LSU EE 4720 **Homework 3** **Due: 7 March 2025**

Formatted 18:17, 21 March 2025

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, 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.

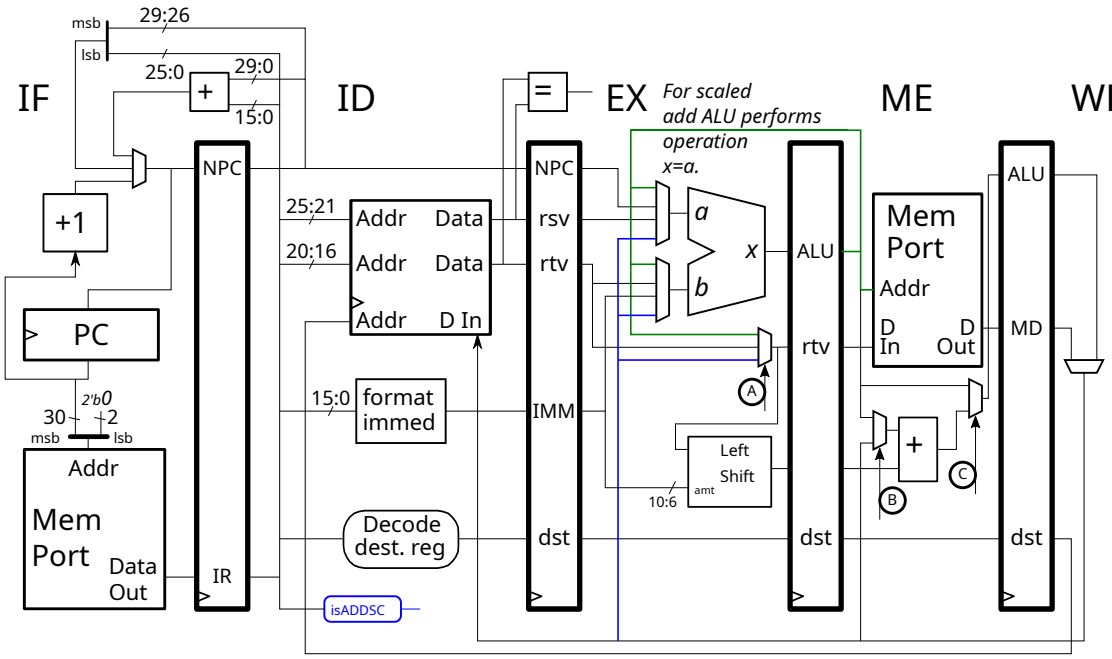
Resources

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.

Homework Background

This assignment asks about hypothetical MIPS instruction `addsc` (scaled addition) that was the subject of 2014 Homework 3 Problem 3. See that assignment and its solution for a description of the `addsc` instruction.

Problem 1: Appearing below is a solution to 2014 Homework 3 Problem 3, though not the same as the posted solutions. Three of the multiplexors have labels on their select signals: A, B, and C.



The incomplete pipeline execution diagram below shows the progress of instructions through the implementation and also the value of the select signals A, B, and C in some cycles. If a select signal value is blank, such as C in cycle 5, then its value does not matter. For example, execution would be correct whether $C = 0$ or $C = 1$ in cycle 5, and so it is blank.

☐ Fill in instructions, including at least one `addsc`, that could have resulted in the execution. ☐ Take care to choose registers so that dependencies and ☐ the use of bypass paths are consistent with the select signal values.

```
# Cycle                    0  1  2  3  4  5  6  7
A                                  0  2  2
B                                     1     0
C                                  0  1     1
# Cycle                    0  1  2  3  4  5  6  7

                           IF ID EX ME WB


                              IF ID EX ME WB

# Cycle                    0  1  2  3  4  5  6  7

                                 IF ID EX ME WB


                                    IF ID EX ME WB

# Cycle                    0  1  2  3  4  5  6  7
```

Problem 2: Consider the *load/use* stall in the execution of the code below on an ordinary MIPS implementation (one without `addsc`):

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

(*a*) Suppose that instead of the code above the assembly code were generated by a compiler that **is aware** of the `addsc` instruction and run on an implementation that implements `addsc`.

☐ Explain how the compiler could avoid the stall.

(*b*) Suppose instead that the original code (at the beginning of the problem) is run on an implementation which includes `addsc` and where `addsc` was encoded (choice of opcode, register fields, etc.) to avoid such stalls. (This could be the same implementation as the previous part.)

☐ Explain how such a stall could be avoided on the original code, with the `add`, by the design of the encoding of `addsc`.

There's another problem on the next page.

Problem 3: Design the following control logic. Some of the logic will need the isADDSC logic block in ID, which detects whether an `addsc` instruction is in ID. An SVG of the diagram can be found at `https://www.ece.lsu.edu/ee4720/2025/hw03-scadd.svg`. It can be edited by Inkscape or any other SVG editor, and by plain-text editors for those who are so disposed.

- ☐ Design control logic for select signal C. *Note: This is easy.*
- ☐ Design control logic for select signal B.
- ☐ Show control logic generating a stall signal for the stalls like those shown in the diagram below.

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

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