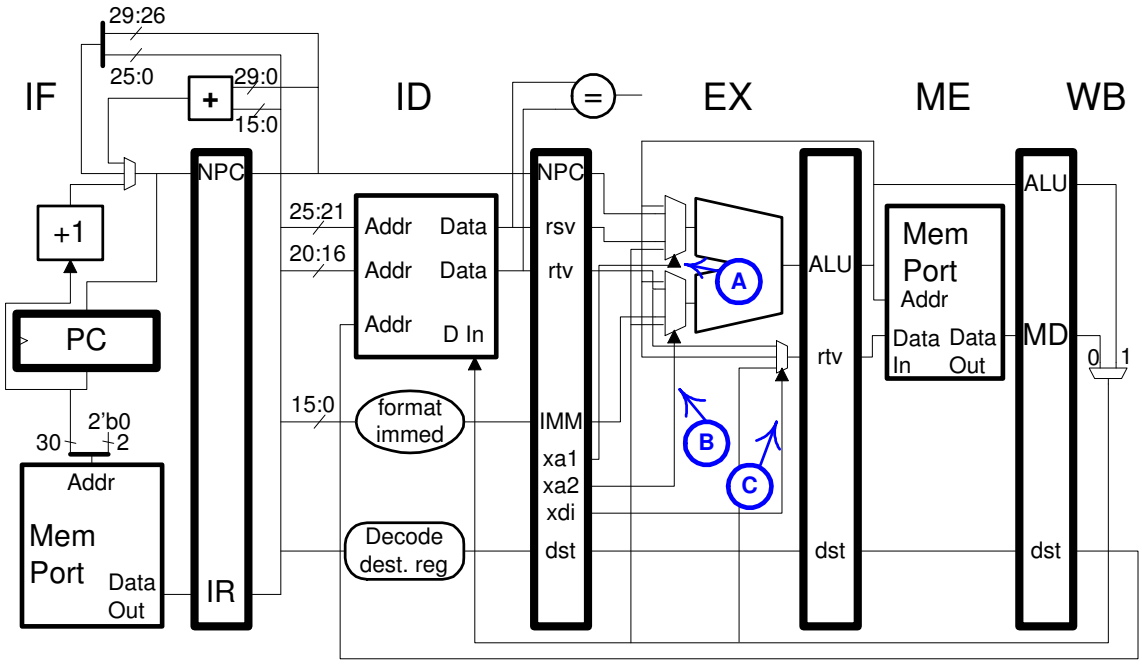
LSU EE 4720 **Homework 2** Solution **Due: 1 October 2007**

For lecture material relevant to this assignment see `http://www.ece.lsu.edu/ee4720/2007f/lsli06.pdf`. *For some background and a list of similar problems see the statically scheduled study guide,* `http://www.ece.lsu.edu/ee4720/guides/ssched.pdf`. *Please make an effort to solve this problem based on an understanding of the material, use the solution to similar problems (if any) only for hints. Feel free to ask questions using the forums, E-mail, or in person. Exam problems will be based on the assumption that students completed (really completed) homework assignments, so don't short-change yourself!*

Problem 1: Consider the following MIPS code and implementation:

```
 # Cycle              0  1  2
 lw r2, 0(r10)        IF ID EX
LOOP:
 lw r1, 0(r2)            IF ID
 add r3, r1, r4
 sw r3, 4(r2)
 bne r3, r5  LOOP
 addi r2, r2, 8
 # Cycle              0  1  2
A:                         2
B:                         2
C:
```



(*a*) Complete the pipeline execution diagram of the execution of the code above on the implementation illustrated for at least the first two iterations. (See the next part for instructions on the "A:", etc.)

Solution appears after part b, below. Note that the branch stalls due to a dependency on the `add` instruction which produces one of the branch source registers, `r3`.

(*b*) After the `addi` instruction three labels are shown, `A:`, `B:`, and `C:`; similar labels are shown, in blue and circled, in the implementation. On the pipeline execution diagram show the values on the wires (which are multiplexor inputs) that those labels point to *only in cycles in which those signals are used*. The values are already shown for cycles 0, 1, and 2. Signals `A` and `B` are used in cycle 2 (but not 0 or 1), signal `C` is not used in cycles 0-2.

Note that the multiplexor inputs are numbered from the top starting at zero.

```
 # SOLUTION
 # Cycle       0  1  2  3  4  5  6  7  8  9  10 11 12 13 14
 lw r2, 0(r10) IF ID EX ME WB
LOOP:
# First Iteration XX
 lw r1, 0(r2)     IF ID -> EX ME WB
 add r3, r1, r4      IF -> ID -> EX ME WB
 sw r3, 4(r2)           IF -> ID EX ME WB
 bne r3, r5  LOOP             IF ID -> EX ME WB
 addi r2, r2, 8                  IF -> ID EX ME WB
# Cycle        0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20
 A:                2     3     3  2        2  0     3  2  ...
 B:                2     2     1  2        2  2     1  2  ...
 C:                                1                    1  ...
# Cycle        0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20
# Second Iteration                       XX
 lw r1, 0(r2)                           IF ID EX ME WB
 add r3, r1, r4                            IF ID -> EX ME WB
 sw r3, 4(r2)                                 IF -> ID EX ME WB
 bne r3, r5  LOOP                                   IF ID -> EX ME WB
 addi r2, r2, 8                                        IF -> ID EX ME WB
# Cycle        0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20
# Third Iteration                                            XX
 lw r1, 0(r2)                                               IF ID EX ME WB
```

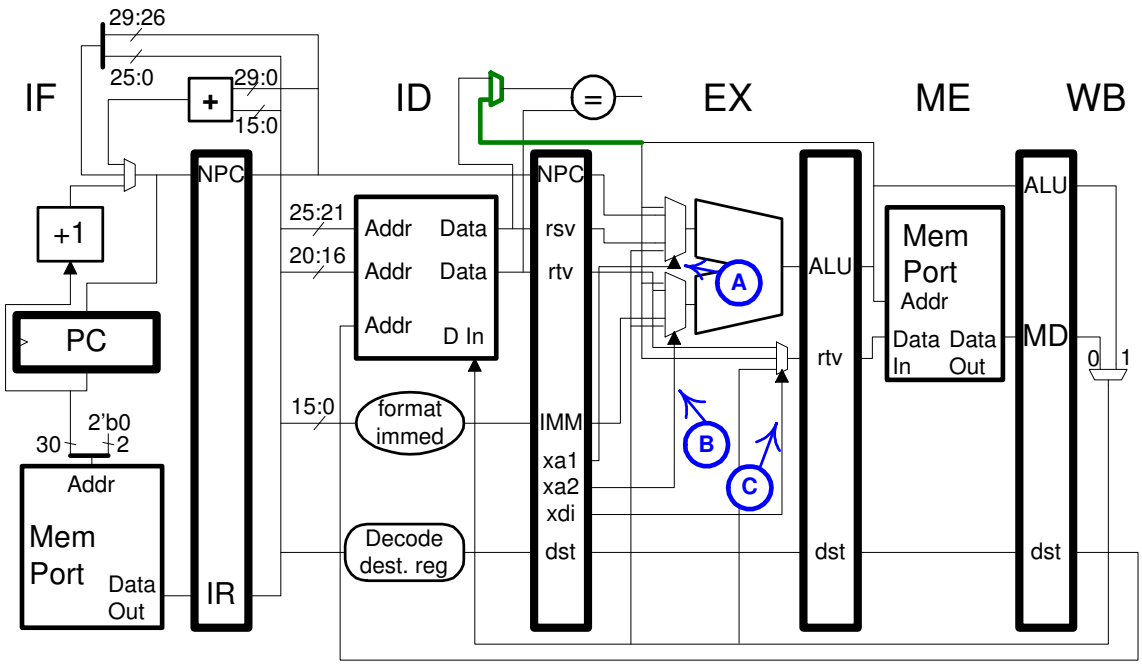
(*c*) Find the CPI of this loop on the illustrated implementation for a large number of iterations.

In the first iteration the load in the loop body stalls because of a dependency with a load outside the loop, obviously that won't happen on subsequent iterations and so the first iteration is not representative. The second iteration starts at cycle 9 (when the first instruction is fetched), the third iteration starts in cycle 16, and so the second iteration takes $16 - 9 = 7$ cycles. Both of these iterations start with the pipeline in an identical state: the `addi` is in ID, the `bne` is in EX, etc. Therefore the third iteration will take exactly the same amount of time as iteration 2, as will all subsequent iterations. Therefore the CPI for a large number of iterations is $\boxed{\frac{16-9}{5} = 1.4}$.

(*d*) Add bypass connection(s) so that the loop above executes as quickly as possible. Show the CPI with those connections.

The stalls in cycles 5 and 12 can't be eliminated by bypasses because the data arrives at the end of cycle 5 and 12, but it would be needed at the *beginning* of cycle 5 and 12 to avoid the stall.

The stall at cycles 8 and 15 can be eliminated because the data is available at the end of cycles 6 and 13, and the branch needs it in the middle of cycles 7 and 14. The added bypasses, shown in green, eliminate the stall.



(*e*) Even with bypass connections the loop above, regrettably, executes with stalls (or at least it should!). Schedule (re-arrange) the code so that it executes without stalls. The scheduled loop should still load and store one value per iteration. Minor changes to the code can be made, such as changing register numbers and immediate values.

The code below executes without a stall with the bypasses added above.

```
 # Scheduled Code

 lw r2, 0(r10)
 lw r1, 0(r2)
LOOP:
 # Cycle            0  1  2  3  4  5  6  7  8
 addi r2, r2, 8     IF ID EX ME WB
 add r3, r1, r4        IF ID EX ME WB
 sw r3, -4(r2)            IF ID EX ME WB
 bne r3, r5  LOOP            IF ID EX ME WB
 lw r1, 0(r2)                   IF ID EX ME WB
 # Cycle            0  1  2  3  4  5  6  7  8
```