Computer Sciences Department
University of Wisconsin-Madison
CS/ECE 552 – Introduction to Computer Architecture
Project Description – Extra Credit

The extra credit components should be implemented atop a working 5-stage pipeline design with caches (Phase 3). Without a working baseline for the mandatory project requirement(s), no extra credit points will be awarded.

Extra Credit Components

Up to a total of 10 points (out of a total of 100 points for the course project) can be potentially obtained by successfully implementing different components of extra credit. The description of different components and their corresponding points are listed below. More than 10 points of extra credit options are listed, to provide you with a variety of options. You will be capped at 10 points of extra credit though.

Note: for the extra credit portion, you must still abide by the 552 Verilog Rules and Filenaming Conventions. Moreover, you should implement these optimizations in a separate folder from your phase 3 submission – i.e., keep a working copy of your phase 3 submission separate from this.

Due Date: Your extra credit optimizations should be turned in by 11:59 PM on May 6th on Canvas.

Submission Requirements

1. You are required to submit a pdf document summarizing your extra-credit design and its differences from the baseline in terms of features, benefits and overheads.
2. You are required to develop one or more test cases, which clearly highlight the benefits of your optimized design, and the results should be shown in the submitted document.
3. You are also required to submit a tar'd or zipped file containing: all the Verilog files of your design, all testbenches used and any other support files.

Since there are many different extra credit optimizations, please explain in your accompanying PDF what your directory structure is.

Extra Credit Optimizations

1. **Branch Decisions in Decode (0 – 1 point)**: As discussed in class, doing your decisions for branches in decode will reduce the number of instructions that need to be flushed and likely improve CPI. You will need to update your forwarding, stall, and flush logic accordingly when you make this optimization. After making this optimization, you should see that performance improves for the tests that use mispredicted branches.

2. **Additional Forwarding Paths (0 - 1 point)**: For Phase 3 you are only required to implement forwarding from beginning of the MEM stage to beginning of EX stage (i.e., X → X forwarding) and forwarding from beginning of the WB stage to the beginning of the EX stage (i.e., M → X forwarding). Thus, there are additional forwarding paths you may implement to improve you CPI:

- Forwarding from the beginning of the WB stage to the beginning of the MEM stage (i.e., M → M forwarding)
- (if you do branch decisions in decode) Forwarding from the beginning of the MEM stage to the beginning of the ID stage (i.e., X → D forwarding) and the beginning of the WB stage to the beginning of the ID stage (i.e., M → D forwarding, assuming you are implementing branches in ID)

Implementing each of these forwarding paths is worth ½ point of extra credit, with a max of 1 point.

3. **Cache Replacement Policy (0-1 points for instruction cache, 0-2 points for data cache)**: Increase the performance of your cache design by using LRU (or another) superior replacement policy. To test the improvement, run all of the tests from Phase 3 and compare the CPI to that of your design before this optimization (all of the tests that are expected to pass for Phase 3 should still pass after this optimization).

4. **Employing "critical word first" data reads from memory to cache (0-2 point for instruction cache, 0-3 points for data cache)**: to service waiting requests early, add support for servicing the requested (critical) word first, before the other words on the same cache line. This will require "hit under miss" behavior so that while your FSM is still filling a previous miss, subsequent instruction fetches, loads, and stores can continue as long as they are to different cache blocks. You must check to see if they are to the same cache block and stall in those cases until the cache block is filled.

To test the improvement, run all of the tests from Phase 3 and compare the CPI to that of your design before this optimization (all of the tests that are excepted to pass for Phase 3 should still pass after this optimization).

5. **Dynamic branch prediction (0-2 points):** implement dynamic branch prediction to reduce the misprediction rate of your processor. To test the improvement, run all of the tests from Phase 3 and compare the CPI to that of your design before this optimization (all of the tests that are excepted to pass for Phase 3 should still pass after this optimization).

6. **Exception Handling (0–2 points)**: As discussed in the Microarchitecture Specification, exception handling is extra credit. If you choose not to implement exception handling, an illegal instruction should be treated as a NOP.

`IllegalOp` is the only defined exception in the WISC-SP22 architecture. It is invoked when the opcode of the currently executing instruction is not a recognized member of the ISA. Upon finding an illegal opcode, the computer shall save the current PC into the reserved register EPC and then

load address 0x02, which is the location of the IllegalOp exception handler. Note that if you choose to implement exceptions, address 0x00 must be a jump to the start of the main program.

After implementing exception handling, you should see that the exception handling tests now pass. To verify the correctness of your exception handling, you should run the tests in `/u/s/i/sinclair/public/html/courses/cs552/spring2022/handouts/testprograms/public/exceptions.list`. The exception handler routines in these programs vary in complexity.

You should also write and submit at least one additional test that demonstrates the handling of exceptions. In this program, your exception handler routine does not need to be complex. At a minimum, it should load 0xBADD into R_7 and then call the RTI instruction.

7. **Synthesizing Your Design (0 – 5 points)**: Successfully synthesizing your design using the Synopsys Design Compiler followed by extracting the synthesized netlist and performing functional simulation by rerunning all the test cases on the post-synthesis netlist. See Synthesis and Synthesis FAQ for details. Synthesizing your design by itself is worth 2 points.

- On top of (7), optimizing your design for low area. (0-1 point)
- On top of (7), optimizing your design for high IPC. (0-1 point)
- On top of (7), optimizing your design for high IPS (instructions per second); i.e., optimizing your design for both high IPC and high synthesis frequency. (0-1 point)

8. **Other optimizations (please discuss with instructor/TAs before proceeding) (0-2 points)**

Note: Please talk to the TAs or Matt before embarking on any of these extra-credit components so as to clearly establish requirements of the extra-credit component.