6.823 Computer Architecture	DUE: 3/13/2002

Timothy Danford, Kilian Pohl, Bryan Russell

Problem Set 3 — *Instructors: Krste Asanovic, Srinivas Devadas*

Problem 3

(a) Assuming that only user processes are running (i.e. only load and store commands are executed), then the only bits that must be written back to the page table are the used and modified (U and M) bits. The U and M bits are the only ones that get updated during a user process due to the load and store commands. If kernel code is running, then all of the bits could actually change. Hence, all of the bits would have to be written back to the page table for this case.

(b) The advantages of a "guaranteed valid" scheme like this are that the valid bit never needs to be checked... time is saved since a TLB cache line can only be in one state instead of two, and space is saved by the reduction of a bit from every line.

The disadvantages are numerous. First, there's a question of how to initialize the TLB when a process first starts. That incurs both the overhead of filling in all the rows and the possible extra logic of figuring out what rows to fill in (like a strange version of prefetching). Second, "guaranteed valid" is also synonymous with "guaranteed overhead". In the previous scheme, it would be possible for a program with a small memory footprint (with few memory pages than lines in the TLB) to have all its VPN-to-PPN mappings stored in the TLB without conflict. On a (compulsory) miss, the page table tree is walked and the appropriate PTE is loaded; in **this** scheme, on the other hand, the overhead of each such compulsory miss is doubled. Not only must the relevant PTE be loaded, but the older (and probably irrelevant) PTE that was initially loaded into the TBL must still be saved to memory.

The final issue is that context switches (which flush the TBL) compound all these problems. Each time a context switch occurs, the TBL must be flushed and reinitialized (which now involves the loading of possibly spurious PTE's instead of simply flipping the invalid bits).

In other words, a few bits and a few cycles to check those bits are saved, at the cost of a much larger amount of time spent waiting for irrelevant PTE's to be written to and read from the memory.

(c) By combining the two valid bits into one (say V'), we have the ambiguous case when V' is 0. First, when V' is 1, we just simply return the PPN/DBN that's in the TLB. When V' is 0, the PTE is either invalid (V bit in the page table is 0) or the TLB entry is invalid and we have to fetch the PTE from the page table. So we must define a new exception that combines these two issues (PTE Invalid and TLB Miss exceptions). After raising this combined exception, for the load and store commands we fetch the PTE from the page table each time and check if the V bit is 1. If so, then we update the TLB and proceed as normal. If the V bit is 0, then we raise a PTE Invalid exception.

(d) For this scheme to work, we must maintain the M (modified) bit. We must require at the start each entry in the page table has the M bit set to 0. Also, if a page becomes invalidated, we reset the M bit to be 0. So, for a load or store using the TLB, we check the M bit. If the M bit is 1, then we return the PPN/DBN as normal since we have modified the page recently and we know that it is valid based on the above conditions. If the M bit is 0 and we are doing a load or store, then we must look in the page table and check the V bit since we cannot know if the page is valid or not. If the V bit is 1 then we return the PPN/DBN as normal, else we raise a PTE Invalid exception. Of course, for a store we change the M bit in the TLB to 1. Also, for a load or store with the TLB Valid bit set to 0 (TLB Miss exception) we must retrieve the PTE from the page table and check the V bit there to see if the PTE is valid (to see if we also raise a PTE Invalid exception).

(e) The scheme from 3.C will lead to better performance. In 3.C, we do extra memory/disk fetches (page table lookup) in the case when V' and V are 0 (for all other cases, we either do the requisite fetches or get the result immediately from the TLB). In 3.D, we do extra memory/disk fetches (page table lookup) in the case when the TLB valid bit is 1 and the M bit is 0. So, for this case if we just do all loads, we will have to always access the page table and get no benefit from the TLB. In the scheme from 3.C, we only do extra memory/disk accesses when a PTE in the TLB is invalid.

(f) We cannot remove the R (resident) bit in the TLB without impacting performance significantly. If we do so, then we do not know if the TLB PTE corresponds to a location in memory or on disk. So, we would have to look at the page table for each access which of course eliminates the benefits of storing PTEs in the TLB.

(g) If we remove the S (supervisor) bit, then we have one of two options to maintain protection for pages. One option is to designate a range of addresses that are for the supervisor and not to be used at all by the user. So, when the user presents a VPN to the TLB, the hardware first checks this VPN to see if it is in the appropriate range and raises a Protection Violation exception if it is not in the correct range. The second option would be to have the hardware or kernel keep a table somewhere that keeps track of which pages are supervisor pages. So, upon checking the VPN we first check the table and see if it is valid for the user or not. Of course, this second method dismisses our goal of reducing bits since we are just moving the S bit from the TLB entry to another place.

(h) For subsequent load operations using the same tag in the TLB, we just keep returning the PPN/DBN. For store operations, we first check the W (write) bit. If W is 1, then we proceed as normal. If W is 0, then we look in the page table and check the W bit for the corresponding PTE. If W is 1 in the page table, then set W to be 1 in the TLB. Else, we keep W as 0 in the TLB. We then proceed with the store operation (note that we must also keep a flag that indicates that we have already looked in the page table, else we will do an infinite loop. If the flag has been set and W is still 0, then we raise a Write Fault exception). Upon writing the PTE back to the page table from the TLB, we check the W bit. If W is 1, then we set the M and U bits in the page table to be 1. Else, we leave them unchanged. Note: if the kernel modifies the U or M bits at all while the PTE is in the TLB, then it must do so in the page table and change the corresponding W bit back to 0 in the TLB in order to keep the TLB consistent.