CS 433: Computer Architecture – Fall 2021
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: November 19, 2021 at 10:00 pm CT
(See course information slides for more 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 solution, please write your name and NETID, your partner's name and NETID, and whether you are an undergrad or grad student.**
- **Name your homework solution file as *firstname_lastname_hw6.pdf***
- **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.**
- **See course information slides for more details.**

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.
0x65, 0x72, 0x20, 0x67

B) How many virtual addresses refer to the first byte of the shaded word in row 0x2_? List them.
Only virtual page 000 maps to physical page 0x2, so there is only one address: 0x04.

C) How many virtual addresses refer to the first byte of the shaded word in row 0x4_? List them.
Virtual pages 001 and 100 map to physical page 0x4, so there are two: 0x18 and 0x48.

D) How many virtual addresses refer to the first byte of the shaded word in row 0x6_? List them.
No virtual addresses map to physical page 0x6, so there are zero addresses.

E) What data is returned if the program loads a word from virtual address 0x5C (01011100)?
Virtual address 0x5C maps to physical address 0x7C, the data is: 0x73, 0x61, 0x79, 0x20

F) What is the result if the program loads a word from virtual address 0x64 (01100010)?
Virtual address 0x64 maps to physical address 0x94. However, this is outside the range of the installed memory, so an exception occurs.

"Exception" is sufficient. ½ credit for saying data is loaded from physical address 0x94.

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.

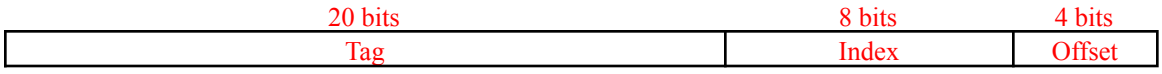
Solution: The virtual address is 40 bits long. Because the virtual page size is 1 MB = 2^20 bytes, and memory is byte addressable, the virtual page offset is 20 bits. Thus, the first 40 - 20 = 20 bits are used for address translation at the TLB. Since the TLB is fully associative, all of these bits are used for the tag, i.e., there are no index bits. When a virtual address is presented for translation, the hardware first checks to see if the 20 bit tag is present in the TLB by comparing it to all other entries simultaneously. If a valid match is found (i.e., a TLB hit) and no protection violation occurs, the page frame number is read directly from the TLB.

Grading:
1 point for noting that the first 20 bits are used for address translation.
1 point for noting that all of these bits are used as a tag and compared against the tag bits in all TLB entries.

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.

Solution: Since the cache is physically indexed and physically tagged, all of the bits for accessing the cache must come from the physical address. However, since the lowest 20 bits of the virtual address form the page offset and are therefore not translated, these 20 bits can be used directly from the virtual address. The remaining 12 bits (of the total of 32 bits in the physical address) must be used after translation. Since the block size is 16 bytes = 2^4 bytes, and memory is byte addressable, the lowest 4 bits are used as block offset. Since the cache is direct mapped, the number of sets is 4 KB / 16 bytes = 2^8. Therefore, 8 bits are needed for the index. The remaining 32 - 8 - 4 = 20 bits are needed for the tag. As mentioned above, the index and offset bits can be used before translation while the tag bits must await the translation for the 12 uppermost bits.



Grading:
1 point for correctly specifying the tag, index, and offset bits. ½ point for specifying any two correctly, no points for specifying only one of them correctly.
1 point for correctly specifying that the lowest 20 bits do not need translation and the upper 12 bits must be translated.

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.

Solution:
Miss penalty = 8 + 50 + (25 * 4) = 158 cycles
CPI = 1.5 + (0.25 * 0.02 * 158) = 2.29

Grading:
1 point for miss penalty calculation.
1 point for CPI calculation.

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?

Solution:
Effective miss penalty = 0.80 * 158 = 126 cycles
CPI = 1.5 + (0.25 * 0.02 * 126) = 2.13
Speedup (over A) = 2.29 / 2.13 = 1.08

Grading:
1 point for miss penalty and CPI calculation.
1 point for speedup statement.

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?

Solution:
Miss penalty = 8 + 50 + (25 * 2) = 108 cycles
CPI = 1.5 + (0.25 * 0.02 * 108) = 2.04
Speedup (over B) = 2.13 / 2.04 = 1.04, so this system is slightly faster than that in part B.

Grading:
1 point for miss penalty and CPI calculation.
1 point for speedup statement. (Note: only need to answer whether B or C is faster.)

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.

Solution: The total number of index+offset bits needed to address a direct mapped 128 KB cache is log2(2^17) = 17. Only 12 of these fit within the offset bits for a 4 KB page. Therefore, 5 are not in the page offset. If the cache associativity is 2^5, then the number of bits for index + offset will be reduced by 5, thus getting all the bits within the page offset. Therefore, we need an associativity of 32.

Grading: There are several ways to solve this problem, including directly applying the formula discussed in class. All reasonable approaches will get full credit.

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.

Solution:
If multiple virtual pages are mapped to the same physical page, say virtual pages 0 and 1 map to physical page 0, then two virtual addresses can map to the same physical address, but have different indices. If a program writes 12 value to one address, then 25 to the other, then reads from the first address it will receive a 12 from the cache, but should see 25.

Grading:
1 point for setting up a page table with aliasing.
1 point for an access pattern that demonstrates the error.

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.

Solution:

To look up a cache line, it takes the 11-bit index from bits 4–14 of the address, which includes the low 3 bits of the virtual page number. It reads tags from the 4 ways in this set, and compares them with tag bits from the physical address, after the read and address translation is complete. If these bits match we have a hit in the fast case. If these bits do not match, then we need to check the other 7 sets corresponding to possible other values for the first 3 bits, in case this data was previously accessed through a different, aliased virtual address. If the data is found , we have a cache hit in the slow case and the data is copied into the first set we checked and invalidated in other sets, to ensure another access through this virtual address will hit quickly. If the data is not found, we have a cache miss and the line must be requested from the next largest level in the memory hierarchy. Once the line is returned, it is stored in the set which was searched first.

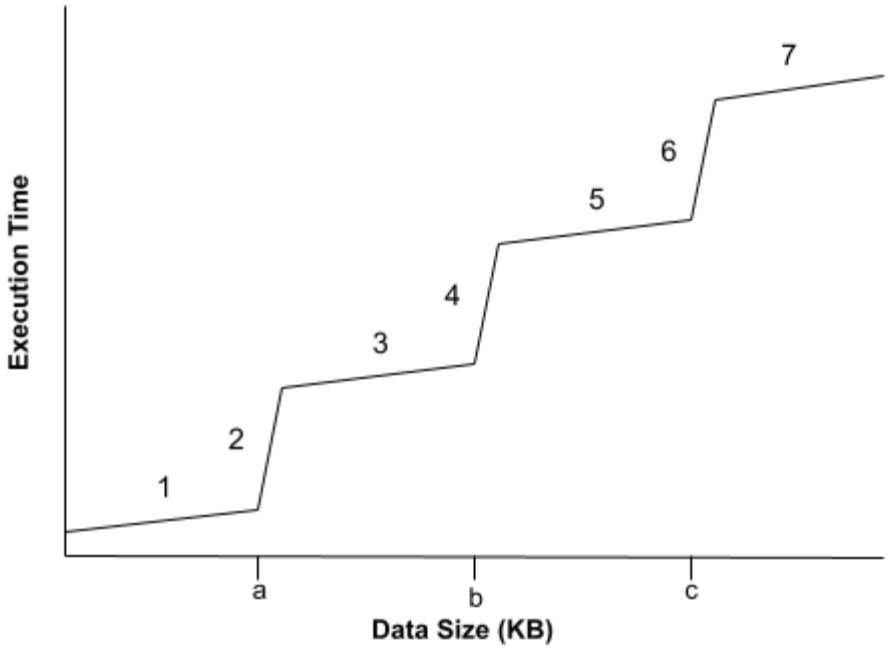
Grading:

1 point for ensuring the cache never has multiple valid lines for a single physical address.
1 point for accessing only 4 lines on a fast hit.
2 points for correct operation.

NOTE: ONLY GRADUATE STUDENTS SHOULD SOLVE THIS PROBLEM

Problem 5 [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	Execution time slowly increases (performance decreases) due to increasing data size but remains at a roughly similar level.
2	At this point, the TLB overflows and execution time sharply increases to handle the increased TLB misses.
3	Execution time again slowly increases due to increasing data size and plateaus at a higher level than before due to overhead from TLB misses.
4	At this point, the data cache overflows, causing a high frequency of cache misses and execution time again sharply increases.
5	Execution time again slowly increases due to increasing data size and plateaus at a high level due to overhead from retrieving data directly from main memory due to cache misses.
6	Execution time again sharply increases due to physical memory filling up and thrashing occurring between disk and physical memory.
7	Execution time is very high due to overhead from TLB misses, cache misses and virtual memory thrashing. It is slowly increasing due to increasing data size.

Grading:
1 point for the explanation for each region.

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
No reasonable guess, since it depends on the page size.

(ii) Page size
No reasonable guess.

(iii) Physical memory size
Reasonable guess is *c* KB, since this is the point at which the execution time shows significant degradations.

(iv) Virtual memory size
No reasonable guess.

(v) Cache size
Reasonable guess is *b* KB, since this is the point at which the execution time shows significant degradation.

For iii and v, *c* and *b* KB are actually only upper bounds, since the actual size of these structures depends on the temporal and spatial reuse in the access stream. (The actual size depends on a property known as the working set of the application.)

Grading:
⅓ point for (i).
⅓ point for (ii).
1 point for (iii).
⅓ point for (iv).
1 point for (v)