LSU EE 4720 **Homework 4** Solution **Due: 3 December 2007**

Problem 1: *For answers to this problems consult the SPARC Architecture Manual Version V8, linked to the course references page.*

Suppose a SPARC V8 trap table has been set up at address `0x12340000`.

(*a*) Write a SPARC V8 program that sets the trap base register (TBR) to that address. Assume the processor is already in privileged mode. *Hint: A correct solution consists of two instructions, a three-instruction program is okay too.*

Browsing the SPARC Architecture Manual Version V8 for information on the trap table one should soon come across the **`wrtbr`** (write trap base register) instruction. The instruction is used in the code fragment below.

```
! SOLUTION
sethi %hi(0x12345000), %l0
wrtbr %l0, 0, %tbr
```

Call the SPARC V8 instruction that writes the TBR *foo*. The ISA definition of `foo` makes it easy to design the control logic and bypassing hardware on **certain** implementations.

(*b*) What about the definition of foo makes the control logic and bypassing hardware design easy on those certain implementations?

The V8 architecture manual definition of **`wrtbr`** states that it is a delayed-write instruction, meaning that if any of the next three instructions attempt to read the TBR the result is undefined. (See the third paragraph on page 134 of the posted version of the manual.)

A five-stage implementation similar to the one used in class would not need to check for true dependencies with the TBR, nor would bypass paths be needed, reducing the cost. For example, in the code below the TBR is written and then read (using **`rdtbr`**). According to the definition the code below would be correct regardless of what the first **`rdtbr`** writes in `g4` (because it is within 3 instructions of the most recent **`wrtbr`**). The same is true for the **`rdtbr`** that writes `g5`. Therefore the implementation does not need to check for a dependency between **`rdtbr`** and preceding instructions in the pipeline. The value placed in `g6` by the last **`rdtbr`** must be the value written to the TBR by the **`wrtbr`**, for this five-stage implementation that requires no special hardware because the TBR write is complete when the last **`rdtbr`** reaches `ID`.

```
! Cycle               0  1  2  3  4  5  6
 wrtbr %l0, 0, %tbr   IF ID EX ME WB
 add %g1, %g2, %g3       IF ID EX ME WB
 rdtbr %tbr, %g4            IF ID EX ME WB
 rdtbr %tbr, %g5               IF ID EX ME WB
 rdtbr %tbr, %g6                  IF ID EX ME WB
```

(*c*) Why not do the same for, say, the `add` instruction?

Because it would be difficult to schedule code without slowing execution by adding lots of `nop` instructions. The first code sample below shows ordinary MIPS code. With sufficient bypass paths the code should execute without a stall on a scalar 5-stage statically scheduled implementation. In the second code fragment there must be at least a 3-instruction separation between an instruction that writes a register and the instruction that reads it. To maintain correctness under that restriction nop's were added, slowing down execution.

It's okay to impose the three-instruction separation restriction on rarely used instructions, such as `wrtbr` because the impact on performance will be tiny. Imposing such a restriction on frequently executed instructions would have too large an impact on execution, not worth the small savings in control and bypass logic.

```
# MIPS as is: No restriction on placement.
LOOP:
 lw r1, 0(r2)
 addi r2, r2, 4
 and r1, r1, r3
 bne r2, r4 LOOP
 add r5, r5, r1

# MIPS with 3-insn separation:
LOOP:
 lw r1, 0(r2)
 addi r2, r2, 4
 nop
 nop
 and r1, r1, r3
 nop
 nop
 bne r2, r4 LOOP
 add r5, r5, r1
```

(*d*) Describe an implementation in which the control logic for `foo` would not be so simple despite the "help" from the ISA definition.

Any implementation in which `wrtbr` writes the TBR *after* the fourth following instruction reads it. This can definitely occur in a 5-way superscalar implementation. In such an implementation logic would be needed to detect the dependency, or else always assume there is such a dependency and stall the pipeline after every `wrtbr` instruction.

Problem 2: Solve the EE 4720 Spring 2007 Final Exam problem 1.

Problem 3: Solve the EE 4720 Spring 2007 Final Exam problem 3.