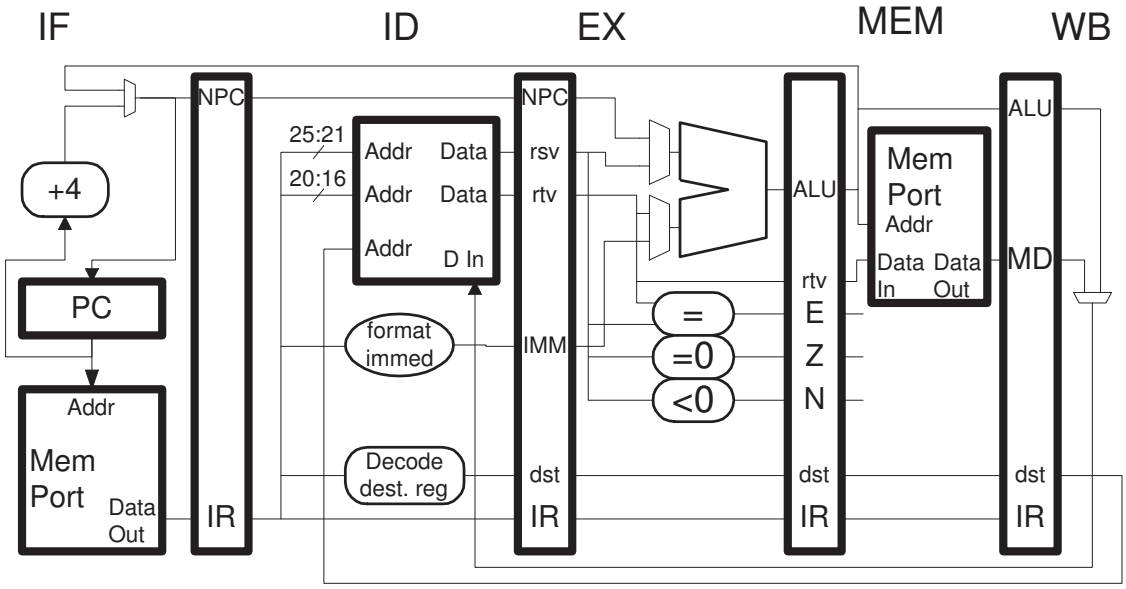
LSU EE 4720 **Homework 2** Solution **Due: 17 September 2010**

Problem 1: Consider the execution of the code fragments below on the illustrated implementation.



- A value written to the register file can be read from the register file in the same cycle. (For example, if instruction A writes `r1` in cycle x (meaning A is in `WB` in cycle x) and instruction B is in `ID` in cycle x, then instruction B can read the value of `r1` that A wrote.)
- As one should expect, the illustrated implementation will execute the code correctly, as defined by MIPS-I, stalling and squashing as necessary.

```
#       SOLUTION   Execution on the resolve-in-ME (illustrated) pipeline
LOOP: # Cycles     0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
 lw r3, 0(r1)      IF ID EX ME WB                          FIRST ITERATION
 add r4, r4, r3       IF ID ----> EX ME WB
 bne r1, r2  LOOP        IF ----> ID EX ME WB
 addi r1, r1, 4                   IF ID EX ME WB
 xor r7, r8, r3                      IF IDx
 sw r4, 16(r5)                          IFx
LOOP: # Cycles     0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
 lw r3, 0(r1)      SECOND ITERATION        IF ID EX ME WB
 add r4, r4, r3                               IF ID ----> EX ME WB
 bne r1, r2  LOOP                                IF ----> ID EX ME WB
 addi r1, r1, 4                                           IF ID EX ME WB
 xor r7, r8, r3                                              IF IDx
 sw r4, 16(r5)                                                  IFx
LOOP: # Cycles     0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
 lw r3, 0(r1)      THIRD ITERATION                                  IF ...
```

(*a*) Show a pipeline execution diagram for this code running for at least two iterations.

Solution appears above.

Grading Note: A common mistake this semester was counting the squashed instructions, `xor` *and* `sw`*, in the formula for the CPI. The CPI is a measure of performance, so it does not make sense to count instructions that were not supposed to be executed.*

- Carefully check the code for dependencies, including dependencies across iterations.
- Base timing on the illustrated implementation, pay particular attention to how the branch executes.

(*b*) Find the CPI for a large number of iterations.

The iteration start times of the first three iterations (based on the `IF` of the first instruction) are 0, 8, and 16. The first iteration and the second iteration each take 8 cycles. The states of the pipeline at the start of the second and third iterations are identical (`lw` in ID, `addi` in ME, etc.) and therefore the third iteration will take the same amount of time as the second. Therefore we can safely say that there are 8 cycles per iteration. Since there are four instructions in the loop the $\boxed{\text{CPI is } \frac{8}{4} = 2}$.

(*c*) How much faster would the code run on an implementation similar to the one above, except that it resolved the branch in `EX` instead of `ME`? Explain using the pipeline execution diagram above, or using a new one. An answer similar to the following would get no credit because "should run faster" doesn't say much: *A resolution of a branch in* `EX` *occurs sooner than* `ME` *so the code above should run faster.*. Be specific, and base your answer on a pipeline diagram.

With the branch resolved in `EX` rather than `ME` the target would be fetched while the branch is in `ME` rather than `WB`, that's one cycle earlier. Sounds good so far. But let's not be hasty, let's do a pipeline diagram, that's shown below.

The diagram shows that the second iteration starts one cycle earlier than before, but there is also a stall in the `lw` because of the dependence with the `addi`. Because of that stall the execution is no faster. The first iteration takes just 7 cycles, but the second takes 8, and so will subsequent iterations. So it's no faster.

```
#       SOLUTION  - Execution on the resolve-in-EX pipeline.
LOOP: # Cycles     0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
 lw r3, 0(r1)      IF ID EX ME WB                          FIRST ITERATION
 add r4, r4, r3       IF ID ----> EX ME WB
 bne r1, r2  LOOP        IF ----> ID EX ME WB
 addi r1, r1, 4                   IF ID EX ME WB
 xor r7, r8, r3                      IFx
LOOP: # Cycles     0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
 lw r3, 0(r1)      SECOND ITERATION     IF ID -> EX ME WB
 add r4, r4, r3                            IF -> ID ----> EX ME WB
 bne r1, r2  LOOP                                IF ----> ID EX ME WB
 addi r1, r1, 4                                           IF ID EX ME WB
 xor r7, r8, r3                                              IFx
LOOP: # Cycles     0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
 lw r3, 0(r1)      THIRD ITERATION                             IF ...
```

Problem 2: *Apologies in advance to those tired of the previous problem.* Consider the execution of the code below on the implementation from the last problem. The code is only slightly modified.

(*a*) Show a pipeline execution diagram for this code, and compute the CPI for a large number of iterations. It should be faster.

```
LOOP:
 add r4, r4, r3
 lw r3, 0(r1)
 bne r1, r2  LOOP
 addi r1, r1, 4
 add r4, r4, r3
 sw r4, 16(r5)
```

The code has been scheduled to avoid dependence stalls, the execution appears below.

```
 #      SOLUTION   Execution on the resolve in ME (illustrated) pipeline
LOOP: # Cycle      0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
 add r4, r4, r3    IF ID EX ME WB
 lw r3, 0(r1)         IF ID EX ME WB
 bne r1, r2  LOOP        IF ID EX ME WB
 addi r1, r1, 4             IF ID EX ME WB
 add r4, r4, r3                IF IDx
 sw r4, 16(r5)                    IFx
LOOP: # Cycle      0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
 add r4, r4, r3                      IF ID EX ME WB
 lw r3, 0(r1)                           IF ID EX ME WB
 bne r1, r2  LOOP                          IF ID EX ME WB
 addi r1, r1, 4                               IF ID EX ME WB
 add r4, r4, r3                                  IF IDx
 sw r4, 16(r5)                                      IFx
LOOP: # Cycle      0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
 add r4, r4, r3                                        IF ...
```

(*b*) How much faster would the code above run on the implementation that resolves branches in EX (from the previous problem)?

An iteration takes 6 cycles on the resolve-in-ME version (shown above); for a CPI of $\frac{6}{4}$. On the resolve-in-EX version the second iteration would start in cycle 5, on cycle earlier, and would not suffer stalls (unlike it's problem 1 counterpart). So it would run with a CPI of $\frac{5}{4}$. The question asked how much faster. A correct answer might be $\frac{5}{4}$ versus $\frac{6}{4}$. (Any reasonable comparison would be just as correct.)

(*c*) Suppose that due to critical path issues, the resolve-in-EX implementation had a slower clock frequency. Let ϕ_{ME} be the clock frequency of the resolve-in-ME implementation (the one illustrated), and ϕ_{EX} be the clock frequency of the resolve-in-EX implementation. Find ϕ_{EX} in terms of ϕ_{ME} such that both implementations execute the code fragment above in the same amount of time. That is, find a clock frequency at which the benefit of a smaller branch penalty is neutralized by the lower clock frequency *on the code fragment above*.

Let c_{EX} denote the number of cycles per iteration of the resolve-in-EX version and define c_{ME} similarly. The execution time per iteration for the two systems are $\frac{c_{\text{EX}}}{\phi_{\text{EX}}}$ and $\frac{c_{\text{ME}}}{\phi_{\text{ME}}}$. Equate the two quantities, $\frac{c_{\text{EX}}}{\phi_{\text{EX}}} = \frac{c_{\text{ME}}}{\phi_{\text{ME}}}$, and solve for ϕ_{EX}:

$$\phi_{\text{EX}} = \phi_{\text{ME}} \frac{c_{\text{EX}}}{c_{\text{ME}}} = \phi_{\text{ME}} \frac{5}{6}$$

Grading Note: Too many students apparently did not give their answers some does-this-make-sense scrutiny. We know that the resolve-in-EX system is faster. Therefore we should expect that it can run at a **lower** *clock frequency and still equal the performance of the resolve-in-ME system.*