EE 4720 **Homework 3** **Due: 15 March 2000**

Problem 1: Show a pipeline execution diagram for the following DLX code fragment on a statically scheduled implementation in which the add functional unit has a latency of 3 (four stages) and an initiation interval of 2 (not the usual 1) and the multiply unit has a latency of 5 (six stages) and has an initiation interval of 3 (not the usual 1).

```
addf f0, f1, f2
addf f3, f0, f4
addf f5, f0, f7
gtf  f0, f8
multf f9, f0, f10
```

Problem 2: Show a pipeline execution diagram for the following DLX code fragment on a statically scheduled implementation in which the add functional unit has a latency of 3 (four stages) and an initiation interval of 1 (the usual 1) and the multiply unit has a latency of 5 (six stages) and has an initiation interval of 1 (the usual 1).

The implementation uses ID-stage branch target calculation. As is true for the pipelines used in class, the branch condition is not bypassed.

Instructions stall in ID to avoid structural hazards.

There are bypass paths from the WB stage to the inputs of the floating-point functional units.

(*a*) What is the CPI for a large number of iterations of the loop?

(*b*) If the multiply functional unit latency were long enough the second iteration would take longer than the first iteration. (An iteration starts when the first instruction is in IF.) What is the smallest such latency?

```
LOOP:
 multd f0, f0, f2
 ld    f4, 0(r1)
 addd  f2, f2, f4
 addi  r1, r1, #8
 sub   r2, r1, r3
 bneq  r2, LOOP
 xor   r10,r11,r12
```

Problem 3: Schedule—but don't unroll—the code from the problem above to avoid as many stalls as possible. Show a pipeline execution diagram of the scheduled code. *Hint: you can change the offset of the load double instruction.*

Problem 4: Unroll the loop below so that two iterations of the original loop form one unrolled loop. Schedule the code so that it executes as efficiently as possible. Assume there will be an even number of iterations and that every register not used in the original code is available and so can be used in the unrolled loop. The loop runs on the implementation described in the second problem.

```
LOOP:
 ld     f0, 0(r1)
 multd  f0, f0, f2
 addd   f0, f0, f4
 sd     8(r1), f0
 addi   r1, r1, #16
 sub    r2, r1, r3
 bneq   r2, LOOP
```