CS433: Computer Architecture – Fall 2019
Homework 6

Total Points: Undergraduates (24 points), Graduates (34 points)

Undergraduate students should only solve the first 4 problems. Graduate students should solve all problems.

Due Date: 11 am, Tuesday, November 19, 2019 (See course information handout for details)

Directions:

- **All students must write and sign the following statement at the end of their homework submission. "I have read the honor code for this class in the course information handout and have done this homework in conformance with that code. I understand fully the penalty for violating the honor code policies for this class." No credit will be given for a submission that does not contain this signed statement.**
- **On top of the first page of your homework solutions, please write your and your partner's name(s) and NETID(s) (indicate which name is the partner), and whether you are an undergrad or grad student. On each successive page, write your NETID.**
- **Please show all work that you used to arrive at your answer. Answers without justification will not receive credit. Errors in numerical calculations will not be penalized. Cascading errors will usually be penalized only once.**

Problem 1 [6 points]

Consider a tiny system with virtual memory. Physical addresses are 8 bits long, but only 2^7 = 128 bytes of physical memory is installed, at physical addresses 0 up to 127. Pages are 2^4 = 16 bytes long. Virtual addresses are 10 bits long. An exception is raised if a program accesses a virtual address whose virtual page has no mapping in the page table, or is mapped to a physical page outside of installed physical memory.

Here are the contents of main memory. To find the physical address of a byte, read the least significant digit from the column label and the most significant digit from the row label. For example, the shaded byte in the second row is at physical address 0x12. All entries are in hexadecimal.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x0_	4e	65	76	65	72	20	67	6f	6e	6e	61	20	67	69	76	65
0x1_	20	79	6f	75	20	75	70	0a	4e	65	76	65	72	20	67	6f
0x2_	6e	6e	61	20	6c	65	74	20	79	6f	75	20	64	6f	77	6e
0x3_	0a	4e	65	76	65	72	20	67	6f	6e	6e	61	20	72	75	6e
0x4_	20	61	72	6f	75	6e	64	20	61	6e	64	20	64	65	73	65
0x5_	72	74	20	79	6f	75	0a	4e	65	76	65	72	20	67	6f	6e
0x6_	6e	61	20	6d	61	6b	65	20	79	6f	75	20	63	72	79	0a
0x7_	4e	65	76	65	72	20	67	6f	6e	6e	61	20	73	61	79	20

Here is the page table. The virtual page number in the left column is mapped to the physical page number in the second column. Virtual page numbers are listed in binary.

Virtual page	Physical page
000	0x2
001	0x4
010	0x1

011	0x5
100	0x4
101	0x7
110	0x9

[1 point for each question]

A) List the four bytes in the word beginning at physical address 0x34.

B) How many virtual addresses refer to the first byte of the shaded word in row 0x2_? List them.

C) How many virtual addresses refer to the first byte of the shaded word in row 0x4_? List them.

D) How many virtual addresses refer to the first byte of the shaded word in row 0x6_? List them.

E) What data is returned if the program loads a word from virtual address 0x5C (01011100)?

F) What is the result if the program loads a word from virtual address 0x64 (01100010)?

Problem 2 [4 points]

Consider a system with the following processor components and policies:

- A direct-mapped L1 data cache of size 4KB (4 x 2^10 bytes) and block size of 16 bytes, indexed and tagged using physical addresses, and using a write-allocate, write-back policy
- A fully-associative data TLB with 4 entries and an LRU replacement policy
- Physical addresses of 32 bits, and virtual addresses of 40 bits
- Byte addressable memory
- Page size of 1MB

Part A [2 points]
Which bits of the virtual address are used to obtain a virtual to physical translation from the TLB? Explain exactly how these bits are used to make the translation, assuming there is a TLB hit.

Part B [2 points]
Which bits of the virtual or physical address are used as the tag, index, and block offset bits for accessing the L1 data cache? Explicitly specify which of these bits can be used directly from the virtual address without any translation.

Problem 3 [6 points]

Consider a hypothetical memory hierarchy with the following parameters. Main memory is interleaved on a word basis with four banks and a new bank access can be started every cycle. It takes 8 processor clock cycles to send an address from the cache to main memory; 50 cycles for memory to access a block; and an additional 25 cycles to send a word of data back from memory to the cache. The memory bus width is 1 word. There is a single level of data cache with a miss rate of 2% and a block size of 4 words. Assume 25% of all instructions are data loads and stores. Assume a perfect instruction cache; i.e., there are no instruction cache misses. If all data loads and stores hit in the cache, the CPI for the processor is 1.5.

Part A [2 points]
Suppose the above memory hierarchy is used with a simple in-order processor (as in Appendix C) and the cache blocks on a load or store until it completes. Compute the miss penalty and resulting CPI for such a system.

Part B [2 points]
Suppose we now replace the processor with an out-of-order processor and the cache with a non-blocking cache that can have multiple load and store misses outstanding. Such a configuration can overlap some part of the miss penalty, resulting in a lower effective penalty as seen by the processor. Assume that this configuration effectively reduces the miss penalty (as seen by the processor) by 20%. What is the CPI of this new system? What is the speedup over the system in Part A?

Part C [2 points]
Start with the system in Part A for this part. Suppose now we double the bus width and the width of each memory bank. That is, it now takes 50 cycles for memory to access the block as before, and the additional 25 cycles now send a *double word* of data back from memory to the cache. What is the miss penalty now? What is the CPI? Is this system faster or slower than that in Part B?

Problem 4 [8 points]

In a virtually indexed, physically tagged cache, the cache set to search is selected using only bits of the virtual address, so virtual-to-physical address translation can proceed in parallel with reading tags for comparison.

In the simplest design, the associativity of the cache is large enough so that the cache index and offset bits together fit entirely into the page offset bits. However, page size remains relatively fixed with architectures while cache size grows with semiconductor technology so this may require a high associativity.

Part A [2 points]
If a processor has 4KB pages and a 128KB level 1 cache, what is the minimum associativity required to use the simple virtually-indexed, physically tagged optimization?

A cache with a high associativity requires examining many ways on each access. This takes more time (even if comparisons are done in parallel) and uses more energy on each cache access, so we might want to keep associativity smaller.

Part B [2 points]

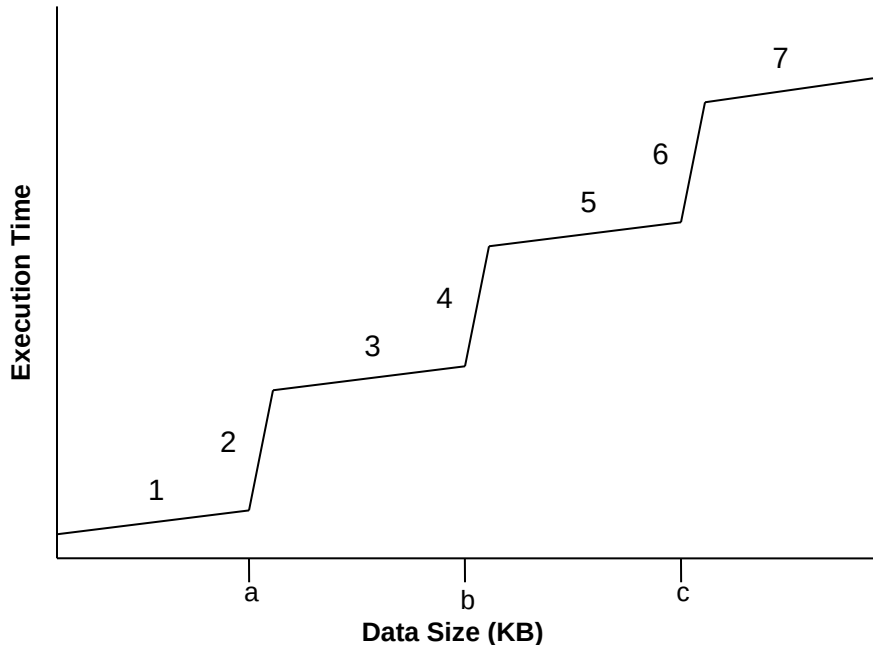
Suppose a cache simply forms a longer index using a few of the least significant bits from the virtual page number. Describe a page table and access pattern where this cache will return incorrect data.

Part C [4 points]

Assume the cache must always work correctly, but accesses as in Part B are rare enough that a performance cost is acceptable for only those accesses. Consider a processor with virtual memory and a 4KB page size, and a 128KB level 1 cache with 16 byte lines and 4 way associativity. Describe a design where the normal cache hits always see the access penalty of a 4-way associative cache but the misses and problem accesses (as in Part B) may incur a larger access penalty. Assume that it takes as long to do an address translation as it takes to access the cache in the normal case. Assume a TLB with zero miss rate. Any cache hit which uses the same virtual page number as the previous access to that cache line must be handled as a normal quick access.

Problem 5 - GRADUATE STUDENTS PROBLEM [10 points]

A student comes to you with the following graph. The student is performing experiments by varying the amount of data accessed by a certain benchmark. The only thing the student tells you of the experiments is that their system uses virtual memory, a data TLB, only one level of data cache, and the data TLB maps a much smaller amount of data than can be contained in the data cache. You may assume that there are no conflict misses in the caches and TLB. Further assume that instructions always fit in the instruction TLB and an L1 instruction cache.



Part A [7 points]

Give an explanation for the shape of the curve in each of the regions numbered 1 through 7.

Region in graph	Explanation
1	
2	
3	
4	
5	
6	
7	

Part B [3 points]

From the graph, can you make a reasonable guess at any of the following system properties? If so, what are they? If not, why not? Explain your answers. (Note: your answers can be in terms of *a*, *b*, and *c*).

(i) Number of TLB entries

(ii) Page size

(iii) Physical memory size

(iv) Virtual memory size

(v) Cache size