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 final exams and homework assignments have problems on FP pipeline variations and control logic.

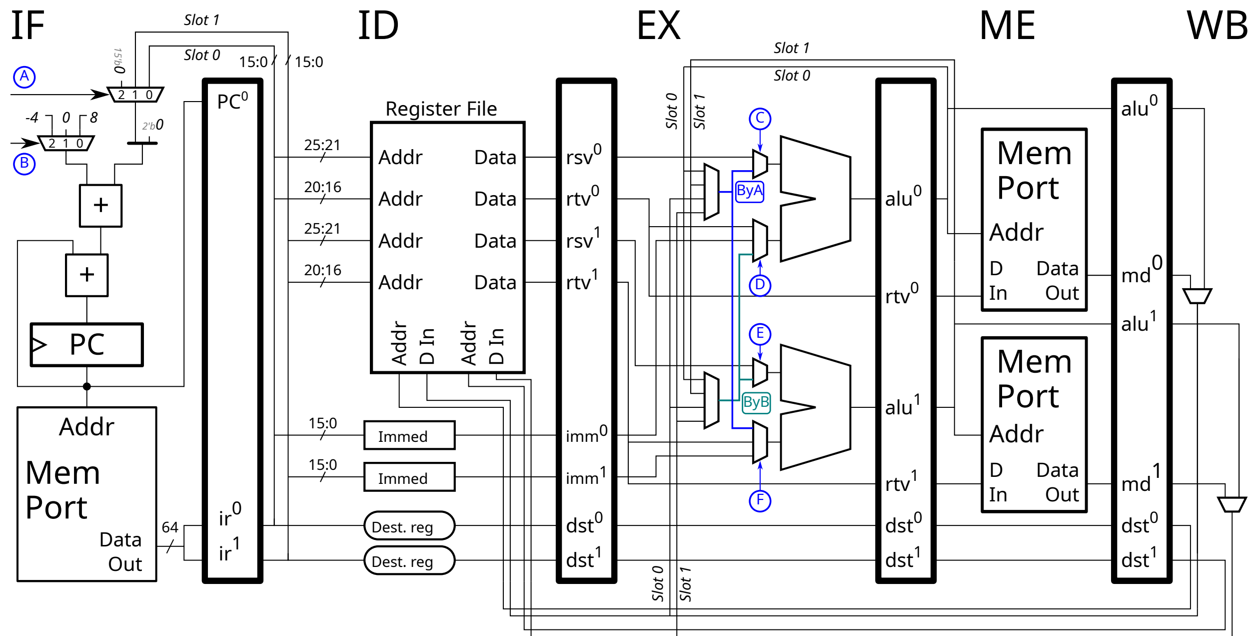
Problems start on the next page.

Problem 1: Solve 2024 Final Exam Problem 2 (both parts). Part (a) asks for code that will use specific bypass connections on a 2-way superscalar MIPS implementation, and part (b) asks for the execution on a 4-way superscalar implementation. Solve the problem without looking at the solution, then look at the solution to check your work. For an example of superscalar execution see the solution to 2023 Final Exam Problem 1c.

Problem 2: Solve the last part of 2025 Final Exam Problem 1, which asks for the execution of code on a four-way superscalar MIPS implementation. The exam illustration has been changed to emphasize hardware computing the branch target. In the original exam the control logic was labeled "Magic Cloud", but that has been changed to just "Control Logic." I will continue to poke fun at the assumption of feasibility of labeled ovals, but in this particular case I want to make clear that the branch control logic is not magic (or to use the computer science theory term, it is not an oracle). In particular the control logic can only compute outputs based on the instructions in `ID`, it can't figure out what instructions are currently being fetched in `IF` (it is not a branch predictor). A common mistake was forgetting that for the illustrated hardware when a branch is in `ID` the next group of instructions are already being fetched in `IF` and that if the branch is taken those must be squashed.

There is another problem on the next page.

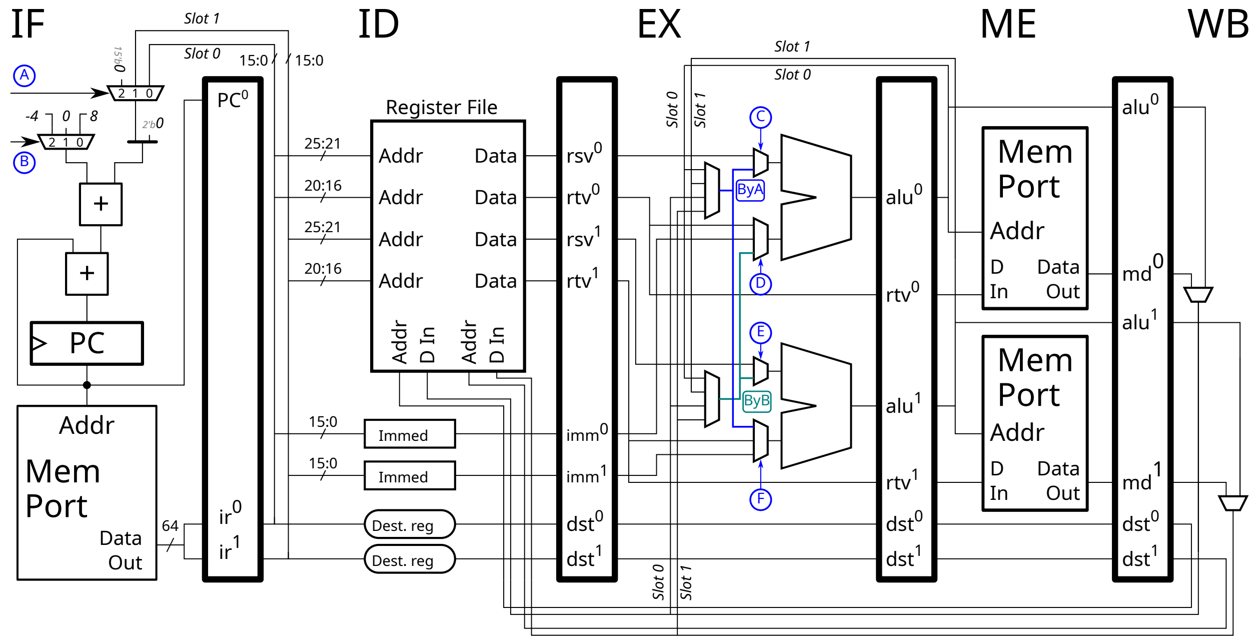
Problem 3: Debuting below is the Lite [tm] 2-way superscalar MIPS implementation. It is different in several ways from the one that we use in class. A larger version can be found on the last page of this assignment, the SVG source is at `https://www.ece.lsu.edu/ee4720/2026/hw06-ss2-lite.svg`.



(*a*) In Lite the cost of bypass hardware has been reduced, and as a result some bypasses that would work on our usual 2-way superscalar won't work here. Bypassed values are carried by wires labeled ByA and ByB. An execution on this hardware appears below.

☐ Show the values that will appear on select signals C, D, E, and F (all in `EX`) in cycles 3 and 4 for the execution below.

```
# Cycle          0    1    2    3    4    5    6
add r1, r12, r13 IF   ID   EX   ME   WB
sub r2, r14, r15 IF   ID   EX   ME   WB
and r3, r11, r10      IF   ID   EX   ME   WB
ori r4, r16, 17       IF   ID   EX   ME   WB

xor r6, r1, r3             IF   ID   EX   ME   WB
or  r7, r3, r1             IF   ID   EX   ME   WB
# Cycle          0    1    2    3    4    5    6

C

D

E

F

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



In the execution on the previous page it was possible to bypass the values of `r1` and `r3` to each instruction that needed them. It should be obvious that Lite can't bypass three or more different registers, such as `r1`, `r2`, and `r3`. But Lite sometimes can't even bypass two registers if one or both are used more than once. Complete the code so that it shows two examples of this. The executions below show a stall because the bypass (when correctly solved) is not possible.

☐ Add registers to the last two instructions in each of the **two** code fragments so that `xor` and `or` use only values of `r1` and `r3` but for which bypasses are not possible.

```
# Cycle          0    1    2    3    4    5    6
add r1, r12, r13 IF   ID   EX   ME   WB
sub r2, r14, r15 IF   ID   EX   ME   WB
and r3, r11, r10      IF   ID   EX   ME   WB
ori r4, r16, 17       IF   ID   EX   ME   WB

xor r6,                    IF   ID   EX   ME   WB
or  r7,                    IF   ID   ->   EX   ME   WB
# Cycle          0    1    2    3    4    5    6
```

```
# Cycle          0    1    2    3    4    5    6
add r1, r12, r13 IF   ID   EX   ME   WB
sub r2, r14, r15 IF   ID   EX   ME   WB
and r3, r11, r10      IF   ID   EX   ME   WB
ori r4, r16, 17       IF   ID   EX   ME   WB

xor r6,                    IF   ID   EX   ME   WB
or  r7,                    IF   ID   ->   EX   ME   WB
# Cycle          0    1    2    3    4    5    6
```

(*b*) The hardware used to compute the PC in Lite is different (more complete) than our usual superscalar implementations. When there is no branch in ID select signal A is set to 2 and B is set to 0. (The code fragments below were taken from 2018 Final Exam Problem 1c, which also asks for more complete 2-way superscalar branch hardware. The comments were written for that problem, but should help in figuring out this problem.)

☐ Show the values of A and B during cycles 0 and 1 in each of the two fragments below.

```
 # Example: Branch in slot 0. Target is TARG = slot_0_pc + 4 + imm0 * 4 = 0x1814
 # Cycle                    0  1  2  3  4  5  6
 0x1000: beq r1, r2, TARG   IF ID EX ME WB       # slot 0 pc = 0x1000
 0x1004: add r4, r5, r6     IF ID EX ME WB       # imm = ( 0x1814 - 0x1004 ) / 4
 0x1008: sub r7, r8, r9        IFx               #     = 0x810 / 4 = 0x204
 0x100c: or  r10, r11, r12     IFx               # TARG = 0x1000 + imm*4 + 4
# Cycle                     0  1  2  3  4  5  6  #      = 0x1000 + 0x204*4 + 4
TARG:                                            #      = 0x1000 + 0x810 + 4
 0x1814: lw r10, 0(r11)           IF ID EX ME WB #      = 0x1814
# Cycle                     0  1  2  3  4  5  6

A

B

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

```
 # Example: Branch in slot 1. Target is TARG = slot_0_pc + 8 + imm1 * 4 = 0x1814
 # Cycle                    0  1  2  3  4  5  6
 0x1000: xori r14, r14, 5   IF ID EX ME WB       # slot 0 pc = 0x1000
 0x1004: beq r1, r2, TARG   IF ID EX ME WB       # imm = ( 0x1814 - 0x1008 ) / 4
 0x1008: add r4, r5, r6        IF ID EX ME WB    #     = 0x80c / 4 = 0x203
 0x1008: sub r7, r8, r9        IFx               # TARG = 0x1000 + imm*4 + 8
 # Cycle                    0  1  2  3  4  5  6  #      = 0x1000 + 0x203*4 + 8
TARG:                                            #      = 0x1000 + 0x80c + 8
 0x1814: lw r10, 0(r11)           IF ID EX ME WB #      = 0x1814
# Cycle                     0  1  2  3  4  5  6

A

B

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

A larger version of Lite appears below the SVG source is at `https://www.ece.lsu.edu/ee4720/2026/hw06-ss2-lite.svg`.