path of hardware pipeline design processor. For the software pipeline design processor, the data need 22.993 ns for data arrive from Pipeline ID/EX register to Pipeline EX/MEM register, which is the critical path of hardware pipeline design processor.

2. **Benchmarking.** Now we are going to compare the performance of your three processor designs in terms of execution time. Please generate a table for each of your final single-cycle, software-scheduled pipeline, and hardware-schedule pipeline designs. The rows should correspond to your synthetic benchmark (i.e., the one with all instructions), grendel (provided with the testing framework), Bubblesort, and, for teams >4, Mergesort. The columns should be # instructions (count using MARS), total cycles to execute (count using your Modelsim simulations), CPI (using the previous two columns to calculate), maximum cycle time (from your synthesis results), and total execution time (using the appropriate previous columns). Note that the applications used to benchmark the single-cycle and hardware-scheduled pipeline applications should be identical and thus the same number of instructions, while the software-scheduled

pipeline programs should be modified to work on the software-scheduled processor and thus should have more instructions. Count software-inserted NOPS as instructions. Make sure to include units and double-check that these results make sense from your first principles!

	Single Cycle Processor					Pipeline Processor Hardware					Pipeline Processor Software				
	#Instruction	Total Cycles	CPI	Maximum Cycle Time (ns)	Total Execution Time (ns)	#Instruction	Total Cycles	CPI	Maximum Cycle Time (ns)	Total Execution Time (ns)	#Instruction	Total Cycles	CPI	Maximum Cycle Time (ns)	Total Execution Time (ns)
Benchmark	81	81	1	39.37	3188.97	81	182	2.25	19.019	3466.21275	138	143	1.04	18.587	2667.60624
grendel.s	2116	2116	1	39.37	83306.92	2116	5363	2.53	19.019	101817.8361	5455	6487	1.19	18.587	120656.5812
bubblesort.s	606	606	1	39.37	23858.22	606	1529	2.52	19.019	29044.29528	1596	1901	1.19	18.587	35301.17388

3. **Performance Analysis.** Analyze the performance of the three applications on the three processors. Explain in your own words why the performance was better on one processor versus another or why some applications may have had a smaller difference in performance between processors versus other applications. This section should reference the above performance table and describe how it was generated including any formulas you used. The section should be about five to seven substantial paragraphs.

   The single-cycle processor has better performance compared to other design processors. According to the table shown above, the overall total execution time for single-cycle processor is lesser than other processors. But ideally, the hardware pipeline design processor has the better performance compared to other design processors. If the forwarding of the data was did correctly in the hardware pipeline processor, the total execution time of the hardware pipeline processor will be the lowest. In the benchmark testing, the processors have smaller difference in performances versus other applications because when there are no branch and jump instructions was being fetch or execute. In grendel and bubblesort testing, the software pipeline design processor has the highest total execution time due to the amount of jumping, branching and NOPs instruction.

4. **Software Optimization.** Identify and describe one software optimization (i.e., assemblylevel software refactoring) that would improve the performance of software on the software-scheduled pipeline relative to the others. Provide an estimate of the performance benefit this change could have given your specific benchmarks.

   There is a technique could improve the performance of software on the software-scheduled pipeline relative to the others is reducing the number of NOPs by switching the order of the instructions without changing the original function of the program. Pipeline stalls occur when a stage in the pipeline must wait for a previous stage to complete before it can proceed. By reordering the instructions, it may be possible to reduce the number and duration of pipeline stalls, improving overall performance. If the number of the NOPs instructions has been reduced, the software-scheduled pipeline performance will be the better one due to less number of instruction being executed. As a result, lower number in total cycles, and could probably achieve the ideal CPI, which is 1.

5. **Hardware Optimization.** Identify and describe at least one different hardware optimization for *each* design that would improve its performance. The optimization cannot be turning it into one of the other designs. Certain optimizations can be beneficial to more than one design – chose one design on which you would apply the optimization. Briefly list the specific set of changes you would have to make to your design to accommodate each optimization (a figure would be helpful). Provide an estimate of the performance benefit each optimization could have given your specific benchmarks.

   One hardware optimization for hardware-scheduled pipeline processor is to implement superscalar execution. Superscalar execution is a technique that allows multiple instructions to be issued and executed simultaneously by duplicating functional units. By using superscalar

execution, the processor can execute more than one instruction per cycle, improving the performance by reducing the overall execution time. To accommodate this optimization, the processor must have multiple functional units for each instruction type, as shown in the figure below. The performance benefit of this optimization will depend on the degree of instruction-level parallelism in the application, but it can potentially improve the performance of instruction-bound applications.

6. **It Depends.** Given the above discussion, you should now understand the interaction between the programs and your hardware designs in terms of performance. Identify or write a program that performs better on a single-cycle processor versus a hardware-scheduled pipeline and another one that performs better on the hardware-scheduled pipeline versus the software scheduled pipeline. Describe your approach to building these programs. If one of these cases is impossible given your designs, argue *quantitatively* why that is the case.

   Program that performs better on a single-cycle processor:
   A program that performs better on a single-cycle processor would be one that has a relatively low number of instructions and has a high data dependency between the instructions.

```
add $t0, $s0, $s1
add $t1, $t0, $s2
add $t2, $t1, $s3
```

   In the program shown above, each instruction depends on the result of the previous instruction. In this case, the single-cycle processor would be more efficient since it can execute each instruction in one clock cycle. The hardware-scheduled pipeline would incur overhead due to the pipeline stages and inter-stage register accesses, which would slow down the execution of the program.

   Program that performs better on a hardware-scheduled pipeline:
   A program that performs better on a hardware-scheduled pipeline would be one that has a high degree of instruction-level parallelism and has a low data dependency between the instructions.

```
add $t0, $s0, $s1
sub $t1, $s2, $s3
add $t2, $t0, $t1
mul $t3, $t2, $s4
```

   In the program shown above, each instruction can be executed independently of the others, and there is no data dependency between them. In this case, the hardware-scheduled pipeline would be more efficient since it can exploit the instruction-level parallelism by executing multiple instructions in parallel, while the software-scheduled pipeline would not be able to take full advantage of the parallelism due to the limited hardware resources.

   It is not possible to build a program that performs better on a single-cycle processor than on a hardware-scheduled pipeline, given the assumption that the hardware-scheduled pipeline is correctly implemented and optimized. The hardware-scheduled pipeline can exploit instruction-level parallelism and overlap the execution of multiple instructions, while the single-cycle processor can only execute one instruction per clock cycle. As a result, the hardware-scheduled pipeline is generally faster than the single-cycle processor for programs with a high degree of instruction-level parallelism.

7. **Challenges.** This term project was challenging for every group. In at least three detailed paragraphs, describe the three most critical challenges your group faced, how you resolved them, and how you could avoid them in the future.

   The first critical challenge is I worked on the project by myself throughout the semester. My team didn't respond to my messages until I reported it to professor. It is hard to work alone because I couldn't multitask. I could only work on a task at a time. This was causing the poor efficiency on the progress of the project. I am glad that the due date for the projects is flexible so that I could submit whenever I successfully complete the processors before the final due date. I set timelines on which tasks must be completed. Therefore, I won't miss submit my term project. In the future, I should report to the instructor as soon as possible so that I could get assigned with another teammate.

   The second critical challenge is the VHDL syntax. The VHDL syntax is very different compared to Java and C programming. One example is the for-loop in VHDL I used in the BarrelShifter component. I was using the for-loop to assign the data according to the number of bit that required to shift. The expression "i in 0 to k", the k must be a constant. Otherwise, the system will show syntax error. I resolved it by using if-else statement in the loop. While doing the shift right logically, I set the k to 31 and the condition to only do regNew(i) <= DATA(i+shiftAmount) if (i+shiftAmount < 31), else regNew(i) <= "0". I have to get familiar with VHDL coding syntax so that I won't struggle on fixing the syntax error.

   The third critical challenge is debugging. The debugging process was the one that I spent most of the time on because the errors could be just assigning wrong signal to the component and causing the wrong result. I kept getting the wrong output for the bgezal and bltzal instructions when I run the test in the given toolflow. The toolflow showed that the register 31 was written at the end of the cycle but it didn't show in the waveform. Therefore, I looked through all my codes and the requirement comment in the give MIPS skeleton code. I found out that the s_RegWr must connect to the register file. I couldn't use another signal to replace the s_RegWr and connect to the register file. I should make sure the required signals are connecting to the required component in the future in order to avoid from this happening.

8. **Demo.** You will be expected to demo your benchmarking process to the Professor and TAs on the project due date. Each member of the project group will be required to be present for the demo, which will take place during regular lab hours. During this time, you will describe the various design tradeoffs of your project parts, describe how they compare to each other, demonstrate simulations of your benchmarked applications, and discuss potential optimizations.