LSU EE 4720 **Homework 4** **Due: 6 March 2026**

Solution *Formatted 16:05, 15 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, RISC-V, or assembler syntax, interpreting error messages, 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. It is okay to make use of AI LLM tools such as ChatGPT, Claude, 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

To solve this assignment you are expected to avail yourself of references provided in class and on the Web site, and to learn how to handle references that are at first hard to understand, and to keep looking (and asking) when the answer isn't in the first place you look. Some of the problems require thought, and you are expected to persevere until you find a solution. It is each student's duty to him or herself to resolve frustrations and roadblocks, 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.

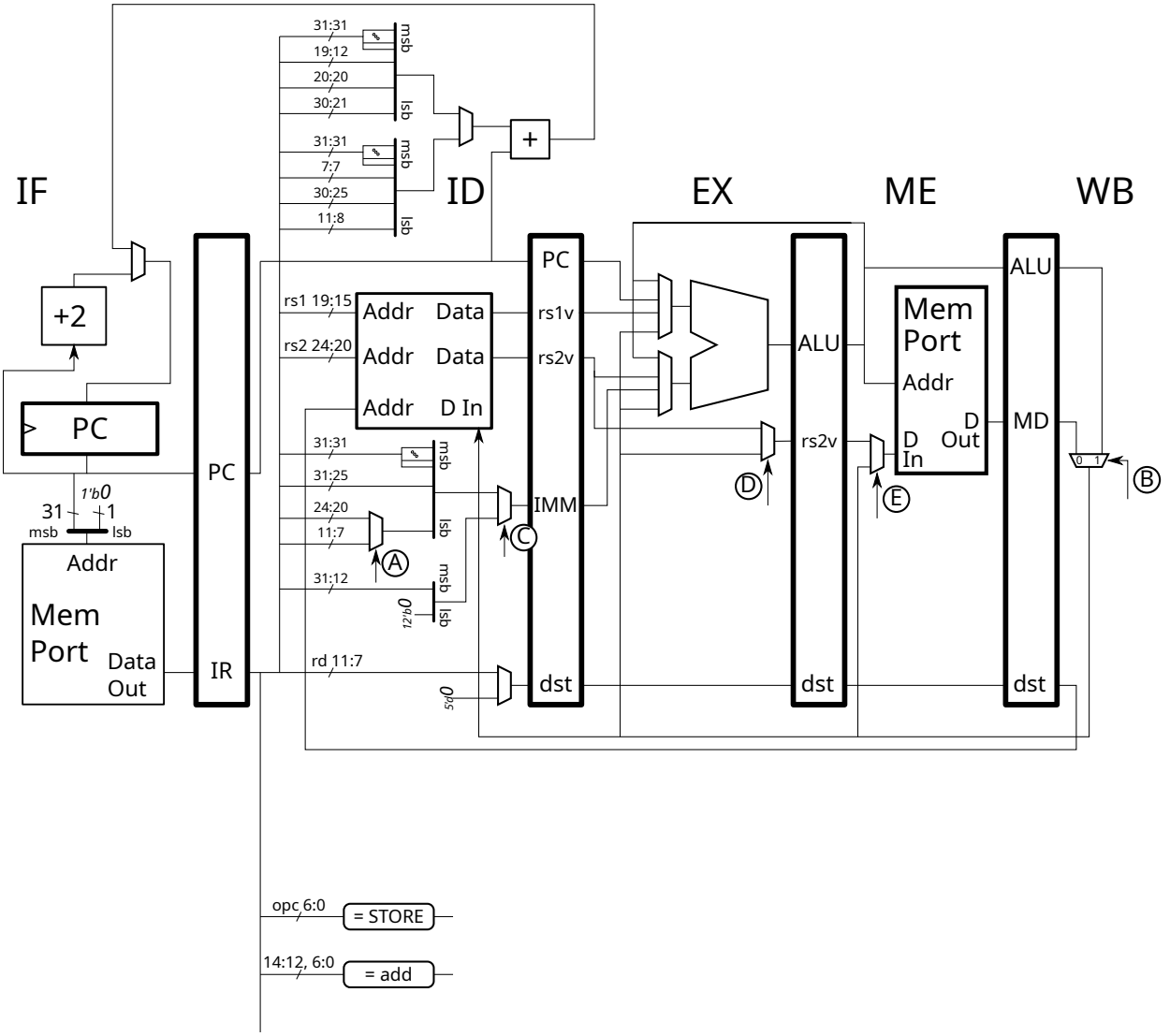
Resources

Many past exams have multiplexor-select-signal design problems, the type of question asked in Problem 2. See, for example, 2024 midterm exam Problem 2. The RISC-V specification is linked to the course site at `https://www.ece.lsu.edu/ee4720//doc/riscv-spec-20250508.pdf` and can be found from the RISC-V site starting at `https://riscv.org/specifications/ratified/`.

Problem 1: Solve 2023 Homework 3 Problem 2, which asks for the select signal values on some multiplexors in a MIPS implementation. Try to solve this problem without looking at the solution. There may be a similar problem on this semester's midterm. Solve the 2023 problem before attempting the next question on this assignment.

Another problem starts on the next page.

Problem 2: Appearing below (and on the last page) is a five-stage implementation of RISC-V RV-32i in which some multiplexor select inputs are labeled. The labels range from A to E. In the subproblems below design control logic for the indicated select signals. An SVG of the illustration can be found at `https://www.ece.lsu.edu/ee4720/2026/hw04-rv-1.svg`, you can use your favorite SVG editor, even if it's not Inkscape, to prepare solutions.



The control logic needs to account for the type of instruction or the exact instruction that's present. There is already control logic that detects store instructions, that's $\boxed{=\text{STORE}}$, and that detects an `add` instruction, that's $\boxed{=\text{add}}$. Note that instruction type STORE refers to many store instructions such as `sw` and `sb`, while `add` just refers to the 3-register `add` instruction. To solve the problems you may need to add additional instruction detection logic. Don't make up instruction types, instead look up the types in the RISC-V specification.

All of the multiplexors for which logic is to be designed have two inputs, making the logic design easier. Because there are two inputs, all one needs to do is detect when one of those inputs is needed and so one only needs to design logic for the easier of the two. For example, for select signal B the 0 input is only used for load instructions, while the 1 input is used for other instructions

that write an integer register. So, use the 0 input if there's a load in `WB` and the 1 input if not. There is no need to consider all of the instructions that could use the 1 input.

Regardless of which stage the select signals are in, the control signals must be computed in `ID` and if necessary sent through the pipeline.

(a) Look at Chapter 2 of the RISC-V specification to figure out which instructions affect signals A and C, and of course take into account the illustrated hardware.

☑ Provide an example of an instruction for which the A signal must be 0 (upper input):

Any type I that uses an immediate, such as `addi r1, r2, 3`.

☑ Provide an example of an instruction for which the A signal must be 1 (lower input):

Any store instruction, such as `sb r1, 0(r2)`.

☑ Provide an example of an instruction for which the C signal must be 1 (lower input):

Either an `lui` or `auipc`, such as `lui r1, 0x1234`.

(b) Design control logic for select signals A, B, and C.

☑ Logic for signals ☑ A, ☑ B, and ☑ C.

☑ The logic must be in `ID`, ☑ pass the select signals through the pipeline when necessary.

☑ Don't forget that the implementation is pipelined and that each stage will be occupied by a different instruction.

The solution appears below (a page or two ahead) in blue.

Signal A was the easiest because the lower mux input was only used by store instructions, and so it could be connected directly to the =STORE box. For type I instructions we need $A = 0$, but since stores are not type I no additional logic is necessary. But what if some other type of instruction is present in `ID`, such as type R? In that case it doesn't matter because for type R the output of the A must is ultimately ignored.

For C the lower input is used by just two instructions, the logic checks for both of them, `lui` and `auipc`.

For B we need to check for a load instruction, since that's the only type of instruction that uses the output of the memory port. The =LOAD checks for a load, and the value is sent down the pipeline to the B mux. Note that the inputs to the B mux were renumbered, now 1 is on the left. It would also be possible to keep the original numbering and use an inverter.

(*c*) Signals D and E are used for bypassing store values. These bypasses should enable the executions shown below (that is, without stalls):

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

# Cycle         0  1  2  3  4  5       Example 2
lw r1, 4(r3)    IF ID EX ME WB
sw r1, 8(r2)       IF ID EX ME WB

# Cycle         0  1  2  3  4  5  6    Example 3
xor r1, r3, r4  IF ID EX ME WB
addi r2, r2, 4     IF ID EX ME WB
sw r1, 8(r2)          IF ID EX ME WB
```

☑ Design control logic for select signals D and E.

☑ The logic must be in `ID`, ☑ pass the select signals through the pipeline when necessary.

☑ Because these multiplexors are only used for stores and have just two inputs the logic will be simpler than the examples shown in the class notes for the ALU multiplexor.

☑ Don't forget that the implementation is pipelined and that each stage will be occupied by a different instruction.

The solution appears below in purple.

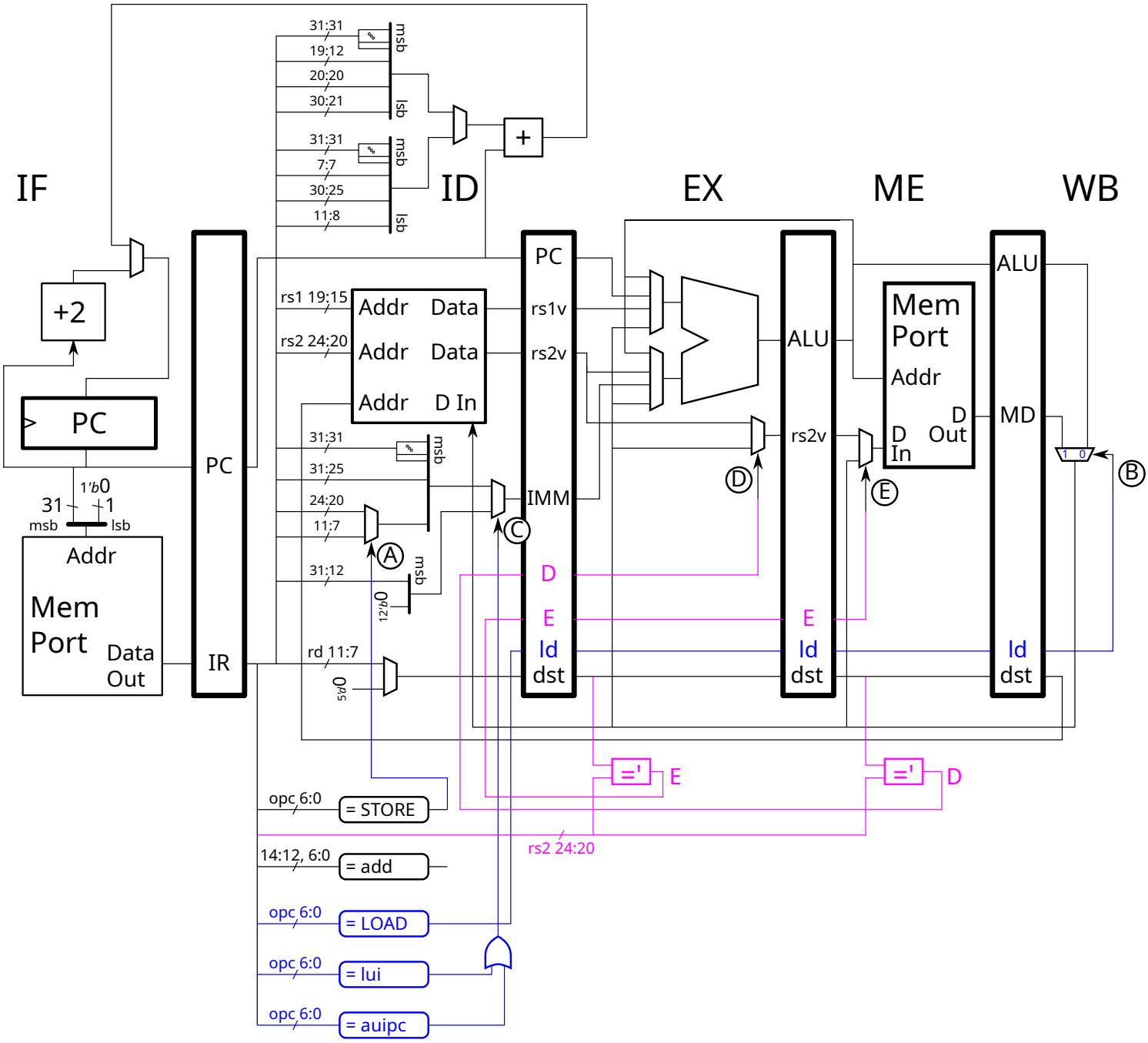
Signal D bypasses from `WB` to `EX`, where there should be a store. A situation where D is used occurs in cycle 4 in example 3. The control logic computes D one cycle earlier, in cycle 3. So, in cycle 3 we need to check if the register being stored, the `rs2` register (`r1` in the example), matches the destination of the instruction in `ME`. The $=$' labeled D makes this comparison. Remember that the output of $=$' is `1` iff the two inputs are equal and not zero. The computed value of D is placed in the pipeline where it will be used in the next cycle.

Signal E bypasses from `WB` to `ME`, where there should be a store. A situation where E is *used* occurs in cycle 4 of both example 1 and example 2. The logic in `ID` must check for a match between the `rs2` register and the instruction in `EX`.

Grading Notes: A serious mistake is to not send signals B, D, and E through the pipeline. Don't forget that an instruction is in only one stage at a time and that each stage has a different instruction.

Another error was to check the destination in the wrong stage. The pipeline diagrams (examples 1, 2, and 3) are there to make it easier to identify where dependent instructions are.

Many students would unnecessarily set D and E to false if there was not a store in `ID`. Because those bypasses are *only* used by stores it does not matter what D and E are when some other instruction is in `ID` and so that extra logic is not needed.


IF
ID
EX
ME
WB
+2
PC
PC
IR
Addr
Mem
Port
Data
Out
1'b0
31
1
msb
lsb
31:31
19:12
20:20
30:21
31:31
7:7
30:25
11:8
msb
lsb
+
rs1 19:15
rs2 24:20
Addr Data
Addr Data
Addr D In
PC
rs1v
rs2v
IMM
D
E
ld
dst
31:31
31:25
24:20
11:7
31:12
msb
12'b0
rd 11:7
5'b0
A
C
D
E
ALU
rs2v
E
ld
dst
Mem
Port
Addr
D
D Out
In
ALU
MD
ld
dst
1 0
B
=' E
=' D
opc 6:0
= STORE
14:12, 6:0
= add
opc 6:0
= LOAD
opc 6:0
= lui
opc 6:0
= auipc
rs2 24:20