LSU EE 4720 **Homework 4** Solution **Due: 20 April 2009**

Problem 1: Solve Fall 2008 Final Exam Problem 4 and the additional questions below.

(*a*) For part (a) provide pipeline execution diagrams for the three systems (5-stage scalar, n-way superscalar, and $5n$-stage superpipelined) running code of your choosing. Refer to these diagrams when answering part (a).

Solution shown below. In the superscalar solution n instructions reside in a stage at one time. in the superpipelined system each stage is split into n stages and the clock frequency is increased by a factor of n.

```
# Scalar System
#
# Cycle             1  2  3  4  5  6 ... n  n+1 ...
1:   add r1, r2, r3  IF ID EX ME WB
2:   or  r9, r2, r3     IF ID EX ME WB
     ...
n:   sub r4, r5, r6                         IF ID EX ME WB
n+1: xor r6, r7, r8                            IF ID EX ME WB
     ...

# n-way Superscalar
#
#Cycle              1  2  3  4  5  6 ... n  n+1 ...
1:   add r1, r2, r3  IF ID EX ME WB
2:   or  r9, r2, r3  IF ID EX ME WB
     ...
n:   sub r4, r5, r6  IF ID EX ME WB
n+1: xor r6, r7, r8     IF ID EX ME WB
     ...

# Superspipelined
#
Cycle                1   2   3   4   5   6   7   8   9   10  11  12  13
1:   add r1, r2, r3  IF1 IF2 ..  IFn ID1 ID2 ..  IDn EX1 EX2 ..  EXn ..
2:   or  r9, r2, r3      IF1 IF2 ..  IFn ID1 ID2 ..  IDn EX1 EX2 ..  EXn ..
     ...
n:   sub r4, r5, r6                  IF1 IF2 ..  IFn ID1 ID2 ..  IDn EX1 ..
n+1: xor r6, r7, r8                      IF1 IF2 ..  IFn ID1 ID2 ..  IDn ..
     ...
```

Problem 2: Consider the three systems from Problem 4 in the final exam. The problem focused on potential (favorable) execution time, which can be achieved when there are few stalls, here we'll be more realistic.

(*a*) Which system will suffer more stalls on typical code? Explain.

The dependence in the code below will not stall the scalar system, will always stall the superpipelined system (since `sub` needs a value in `EX1` at the latest, but the `add` doesn't have it ready until `ME1`, $n-1$ cycles too late), and will sometimes stall the superscalar system. The non-stall superscalar case is shown below, note that the `add` is the last instruction of a group so the `sub` starts one cycle later. Therefore $\boxed{\text{the superpipelined system suffers the most stalls}}$.

```
n:   add r1, r2, r3  IF ID EX ME WB
n+1: sub r4, r1, r5     IF ID EX ME WB
```

(*b*) Invent a quantitative measure of implementation (not program) stall potential and apply it to the three systems. The answer should include a formula for each system (giving the stall potential); the superscalar and superpiplined formulas should be in terms of n. *Hint: think about the average or minimum distance between two dependent instructions needed to avoid a stall.* The formulas should be consistent with your answer to the first part.

Call the measure the *average stall distance*. Let $a \in \{0, \ldots, A-1\}$ be the set of possible instruction locations (the address divided by 4 in MIPS) and let $s(a)$ denote the minimum number of instructions between an **add** instruction at a and a dependent **sub** instruction (so that **sub** would be at location $a+1+s(a)$). For the scalar MIPS system $s(a) = 0$ for all a (there are no possible stalls between an add and a subtract). For the superscalar system

$$s(a) = \begin{cases} 0, & \text{if } a \bmod n = n-1; \\ 1, & \text{otherwise.} \end{cases},$$

and for the superpipelined system $s(a) = n$.

Define the *average stall distance* to be $\frac{1}{A}\sum_{a=0}^{A-1} s(a)$. The average stall distance for the scalar system is 0, for the superscalar system it is $\frac{n-1}{n}$ and for the superpipelined system it is n.