Computer Architecture and Engineering

CS152 Quiz #3

March 18th, 2008

Professor Krste Asanovic

Name:_____

This is a closed book, closed notes exam.

80 Minutes

10 Pages

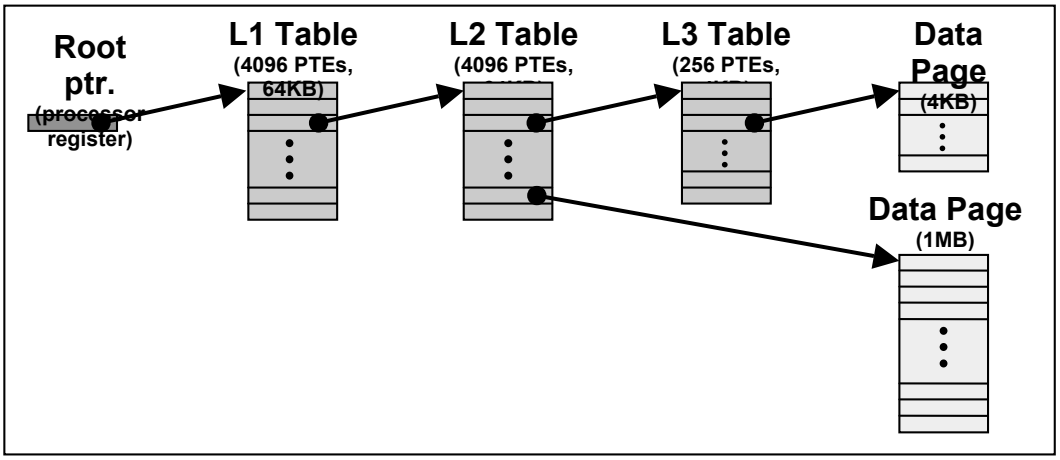
Notes:

- **Not all questions are of equal difficulty, so look over the entire exam and budget your time carefully.**
- **Please carefully state any assumptions you make.**
- **Please write your name on every page in the quiz.**
- **You must not discuss a quiz's contents with students who have not yet taken the quiz. If you have inadvertently been exposed to the quiz prior to taking it, you must tell the instructor or TA.**
- **You will get no credit for selecting multiple choice answers without giving explanations if the instructions ask you to explain your choice.**

Writing name on each sheet	**_____**	**1 Point**
Question 1	**_____**	**30 Points**
Question 2	**_____**	**25 Points**
Question 3	**_____**	**24 Points**
TOTAL	**_____**	**80 Points**

Problem Q3.1: Page Size and TLBs 30 Points

The configuration of the hierarchical page table in this problem is similar to the one in Handout #3 (included at the back of the exam for your convenience), but we modify two parameters: 1) this problem evaluates a virtual memory system with two page sizes, 4KB and 1MB (instead of 4 MB), and 2) all PTEs are 16 Bytes (instead of 8 Bytes). The following figure summarizes the page table structure and indicates the sizes of the page tables and data pages (not drawn to scale):



The processor has a data TLB with 64 entries, and each entry can map either a 4KB page or a 1MB 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 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, the starting address of the arrays are given below):

```
byte A[1048576]; // 1MB array 0x00001000000
byte B[1048576]; // 1MB array 0x00001100000
byte C[1048576]; // 1MB array 0x00001200000

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

Assume that the above program is the only process in the system, and ignore any instruction memory or operating system overheads. The data TLB is initially empty.

Problem Q3.1.A **5 Points**

Consider the execution of the program. Assume that there is no data cache, and that each memory lookup has 100 cycle latency. Note that the program loop accesses memory one byte at a time.

If all data pages are 4KB, compute the ratio of cycles for address translation to cycles for data access.

Problem Q3.1.B **5 Points**

If all data pages are 1MB, compute the ratio of cycles for address translation to cycles for data access.

Problem Q3.1.C **10 Points**

For this question, assume that in addition we will add a PTE cache with single cycle latency. A PTE cache will cache page table entries from any level in order to speed up address translation. If this PTE cache has unlimited capacity and is fully associative, compute the ratio of cycles for address translation to cycles for data access for the 4KB data page case.

Problem Q3.1.D **5 Points**

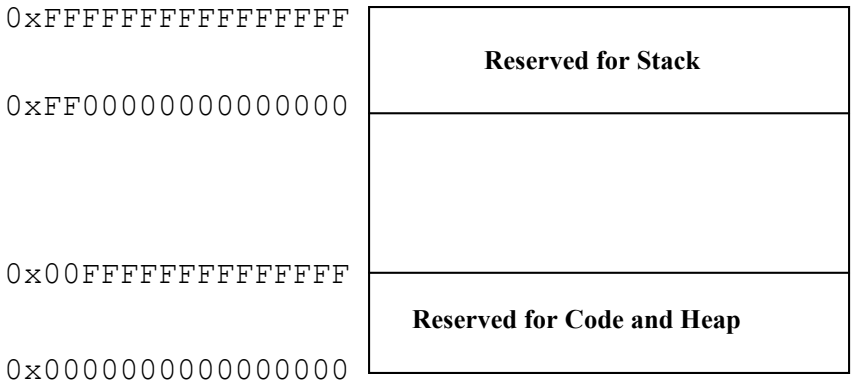
When running this code on a system with both a PTE cache and a TLB, is there any benefit to caching L3 PTE entries in the PTE cache? Explain.

Problem Q3.1.E **5 Points**

What is the minimum capacity (number of entries) needed in the PTE cache to get the same performance on this code as an unlimited PTE cache? (Assume that the fully associative PTE cache does not cache L3 PTE entries and all data pages are 4KB)

Problem Q3.2: 64-bit Address Spaces 25 Points

Ben read in a recent technical journal that many current implementations of 64-bit ISAs implement only part of the large virtual address space. They usually segment the virtual address space into three parts: one used for stack, one used for code and heap data, and the third one unused. Ben's proposed processor's virtual address space is shown below:



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.

Alyssa P. Hacker is unhappy with the large hole in the virtual address space given by Ben's scheme. She decides that a hashed page table is the way to go. Again, the machine has a 64-bit virtual address and 4KB pages. The hardware paging system has only one page table with **64 slots**, each containing **8 PTEs**. Alyssa decides to use ***X mod 64*** as the hash function to select a slot, where ***X*** is the VPN. The page table resides in memory and Alyssa's design has **no TLB**, so **each** PTE read requires one memory access. During a page table lookup, **all PTEs in each slot** are searched sequentially until a match is found (each of these sequential accesses is a memory access). If there is a miss in the page table, a trap is raised and a software handler will refill the page table, with each refill requiring 10 memory accesses on average.

Problem Q3.2.A **12 Points**

Alyssa is happy with her new modification and runs a very simple benchmark that repeatedly loops over an array of 2^{21} bytes, reading one byte at a time in sequential address order. **On average in the steady state, how many memory accesses are performed for each byte read by the user program?** *Ignore the memory traffic for instruction fetch, assume that the array starts on a page boundary, and there are no other memory accesses in the user code apart from the single byte memory accesses.*

Problem Q3.2.B **5 Points**

Alyssa now decides to add a one-entry TLB to the hashed paging system. What should the replacement policy for the TLB be? **Circle** the most appropriate of the following choices and **explain**:

a) FIFO
b) LRU
c) Random
d) Doesn't matter, all of the above give the same performance

Problem Q3.2.C **8 Points**

Given your answer to the previous question, what is now the average number of memory

accesses per user byte read for Alyssa’s benchmark?

Problem Q3.3: Virtual Memory and Caches 24 Points

Problem Q3.3.A **8 Points**

Ben Bitdiddle is designing a one level 4-way set associative cache that is virtually indexed and virtually tagged. He realizes that such a cache suffers from a *homonym* aliasing problem. The *homonym* problem happens when two processes use the same virtual address to access different physical locations. Ben asks Alyssa P. Hacker for help with solving this problem. She suggests that Ben should add a PID (Process ID) to the virtual tag. Does this solve the *homonym* problem? Explain.

Problem Q3.3.B **8 Points**

Another problem with virtually indexed and virtually tagged caches is called the *synonym* problem. The *synonym* problem happens when distinct virtual addresses refer to the same physical location. Does Alyssa's idea solve this problem? Explain.

Problem Q3.3.C **8 Points**

Ben thinks that a different way of solving *synonym* and *homonym* problems is to have a direct mapped cache, rather than a set associative cache. Is he right? Explain.

END OF EXAM SECTION
THIS PAGE PURPOSEFULLY LEFT BLANK

Hierarchical Page Table Supporting Variable-Sized Pages

Small fixed-sized pages (e.g. 4 KB) reduce internal fragmentation and the page fault penalty compared to large fixed-sized pages. However, when we run an application with a large working set, they may degrade a processor's performance by incurring a number of TLB misses because of their small *TLB reach*. Therefore, researchers have proposed to support variable-sized pages to increase the TLB reach without losing the benefits of small fixed-sized pages. Many modern processor families (e.g. UltraSparc, PA-RISC, MIPS) and operating systems (e.g. Sun Solaris, SGI IRIX) support this feature.

In this handout, we present an example implementation of variable-sized pages, supporting only two page sizes: 4 KB and 4 MB. Assume that the system uses 44-bit virtual addresses and 40-bit physical addresses. 4KB pages are mapped using a three-level hierarchical page table. 4MB pages are mapped using the first two levels of the same page table. An L2 Page Table Entry (PTE) contains information which indicates if it points to an L3 table or a 4MB data page. All PTEs are 8 Bytes. The following figure summarizes the page table structure and indicates the sizes of the page tables and data pages (not drawn to scale):

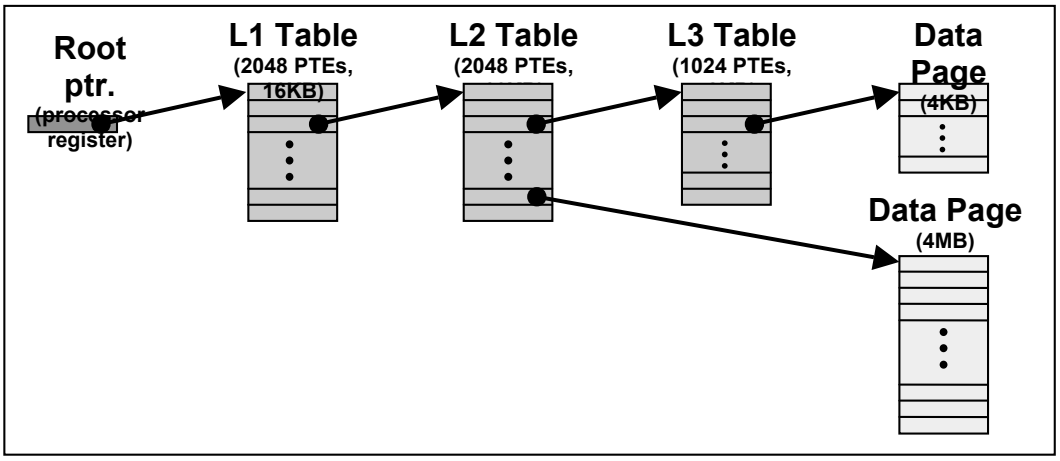


Figure H8-A. Example implementation of variable-sized pages.

Page Table Entries and Translation Lookaside Buffers

Each Page Table Entry (PTE) will map a virtual page number (VPN) to a physical page number (PPN). In addition to the page number translation, each page table entry also contains some permission/status bits.

Bit Name	Bit Definition
PPN / DBN	Physical Page Number / Disk Block Number
V (valid)	1 if the page table entry is valid, 0 otherwise
R (resident)	1 if the page is resident in memory, 0 otherwise
W (writable)	1 if the page is writable, 0 otherwise
U (used)	1 if the page has been accessed *recently*, 0 otherwise
M (modified)	1 if the page has been modified, 0 otherwise
S (supervisor)	1 if the page is only accessible in supervisor mode, 0 otherwise

Each entry in the Translation Lookaside Buffer (TLB) has a tag that is matched against the VPN and a TLB Entry Valid bit (note, the TLB Entry Valid bit is not the V bit shown in the table above). The TLB Entry Valid bit will be set if the TLB entry is valid. Each TLB entry also contains all the fields from the page table that are listed above.

A TLB miss (VPN does not match any of the tags for entries that have the TLB Entry Valid bit set) causes an exception. On a TLB miss kernel software will load the page table entry into the TLB and will restart the memory access. (Kernel software can modify anything in the TLB that it likes and always runs in supervisor mode). If the entry being replaced was valid, then the kernel will also write the TLB entry that is being replaced back to the page table.

Hardware will set the used bit whenever a TLB hit to the corresponding entry occurs. Similarly, the modified bit (in the TLB entry) will be set when a store to the page happens.

All exceptions that come from the TLB (hit or miss) are handled by software. For example, the possible exceptions are as follows:

TLB Miss:	VPN does not match any of tags for entries that have the TLB Entry Valid bit set.
Page Table Entry Invalid:	Trying to access a virtual page that has no mapping to a physical address.
Write Fault (Store only):	Trying to modify a read-only page (W is 0).
Protection Violation:	Trying to access a protected (supervisor) page while in user mode.
Page Fault:	Page is not resident.

(Unless noted, exceptions can occur for both loads and stores.)