LSU EE 4720 **Homework 2** **Due: 13 February 2026**

Formatted 14:25, 10 February 2026

Collaboration Rules

Each student is expected to complete his or her own assignment. It is okay to work with other students and to ask questions in order to get ideas on how to solve the problems or how to overcome some obstacle (be it a question of MIPS, RISC-V or their assembler syntax, how a part of the problem might be solved, etc.) It is also acceptable to seek out resources for help on MIPS, RISC-V, etc. For this assignment (2026 Homework 2), please read the MIPS and RISC-V documentation *before* seeking out help. Attempt to learn by reading the indicated material.

Problem 1: Solve 2025 Final Exam Problem 5(a) and 5(b) **by hand**. Show all field values, including the `func` and `opcode` fields. You can find MIPS 32 ISA (architecture) manuals linked to the course references Web page. Solving *by hand* means to solve the problem by looking up the instructions in the respective ISA (architecture) manuals and figuring out what to put in the fields. Do not "solve" the problem by entering the code into a file and, say, using the SPIM simulator to view encoded output. Also do not "solve" the problem by prompting an LLM.

There's more problems on the next page.

Problem 2: First, familiarize yourself with RISC-V by reading Chapter 1 of Volume I of the RISC-V specification, especially the Chapter 1 Introduction and Sections 1.1 and 1.3. Skip Section 1.2 unless you are comfortable with operating system and virtualization concepts. Other parts of Chapter 1 are interesting but less relevant for this problem. The questions in this assignment are about RISC-V RV32I. Also read the material in Chapter 2 relevant to this question. **Solve this question by reading the material in Chapter 2 and other parts of the RISC-V specification, not by other means.** Test questions will be written for students that have read these chapters. Chapter 2 provides the format and details of base RV32I instructions, but it does not provide opcode and related values. Those can be found in the chapter titled "RV32/64G Instruction Set Listing," which is Chapter 36 in the May 2025 version of the spec (it was Chapter 34 in 2024).

(*a*) Show the corresponding RISC-V instruction for the MIPS code below. *Hint: The mnemonic is not* `sll`*, but it's close.*

☐ Show the corresponding RISC-V instruction.

☐ Show encoding of the RISC-V instruction. ☐ Label fields, and show specific values. to the format that this particular instruction uses.

```
sll r9, r8, 7
```

(*b*) Convert the MIPS I code fragment below to RISC-V RV32I. Pay attention to differences between MIPS and RISC-V branch and load-upper-immediate instructions.

☐ Convert fragment from MIPS I to RISC-V RV32I. ☐ Note the difference in RISC-V branch behavior. ☐ Reduce the number of instructions where possible.

```
 bne r5, r6, SKIP
 sw r4, 5(r6)
 lui r8, 0x5678
 addi r8, r8, 0x4000
SKIP:
 add r9, r8, r10
```

☐ Show the encoding of the RISC-V branch instruction used in the converted code fragment. Pay attention to the branch displacement (called and offset in the RISC-V documentation).

There's another problem on the next page.

Problem 3: In a skeptical mood I prompted ChatGPT 5.2 with: "Is the RISC zero register overrated?" It responded, in part:

```
Hot take: **the RISC zero register is neither overrated nor
sacred**its a very pragmatic hack that aged surprisingly well, but
its not universally best.

Lets unpack it like engineers, not cultists :-)

### 1. Instruction set simplification

You get a bunch of operations for free without extra opcodes:

* ‘move rd, rs‘ → ‘add rd, rs, x0‘
* ‘clear rd‘ → ‘add rd, x0, x0‘
* ‘neg rd, rs‘ → ‘sub rd, x0, rs‘
* Compare against zero without loading a literal

Thats real silicon and encoding savings, especially in early RISC
designs where every opcode bit mattered.
```

Yes, it's important to reduce the number of opcodes. But is that really a good list of pseuodinstructions that can be used by ISAs with a zero register so that they don't need to have real instructions, such as `move`, that would use a valuable opcode?

(*a*) In MIPS-I, which of the first three pseuodinstructions in the answer above (`move`, `clear`, `neg`), can be done *without* a zero register?

☐ If possible, show MIPS-I instruction without a zero register that can be used for `move`, `clear`, and `neg`. ☐ Don't just put a zero into some register and use that.

(*b*) The fourth "compare-against-zero" bullet might have referred to conditional branches. Consider mnemonics `bnez` (branch not equal to zero) and `bgtz` (branch greater than zero).

☐ Are `bnez` and `bgtz` examples of operations that one gets for free in MIPS? The word *free* is in the sense used by ChatGPT 5.2 in the `You get a bunch of operations "for free"` sentence. Answer for ☐ `bnez` and for ☐ `bgtz`.