LSU EE 4720 **Homework 5** Solution **Due: 12 April 2021**

☣ Do not hand in paper copies. Instead, E-mail your solution to koppel@ece.lsu.edu. The preferred format is a PDF file.

Problem 1: Look over SPECcpu2017 run and reporting rules, available at `http://www.spec.org/cpu2017/Docs/runrules.html`. Start with Sections 1.1 to 1.5 and read other sections as needed to answer the questions below.

(*a*) Section 1.2.3 of the run and reporting rules lists several assumptions about the tester.

Consider the following testing scenario: The SUT (system being benchmarked) is a new product and that the tester works for the company that developed it. The company spent lots of money developing the product and their potential customers will use SPECcpu2017 when making buying decisions.

☑ Explain why assumptions b and c seem reasonable given the testing scenario above.

It is in the company's interest that benchmark scores are high, so we can safely assume that they choose someone knowledgeable about the SUT (item c) and give that person time and other resources to learn what compilation options and other configuration details can be changed and how to best set them for the SUT (item b). Even if the company intended to submit a misleading benchmark run, they would make sure that the tester new what the rules said.

☑ Explain why assumption d also seems reasonable, given other stipulations set forth in the run and reporting rules (and discussed in class).

We can safely assume honesty on the part of the tester because any dishonesty would quickly be caught. The rules require that any publicly divulged results be accompanied by a config file which can be used to reproduce the results. We can safely assume that the company expects its competitors to buy a system and use the config file to reproduce the results. Further, the company would not expect its competitors to keep quite about any discrepancies.

(*b*) The SPECcpu benchmarks can be prepared at base and peak tuning levels (or builds). These are described in Section 1.5. Section 2.3.1 stipulates that base optimizations are expected to be safe.

☑ What is an unsafe optimization? (Points deducted for irrelevant or lengthy answers, especially if they appear copied.)

An optimization should transform unoptimized code into optimized code which does exactly the same thing, but does so faster, using less energy, using fewer instructions, or realizing some other benefit. An optimization is unsafe if sometimes the transformed code computes different results than the unoptimized code. In practice, compilers do not attempt unsafe optimizations because programmers have enough problems finding their own bugs. The last thing they need to worry about is a bug being introduced by the compiler.

Item e in Section 2.3.1 states that there is evidence an optimization is unsafe if it is used to prepare a run of the benchmarks and the outputs fail to validate. Presumably a company preparing the benchmark run in this scenario would have their compiler people fix the unsafe optimization so that it is correct (validates) on the SPEC benchmarks, but is still unsafe on other code.

Does that mean peak optimizations are unsafe? Does that mean peak results can be obtained with unsafe, don't-try-this-at-home optimizations?

☑ Why would it be bad if peak results were obtained with unsafe optimizations?

Because in that case peak results would not be useful to many people because they could not rely on their code reaching similar performance levels while computing correct results.

☑ What rules ensure that optimizations used to obtain peak results aren't too unsafe?

Section 1.3.2 stipulates that components used to build the benchmarks (especially compilers) be of production quality, rather than something quickly thrown together to complete one job, or a buggy prototype. It would not be very meaningful if the rules required that components be of production quality but went no further. Otherwise when challenged a tester would respond, "we believe that all of the components used to build the benchmarks are of production quality or better." To provide some objective criteria, the rules require that the components are real products. (A non-real product is one that violates the items in 1.3.2. For example, the customer is not aware of the name of the product, the product will not be available for years, is undocumented and there's no support either.) These criteria are not a guarantee that the optimizations will be safe because there are (or have been) companies that sell lousy products.

Problem 2: The illustration below is our familiar 5-stage MIPS implementation with the destination register mux and an immediate mux shown. Modify it so that it is consistent with the RISC-V RV32I version as described below. The modifications should include datapath and labels, but not control logic. For this problem use RISC-V specification 20191213 available at
`https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf`.

The Inkscape SVG source for the image is at
`https://www.ece.lsu.edu/ee4720/2021/hw05-mips-id-mux.svg`. It can be edited with your favorite SVG or plain text editor.

In some ways RISC-V is similar to MIPS, but there are differences. Pay attention to the encoding of the store instructions. Also pay attention to how branch and jump targets are computed.

Be sure to change the following:

- ☑ Bit ranges at the register file inputs.
- ☑ The bit ranges used to extract the immediate.
- ☑ The bit ranges used for the offsets of branch and jump instructions and the hardware used to compute branch and jump targets.
- ☑ The inputs to the destination register mux (which connects to the `dst` pipeline latch).
- ☑ The names used in the pipeline latches.
- ☑ Add or remove unneeded pipeline latches. (Such changes will be needed for branches and jumps.)

Consider the following instructions:

- ☑ Two-register and immediate arithmetic instructions, such as `add` and `addi`.
- ☑ The `lui` instruction (which is similar but not identical to MIPS' lui).
- ☑ Branch instructions as well as `jal` and `jalr`.
- ☑ The load and store instructions. (Only the store instructions will require a change beyond what is required for arithmetic instructions.)

Note:

- Do not show control logic such as logic driving mux select inputs.
- Do not show the logic to decide whether a branch is taken.

The solution diagram appears on the next page, beneath the original MIPS implementation.

A common mistake was including `r31` and some second field in the destination-register mux. RISC-V does not have any assumed destination register, so `r31` is not needed. Also, if an instruction does write a register, then the register is always specified in the `rd` field. (That's why the immediate for store instructions is a special case.)

Another common mistake was using NPC to compute branch and jump targets. In RISC-V branch and jump displacements are added to the PC of the branch or jump.

SVG source at `https://www.ece.lsu.edu/ee4720/2021/hw05-mips-id-mux.svg`.

Solution appears below, beneath the MIPS implementation.