Universität des Saarlandes
FR 6.2 - Informatik
Prof. Dr. W.J. Paul
Dipl.-Ing. Christoph Baumann

Exercise Sheet 10
Computer Architecture II

(Due: Jan 14th, 2014)

Exercise 1: (New Internal Interrupt Semantics) **(6+4+10 Points)**

In the lecture we noticed that it is a bad idea to accumulate internal interrupts in the sequential MIPS semantics. For instance, when a misalignment interrupt occurs on fetch, execution should halt and jump to the interrupt service routine immediately and it should not be possible that further internal interrupts, e.g., *ill*, or *ovf* are signalled. Moreover we have now two different misalignment interrupts, one for misalignment on fetch (*malf*) ad one for misalignment on load-store (*malls*).

a) Make a list of internal interrupts that can occur in the different phases of instruction execution in the MIPS machine with interrupts and address translation. Give a new definition of $iev(c)$ such that it contains no active interrupt signals from two different phases!

b) Adapt the MIPS implementation to the new specification!

c) Given a MIPS hardware computation (h^t) according to some external input sequence eev_h^t and an ISA state c which is consistent with h^0, i.e.:

$$sim(c, h^0)$$

Assume that the hardware gets interrupted during execution of the next ISA step. Prove that both machines are consistent again after the jump to the interrupt service routine for the special purpose register component.

$$\exists t > 0, eev_{isa}.\ \delta(c, eev_{isa}).spr = h^t.spr$$

Exercise 2: (Page Fault Handler) **(8+12 Points)**

In this exercise we want to design a page fault handler for our sequential MIPS with interrupts, multi-level address translation, TLB, and devices. The page tables are set up such that we map a 2 GiB fraction of the swap disk into the upper half of main memory, this means we only consider virtual addresses $a \in \mathbb{B}^{32}$ with $a.px_2[9] = 1$. There is a function $hdmap : \mathbb{B}^9 \rightarrow \mathbb{B}^{18}$ which maps 4 MiB sectors in the RAM to 4 MiB sectors on the disk.

$$d.m_{22}(a.px_2 \circ 0^{22}) = d.sm_{10}(hdmap(a.px_2) \circ 0^{10})$$

The page tables are set up to provide an identical translation of virtual to physical addresses. They are located in the lower half of main memory and can be assumed to be set up correctly, however not all pages of the page table may be present in physical memory. The root page table is stored on the disk at page $root \in \mathbb{B}^{28}$ followed by the 2nd level page tables in ascending order of their page index in the root table.

A page fault handler gets as an argument the faulty effective address *ea* in the *edata* special purpose register. It then rewalks the page tables in order to find out where the page fault occurred. For simplicity we only assume here page faults due to pages that are not present, i.e., we exclude page faults due to rights violations. In case a page is not present, it is loaded from the swap disk using a hard disk port located at the double-page-aligned address $\mathit{hdbase} \in \mathbb{B}^{32}$ in lower memory, i.e., $\mathit{hdbase}[12:0] = 0^{13}$ and $\mathit{hdbase}[31] = 0$. Moreover the disk port and the page tables are disjoint. Since we have a fixed mapping from the subset of swap memory pages to the virtual physical memory pages and an identity mapping in the page tables, we do not have to swap out pages to make space for the new one, or invalidate old walks in the TLB. After the page is swapped, in the handler sets the present bit and continues walking until the address translation is complete.

a) Give a formal specification for the page fault handler, i.e. state the post-condition that should hold after it has finished executing relating to the pre-state of the system!

b) Provide a detailed implementation of the page fault handler in pseudo-code. Comment your code as necessary such that it is understandable!