CS/ECE 552
Spring 2022
Homework 4
Due 11:59 PM Central Time on Friday, March 25th, 2022

You should do this assignment on your own, although you are encouraged to talk with classmates in person or on Piazza about any issues you may have encountered. If you take code from another student, you will be penalized according to the UW Student Code of Conduct, as laid out in the syllabus. The standard late assignment policy applies: you may submit up to 2 days late with a 10% penalty for each late day. See *Hand in* directions for additional details.

Total Points: 12

This homework assignment requires writing assembly tests for your project. A reminder of some important documents:

1. Info about how to write assembly code and also about how to use the assembler can be found in the Using the assembler page.
2. Details about what each instruction means is available in the ISA specification.
3. The WISC-SP22 Simulator-Debugger page has information on the simulator for the WISC-SP22 ISA.
4. Other useful documents on how to write tests:
    a. Other example test programs to understand syntax etc.
    b. Test Programs FAQ

Baseline provided (template) files:
/u/s/i/sinclair/public/html/courses/cs552/spring2022/handouts/verilog_code/hw4/hw4-templates.tgz

The work flow we will follow is:

1. Write test in WISC-SP22 assembly language.
2. Assemble using assembler `assemble.sh`
3. Simulate the test in the simulator and make sure your test is doing what you thought it was doing. Use the simulator: `wiscalculator` (binary accessible on the CSL machines: `/u/s/i/sinclair/public/html/courses/cs552/spring2022/handouts/bins/wiscalculator`)
4. Write up your explanation of each test.

Below is a short demo:

```
prompt% assemble.sh add_0.asm
Created the following files
loadfile_0.img  loadfile_1.img  loadfile_2.img  loadfile_3.img
loadfile_all.img  loadfile.lst

prompt% wiscalculator loadfile_all.img
```

```
WISCalculator v1.0
Author Derek Hower (drh5@cs.wisc.edu)
Type "help" for more information

Loading program...
Executing...
lbi r1, -1
INUM:        0 PC: 0x0000 REG: 1 VALUE: 0xffff
lbi r2, -1
INUM:        1 PC: 0x0002 REG: 2 VALUE: 0xffff
add r3, r1, r2
INUM:        2 PC: 0x0004 REG: 3 VALUE: 0xfffe
halt
program halted
INUM:        3 PC: 0x0006
Program Finished

prompt%
```

The simulator will print a trace of each instruction along with the state of the relevant registers. You should examine these to make sure that your test is indeed doing what is expected.

Suggestions:

- Identify corner cases. Think about possible bugs in the hardware with pipelining.
- Ideally tests should be short and target specific cases, NOT all cases at once.
- Write comments in your assembly code explain what the test is doing (comments use "//" for our assembler)
- Make sure that all of your assembly files will assemble. **You won't get credit for malformed assembly.**
- The goal of this problem is to make sure you understand the ISA and develop targeted tests for the (pipelined) hardware. Understanding the ISA is required (and extremely helpful!) before building (pipelined) hardware for it!

Problem 1 [6 points]

a) [3 points] Based on the below table, write three unique tests that use a specific instruction based on the first letter of your last name. In addition to using this instruction, your assembly tests should test different parts of the pipeline (e.g., different forwarding paths or stalls, or different corner cases for the instruction such as sign extension).

First Letter of Last (Family) Name	Instruction
A	ROLI
B	SLLI
C	RORI
D	SRLI
E	ST
F	LD
G	STU
H	BTR

I	ROL
J	SRL
K	ROR
L	SLL
M	SEQ
N	SLT
O	SLE
P	SCO
Q	SIIC & RTI
R	SLBI
S	J
T	JR
U	JAL
V	JALR
W	XOR
X	XORI
Y	ANDN
Z	ANDNI

It is fine if your tests contain other instructions (e.g., LBI to initialize registers, HALT to stop the program, and NOPs to test various pipeline stalls, forwarding cases, or branch cases), but each of your tests must include the corresponding instruction above as well. For example, the first letter in "Sinclair" is 'S', so if I were writing tests each one must include at least one J (Jump) instruction. Note: for 'Q' both SIIC and RTI should be included because both are needed for the program to properly return from the exception handler.

b) [3 points] Write an explanation of your programs including where and why forwarding takes place. Write your answer in `p1.txt` (i.e., explain all three programs in the same file).

Problem 2 [6 points]

a) [4 points] Write two unique assembly programs to demonstrate why branch prediction is necessary and useful. Write your code in `p2-1.asm` and `p2-2.asm`, respectively.

b) [1 point] Write an explanation of your programs and how branch prediction helps in `p2.txt`.

c) [1 point] Will branch prediction always take only 1 cycle? Write your answer in `p2.txt`.

What to Hand In

To submit this assignment, zip or tar your Verilog files together and submit them as a **single file named <netID>-hw4.tgz or <netID>-hw4.zip** on Canvas. Inside this tarball/zip, should be a top-level directory (e.g., hw4), which contains hw4_1/ and hw4_2/; inside hw4_1 should be all files for problem 1 and so on for other problems – you must keep this directory structure. For example, my Net ID is msinclair, so my submission would be called msinclair-hw4.tgz (or msinclair-hw4.zip) and in it would be hw4, which itself has 2 files/sub-folders: hw4_1/ and hw4_2/. **If you need the same file for multiple problems (e.g., problem 1 and problem 2), you should have a copy of that file in each part**. *Also, please do not create a top-level folder above these.* For example, my Net ID is msinclair, so my submission would be called

msinclair-hw4.tgz (or msinclair-hw4.zip) and in it would be hw4_1/ and hw4_2/. If you don't have experience with tar, I recommend consulting tutorials such as this one.

Graphically your handin directory should look like:

- hw4 (Top level folder) // tar/zip this as <netID>-hw4.tgz or <netID>-hw4.zip
  - hw4_1 (folder containing everything for Problem 1, including p1.txt)
  - hw4_2 (folder containing everything for Problem 2, including p2.txt)

If an assembly file is missing or does not compile, you will automatically get **zero points** for that part.

Note: no schematics are needed for this assignment.

Verifying Your Handin

We have also created a script to check that your submission correctly follows the format, located at:

```
/u/s/i/sinclair/public/html/courses/cs552/spring2022/handouts/scripts/hw4/verify_submission_format.sh <netID>-hw4.tgz
```

You should run this script before submitting in order to ensure that you don't lose points for incorrectly formatting your submission! Additional details about the handin script, including how to run and examples, are available here.

This version of the handin script checks that p1-1.asm, p1-2.asm, p1-3.asm, p2-1.asm, and p2-2.asm compile and run correctly.