

# 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 DeadPrep 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.

In this report we will be evaluating three different processors we made over the semester. We started by creating a single cycle design. Then, for project 2 we created a pipelined processor. For the first part of project 2 we had to avoid hazards by using software. In our second part we implemented hardware that would avoid these hazards. In this report we will tear them apart and compare their performance.

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					
	# Instructions	Total Cycles	CPI	Max Cycle Time (ns)	Total Execution Time (ns)
Synthetic Benchmark	43	43	1	48.366	2079.74
Grendel	2116	2116	1		102342.46
Bubblesort	611	611	1		29551.63
Software-Scheduled Pipeline					
	# Instructions	Total Cycles	CPI	Max Cycle Time (ns)	Total Execution Time (ns)
Synthetic Benchmark	110	119	1.082	20.732	2467.52
Grendel	6852	7113	1.038		147453.78
Bubblesort	1490	1571	1.054		32558.78
Hardware-Scheduled Pipeline					
	# Instructions	Total Cycles	CPI	Max Cycle Time (ns)	Total Execution Time (ns)
Synthetic Benchmark	43	54	1.256	25.316	1367.27
Grendel	2116	2889	1.365		73121.22
Bubblesort	611	789	1.291		19969.29

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.

**# Instructions**, the number of instructions in the program. Overall, the programs run by Single-Cycle Processor have the same number of instructions as the Hardware-Scheduled Pipeline Processor because these processors are not manually inserting nops into the program to act as stalls, unlike the Software-Scheduled Pipeline Processor. Therefore, there is an increased number of instructions in the programs run by the Software-Scheduled Pipeline Processor.

**CPI**, Cycles per instruction is calculated by using  $\text{Total Cycles} / \# \text{ Instructions}$ :

- 1) For the Single-Cycle Processor, the CPI is 1. It takes 1 cycle for 1 instruction to be completed.
- 2) For the Software-Scheduled Pipeline Processor, the CPI is between 1.0 and 1.1. It takes roughly 1.0 to 1.1 cycles for 1 instruction to be completed.
- 3) For the Hardware-Scheduled Pipeline Processor, the CPI is between 1.3 and 1.4. It takes roughly 1.3 to 1.4 cycles for 1 instruction to be completed.

The Hardware-Scheduled Pipeline Processor has a higher CPI than the Single-Cycle Processor even though the programs that they are running have the same number of instructions. This is because the Hardware-Scheduled Pipeline Processor needs to insert nops to stall and flush the pipeline in order to eliminate data and control hazards, hence the increased cycle count for each program.

The same reason goes for Software-Scheduled Pipeline Processor having a higher CPI than the Single-Cycle Processor. We would need to manually insert nops between instructions in the assembly code to eliminate hazards. Therefore, there is an increased number of instructions as well as an increased number of cycles. Even though the CPI of the Software-Scheduled Pipeline Processor is higher than the CPI of the Hardware-Scheduled version, this does not mean that the performance of the Software-Scheduled Pipeline is better because it takes more instructions as well as more cycles per program compared to the Hardware-Scheduled Pipeline.

**Max Cycle Time**, the cycle period or time required to complete a cycle:

The Single-Cycle Processor has the highest Cycle period because its datapath is not divided into five stages, unlike the Software and Hardware-Scheduled Pipeline Processors. Therefore, the max cycle time of a Single-Cycle Processor will be its critical path, which in our case is 48.366 ns.

Software and Hardware-Scheduled Pipeline Processors split their datapath into five stages, namely Fetch (IF), Decode (ID), Execute (EX), Memory (MEM), and Writeback (WB). To get the maximum cycle time of these pipeline processors. First, we need to know the critical path of each stage (Time needed for data to be transmitted from one stage to another, i.e., From IF to ID stage.), and the max cycle time would be the maximum critical path in the five stages (Max {IF, ID, EX, MEM, WB}).

A logical explanation of why our Hardware-Scheduled Pipeline has a higher cycle time than the Software-Scheduled version is because of the forwarding, stalling, and flush units that is not present in the Software-Scheduled version. These components will increase the critical path of the stages that it is located in and, in turn increasing the max cycle time.

**Execution Time**, (ET) is calculated using the formula :

$$ET = \# \text{ Instructions} * CPI * \text{Max Cycle Time}.$$

Our Software-Scheduled Pipeline Processor has the worst performance overall, with the highest Execution Time in all 3 programs. This is because the Software-Scheduled Pipeline programs have roughly 2.5 times more instructions than the programs run by the other processors due to the insertion of nops in the programs. Again, these nops are necessary because the Software-Scheduled Pipeline does not have any hazard detection units (stall, forward, flush). So even though the Software-Scheduled Pipeline Processor has a lower CPI and Cycle Time than the Hardware-Scheduled version, the number of instructions that it needs to execute because of the lack of hazard detection is 2.5 times more, and it is enough to offset the lower CPI and Cycle Time.

Our Single-Cycle Processor has a worse performance than the Hardware-Scheduled version. This is because of the higher max cycle time which offsets the lower CPI. Even though our Single-Cycle Processor has a CPI of 1 which is about 30% lower than the Hardware-Scheduled Pipeline Processor, it has an about 50% higher max cycle time than the Hardware-Scheduled Pipeline Processor, which results in a higher Execution Time and lower performance.

### **In terms of Application/Program**

There is a lower difference in performance if a processor executes a program with a lot of data hazards such as grendel. Between the Single-Cycle processor and the Hardware-Scheduled Pipeline Processor:  $\text{Speedup} = ET_{SC} / ET_{HSP}$

#### **In terms of grendel:**

$$\text{Speedup} = ET_{SC} / ET_{HSP} = 102342.46 / 73121.22 = 1.4 = 40\%$$

### **In terms of synthetic benchmark:**

The speedup of synthetic benchmark is  $= ET_{SC} / ET_{HSP} = 2079.74 / 1367.27 = 1.521 = 52\%$

The synthetic benchmark program has a lot less data and control hazards compared to grendel because there is only two branching instructions (beq, bne) as well as three jump instructions (j, jr, jal), not to mention that it has a lot less data hazards (arithmetic instructions after lw, lh, lb, lhu, lbu) compared to grendel. The stalls and flushes because of hazards are more frequent in grendel, hence more cycles need to be used to eliminate these hazards, resulting in a lower performance and execution time.

4. **Software Optimization.** Identify and describe one software optimization (i.e., assembly level 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.

One software optimization is reorganizing your code. This could help avoid data hazards. For example if you have an add that writes to a certain register and you do another operation that uses that specific register you can add other instructions after the first instruction to avoid the hazard.

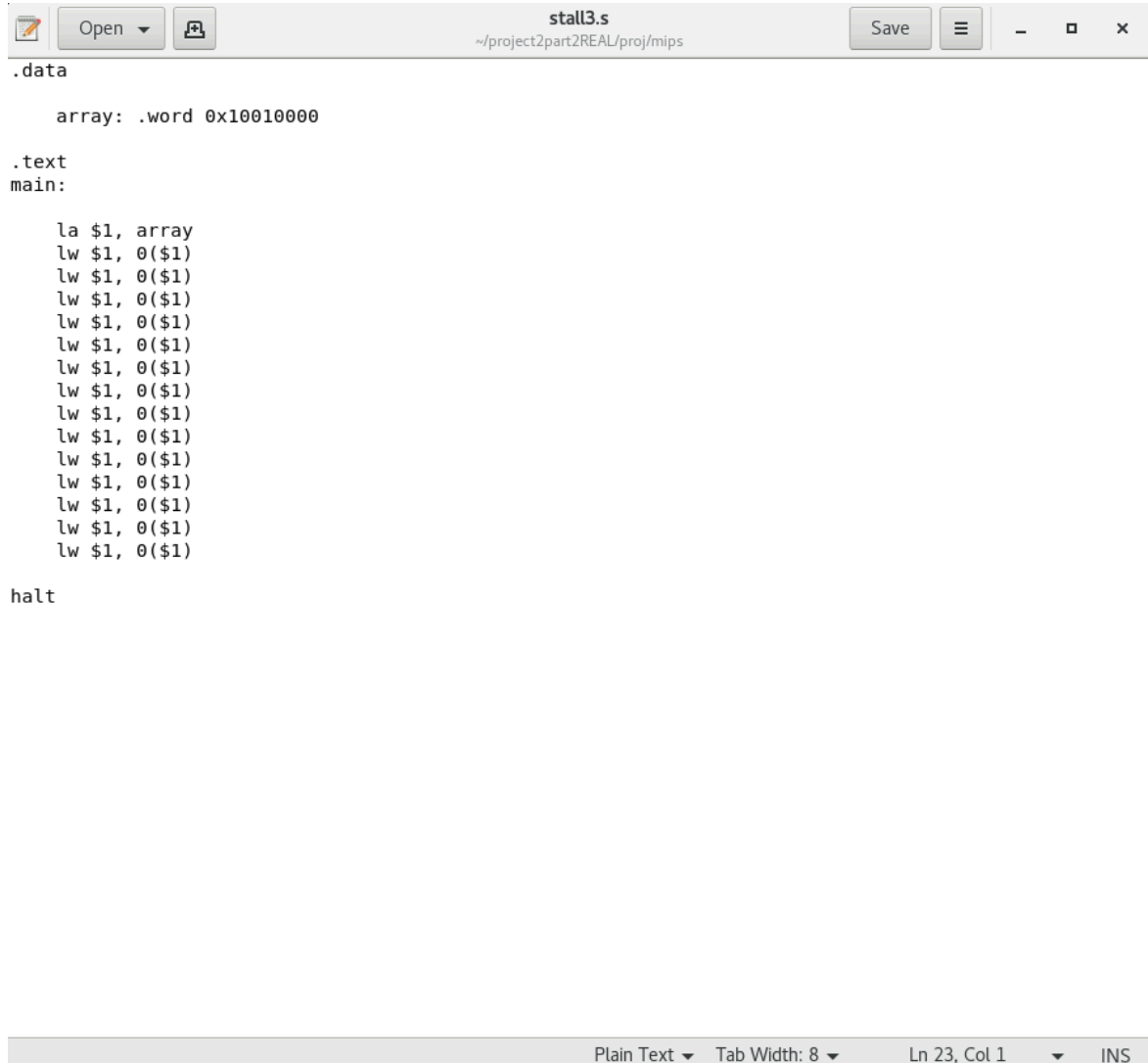
5. **Hardware Optimization.** Identify and describe at least one different hardware optimization for each design that would improve its performance. The optimization cannot be turning it into one of the other designs. Certain optimizations can be beneficial to more than one design – chose one design 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.

Branch prediction would help with performance. Our processor always assumes that you don't take the branch. However, if in your code you have a loop that constantly takes the branch we have a constant squash in our processor. If we had a dynamic branch prediction it would avoid that squash as it would predict that it would take that branch and have a cache with that instruction value which would avoid a squash. This would improve performance by reducing the instructions by one for each iteration of the loop.

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 runs better on Single-Cycled than Hardware-Scheduled Pipeline:



```
.data
    array: .word 0x10010000

.text
main:
    la $1, array
    lw $1, 0($1)
    lw $1, 0($1)
    lw $1, 0($1)
    lw $1, 0($1)
    lw $1, 0($1)
    lw $1, 0($1)
    lw $1, 0($1)
    lw $1, 0($1)
    lw $1, 0($1)
    lw $1, 0($1)
    lw $1, 0($1)
    lw $1, 0($1)
    lw $1, 0($1)
    lw $1, 0($1)
    lw $1, 0($1)
    halt
```

Plain Text ▼ Tab Width: 8 ▼ Ln 23, Col 1 ▼ INS

### For Single-Cycle Processor:

```
Testing file: proj/mips/stall3.s
Mars simulation: pass
Modelsim simulation: pass
Test Result: pass
Mars Instructions: 17
Processor Cycles: 17
CPI: 1.0
Results in: output/stall3.s
-----
```

# Inst = 17

Cycles = 17

CPI = 1.0

### For Hardware-Scheduled Pipeline:

```
bash-4.4$ ./381_tf.sh test proj/mips/stall3.s
Using LAB Python Environment
Testing
All VHDL src files compiled successfully
Testing file: proj/mips/stall3.s
Mars simulation: pass
Modelsim simulation: pass
Test Result: pass
Mars Instructions: 17
Processor Cycles: 34
CPI: 2.0
Results in: output/stall3.s
-----
```

# Inst = 17

Cycles = 34

$$\text{CPI} = 2.0$$

Therefore:

$$\text{ET}_{\text{SC}} = \# \text{ Instructions} * \text{CPI} * \text{Max Cycle Time}$$

$$\text{ET}_{\text{SC}} = 17 * 1.0 * 48.366$$

$$\text{ET}_{\text{SC}} = 822.222 \text{ ns}$$

$$\text{ET}_{\text{HSP}} = \# \text{ Instructions} * \text{CPI} * \text{Max Cycle Time}$$

$$\text{ET}_{\text{HSP}} = 17 * 2.0 * 25.316$$

$$\text{ET}_{\text{HSP}} = 860.744 \text{ ns}$$

$$\text{ET}_{\text{HSP}} > \text{ET}_{\text{SC}}$$

Hardware-Scheduled Pipeline has a longer Execution time than Single-Cycled. Therefore Single-Cycled Processor has a better performance running this program.



**Program that performs better on the hardware-scheduled pipeline versus the software scheduled pipeline: Bubblesort.s**

```
.data
arr: .word 7 23 4 12 19 6 5 8 33 72 12 # Unsorted array
n: .word 11 # Length of array

.text
main:
la $t1, arr # $t1 = &(arr[0])
lw $a0, n # $s0 = n = 11

add $s0, $zero, $zero # i = 0
add $s1, $zero, $zero # j = 0
add $s2, $zero, $zero # swapped = 0

subiu $s3, $a0, 1 # n - 1 = 10

loop:
beq $s0, $s3, exit # if (i == n - 1) exit loop
add $s2, $zero, $zero # swapped = false

    innerLoop:
        subu $s4, $s3, $s0 # n - i - 1
        beq $s1, $s4, exitInner # if (j == n - i - 1) exit inner loop

        sll $t5, $s1, 2 # j * 4
        add $t6, $t1, $t5 # &arr[j]
        lw $t3, 0($t6) # arr[j]
        addi $t7, $t6, 4 # &arr[j+1]
        lw $t4, 0($t7) # arr[j+1]
        addi $s1, $s1, 1 # j++
        bgt $t3, $t4, swap # if(arr[j] > arr[j+1]) swap
        j innerLoop

        swap:
        addi $s2, $zero, 1 # swapped = true
        sw $t3, 0($t7) # arr[j + 1] = arr[j]
        sw $t4, 0($t6) # arr[j] = arr [j + 1]
        j innerLoop

    exitInner:

beq $s2, $zero, exit # if (swapped == false) exit
addi $s0, $s0, 1 # i++
add $s1, $zero, $zero # j = 0
j loop # loop

exit:
halt
```

#### For Hardware-Scheduled:

```
bash-4.4$ ./381_tf.sh test proj/mips/Proj1_bubblesort.s
Using LAB Python Environment
Testing
All VHDL src files compiled successfully
Testing file: proj/mips/Proj1_bubblesort.s
Mars simulation: pass
Modelsim simulation: pass
Test Result: pass
Mars Instructions: 611
Processor Cycles: 789
CPI: 1.29
Results in: output/Proj1_bubblesort.s
-----
```

#### For Software-Scheduled:

```
bash-4.4$ ./381_tf.sh test ./proj/mips/Proj1_bubblesort.s
Using LAB Python Environment
Testing
All VHDL src files compiled successfully
Testing file: proj/mips/Proj1_bubblesort.s
Mars simulation: pass
Modelsim simulation: pass
Test Result: pass
Mars Instructions: 1490
Processor Cycles: 1571
CPI: 1.05
Results in: output/Proj1_bubblesort.s
```

$$ET_{HSP} = \# \text{ Instructions} * CPI * \text{Max Cycle Time}$$

$$ET_{HSP} = 611 * 1.29 * 25.316$$

$$ET_{HSP} = 19969.29 \text{ ns}$$

$$ET_{SSP} = \# \text{ Instructions} * CPI * \text{Max Cycle Time}$$

$$ET_{SSP} = 1490 * 1.5 * 20.732$$

$$ET_{SSP} = 32558.78 \text{ ns}$$

$$ET_{SSP} > ET_{HSP}$$

Execution time of Software-Scheduled is higher than Hardware-Scheduled, therefore performance of Hardware-Scheduled is better than Software-Scheduled while running bubblesort.s

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.

Three issues we faced were communication, time, and knowledge.

In our group we would get tunnel vision at times not communicating with each other. We also were not on the same page at times. We should have had more discussions about what each other were doing. We should have also had more discussions about design choices. We would sometimes just do stuff individually and should have done it as a team.

Time was a large problem for us. We were both taking a lot of other credits so it was hard to find the time to work on the project. It felt like we were trying to squeeze in time to work on it. If we were not as busy it would not be as bad.

The last issue we faced was knowledge. We lacked knowledge about certain things. It caused us to run into problems not knowing how things worked. We also had to spend time researching and figuring out the correct way to implement things. We also implemented things wrong the first time from a lack of knowledge and had to reimplement things which was troublesome as we already didn't have a lot of time to work on the project.

There were things we would do differently if we were to do this project again. These were three of the big things we faced and will try to keep in mind for future projects.

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.