CMPE 110 Computer Architecture
Fall 2015, Homework #4

Computer Engineering
UC Santa Cruz

November 23, 2015

Name: _____

Email: _____

Submission Guidelines:

- This homework is due on Friday 12/4/15.
- The homework must be submitted to ecommons by 7:59am.
  - Anything later is a late submission
- **Please write your name and your UCSC email address**
- **The homework should be "readable" without too much effort**
  - The homework must be typed and submitted as a single file in PDF format
  - Please name your homework file cmpe110-hw4-yourcruzid.pdf
  - Please keep your responses coherent and organized or you may lose points
- Provide details on how to reach a solution. **An answer without explanation gets no credit. Clearly state all assumptions.**
- Points: $64 = 14 + 15 + 7 + 28$

Question	Part A	Part B	Part C	Total
1	–	–	–	
2				
3	–	–	–	
4			–	
Total				

Question 1. Cache Coherence Transactions (14 points)

Assume that you have the following multiprocessor system configuration:

- 4 CPU Cores (C0, C1, C2, and C3)
- MESI coherence protocol
- Each core has a 32 KB private L1 Cache (2-way set-associative, uses write-back scheme, with a 64 byte cacheline)
- Shared L2 between all cores (512 KB cache, 8-way set-associative, with a 64 byte cacheline)

Given the following sequence of memory accesses, write the state of each cacheline accessed in the respective core's L1 cache.

Hint: C0 performs a Load operation on address 0x00ffabc0. There will be a miss at both L1 and L2 (due to a cold cache). The requested cache line is fetched from DRAM and mapped to L2 and then on to L1. Thus, this access causes the cacheline to have an Exclusive (E) state in C0.

	Core	**Request Type**	**Address**	**C0 L1 State**	**C1 L1 State**	**C2 L1 State**	**C3 L1 State**
1	0	Load	`0x00ffabc0`	E	-	-	-
2	0	Store	`0x00ffabc8`				
3	1	Load	`0x00ffabd4`				
4	1	Store	`0x00ffabd8`				
5	1	Load	`0x00afabc0`				
6	2	Load	`0x00afabc8`				
7	1	Load	`0x00bfabf0`				
8	0	Load	`0x00ffabc0`				

Question 2. Cache Coherence Protocols (15 points)

Consider a processor with 4 cores. In each of the following parts you will fill out a table for a different cache coherence protocol. For the following memory references, show the state of the cache line containing the variable x in each core's cache. Consider the cacheline only in L1.

Question 2.A VI Protocol (5 points)

Assume for this part that we use the VI protocol for cache coherence. Fill out the table as specified at the beginning of the question.

	Core	Request Type	C0 Cache Line State	C1 Cache Line State	C2 Cache Line State	C3 Cache Line State
1	0	Read x				
2	1	Read x				
3	2	Read x				
4	3	Write x				
5	1	Read x				

Question 2.B MSI Protocol (5 points)

Assume for this part that we use the MSI protocol for cache coherence. Fill out the table as specified at the beginning of the question.

	Core	Request Type	C0 Cache Line State	C1 Cache Line State	C2 Cache Line State	C3 Cache Line State
1	0	Read x				
2	1	Read x				
3	2	Read x				
4	3	Write x				
5	1	Read x				

Question 2.C MESI Protocol (5 points)

Assume for this part that we use the MESI protocol for cache coherence. Fill out the table as specified at the beginning of the question.

	Core	Request Type	C0 Cache Line State	C1 Cache Line State	C2 Cache Line State	C3 Cache Line State
1	0	Read x				
2	1	Read x				
3	2	Read x				
4	3	Write x				
5	1	Read x				

Question 3. Virtual Memory (7 points)

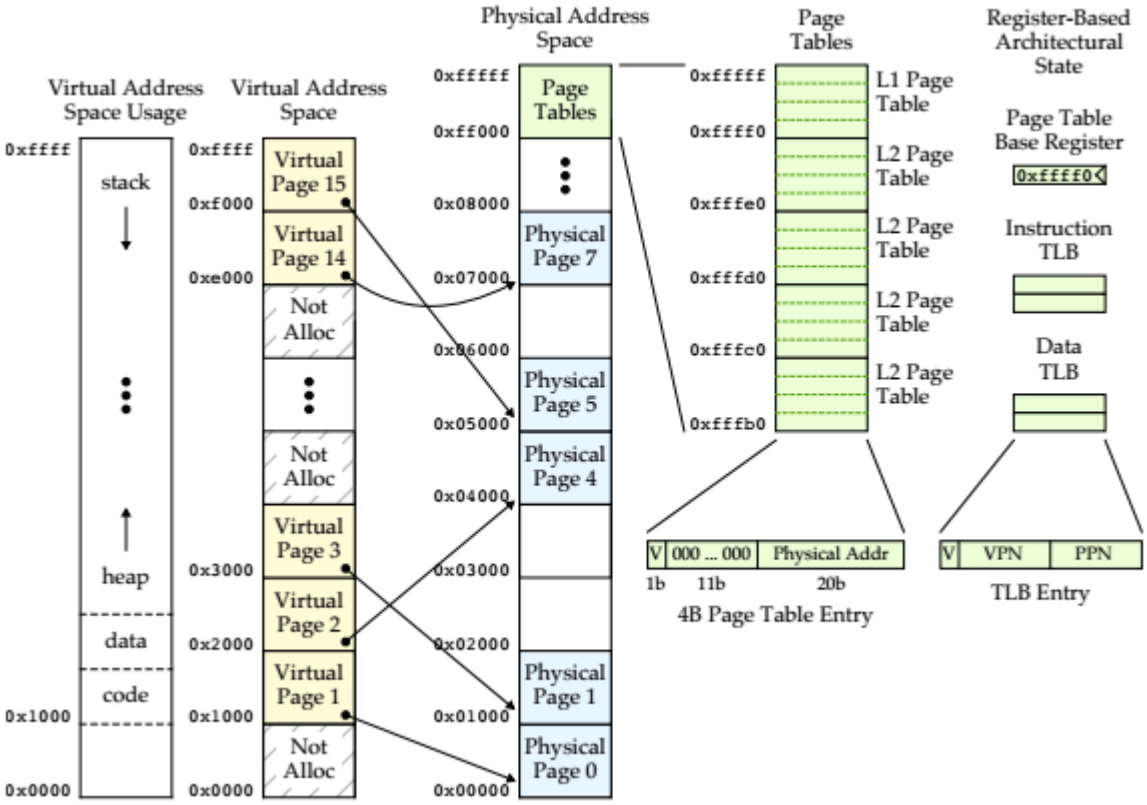
Suppose a computer uses a 40-bit virtual address with 16 KB pages and has 4 bytes per page table entry. Answer the following questions:

- What is the number of pages in the virtual address space?
- What is the maximum size of addressable physical memory in the system?
- If an average process size is 8 GB, what would be the size of a one-level page table? A two-level page table?

Question 4. Page-Based Memory Translation (28 points)

In this problem, we will be exploring a small-scale page-based memory translation system that uses 4 KB pages, a two-level page-table, and a two-entry translation-lookaside buffer (TLB). For all parts, we will assume that all addresses are byte addresses, virtual addresses are 16 bits (i.e., we have 64KB of virtual memory), and physical addresses are 20 bits (i.e., we have 1MB of physical memory). We have more physical memory than virtual memory to enable multiple programs to be resident in physical memory at the same time. While these small memory spaces are not realistic, they will help simplify the problem.

Assume program A was running for some amount of time, was context swapped by the operating system so that program B could run, and is now being context swapped back onto the processor. This means that some amount of physical memory has already been allocated to program A and the two-level page table is already initialized and stored in physical memory. However, since a context swap flushes the TLB, all entries in the TLB are now invalidated. The Figure below shows the state of the system when we restart execution of program A. Note that only five virtual pages have been allocated to program A; the remaining 11 virtual pages are unallocated. The L1 page table and each L2 page table has four entries. The page tables are stored at the very top of the physical memory address space. The L1 page table starts at address `0xffff0` and the L2 page tables are directly below the L1 page table. All page-table entries (PTEs) are assumed to be four bytes: one valid bit, 11 bits that are always zero, and 20 bits for a physical address. A page-table base register is already initialized to point to the base of the L1 page table.



4.A Two-Level Page Tables (14 points)

A two-level page table is a space-efficient way to translate virtual addresses to physical addresses. The L1 page table entries point to L2 page tables, and the L2 page table entries point to the corresponding page in physical memory. The virtual address is used to "walk" the page table. Some bits of the virtual address are used to index into the L1 page table, different bits of the virtual address are used to index into the L2 page table, and finally the page offset bits are used to index into the physical page. **Clearly indicate which bits of the virtual address are used for: (a) the page offset, (b) the virtual page number, (c) indexing into the L1 page table, and (d) indexing into the L2 page table.**

The L1 page table has four PTEs, and there are four L2 page tables each with four PTEs for a total of 20 PTEs. These page tables are stored in physical memory. **Create a table similar to the one shown on the right which shows the contents of physical memory where the page tables for program A reside.** We have provided one page-table entry for the L1 page-table to get you started.

As an aside, we probably should not have pre-allocated all five page tables! As you will see, only a subset of these page tables actually need to be allocated, so by pre-allocating all five pages we have mitigated the key advantage of a two-level page table compared to a one-level page table. Please note that if all of the PTEs in a L2 page table are invalid then then there should not be a valid PTE entry in the L1 page table pointing to this L2 page table. In other words, lets try and capture the idea that we would not really need to allocate L2 page tables for which all entries are invalid.

	Page-Table Entry	
Paddr of PTE	**Valid**	**Paddr**
0xffffc		
0xffff8		
0xffff4		
0xffff0	1	0xfffb0
0xfffec		
0xfffe8		
0xfffe4		
0xfffe0		
0xfffdc		
0xfffd8		
0xfffd4		
0xfffd0		
0xfffcc		
0xfffc8		
0xfffc4		
0xfffc0		
0xfffbc		
0xfffb8		
0xfffb4		
0xfffb0		

Contents of Physical Memory with Page Tables

Question 4.B Translation-Lookaside Buffer (14 points)

A two-level page table requires two additional memory accesses for every instruction or data memory request. A translation-lookaside buffer (TLB) can be used to cache translations and provide single-cycle mappings between virtual to physical addresses. Each TLB entry includes a valid bit, virtual page number (VPN), and physical page number (PPN). TLBs are usually flushed on a context swap. This is one step in implementing memory protection. Flushing the TLB prevents one program from accidentally using an old translation in the TLB to access physical memory allocated to a different program. Unfortunately, this results in TLB misses when a program restarts execution.

We will assume that program A was in the middle of copying a large amount of data from the stack to the heap when it was context swapped. Now that program A is restarting, it will continue copying the data from the stack to the heap. This results in the following address stream:

`0xeff4`, `0x2ff0`, `0xeff8`, `0x2ff4`, `0xeffc`, `0x2ff8`, `0xf000`,
`0x2ffc`, `0xf004`, `0x3000`, `0xf008`, `0x3004`, `0xf00c`, `0x3008`

We will be focusing on a two-entry, fully associative TLB exclusively for data memory accesses (i.e., instruction memory accesses use a different TLB). Assume the TLB uses a least-recently used replacement policy. **Create a table similar to the one shown below which shows the state of the TLB during the given sequence of data memory request transactions.** To get you started, we have filled in the table for the first transaction. Use a dash (–) to indicate an invalid TLB entry (recall that all TLB entries are initially invalid). Fill in the VPN and page offset for each transaction before updating the VPN and PPN of each TLB entry after each transaction. Indicate which accesses result in a TLB miss or hit. Indicate the total number of memory accesses for each transaction (i.e., include any accesses to the page tables and the actual access corresponding to the memory transaction). Include the total number of TLB misses and the TLB miss rate in your table. *You only need to fill in elements in the table when the value changes! Remember that the TLB entries should always reflect the state of the TLB before the corresponding transaction on that row executes!*

Transaction Address	VPN	Page Offset	m/h	Total Num Mem Accesses	TLB Way 0		TLB Way 1	
					VPN	PPN	VPN	PPN
`0xeff4`	`0xe`	`0xff4`	m	3	–	–	–	–
`0x2ff0`	...				0xe	0x07		
`0xeff8`	...							
Number of Misses =								
Miss Rate =								

TLB Contents Over Time