15-213: Introduction to Computer Systems
Written Assignment #6

This written homework covers virtual memory.

Directions

Complete the question(s) on the following pages with single paragraph answers. These questions are not meant to be particularly long! Once you are done, submit this assignment on Canvas.

Below is an example question and answer.

Q: Please describe benefits of two's-complement signed integers versus other approaches.

A: Other representations of signed integers (ones-complement and sign-and-magnitude) have two representations of zero (+0 and -0), which makes testing for a zero result more difficult. Also, addition and subtraction of two's complement signed numbers are done exactly the same as addition and subtraction of unsigned numbers (with wraparound on overflow), which means a CPU can use the same hardware and machine instructions for both.

Grading

Assignments are graded via *peer review:*

1. Three other students will each provide short, constructive feedback on your assignment, and a score on a scale of 1 to 20. You will receive the maximum of the three scores.
1. You, in turn, will provide feedback and a score for three other students (not the same ones as in part 1). We will provide a *rubric*, a document describing what good answers to each question look like, to assist you. You receive five additional points for completing all of your peer reviews.

Due Date

This assignment is due on July 20 by 11:59pm Pittsburgh time (currently UTC-4). Remember to convert this time to the timezone you currently reside in.

Peer reviews will be assigned roughly 12 hours later, and are due a week after that.

Question #1

Virtual addressing is one of the most innovative ideas in computer science. How does virtual memory keep address spaces separate, and how can it share information between address spaces?

Memory is divided into pages, and the system maps virtual memory pages into physical memory pages according to one or more page table(s). The mappings are controlled by the OS.

Therefore, different virtual pages can be mapped into different physical pages, and memory of different processes don't interfere with others'. So each process has the illusion of having separated and full address spaces.

Meanwhile, the OS can make some parts of virtual memory be mapped into a same part of physical memory, so processes can share memory with others if needed.

Question #2

Assume a System that has

- 1MB of virtual memory
- 4KB page size
- 512KB of physical memory
- A two way set associative TLB with 8 total entries

How many bits are needed to represent the virtual address space? How many bits are needed to represent the physical address space? How many bits are needed to represent a page offset? How many bits are needed to represent the TLBI (TLB index)?

20 bits are needed to represent the virtual address space

19 bits are needed to represent the physical address space

12 bits are needed to represent a page offset

2 bits are needed to represent the TLBI, because there are 4 sets

Below is the TLB of this system:

Index	Tag	PPN	Valid
0	0x06	0x15	1
	0x18	0x58	0
1	0x3F	0x27	0
	0x2A	0x72	1
2	0x1B	0x35	1
	0x28	0x4B	0

3	0x0D 0x1D	0x15 0x5F	1 0

Use the TLB to translate the virtual address 0x18613. What is the VPN? What is the TLB index? What is the TLB tag? Is it a TLB hit? If so, what is the physical address?

0x18613 = 00011000 011000010011

VPN = 00011000 = 0x18

TLBI = 00 = 0

TLBT = 000110 = 0x6

It's a TLB hit, and physical address is 0x15613

Use the TLB to translate the virtual address 0x77513. What is the VPN? What is the TLB index? What is the TLB tag? Is it a TLB hit? If so, what is the physical address?

0x77513 = 01110111 010100010011

VPN = 01110111

TLBI = 11 = 0x3

TLBT = 011101 = 0x1D

It's a TLB hit, the physical address is 0x5F513

Question #3

You are tasked with the design and implementation of a virtual addressing system. The system has a 64-bit address space and 2KB page ~~tables~~ size. Each page table entry is 64 bits. How much memory would be necessary for a single-level page table using the aforementioned parameters? Comparing this to the 2KB page tables, which scheme would you use if you wanted to optimize for space and why? However, what is one advantage in using a single-level page table over multi-level page tables? How can we offset this disadvantage of the multi-level page tables? (Hint: what other pieces of hardware are available to speed up page table entry look-ups?)

We need to have enough page table entries for 2^64 bytes / 2KB = 2^53 pages. Since each PTE takes up 8 bytes, we need 2^56 bytes = 64 PB for a single-level page table.

To optimize space, we can use multi-level page tables. The lowest level of page table stores page entries of small memory regions (such as 1GB), and each entry in the upper-level table points to a bucket of some lower-level entries. Thus, one entry in the upper-level table manages a large chunk of memory. But most importantly, if a region of virtual memory is not present in the physical memory, the bucket of page entries of this region is omitted, saving us a large space in the lower-level page tables.

The disadvantage of using a multi-level page table, compared to using a single-level page table, that it is slow since we need to look up page tables of every level.

To overcome this problem, we can use a TLB cache to store most commonly used PTE. If a PTE is in the TLB, we can skip checking page tables.