LSU EE 4720 **Homework 3** Solution **Due: 18 April 2007**

Some of the questions below are about the interrupt mechanisms defined for the MIPS32, SPARC V8, and PowerPC 2 ISAs. MIPS and SPARC interrupt mechanisms were covered in class, PowerPC's mechanism was not. All are documented in manuals linked to the class references page, `http://www.ece.lsu.edu/ee4720/reference.html`. *When using these references keep in mind that interrupt terminology differs from ISA to ISA and that you are not expected to understand (at least on first reading) most of what is in these manuals. Finding the right manuals and the relevant pages in those manuals is part of this assignment's learning experience.*

Problem 1: Consider a load instruction that raises an exception due to a fixable problem with a memory address (for example, a TLB miss, whatever that is) on an implementation of MIP32, SPARC V8, and PowerPC 2.

(*a*) Where does each ISA say the address of the faulting instruction (the load) should be put? Give the exact register name, number, or both (if available).

MIPS32: Coprocessor 0 register **EPC**, register number 14. (If the faulting instruction is not in a delay slot then **EPC** is set to the address of the faulting instruction, otherwise the address of previous instruction).

SPARC V8: After advancing to a new register window, the address of the faulting instruction is put in register `l1` (also called `r17`). The address of the next instruction (which could be a CTI target) is put in register `l2` (also called `r18`).

PowerPC: The address of the load instruction is written to SRR0.

(*b*) Where does each ISA say to put the memory address that the load attempted to load from?

MIPS32: Coprocessor 0 register `BadVAddr`, register number 8.

SPARC V8: The memory management unit's (MMU) fault status register.

PowerPC: Register **DAR**.

Problem 2: Is PowerPC's equivalent of a trap table more similar to SPARC's trap table or to MIPS'? Explain and describe how specific elements are the same or different. Look at table placement, size, number of entries, and perhaps other characteristics.

Here is a summary of table characteristics.

MIPS32: The trap "table" is actually spread across several locations, defined by the ISA. At some locations there is a single handler, at others a set of four of handlers (as MIPS and PowerPC). Normally the cache exception table starts at `0xa0000000` and the table for other non-reset, non-debug exceptions starts at `0x80000000`. The number of entries under normal operations is about 8, the smallest gap between entries is 32 instructions.

SPARC V8: Base address is value stored in **TBR** (trap base register), written by software during OS startup. Table size: 256 entries, each entry is four instructions.

PowerPC: Table starts at address 0 and spans 4096 characters. There are 15 entries with defined uses. Based on Figure 29 one might assume handler size is 32 instructions based on the smallest space between defined entries.

Comparison: SPARC has the largest table (256 entries) and one that can be placed anywhere. The benefit of many entries is that the handler might spend less branching to the appropriate code (because for each entry there is just one place to jump). (This is probably not that big an advantage since the table would not be used that often.)

PowerPC is similar to SPARC in that all entries are in one table, not spread across the address space as in MIPS. However like MIPS the location of PowerPC's table is defined by the ISA. The number of entries is closer to that of MIPS than SPARC. Overall the PowerPC table seems more similar to MIPS.

Grading Note: If reasonably argued, "closer to SPARC" would also be considered a correct answer.

Problem 3: In class a precise exception was defined as one in which, to the handler it appears that all instructions before the faulting instruction have completed normally and that the faulting instruction and those following it have not executed (correctly or otherwise). PowerPC calls certain exceptions precise even though they violate this rule. What are they and how is this violation justified in the manual?

Data segment exceptions raised by certain load and store instructions. Under some circumstances when the handler starts the faulting load or store will have written some registers, violating a condition normally associated with precise exceptions. The ISA "book" points out that what's important is the ability to re-execute the instruction, and so implementations can have such loads and stores write registers so long as no information needed to re-execute them is lost.

Problem 4: Solve Fall 2006 Final Exam Problem 1. *Note: At the time this was assigned the solutions were not available.*

See posted solution to exam.