Homework 2 (due September 17, 2019)

Question 1: As discussed in class, even with forwarding hardware, the pipeline has to be stalled when a "lw" instruction which loads data into a register "Rt" is followed by an instruction which reads from register "Rt". In this question, you will explore an exception to this rule in the case where the instruction following "lw \$Rt, I1(\$R1)" is "sw \$Rt, I2(\$R2)" that saves the content of "Rt" (just loaded) into memory.

a. Argue that in this case, it is possible to avoid stalling by providing a forwarding path between the MEM/WB buffer to the input of the data memory.
b. Draw a simple diagram for the part of the data path between the EX/MEM and MEM/WB buffers showing the forwarding path described in part (a).

Question 2: In this question, we will consider a 10-stage pipeline in which target addresses are evaluated in the second stage and branch conditions are evaluated in the fifth stage (no delayed branching is assumed). In this pipeline, predicting that a branch is always taken means that if the branch instruction (at address PC) enters the pipeline at time t, then the instruction following the branch (the instruction at PC+4) will enter the pipeline at time t+1 and the instruction at the branch target will enter the pipeline at time t+2. When the branch instruction is in the fifth stage, the branch condition will be known and some of the instructions in the pipeline will be changed into no-ops. In the following, assume that 20% of the instructions are branch instructions and that 60% of the branches are "taken".

a) What is the extra CPI that results from control hazards when the architecture assumes that branches are always "not taken"?
b) What is the extra CPI that results from control hazards when the architecture assumes that branches are always "taken"?
c) Assuming that 20% of the instructions are branches and denote by X the probability that the branches are "taken". What is the maximum X for which the "branches always not taken" scheme is more efficient than the "branches always taken" scheme?

Question 3:

Consider the following loop that computes z(i) = a*w(i) + y(i) for i=0,…,99 and assume that it executes on an architecture similar to the one shown in slide 31 (of Appendix C slides) except that the pipeline depths of the multiplication and addition units are 4 and 2, respectively (instead of 7 and 4):

```
L: fld       f2, 0(x1)        ; load w(i)  -- initially x1 points to w(0).
   fmult.d   f4, f2, f0       ; multiply a*w(i)
   fld       f6, 800(x1)      ; load y(i)  -- vector y is allocated immediately after vector w
   fadd.d    f6, f4, f6       ; add a*w(i) + y(i)
   fsd       1600(x1), f6     ; store z(i) – vector z is allocated immediately after vector y
   addi      x1, x1, 8        ; increment index
   bne       x1, x2, L        ; loop if not done  -- x2 contains the address of w(0) + 800
   add       x3, x4, x5
   add       x6, x7, x8
```

(a) Identify the forwarding paths needed to minimize pipeline stalling. Identify each forwarding paths by (sourse intersatge buffer – destination interstage buffer). For example in the 5 stage pipeline, the two forwarding paths are (EX/MEM – ID/EX) and (MEM/WB – ID/EX). Note that there is only one buffer between ID and EX, M1 and A1 (call it ID/EX).

(b) Construct a table similar to the one in slide 33 (Figure C.32) and determine the number of clock cycles needed to complete the execution of one iteration of the loop (starting from the time when the "fld f2, 0(x1)" is in the IF stage to the time when the "bne x1, x2, L" is in the EX stage. Here it is assumed that the branch is resolved in the EX stage.

(c) How many cycles will it take to execute 100 iterations of this loop. Assume that the normal "predict not taken" scheme is used and that the branch is resolved in the EX stage.

(d) It is possible to replace the two instructions

```
    fsd         1600(x1), f6   ; store z(i)
    addi        x1, x1, 8      ; increment index
```

by

```
    addi        x1, x1, 8      ; increment index
    fsd         1592(x1), f6   ; store z(i)
```

without affecting the correctness of the code. How will this affect the number of cycles needed to execute each iteration of the loop?

Question 4: In this question, assume the same architecture as in Question 3.

(a) Show the original loop of Question 3 after unrolling it (the new loop will iterate only 50 times instead of 100 times).

(b) Construct a table similar to the one in Question 3 to determine the number of clock cycles needed to complete the execution of one iteration of the unrolled loop.

(c) What is the speedup resulting from unrolling the loop.

(d) Do you expect further speedup if the loop is unrolled more than once?