LSU EE 4720 **Homework 3** **Due: 25 February 2026**

Solution *Formatted 16:09, 2 March 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 or assembler syntax, interpreting error messages, how a part of the problem might be solved, etc.) It is also acceptable to seek out assembly language resources for help on MIPS, etc. It is okay to make use of AI LLM tools such as ChatGPT and Copilot to generate sample code. (Do not assume LLM output is correct. Treat LLM output the same way one might treat legal advice given by a lawyer character in a movie: it may sound impressive, but it can range from sage advice to utter nonsense.)

After availing oneself to these resources **each student is expected to be able to complete the assignment alone**. Test questions will be based on homework questions and **the assumed time needed to complete the question will be for a student who had solved the homework assignment on which it was based**.

Student Expectations

Some of the problems require thought (the one on page 3 of the final exam), and students are expected to persevere until they find a solution. It is each student's duty to him or herself to resolve frustrations and roadblocks quickly, hopefully helped along by the satisfaction of making progress. There are plenty of old problems and solutions to look at. One way to resolve issues is to ask Dr. Koppelman or others for help.

Resources

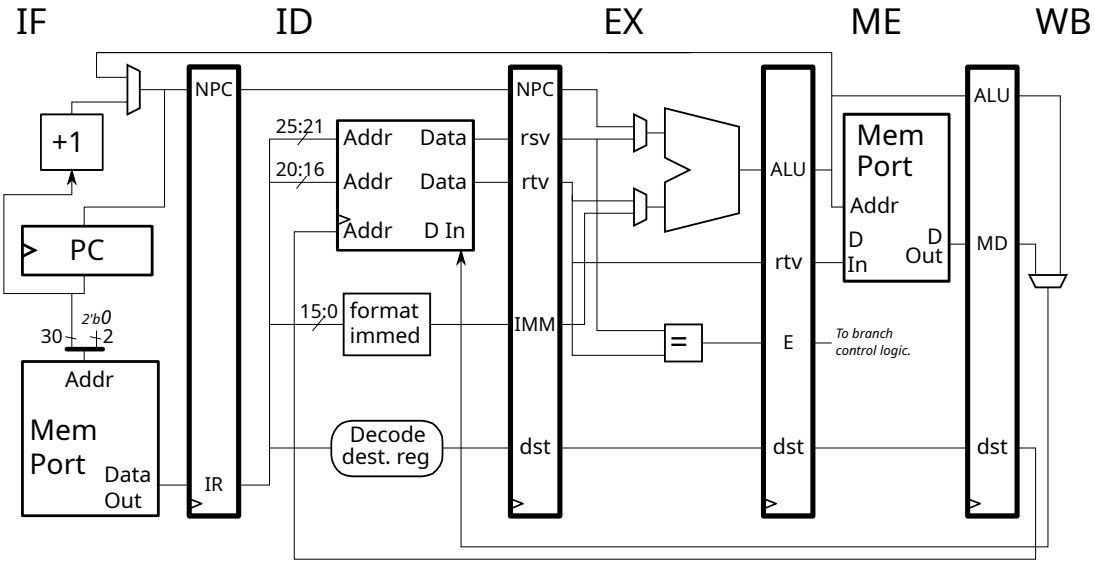
For examples of pipeline execution diagrams of given code fragments running on given MIPS implementations see past midterm exams (and final exams, but mostly midterms). The solutions to almost all past midterms in this course are available. A good place to start would be 2023 Midterm Exam Problem 2, 3, 4, and 5.

Problem 1: *Solve this problem* **after** *Problem 2. It appears before Problem 2 so that you don't somehow forget it.* Complete the first three parts 2025 Final Exam Problem 1, which asks for pipeline execution diagrams of MIPS implementations. One of those MIPS implementations is different than others covered in the past. Solve the parts on page 2, 3, and 4. Do not solve the part with the superscalar implementation (multiple ALUs in `EX`) on page 5.

See the posted final exam solution.

There is another problem are on the next page.

Problem 2: *Note: The following problem was assigned in all but one of the last nine years, and its solution is available. DO NOT look at the solution unless you are lost and can't get help elsewhere. Even in that case just glimpse.* Appearing below are **incorrect** executions on the **illustrated implementation**. For each one explain why it is wrong and show the correct execution.



(*a*) Explain error and show correct execution.

```
# Cycle           0  1  2  3  4  5  6  7
lw r2, 0(r4)      IF ID EX ME WB
add r1, r2, r7       IF ID EX ME WB
```

The `add` depends on the `lw` through `r2`, and for the illustrated implementation the `add` has to stall in `ID` until the `lw` reaches `WB`.

```
# Cycle           0  1  2  3  4  5  6  7     SOLUTION
lw r2, 0(r4)      IF ID EX ME WB
add r1, r2, r7       IF ID ----> EX ME WB
```

(*b*) Explain error and show correct execution.

```
# Cycle           0  1  2  3  4  5  6  7
add r1, r2, r3    IF ID EX ME WB
lw r1, 0(r4)         IF ID -> EX ME WB
```

There is no need for a stall because the `lw` writes `r1`, it does not read `r1`.

```
# Cycle           0  1  2  3  4  5  6  7  SOLUTION
add r1, r2, r3    IF ID EX ME WB
lw r1, 0(r4)         IF ID EX ME WB
```

(*c*) Explain error and show correct execution.

```
# Cycle          0  1  2  3  4  5  6  7
add r1, r2, r3   IF ID EX ME WB
sw r1, 0(r4)        IF ID -> EX ME WB
```

A longer stall is needed here because the **sw** reads **r1** and it must wait until **add** reaches **WB**.

```
# Cycle          0  1  2  3  4  5  6  7  SOLUTION
add r1, r2, r3   IF ID EX ME WB
sw r1, 0(r4)        IF ID ----> EX ME WB
```

(*d*) Explain error and show correct execution.

```
# Cycle          0  1  2  3  4  5  6  7
add r1, r2, r3   IF ID EX ME WB
xor r4, r1, r5      IF ----> ID EX ME WB
```

The stall above allows the **xor**, when it is in **ID**, to get the value of **r1** written by the **add**; that part is correct. But, the stall starts in cycle 1 *before* the **xor** reaches **ID**, so how could the control logic know that the **xor** needed **r1**, or for that matter that it was an **xor**? The solution is to start the stall in cycle 2, when the **xor** is in **ID**.

```
# Cycle          0  1  2  3  4  5  6  7   SOLUTION
add r1, r2, r3   IF ID EX ME WB
xor r4, r1, r5      IF ID ----> EX ME WB
```