CISC 340 – Project 2

Due date: Class time, Wednesday, April 1
(10 points)

1. Purpose

This project is intended to help you understand multicycle CPU designs, how such a design executes machine instructions, and how to compare the performance of such design to that of a single-cycle design.
You will write procedural C.
You will use gcc on the departmental linux system (rusty).
You will do this project with your partner.

2. Problem

This project has two parts:

1. You will write a multicycle behavioral simulator for UST-3400 machine code.
2. You will write a simple assembly benchmark suite to compare the performance of the multicycle design and the single-cycle design.

3. UST-3400 Multicycle Design (Pappy Van saWinkle) (50%)

For this project, you will only need to know how the UST-3400 single-cycle simulator from project 1 works. You can reference project 1 and class lecture notes for details.

We are now extending the single-cycle UST-3400 design to a multicycle design. The idea that we are exploiting here is that different instructions have different datapaths. These different datapaths take different amounts of time! In the single-cycle design the CPU clock is set to the time of the slowest instruction. The multicycle design is the antithesis of the single-cycle as the multicycle design sets the CPU clock to the time of the fastest instruction. All instructions that take longer than that time require multiple cycles to complete.

We are going to assume the following breakdown:

Instruction	Number of cycles
add	2
nand	2
lw	8
sw	8
beq	4
jalr	1
noop	1
halt	1

Your multicycle simulator should print the processor state and the total number of cycles executed, just like your single-cycle simulator.

5. Benchmark suite (50%)
The second part of this assignment is to write a suite of UST-3400 assembly language programs to demonstrate the difference in performance between your single-cycle and multicycle designs.

Your benchmark suite should include programs that are “heavy” on each of the following instruction groups:

- add/nand
- lw/sw
- beq
- jalr

as well as programs that are a balance thereof.

5.1 Performance analysis
There is no point to the benchmark suite without some analysis. Once you have developed your benchmark suite you should run it on both your single-cycle and multicycle designs and record the number of cycles executed in each test. You should then write up a 3-4 page document (double spaced) outlining the performance of the multicycle machine relative to the single-cycle machine. It is important to compare your experimental results to theoretically predicated behavior. Make sure to detail what equations you use (and those equations should be nicely formatted). **NOTE:** You may assume the clock in the single-cycle design is 9 times slower than the clock in multicycle design.

6. Grading
I will grade primarily on functionality, including simulating all instructions, correct performance statistics, and comprehensiveness of the test suite. I won’t be docking you points for program style. That shouldn’t prevent you from using good coding practices, however.

The best way to debug your programs is to generate your own test cases, figure out the correct answers by hand, and compare your program's output to the correct answers. This is also one of the best ways to learn the concepts in the project.

Your benchmark suite will be graded according to how thoroughly its tests the UST-3400 simulators.

Finally, grading will be performed on the departmental Linux machines (rusty). Make sure your programs run correctly on this platform.

Printing execution statistics will be the final output of the simulator. Use the following updated code:

```
void print_stats(int n_instrs, int n_cycles){
        printf("INSTRUCTIONS: %d\n", n_instrs); // total executed instructions
        printf("CYCLES: %d\n", n_cycles); // total executed cycles
}
```

8. Turning in the Project
The entirety of the project will be turned in via canvas. A hard copy of your overview document should also be submitted in class on the due date. The following components (**and structure!**) are necessary for submission:

- Directory for your project named project2_username1_username2

- code for your simulator
- a Makefile to compile your code (include a clean rule!)
- Sub-Folder of assembler benchmark test files (assembly code, and machine!)
- A README file with instructions and explanations of files.
- One overview documents (pdf) telling me how your multicycle simulator works. This document should also communicate any difficulties you had in developing the program, as well as any shortcomings that remain (e.g., did not finish implementing jalr).
- One document (pdf) outlining the results of the test suite and the performance of the multi-cycle design relative to the single-cycle design.

Place all of this stuff inside a directory named “project2_username1_username2” using whatever your login name is. You can then zip this directory (project2_username1_username2) using the linux command
“zip -r project2_username1.zip project2_username”. Submit the zip file via Canvas on the due date.

Mac users: creating a zip on a mac includes an unneeded __MACOSX directory. **Do not include this directory.** E.g., do your zipping on a linux or windows machine.