CSI 250 - Cache Exercise

1. Given the following pieces of information:
   262144 bytes of main memory
   65536 bytes of cache
   8 byte blocks
   (a) How many bits does it take to describe an address in main memory.
   (b) How many blocks are there in main memory?
   (c) How many blocks of memory does the cache hold?
   (d) How many bits does it take to address a block in the cache?
   (e) In a direct-mapped cache, how many bits is the cache tag?
   (f) In a 2-way set-associative cache, how many sets are there in the cache?
   (g) In a 2-way set-associative cache, how many bits is in the cache tag?
   (h) In a direct-mapped cache whose cache is initially invalid, what is the hit ratio of the following byte accesses. The following are byte addresses, not blocks. The left column accesses happen first. The second column begins with the 17th byte access.

11 0 000 0000 0000 0 000	11 0 000 0000 0000 0 000
11 0 000 0000 0000 0 001	10 0 000 0000 0000 0 000
11 0 000 0000 0000 0 010	11 0 000 0000 0000 0 001
11 0 000 0000 0000 0 011	10 0 000 0000 0000 0 001
11 0 000 0000 0000 0 100	11 0 000 0000 0000 0 010
11 0 000 0000 0000 0 101	10 0 000 0000 0000 0 010
11 0 000 0000 0000 0 110	11 0 000 0000 0000 0 011
11 0 000 0000 0000 0 111	10 0 000 0000 0000 0 011
11 0 000 0000 0000 1 000	11 0 000 0000 0000 0 100
11 0 000 0000 0000 1 001	10 0 000 0000 0000 0 100
11 0 000 0000 0000 1 010	11 0 000 0000 0000 0 101
11 0 000 0000 0000 1 011	10 0 000 0000 0000 0 101
11 0 000 0000 0000 1 100	11 0 000 0000 0000 0 110
11 0 000 0000 0000 1 101	10 0 000 0000 0000 0 110
11 0 000 0000 0000 1 110	11 0 000 0000 0000 0 111
11 0 000 0000 0000 1 111	10 0 000 0000 0000 0 111

   (i) Using the access sequence from above, what is the hit ratio with a 2-way set-associative cache.
   (j) If a cache hit is one clock cycle and a cache miss is 40 cycles, what is the average memory access time in the previous two scenarios. What would the average be without a cache?
2. Answer the first 7 questions from above for the following parameters:
   Memory size = 32 megabytes. Treat a megabyte as 2^{20}
   Cache size = 64 kilobytes. Treat a kilobyte as 2^{10}
   Block size = 32 bytes
3. If the cache hit ratio is 90% and cache hits are 1 cycle, but cache misses are 40 cycles, what is the average memory access time?
4. 3 layers of cache are available. The first cache has 1 cycle access but it only has a 50% hit ratio. Access to the second cache is 10 cycles with 75% hit ratio. The third has 20 cycles access with a 90% hit ratio. Access to main memory happens the rest of the time at 40 cycles.
   (a) If only the first cache is used, what is the average memory access time?
   (b) If only the second cache is used, what is the average memory access time?
   (c) If only the third cache is used, what is the average memory access time?
   (d) Extra Credit: What is the average memory access time if all the caches are used?

1 Solutions

1. Given the following pieces of information:
   262144 bytes of main memory 65536 bytes of cache 8 byte blocks

   (a) Full address in main memory - 262144 = 2^{18} - answer: 18 bits

   (b) Blocks in main memory? 262144/8 = 32768 blocks

   (c) Blocks in cache: 65536 / 8 = answer: 8192 blocks

   (d) bits to address 8192 blocks - 8192 = 2^{13} answer: 13 bits

   (e) direct mapped cache tag bits: Every main memory block (32768) must map to a cache block (8192). So each cache block gets: 32768/8192 = 4 main memory blocks mapped to it. $4 = 2^2$. And thus the tag is answer: 2 bits. We can get this answer another way. Addresses in main memory are 18 bits long. 3 bits (called the offset) are required to know where you are in the block because there are 8 byte blocks. 13 bits are needed to know which cache block is mapped to. This leaves only 2 bits left over for the cache. Answer: 2 bits

   (f) Since the cache has 8192 blocks and sets are of size 2, there are 8192/2 sets = answer: 4096

   (g) Given 4096 sets = 2^12. Every main memory block maps to a set. Each set gets 32768/4096 = 8 main memory blocks mapped to it. $8 = 2^3$. So answer: 3 bits. In another way. It requires 12 bits to determine which set an address belongs to and 3 bits to know where in the block you belong. This leaves a tag of 3 bits long (18 - 12 - 3 = 3).

   (h) The first 8 accesses all have the same middle 13 numbers. This means they map to the same cache line. The first 2 numbers are the tag. The last 3 numbers are the offset to get which byte we care about within a block. So the middle 13 numbers tell us which cache line we're mapping to. Since all the tags are the same and the next 13 numbers are the same too, these refer to the same main memory block. So the first access is a miss and the next seven are hits. The same analysis is true of the next 8. The first is a miss. The tag is the same as before, but it is in another cache line, so that doesn't matter. So in this first column of accesses, things are progressing fairly well and normally. There's a sequential access through memory. 2 cache misses and 14 cache hits. However, in the second column, things change. The middle 13 bits always match but the first two bits - the tags are alternating. The offset bytes are changing, but they don't affect our cache. This could happen because you're copying one array to another byte by byte. but in the process, you're alternating between two different places in memory and disrupting the cache. These arrays are perfect spaced to do this. In any case, the first access is already in the cache, that's a cache hit. But the second has a valid entry in that cache line and thus it replaces the cache line there with this one so it is a miss. And this same logic follows for the next 14 cache accesses. So in the second column there is 1 hit (the first access) and every other access forces a replacement of the hold with the new. Causing 15 cache misses. In total in this block there are 15 cache hits and 17 cache misses. (Quite horrible performance). Hit ratio = 15 hits / 32 accesses which is roughly 46.9%. Not a great hit ratio. You can see that "normal" and locality principle obeying accesses do okay, but the second column shows how the cache is actually useless in a very certain situation.

(i) In set associative, the first 3 numbers are the tag. Still the same situation applies for the first 17 accesses. In the 18th access (10 0000 0000 0000 0 000), this caused a cache miss and replacement in the direct-mapped cache. But in the set-associative cache, we have another block in the set that we can put this memory into. So, in the second block of the set "000 0000 0000 0 000" we can place this block with the tag "100" in it. That set also has a block with the tag "110" in it. So this access is a miss. But all subsequent accesses are hits. So, this is 3 misses and 29 hits. The hit ratio is 29/32 which is roughly 90.6%. This is actually the best that this short run of accesses can do because those three blocks must be loaded into the cache.

(j) Direct Mapped Cache: 15 accesses of 1 cycle and 17 of 40 cycles gives 15*1+17*40 = 695 cycles total. On average this is 695/32 or 21.7 cycles. In the 2-way set-associative cache there are 29 accesses of 1 cycle and 3 of 40 cycles giving: 29*1+3*40 = 149 cycles total. On average this is 149/32 or 4.65 cycles. Without a cache, access time would be 40 cycles on average!

2. (a) 32mb = 2^{25} = 25
   (b) $2^{20}/2^5 = 2^{20}$
   (c) $2^{16}/2^5 => 2^{11}$
   (d) 11
   (e) 25-11-5 = 9 bit tags
   (f) $2^{11}/2 = 2^{10}$
   (g) 25-10-5 = 10 bit tags

3. 90% of the time they're 1 cycle. The other 10% they're 40 cycles. Average memory access time is then 0.9*1 + 0.1 *40 = 4.9 cycles. (10 times speed up over not using a cache).

4. These parts all have the same computation as the previous question.
   (a) 0.5 * 1 + 0.5 * 40 = 20.5 cycles access time on average
   (b) 0.75 * 10 + 0.25* 40 = 17.5 cycle access time on average (Choose this one!!)
   (c) 0.9 * 20 + 0.1 * 40 = 22 cycle access time on average
   (d) If we could use all three, then we have a series of degrading miss chances. Half will be accessed in 1 cycle = 0.5*1. The other half hit the rest of the memory hierarchy. The hit ratio of the second cache was 0.75 alone. But 0.5 of those were hits to the first cache. So, only 0.75-0.5 = 0.25 or 25% of the total accesses will be hits in the second cache. Thus, 0.25*10. Of the remaining 25% that are misses, some of those are caught by the third cache. 0.9-0.75 = 0.15. Thus the third cache contributes 0.15*20 to the average. The last 10% are hits on the main memory. So total time is:
   $0.5 * 1 + 0.25 * 10 + 0.15 * 20 + 0.1 * 40 = 10$ clock cycles on average.
   Hopefully this number really sells how the memory hierarchy can really work, even if the performance at each level doesn't seem like it may contribute much and even in a rather poor cache performance situation. And the results look way better than any of the caches alone.