Objective: In this recitation, you work through some examples of cache designs and access patterns.

Complete as much of this as you can during recitation. If you run out of time, please complete the rest at home.

1. Cache performance
Suppose you have a three-level memory hierarchy. The L1 cache has a latency of 1 cycle and hits 80% of the time; the L2 cache has a latency of 6 cycles and hits 90% of the time; the L3 cache has a latency of 30 cycles and hits 95% of the time. The latency of memory is 200 cycles.

What is the average access time of L3 cache? L2 cache? L1 cache?

2. Cache tracing
You have a 16-bit machine with a 512B cache that is 4-way set-associative. Blocks are 16B.

1) How many frames does it have?
2) How many sets?
3) Divide up the 16-bit address into its three fields, for purposes of accessing this cache. How many block offset bits are there? How many set index bits are there? How many tag bits are there?
4) What address will have block tag 1, index 2, and block offset 3? You may find it easier to express your answer in hex.
5) Starting from a cold cache, suppose we access addresses \{0, 1, 2, 3\}. Will the last access be a hit or miss? If a miss, will it be a compulsory, conflict, or capacity miss?
6) Starting from a cold cache, suppose we access addresses \{0, 256\}. Will the last access be a hit or miss? If a miss, will it be a compulsory, conflict, or capacity miss?
7) Starting from a cold cache, suppose we access addresses \{0, 256, 512, 0\}. Will the last access be a hit or miss? If a miss, will it be a compulsory, conflict, or capacity miss?
8) Starting from a cold cache, suppose we access addresses \{0, 256, 512, \ldots, 8192, 0\}\(^1\) (34 total accesses). Will the last access be a hit or miss? If a miss, will it be a compulsory, conflict, or capacity miss?

---

\(^1\) Updated 2017-11-2: The previous version ended on address 8192, which made it a compulsory miss (we’ve never seen 8192’s block before). We now end on 0, to get a more interesting kind of miss. Thanks for Brian Nieves for the correction.
Hint: As before, recall this slide summarizing the cache arithmetic:

**Cache structure math summary**

- Given capacity, block_size, ways (associativity), and word_size.
- Cache parameters:
  - num_frames = capacity / block_size
  - sets = num_frames / ways = capacity / block_size / ways
- Address bit fields:
  - offset_bits = log₂(block_size)
  - index_bits = log₂(sets)
  - tag_bits = word_size - index_bits - offset_bits
- Numeric way to get offset/index/tag from address:
  - block_offset = addr % block_size
  - index = (addr / block_size) % sets
  - tag = addr / (sets*block_size)