# CS252 HOMEWORK 2

Due Tues. 10/9/07

**Problem 1:**

In the following problem, use a simple pipelined RISC architecture with a branch delay cycle. The architecture has pipelined functional units with the flowing execution cycles:

1. Floating point op: 3 cycles
2. Integer op: 1 cycles

The following table shows the minimum number of intervening cycles between the producer and consumer instructions to avoid stalls. Assume 0 intervening cycle for combinations not listed.

| **Instruction producing result** | **Instruction using result** | **Latency in clock cycles** |
|---|---|---|
| FP ALU op | Another FP ALU op | 2 |
| FP ALU op | Store and move double | 2 |
| Load double | FP ALU op | 1 |
| Load double | Store double | 0 |

As an example, if the result of an ADD.D instruction is used by the next instruction, which is also ADD.D, there has to be two stalls between these two instructions.

The following code computes a 3-tap filter. R1 contains address of the next input to the filter, and the output overwrites the input for the iteration. R2 contains the loop counter, already decremented by 1. The tap values are contained in F12, F13, F14.

```
# F0 = previous value V[-1]
# F1 = V[-2]
# Assume at least one iteration left
# Assume loop counter already decremented by 1
LOOP:   MULT.D   F3, F1, F14   # multiply V[-2] by T[2]
        MULT.D   F2, F0, F13   # multiply V[-1] by T[1]
        ADD.D    F3, F3, F2    # add results
        MOV.D    F1, F0        # move V[-1] to next iter's V[-2]
        L.D      F0, 0(R1)     # load V[0]
        MULT.D   F2, F0, F12   # multiply V[0] by T[0]
        ADD.D    F3, F3, F2    # add results
        S.D      F3, 0(R1)     # store result
        ADDI     R1, R1, 8     # increment pointer
        BGTZ     R2, LOOP      # done?
        SUBI     R2, R2, 1     # decrement loop counter
```

a. How many cycles does the current code take for each iteration? Show the stalls.

b. Rearrange the code without unrolling to reduce the cycles per iteration by 2 or more. Do not modify any instructions yet. Show execution cycle next to each code line.

c. Unroll your code in b. **twice** (so it contains 3 iterations). Replace registers so that they are not duplicated, drop unnecessary instructions, and modify instructions as needed for proper operation. Make sure to document necessary changes for loop input. Do not reschedule code yet.

d. Schedule your code from c. to minimize number of stalls.

e. What is the effective cycle per iteration for the unrolled loop, where the iteration is referring to an iteration for the original code?

**Problem 2:**

a) What is static branch prediction? What is the downside to it?

b) Why is it important for Tomasulo to issue instructions in-order?

c) What factors can limit the performance of Tomasulo's algorithm?

**Problem 3:**

The following results are seen from a simulation study of five floating-point benchmarks and two integer benchmarks from the SPEC92 suite. The branch misprediction rate nearly doubles from 5% to 9.1% going from 1 thread to 8 threads in an SMT processor. However, the wrong-path instructions fetched (on a misprediction) drops from 24% on a single-threaded processor to 7% on an 8-thread processor.

a. What causes the increase in branch misprediction rate?

b. Why is there a decrease in the number of wrong-path instructions fetched even though there is an increase in branch misprediction rates? (*Hint:* This is related to the scheduling of threads.)