Computer Architecture and Engineering

CS152 Quiz #2

March 7th, 2016

Professor George Michelogiannakis

Name: _____<ANSWER KEY>_____

This is a closed book, closed notes exam.
80 Minutes. 15 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 other students who have not taken the quiz. If you have inadvertently been exposed to a 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	________________	29 Points
Question 2	________________	30 Points
Question 3	________________	18 Points
Question 4	________________	22 Points
TOTAL	________________	100 Points

Name ____________________________________

Question 1: Doppleganger Cache [29 points]

For this question, we will consider a new cache design that was proposed in 2015, named the doppleganger cache. The doppleganger cache is motivated by approximate computing and essentially implements lossy compression of data in cache lines, where multiple tags share the same cache line. A simplified diagram is as follows:

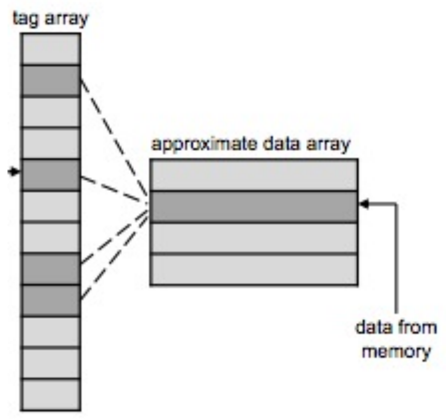

Figure 1: Doppleganger cache overview. "Data from memory" refers to incoming data in case of a cache miss.

As shown, multiple tag array entries point to the same data array line. This results in different memory addresses (different tag array entries) being serviced by the same data array line. The benefit of this design is reducing overall cache area by reducing the size of the data array, but not the coverage (entries) in the cache because the tag array preserves its number of entries.

For a more detailed view, you can use Figure A-1 in appendix A, with the modification that the data array has fewer entries than shown in the original direct-mapped cache.

For this question, we consider a doppleganger cache with 128 32-byte lines (4KB data array and blocks of 32-bytes or eight 4-byte words) and 512 tag array entries. Tag array entry I maps to data array line I % 128. For example, tag arrays 0, 128, 256, and 384 all map to (use) data array entry 0. You can assume direct mapping.

When a read miss occurs, data arrives from memory into the appropriate location of the data array as determined by the tag array entry that was used. If the data array already contains a valid entry for a different tag array, the new data are "merged" with lossy compression. Therefore, the end result is that both the previous tag array and the new tag array entry now point to the same data array entry.

Name ___________________________________

Q1.A Three C's of Cache Misses [3 points]

Compare the 3C's of cache misses for the doppleganger cache as provided, to a 256-entry direct-mapped cache. State which of the 3C's of cache misses increase, decrease, or remain the same for the doppleganger cache and why. How do you expect the cache miss rate to change compared to the direct mapped cache? Explain your reasoning to receive credit.

Conflict: Should not change because the size of the tag array remains the same.
Cold: Does not change.
Capacity: Doppleganger cache has fewer capacity misses because it has a longer tag array.
Therefore, doopleganger cache should have fewer misses overall.

Q1.B Read Lookup [4 points]

Now lets assume that any tag array entry can map to **any** data array entry (line). That is, tag array entries are extended with a field "data array entry" to point to the data array entry the tag refers to. Now the cache needs to not only hit or miss like we have seen for each access, but also determine which data array entry to access in the case of a hit.

Name one advantage of this modified design. Name one disadvantage. Explain.

One advantage is that two tag arrays do not have to map to the same data array entry. Therefore, the cache gains more flexibility in providing separate data for different tag entries.

One disadvantage is that the cache has to wait for the result of the tag array before accessing the data array, or access the entire data array in parallel much like a fully set-associative cache.

Name ______________________________

Q1.C Write Policy [5 points]

Back to the original doppleganger design in the description and Q1.A (ignore the modification in Q1.B), would you choose a write back/write allocate policy or a write through/no allocate policy in the doppleganger cache? Think of any different steps compared to the standard direct-mapped cache and how one write can affect multiple tag entries. Is there any functional (i.e., different data for the same read) difference between write back and write through in this cache?

For write back, we need to create a new entry similar to a read miss with the new data in case of a write miss. Write back requires a valid bit in every tag array. An important difference with the "standard" direct mapped cache is that a write for one tag entry can affect the data of another tag array entry (or entries) if they point to the same data array entry. For instance, if tag array A already exists and a write miss occurs, tag array B will become valid and change A's data if both A and B point to the same data array entry.

With a write through policy, the above "data interference" is avoided. Therefore, in the above example, tag A will still point to its original data. This is a functional difference.

Q1.E Cache Parameters [4 points]

Fill out the following table showing how many **bits** are in each of the caches with the assumptions in the problem description (keep ignoring Q1.B), the direct-mapped cache with 512 entries, and for a write back/allocate policy. Assume 32-bit addresses, byte addressable addresses, and 32-byte blocks.

	Unmodified Direct-mapped	Doppleganger
# tag array bits	2 bits byte offset, 3 bits for block offset, 9 bits for the index, so 32-12 = 18 tag. Add a valid bit. 20 * 512 = 10240	Same for direct mapped (left).
# data array bits	512 * 32 bytes = 131072 Also 1 dirty bit (512 total) So 131072 + 512 = 131584	128 * 32 bytes = 32768 Also 1 dirty bit (512 total) So 32768 + 512 = 33280

Name ________________________________

Q1.F TLB and virtual indexing [8 points]
Now lets assume that we want to all a TLB *in parallel* with the cache. To make this happen, do we need a virtual or physical tag, and a virtual or physical index?

Virtual index, physical tag.

Using your answer above, 4KB pages, and the cache organization in this problem, is it possible to have aliasing in this cache? In other words, could there be a situation where multiple copies of the same physical page exist in the cache? Describe why not, or if there is such a case how it can happen *and how to solve it*.

Page number (virtual or physical)	Page offset

(the index uses bits starting from the least significant after the cache block offset bits)

If the index bits are many more than the page offset, the most significant part of the index will be part of the translated portion of the virtual address. So two different virtual addresses point to different tag lines in the cache. If they map to the same physical page, we can have two copies of the physical page.

One way to solve this is to change this cache's mapping of tag array entry to data array mapping such that tags that would otherwise duplicate the physical page now point to the same data array entry instead. Other ways such as those we talked in the lecture are also acceptable.

Q1.G Two-way set associative [5 points]
Describe a modification to the doppleganger cache that makes it two-way set associative but keeps one data array. Don't forget to describe how the data array gets accessed. You can change the mapping from tag to data entries that this problem provides.

The key is the tag array. We can make that two-way set associative like Appendix B by having two tag entries per tag array line. Each of the two tags can point to a different data array entry, or the same one.

Question 2: Page Tables [30 points]

For this question, you are going to study a 32-bit machine that uses a hierarchical page table scheme. Assume that each page table segment can hold 512 entries. In other words, the first level of the page table has 512 entries. Each entry points to another segment that can hold 512 entries, etc. Virtual addresses are 32 bits long, This machine uses 4KB pages. Assume that the operating system is smart enough to only allocate the minimal needed physical memory for each program. In all questions, **explain** your answers.

Q2.A Bit indices [6 points]

Given the above information, and assuming you want to use as many bits as possible of the address, how many bits for the page offset? How many levels for the hierarchical page table? How many index bits per level?
In your above answer, it is possible that some bits cannot be assigned to anything given the constraints. What can you change from the constraints to assign all bits?

12 bits for the page offset. 9 bits for each page table segment. Therefore it's a two-level hierarchical page table with 2 left over (unassigned) bits. To assign those bits we can increase the page size to 16KB. Or we can increase each page table segment.

Q2.B DRAM Size [4 points]

With your original answer from Q2.A (without the modification to fit the unassigned bits), what is the maximum DRAM size we can have in this system?

512^2 entries in the last (second) level of the page table. 4KB each entry. Therefore, 1073741824 bytes (roughly 1GB).

Q2.C Time for a TLB Refill [6 points]

Now lets assume that the latency to access a page table segment is <number of entries> * 1ns. So for 512 entries it is 512ns. Also assume that there is a fixed penalty (for example for pointer chasing) to go from one level to the next that is 10ns. Calculate the latency to walk the page table you designed in Q2.A. Can you propose a modification to this page table structure that has a lower latency but maps to the same or more amount of DRAM (i.e., no less)?

512+10+512=1034ns for the page table of Q2.A.
Having a three-level page table with 256-entry page table segments is faster and maps to more DRAM. Increasing the page size has the same effect.

Q2.D Page Table Size [4 points]

What is the size of the page table if a program allocates the maximum it is allowed to (assume the page table structure from Q2.A. Assume a page table entry in any level is 8 bytes.

What is the size of the page table if a program allocates only one page?

There are 512 + 1 = 513 page table segments. Each has 512 entries of 8 bytes each: 2101248

If a program allocates only one page, the first level still needs 512 entries (one segment), but only one page table segment exists in the second level. So 1028 entries of 8 bytes each: 8192

Q2.E Reducing Page Walk Size [10 points]
Based on the page table structure of Q2.A, assume you want to speed up the translation of some important pages, such as pages belonging to the operating system. Let's assume that pages 0 to 127 have to be resolved with the minimum latency possible, that is smaller than all other pages. Pages from 128 and upward are to be translated with higher latency.

Describe how you would do this while not decreasing the amount of DRAM we can map in this machine (similar to Q2.C). You can modify an existing structure or add a new structure, as long as you describe what the functionality should be. However, make sure you maintain the same first-level structure (i.e., there still is one first page table segment).

For the modified structure you propose, calculate the latency to translate pages 0 to 127, and then pages 128 and upwards. You can use the assumption of <number of entries> * 1ns for any array structure you propose. Also, to go from one level of a hierarchical page table to another, add 10ns.

There are many possible answers. One easy way is to modify the first level such that it holds the 128 entries that are to be translated fast with no need to access further levels of the page table. So entries 0 to 127 will contain a translation in the first level. The other entries in the first level will contain a pointer to the second level. However, in order to be able to map to the same amount of DRAM, we now need three levels. So the fast pages are translated in 512ns (single level access), whereas other pages are translated in 512+10+512+10+512ns = 1536ns.