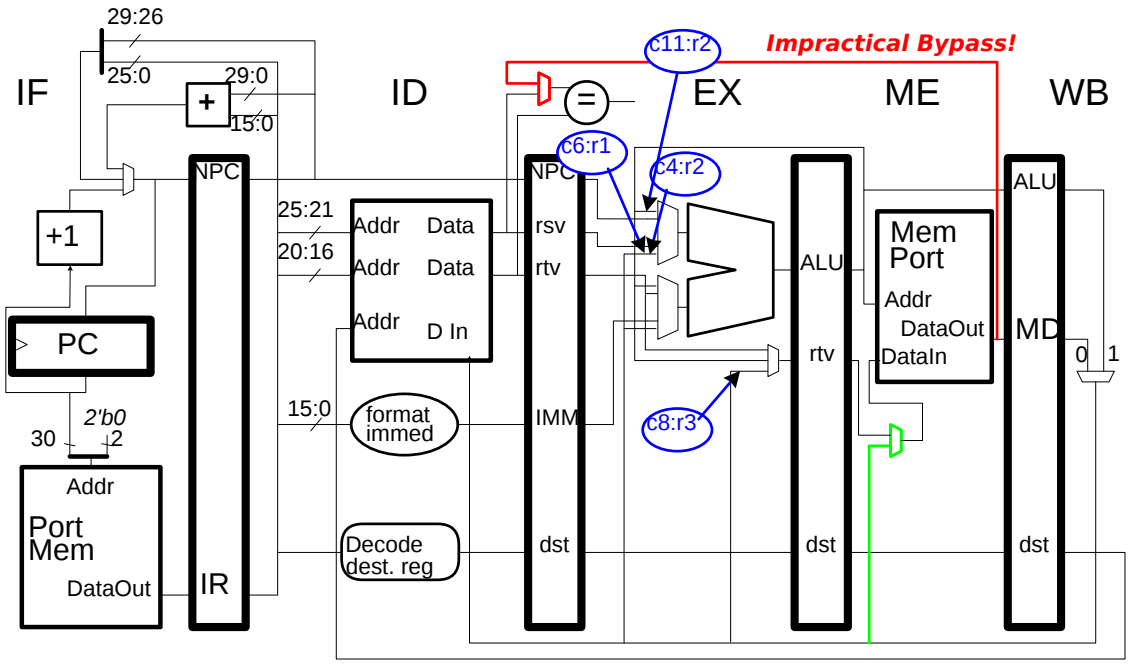
LSU EE 4720 **Homework 1** Solution **Due: 2 March 2011**

Problem 1: The MIPS code below executes on the illustrated implementation. The loop iterates for many cycles.




```
# SOLUTION
 lw r2, 0(r5)       IF ID EX ME WB
LOOP:  # Cycles     0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
 lw r1,  0(r2)         IF ID -> EX ME WB                    FIRST ITERATION
 lw r3,  0(r1)            IF -> ID -> EX ME WB
 sw r3, 4(r2)                  IF -> ID -> EX ME WB
 bne r3, r0  LOOP                    IF -> ID EX ME WB
 addi r2, r3, 0                            IF ID EX ME WB
# Cycles            0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
 lw r1,  0(r2)      SECOND ITERATION           IF ID EX ME WB
 lw r3,  0(r1)                                    IF ID -> EX ME WB
 sw r3, 4(r2)                                        IF -> ID -> EX ME WB
 bne r3, r0  LOOP                                          IF -> ID EX ME
 addi r2, r3, 0                                                  IF ID EX
# Cycles            0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
 lw r1,  0(r2)      THIRD ITERATION                                  IF ID
```

(*a*) Show a pipeline execution diagram for enough iterations to determine the CPI. Compute the CPI for a large number of iterations.

Pipeline diagram appears above. Note that execution is shown to the start of the third iteration. That was necessary to insure that a repeating pattern has been established, meaning that the state of the pipeline was the same in consecutive iterations. The first iteration starts at cycle 1 (by definition with the fetch of the first instruction of the loop), the pipeline

has the first instruction in IF and a non-loop instruction in ID. The second iteration starts in cycle 9, there the ID stage has a different instruction than in the first iteration. The third iteration starts in cycle 16, the stage contents here are the same as the start of the second iteration, **addi** in ID, **bne** in EX, and **sw** in ME. Therefore whatever happens in the second iteration will happen in the third iteration, and all following iterations (so long as the loop branch is taken).

The second iteration takes $16 - 9 = 7$ cycles, so the $\boxed{\text{CPI is } \frac{7}{5} = 1.4}$.

(*b*) Show when each bypass path is used. Do so by drawing an arrow to a multiplexor input and labeling it with the cycles in which it was used and the register. For example, something like $\boxed{\text{C10/r9}} \longrightarrow$ to show that the input is used in cycle 10 carrying a value for **r9**.

The labels are shown in the diagram above in blue.

Problem 2: Continue to consider the pipeline and code from the previous problem. The store instruction and the branch could both benefit from a new bypass connection.

(*a*) Show a new bypass connection for the store.

The store needs the value of the preceding load. That's available too late for the bypass connection in the EX stage. A new bypass has been added to the ME stage, that is shown in green.

(*b*) Indicate the impact of the new store bypass connection on critical path length.

The memory port is assumed to be on the critical path, however it would be reasonable to assume that it's the address input and data output that are critical. If so, the added multiplexor would not increase critical path.

(*c*) Show a new bypass connection needed by the branch.

Bypass needed from ME to the comparison unit in ID. Though it is impractical, it's shown in red.

(*d*) Indicate the impact of the new branch bypass connection on critical path length.

The output of the memory port will not be available until the very end of the cycle, so this bypass would certainly lengthen the critical path and so should not be added.

(*e*) Suppose that the cost of the two bypass connections were equal and that both had no critical path impact. If only one could be added to an implementation which would you add? Base your answer not on the example code above, but on what you consider to be typical programs.

The branch bypass. Branches occur frequently and its reasonable that it would use a value loaded by a nearby instruction. The store bypass, since it's only useful when the bypassed value is from an immediately preceding load, would only be useful for programs that are copying data from one area of memory to another, and there are ways of separating such load/store pairs.