CS152 Computer Architecture and Engineering

February 28, 2008

Virtual Memory and Address Translation

*Assigned February 28*

Problem Set #3

*Due March 11*

**http://inst.eecs.berkeley.edu/~cs152/sp08**

The problem sets are intended to help you learn the material, and we encourage you to collaborate with other students and to ask questions in discussion sections and office hours to understand the problems. However, each student must turn in their own solutions to the problems.

The problem sets also provide essential background material for the quizzes. The problem sets will be graded primarily on an effort basis, but if you do not work through the problem sets you are unlikely to succeed at the quizzes! We will distribute solutions to the problem sets on the day the problem sets are due to give you feedback. Homework assignments are due at the beginning of class on the due date. Homework will not be accepted once solutions are handed out.

This material will be on Quiz 3, not Quiz 2.

## Problem 3.1: Virtual Memory Bits

*This problem requires the knowledge of Handout #3 (Virtual Memory Implementation) and Lecture 9. Please, read these materials before answering the following questions.*

In this problem we consider simple virtual memory enhancements.

### Problem 3.1.A

Whenever a TLB entry is replaced we write the entire entry back to the page table. Ben thinks this is a waste of memory bandwidth. He thinks only a few of the bits need to be written back. For each of the bits explain why or why not they need to be written back to the page table.

With this in mind, we will see how we can minimize the number of bits we actually need in each TLB entry throughout the rest of the problem.

### Problem 3.1.B

Ben does not like the TLB design. He thinks the TLB Entry Valid bit should be dropped and the kernel software should be changed to ensure that all TLB entries are always valid. Is this a good idea? Explain the advantages and disadvantages of such a design.

### Problem 3.1.C

Alyssa got wind of Ben's idea and suggests a different scheme to eliminate one of the valid bits. She thinks the page table entry valid and TLB Entry Valid bits can be combined into a single bit.

On a refill this combined valid bit will take the value that the page table entry valid bit had. A TLB entry is invalidated by writing it back to the page table and setting the combined valid bit in the TLB entry to invalid.

How does the kernel software need to change to make such a scheme work? How do the exceptions that the TLB produces change?

## Problem 3.1.D

Now, Bud Jet jumps into the game. He wants to keep the TLB Entry Valid bit. However, there is no way he is going to have two valid bits in each TLB entry (one for the TLB entry one for the page table entry). Thus, he decides to drop the page table entry valid bit from the TLB entry.

How does the kernel software need to change to make this work well? How do the exceptions that the TLB produces change?

## Problem 3.1.E

Compare your answers to Problem 3.1.C and 3.1.D. What scheme will lead to better performance?

## Problem 3.1.F

How about the R bit? Can we remove them from the TLB entry without significantly impacting performance? Explain briefly.

## Problem 3.1.G

The processor has a kernel (supervisor) mode bit. Whenever kernel software executes the bit is set. When user code executes the bit is not set. Parts of the user's virtual address space are only accessible to the kernel. The supervisor bit in the page table is used to protect this region—an exception is raised if the user tries to access a page that has the supervisor bit set.

Bud Jet is on a roll and he decides to eliminate the supervisor bit from each TLB entry. Explain how the kernel software needs to change so that we still have the protection mechanism and the kernel can still access these pages through the virtual memory system.

## Problem 3.1.H

Alyssa P. Hacker thinks Ben and Bud are being a little picky about these bits, but has devised a scheme where the TLB entry does not need the M bit or the U bit. It works as follows. If a TLB miss occurs due to a load, then the page table entry is read from memory and placed in the TLB. However, in this case the W bit will always be set to 0. Provide the details of how the rest of the scheme works (what happens during a store, when do the entries need to be written back to memory, when are the U and M bits modified in the page table, etc.).

## Problem 3.2: Page Size and TLBs

*This problem requires the knowledge of Handout #3 (Virtual Memory Implementation) and Lecture 10. Please, read these materials before answering the following questions.*

Assume that we use a hierarchical page table described in Handout #3.

The processor has a data TLB with 64 entries, and each entry can map either a 4KB page or a 4MB page. After a TLB miss, a hardware engine walks the page table to reload the TLB. The TLB uses a first-in/first-out (FIFO) replacement policy.

We will evaluate the memory usage and execution of the following program which adds the elements from two 1MB arrays and stores the results in a third 1MB array (note that, 1MB = 1,048,576 Bytes):

```
byte A[1048576]; // 1MB array
byte B[1048576]; // 1MB array
byte C[1048576]; // 1MB array

for(int i=0; i<1048576; i++)
  C[i] = A[i] + B[i];
```

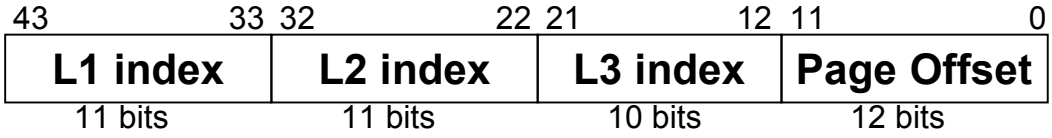
We assume the A, B, and C arrays are allocated in a contiguous 3MB region of physical memory. **We will consider two possible virtual memory mappings**:

- **4KB**: the arrays are mapped using 768 4KB pages (each array uses 256 pages).
- **4MB**: the arrays are mapped using a single 4MB page.

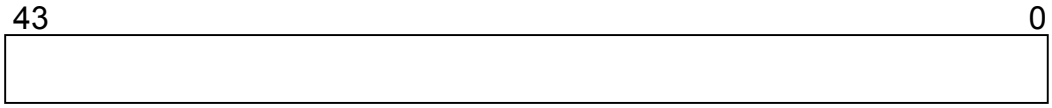
For the following questions, assume that the above program is the only process in the system, and ignore any instruction memory or operating system overheads. Assume that the arrays are aligned in memory to minimize the number of page table entries needed.

**Problem 3.2.A**

This is the breakdown of a virtual address which maps to a 4KB page:




Show the corresponding breakdown of a virtual address which maps to a 4MB page. Include the field names and bit ranges in your answer.




**Problem 3.2.B** **Page Table Overhead**

We define page table overhead (PTO) as:

$$\textbf{PTO} = \frac{\text{Physical memory that is allocated to page tables}}{\text{Physical memory that is allocated to data pages}}$$

For the given program, what is the PTO for each of the two mappings?

$\textbf{PTO}_{\textbf{4KB}} =$ _____

$\textbf{PTO}_{\textbf{4MB}} =$ _____

**Problem 3.2.C** **Page Fragmentation Overhead**

We define page fragmentation overhead (PFO) as:

$$\textbf{PFO} = \frac{\text{Physical memory that is allocated to data pages but is never accessed}}{\text{Physical memory that is allocated to data pages and is accessed}}$$

For the given program, what is the PFO for each of the two mappings?

$\textbf{PFO}_{\textbf{4KB}}$ = _____

$\textbf{PFO}_{\textbf{4MB}}$ = _____

**Problem 3.2.D**

Consider the execution of the given program, assuming that the data TLB is initially empty. For each of the two mappings, how many TLB misses occur, and how many page table memory references are required per miss to reload the TLB?

| | Data TLB misses | Page table memory references (per miss) |
|---|---|---|
| **4KB:** | | |
| **4MB:** | | |

**Problem M2.8.E**

Which of the following is the best estimate for how much longer the program takes to execute with the 4KB page mapping compared to the 4MB page mapping?
Circle one choice and **briefly explain** your answer (about one sentence).

**1.01×** **10×** **1,000×** **1,000,000×**

## Problem M3.3: 64-bit Virtual Memory

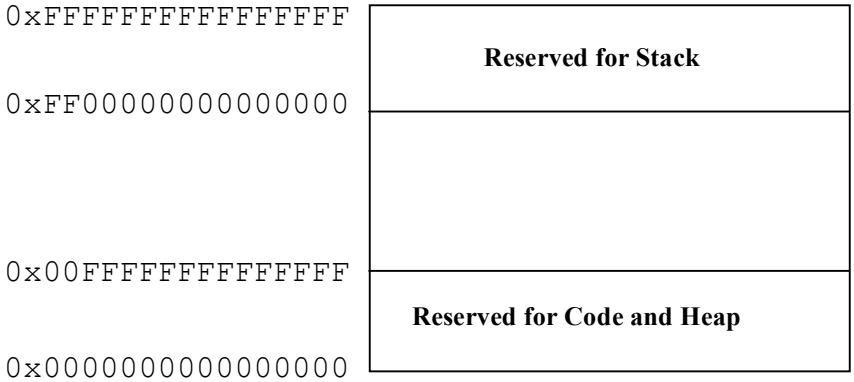
This problem examines page tables in the context of processors with a 64-bit addressing.

### Problem 3.3.A — Single level page tables

For a computer with 64-bit virtual addresses, how large is the page table if only a single-level page table is used? Assume that each page is 4KB, that each page table entry is 8 bytes, and that the processor is byte-addressable.

### Problem 3.3.B — Let's be practical

Many current implementations of 64-bit ISAs implement only part of the large virtual address space. One way to do this is to segment the virtual address space into three parts as shown below: one used for stack, one used for code and heap data, and the third one unused.




A special circuit is used to detect whether the top eight bits of an address are all zeros or all ones before the address is sent to the virtual memory system. If they are not all equal, an invalid virtual memory address trap is raised. This scheme in effect removes the top seven bits from the virtual memory address, but retains a memory layout that will be compatible with future designs that implement a larger virtual address space.

The MIPS R10000 does something similar. Because a 64-bit address is unnecessarily large, only the low 44 address bits are translated. This also reduces the cost of TLB and cache tag arrays. The high two virtual address bits (bits 63:62) select between user, supervisor, and kernel address spaces. The intermediate address bits (61:44) must either be all zeros or all ones, depending on the address region.

How large is a single-level page table that would support MIPS R10000 addresses? Assume that each page is 4KB, that each page table entry is 8 bytes, and that the processor is byte-addressable.

## Problem 3.3.C — Page table overhead

A three-level hierarchical page table can be used to reduce the page table size. Suppose we break up the 44-bit virtual address (VA) as follows:

| VA[43:33] | VA[32:22] | VA[21:12] | VA[11:0] |
|---|---|---|---|
| 1st level index | 2nd level index | 3rd level index | Page offset |

If page table overhead is defined as (in bytes):

$$\frac{\text{PHYSICAL MEMORY USED BY PAGE TABLES FOR A USER PROCESS}}{\text{PHYSICAL MEMORY USED BY THE USER CODE, HEAP, AND STACK}}$$

Remember that a complete page table page (1024 or 2048 PTEs) is allocated even if only one PTE is used. Assume a large enough physical memory that no pages are ever swapped to disk. Use 64-bit PTEs. What is the smallest possible page table overhead for the three-level hierarchical scheme?

Assume that once a user page is allocated in memory, the whole page is considered to be useful. What is the largest possible page table overhead for the three-level hierarchical scheme?

## Problem 3.3.D — PTE Overhead

The MIPS R10000 uses a 40 bit physical address. The physical translation section of the TLB contains the physical page number (also known as PFN), one "valid," one "dirty," and three "cache status" bits.

What is the minimum size of a PTE assuming all pages are 4KB?

MIPS/Linux stores each PTE in a 64 bit word. How many bits are wasted if it uses the minimum size you have just calculated? It turns out that some of the "wasted" space is recovered by the OS to do bookkeeping, but not much.