

*An EPS version of the MIPS FP implementation used in some of the problems below can be found at [http://www.ece.lsu.edu/ee4720/2015/mpipei\\_fp.eps](http://www.ece.lsu.edu/ee4720/2015/mpipei_fp.eps) and an easy-to-edit Inkscape SVG version can be found at [http://www.ece.lsu.edu/ee4720/2015/mpipei\\_fp.svg](http://www.ece.lsu.edu/ee4720/2015/mpipei_fp.svg).*

**Problem 1:** Solve 2014 Midterm Exam Problem 2, which asks for a stall-in-ME version of our floating-point pipeline. A solution to this problem is available but use it only if you are stuck, and after you are finished to check your answer. If you got it wrong, then solve the problem again without looking at the solution.

**Problem 2:** Solve 2014 Final Exam Problem 1, which asks for an execution diagram of code running on the solution to the 2014 Midterm Problem 2.

*There's another problem on the next page.*

**Problem 3:** In the FP implementation on the next page (which is the same as the one used in class) an `add.s` instruction can stall due to an earlier `mul.s`, see the example below.

```
# Execution of code on the illustrated implementation.
# Cycle          0  1  2  3  4  5  6  7  8  9
mul.s f0, f1, f2  IF ID M1 M2 M3 M4 M5 M6 WF
add.s f6, f7, f8      IF ID A1 A2 A3 A4 WF
add.s f3, f4, f5          IF ID -> A1 A2 A3 A4 WF
and r6, r7, r8          IF -> ID EX ME WB
```

To avoid the stall consider the *fpa-4/6* design in which an `add.s` instruction that would stall taking the usual route instead enters the FP pipeline at the M1 unit. Assume that the M1 unit's control signal (not shown and not part of the problem) will command it to pass the values at its inputs to its outputs unchanged when it is carrying `add.s` operands. Then at the appropriate time it crosses over to A1 and continues through the remaining adder stages. An `add.s` not facing a WF structural hazard stall would go from ID to A1, as in the usual design. See the execution below.

```
# Desired execution on the fpa-4/6 implementation.
# Cycle          0  1  2  3  4  5  6  7  8  9 10
mul.s f0, f1, f2  IF ID M1 M2 M3 M4 M5 M6 WF
add.s f6, f7, f8      IF ID A1 A2 A3 A4 WF      # Uses 4-stage (normal) path.
add.s f3, f4, f5          IF ID M1 M2 A1 A2 A3 A4 WF  # Uses 6-stage (M1 M2..) path.
and r6, r7, r8          IF ID EX ME WB
```

(a) Modify the pipeline to implement *fpa-4/6*.

- Show the datapath for the operands crossing from the multiply to the add unit.
- Show the control logic. The control logic should only send `add.s` into M1 if it would stall taking the usual route.
- The control logic should include the `we`, `fd`, and `xw` signals, and signals for any multiplexors that you add.
- As always, pay attention to cost and critical path.

(b) In the code fragment above the `add.s f3` goes from ID to M1. If it had gone from ID to M2 it would have still avoided the WF hazard and it also would have finished one cycle earlier. Consider an *fpa-4/5/6* design in which an `add.s` can start at A1, M2, or M1, using the first one that avoids a stall. Provide a code example that would finish sooner on an *fpa-4/5/6* design than on an *fpa-4/6* design. *Hint: A correct answer can add just one more instruction to the code fragment above.*

(c) Is the *fpa-4/5/6* design better than the *fpa-4/6* design? Justify your answer using reasonable cost estimates and made-up properties of typical user programs. Either yes or no is correct, credit will be given for the justification.

