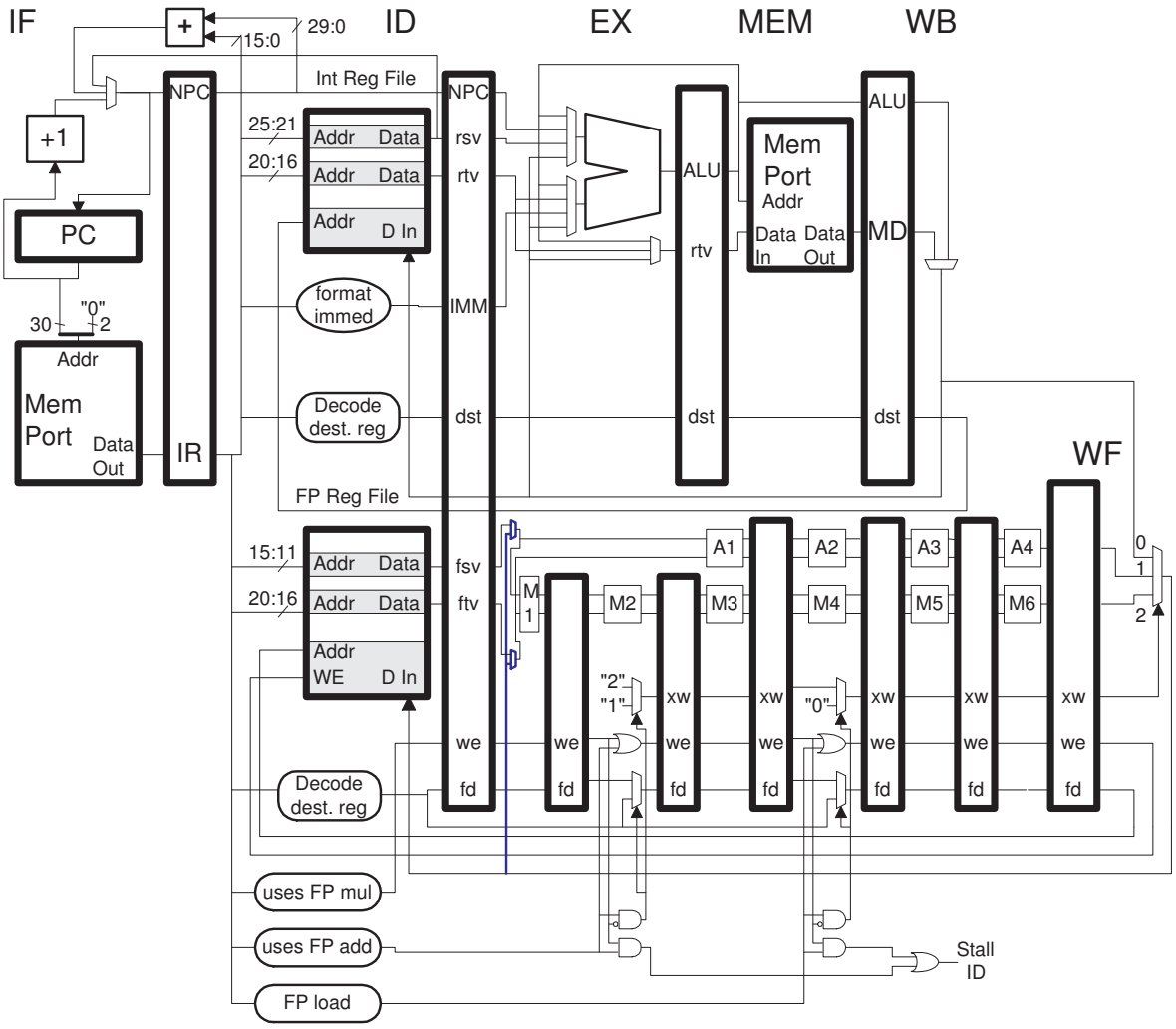
LSU EE 4720 **Homework 3** Solution **Due: 19 April 2010**

Problem 1: The code below executes on the illustrated MIPS implementation. Assume that any reasonable bypasses needed for the FP operands are available, even though they are not shown in the illustration. A bypass is reasonable if it does not have a significant impact on clock frequency and if it does not use circuitry that can predict the future.



```
# SOLUTION
LOOP:          #    0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
 ldc1 f0, 0(r1)     IF ID EX ME WF
 mul.d f2, f0, f4      IF ID -> M1 M2 M3 M4 M5 M6 WF
 add.d f6, f6, f2         IF -> ID -------------> A1 A2 A3 A4 WF
 bne r1, r2, LOOP               IF -------------> ID EX ME WB
 addi r1, r1, 8                                   IF ID EX ME WB
#                   Second Iteration Below
LOOP:          #    0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20 21 22
 ldc1 f0, 0(r1)                                      IF ID EX ME WF
 mul.d f2, f0, f4                                       IF ID -> M1 M2 M3 M4 M5 M6 WF
 add.d f6, f6, f2                                          IF -> ID -------------> A1 A2
 bne r1, r2, LOOP                                                IF -------------> ID EX
 addi r1, r1, 8                                                                    IF ID
LOOP:          #    0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20 21 22
#                   Third Iteration Below
 ldc1 f0, 0(r1)                                                                       IF
```

(*a*) Show a pipeline execution diagram covering enough iterations to compute the CPI. Don't forget to check code for dependencies.

Solution appears above. The state of the pipeline is the same at the start of the second and third iteration (cycles 11 and 22) and so the second iteration can be used to compute CPI. . ..

(*b*) Compute the CPI.

. . . which is $\frac{22-11}{5}$.

(*c*) Remember, that some bypass paths are assumed present though not illustrated. Add the needed paths to the implementation and show when they are used.

The bypass paths are from `WF` to the `M1` and `A1` inputs, they appear in blue. They are used by the `ldc1` bypassing to the `mul.d` and `mul.d` bypassing to the `add.d`.

Problem 2: Precise exceptions are necessary for integer instructions, but only Nice To Have for floating-point instructions. Suppose exception conditions, such as overflow, were detected in `A4` and `M6` in the pipeline from the previous problem.

```
# Pipeline diagram for solution.
# Cycle:           0  1  2  3  4  5  6  7  8  9  10 11 12
mul.d f2, f0, f4   IF ID M1 M2 M3 M4 M5 M6 WF
add.d f6, f6, f2      IF ID ------------> A1 A2 A3 A4 WF
and r3, r3, r5           IF ------------> ID EX ME WB
addi r1, r1, 8                            IF ID EX ME WB
```

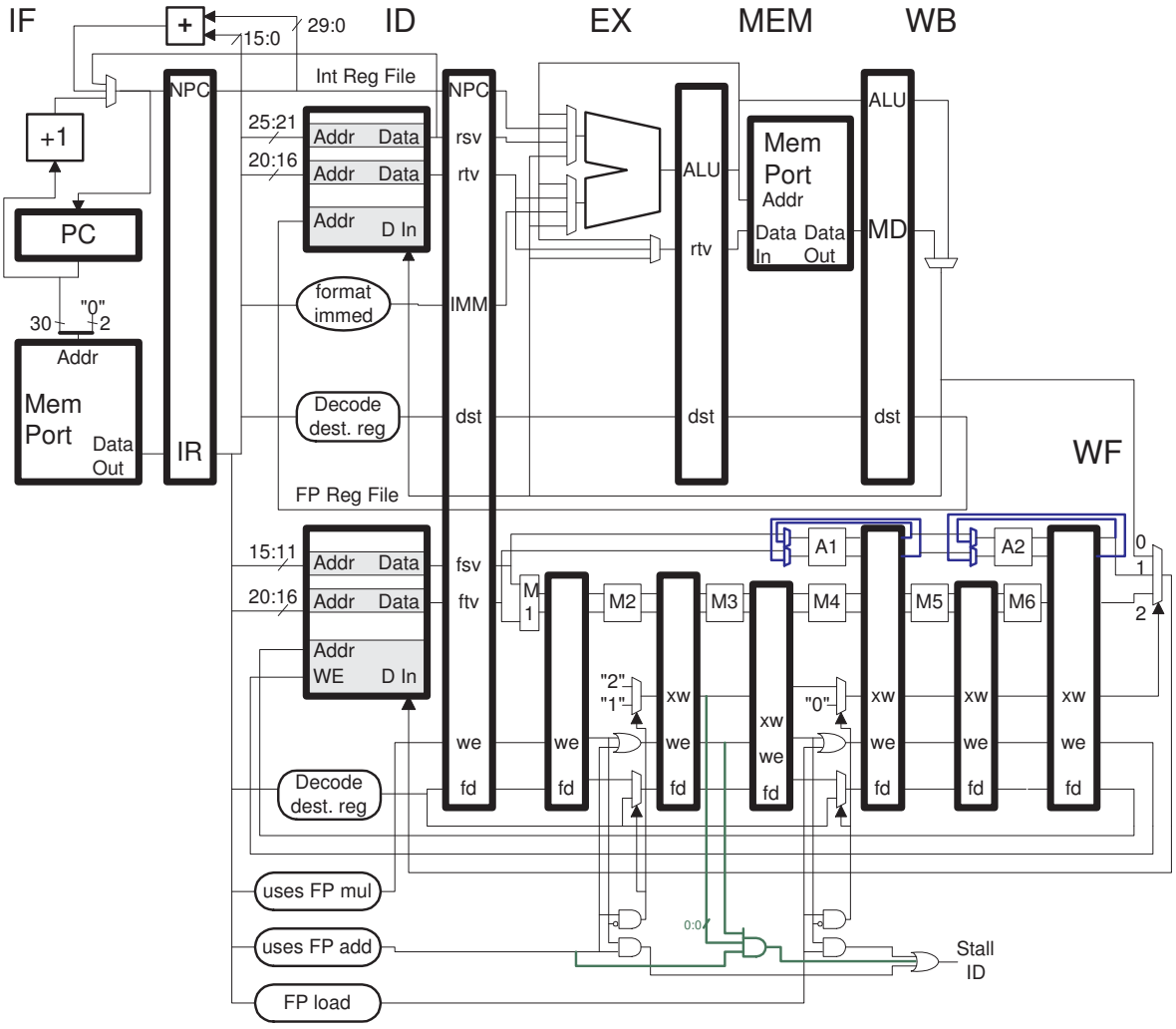
(*a*) For the code fragment above, would a `mul.d` exception detected in `M6` be precise? Explain in terms of architecturally visible storage (register and memory values) when the handler starts. (Note that in general exceptions detected in `M6` would not be precise, but the question is only asking about the fragment above.)

Yes, because when the multiply is in `M6` none of the following instructions has written a register, so they could be squashed. Registers `f6`, `r3`, and `r1` will reflect execution up to the `mul.d` instruction, a requirement of precise exceptions.

(*b*) For the code fragment above, would a `add.d` exception detected in `A4` be precise? Explain in terms of architecturally visible storage when the handler starts.

No, because it would be too late to prevent the `and` instruction from writing `r3`.

Problem 3: The MIPS implementation below has a fully pipelined FP add unit. Replace the FP add unit with one that has an initiation interval of 2 and a total computation time of 4 cycles. Note that the time to compute a floating point sum is the same on the original and replacement adder.



The new adder has two stages, `A1` and `A2`, each has two inputs (like their fully pipelined counterparts), and each has two outputs. In the first cycle of computation the source operands are placed at the inputs to `A1`, in the second cycle of computation the values at the outputs of `A1` at the end of the first cycle are placed at the inputs to `A1`. In the third cycle the values at the outputs of `A1` at the end of the second cycle are placed at the inputs `A2`, and in the fourth cycle the inputs to `A2` are the values at the outputs of `A2` at the end of the third cycle. The sum is available from the upper output of `A2` at the end of the fourth cycle.

(*a*) Replace the FP adder datapath with the one described above.

Solution appears above in blue. Multiplexors at the adder FU inputs select the proper values to process. Note that some cost is saved by having fewer pipeline latches for intermediate data but that is partly offset by the need for the multiplexors.

(*b*) Modify the control logic for the new adder. Be sure to account for the structural hazard when there are two consecutive FP add instructions.

Solution appears above in green. Assuming the multiply is still fully pipelined the pipeline latches carrying **xw**, **we**, and **fd** are still needed and so can't be removed. (If the multiply also had an initiation interval of two then half the number of pipeline latches would be needed.) The control logic for detecting the **WF** structural hazard does not need to change since operations take the same number of cycles to reach **WF**. It is only necessary to add control logic to detect the structural hazard, and that is done by examining the **we** bit in the M3 stage, to see if that stage is occupied, and the LSB of **xw**, to determine if M3 is occupied by an add operation.