CprE 381 Homework 10

[Note: The homework below focuses on the details of caching and cache design. Once you have completed this homework you should be able to motivate the need for caches using specific program examples. You should also be able to configure and hand-simulate a range of caches.]

1. Cache Configuration and Simulation

   In this problem we will consider several cache designs for a processor implementing the MIPS ISA *[Note that this has implications needed to answer the below questions].* Assume that the block offset is four bits and the index is four bits.

   a. What is the cache block size in bytes? words? double words?

      Since the block offset is 4 bits and MIPS is byte-addressable, blocks are $2^4 = 16$ bytes or 4 words or 2 double words.

   b. How many sets does this cache have? *[Hint: note that both direct-mapped and fully-associative caches can be considered to have sets.]*

      Since the set index is 4 bits, there are $2^4 = 16$ sets.

   c. Record both the amount of data and meta-data (in bits) this cache holds if it is direct-mapped, two-way set associative, and four-way set associative.

      If there are 16 sets and 16 data bytes per set each way requires 16*16=256 bytes of data. Tags are 32-4-4=24 bits and each block also has an invalid and a dirty bit. Therefore there are 16*26=416 bits of meta-data or 52 bytes of meta-data per way. A direct-mapped cache has one way so it has 256B of data and 52B of meta-data. A two-way set-associative cache has two ways so it has 512B of data and 104B of meta-data. A four-way set-associative cache has two ways so it has 1024B of data and 208B of meta-data. Note that this does not include any meta-data needed for the replacement policy in the set-associative caches.

   d. Simulate (i.e., complete the following table) the direct-mapped and four-way set associative cache with respect to the following series of memory accesses. Provide the reason for each miss. Assume the caches have no valid entries to begin with and use a least-recently-used (LRU) replacement policy.

Memory Access	Direct-Mapped	4-way Set Associative
`0x1001FEA0`	Miss (invalid)	Miss (4x invalid)
`0x1001EFA4`	Miss (tag mismatch)	Miss (tag mismatch; 3x invalid)
`0x1001FEA8`	Miss (tag mismatch)	Hit (way 0)
`0x100100A0`	Miss (tag mismatch)	Miss (2x tag mismatch; 2x invalid)
`0x100100B0`	Miss (invalid)	Miss (4x invalid)
`0x100100C0`	Miss (invalid)	Miss (4x invalid)
`0x10011FA1`	Miss (tag mismatch)	Miss (3x tag mismatch; 1x invalid)
`0x1001EEA2`	Miss (tag mismatch)	Miss (4x tag mismatch)

0x1001EFAF	Miss (tag mismatch)	Miss (4x tag mismatch)
0x100100A2	Miss (tag mismatch)	Hit (way 2)

Write the valid entries in the final state of each cache using the format <set#, way#, tag>. What was the hit rate of each cache?

Direct-mapped:
<Set A, Way 0, 0x100100>
<Set B, Way 0, 0x100100>
<Set C, Way 0, 0x100100>

4-way set-associative:
<Set A, Way 0, 0x1001EF>
<Set A, Way 1, 0x1001EE>
<Set A, Way 2, 0x100100>
<Set A, Way 3, 0x10011F>
<Set B, Way 0, 0x100100>
<Set C, Way 0, 0x100100>

The direct-mapped cache has a hit rate of 0, while the four-way set-associative cache has a hit rate of 2/10 = 0.2.

2. Cache Write Nuances
   a. ZyBooks (Textbook) 5.19.6 all (a-c)
   b. ZyBooks (Textbook) 5.19.7 all (a-b)

5.19.6 a

The L1 cache has a low write miss penalty while the L2 cache has a high write miss penalty. A write buffer between the L1 and L2 cache would hide the write miss latency of the L2 cache. The L2 cache would benefit from write buffers when replacing a dirty block, since the new block would be read in before the dirty block is physically written to memory.

5.19.6 b

On an L1 write miss, the word is written directly to L2 without bringing its block into the L1 cache. If this results in an L2 miss, its block must be brought into the L2 cache, possibly replacing a dirty block which must first be written to memory.

5.19.6 c

After an L1 write miss, the block will reside in L2 but not in L1. A subsequent read miss on the same block will require that the block in L2 be written back to memory, transferred to L1, and invalidated in L2.

5.19.7 a

When the CPI is 2, there are, on average, 0.5 instruction accesses per cycle. 0.3% of these instruction accesses cause a cache miss (and subsequent memory request). Assuming each miss requests one block, instruction accesses generate an average of 0.5*.003*64 = 0.096 bytes/cycle of read traffic.
25% of instructions generate a read request. 2% of these generate a cache miss; thus, read misses generate an average of 0.5*0.25*0.02*64 = 0.16 bytes/cycle of read traffic.
10% of instructions generate a write request. 2% of these generate a cache miss. Because the cache is a write-through cache, only one word (4 bytes) must be written back to memory; but, every write is written through to memory (not just the cache misses). Thus, write misses generate an average of 0.5*0.1*4 = 0.2 bytes/ cycle of write traffic. Because the cache is a write-allocate cache, a write miss also makes a read request to RAM. Thus, write misses require an average of 0.5*0.1*0.02*64 = 0.064 bytes/cycle of read traffic.
Hence: The total read bandwidth = 0.096 + 0.16 + 0.064 = 0.32 bytes/cycle, and the total write bandwidth is 0.4 bytes/cycle.

5.19.7 b

The instruction and data read bandwidth requirement is the same as in Part a. With a write-back cache, data is only written to memory on a cache miss. But, it is written on every cache miss (both read and write), because any line could have dirty data when evicted, even if the eviction is caused by a read request. Th us, the data write bandwidth requirement becomes 0.5*(0.25 + 0.1)*0.02*0.3*64 = 0.0672 bytes/cycle.

3. Cache Hierarchy Performance
    a. ZyBooks (Textbook) 5.19.10 all (a-g)

5.19.10 a

P1	1.515 GHz
P2	1.11 GHz

Cycle time can be equated to the L1 hit time:

Cycle time = L1 Hit Time

Then, to evaluate the clock rate we perform the following division:

Clock rate = 1 / Cycle time

Doing this for both P1 and P2 yields the clock rates shown in the table above.

5.19.10 b

P1	6.26 ns	9.56 cycles
P2	5.10 ns	5.68 cycles

Recall that AMAT (Average Memory Access Time) refers to the average time that it takes to access memory by accounting for both hits and misses.

The formula is as follows:

AMAT = Hit time + (Miss rate * Miss penalty)

This can also be given in the context of cycles:

AMAT = Cycles for a hit + (Miss rate * Miss penalty)

AMAT calculation for P1:

Hit time for P1 = 0.66 ns
Miss rate for P1 = 8.0 %
Miss penalty = 70 ns (going to main memory)

AMAT (P1) = 0.66 ns + (0.08 * 70) = 6.26 ns

AMAT (P1) = 1 cycle + (0.08 * ceil(70/0.66)) = 9.56 cycles

AMAT calculation for P2:

Hit time for P2 = 0.90 ns
Miss rate for P2 = 6.0 %
Miss penalty = 70 ns (going to main memory)

AMAT (P2) = 0.90 ns + (0.06 * 70) = 5.10 ns

AMAT (P2) = 1 cycle + (0.06 * ceil(70/0.90)) = 5.68 cycles

5.19.10 c

P1	12.64 CPI	8.34 ns per inst
P2	7.36 CPI	6.63 ns per inst

CPI calculation for P1:

The CPI can be broken into three parts. This will be shown below in the calculations.

An instruction needs at least 1 cycle.

$CPI_A = 1$

By the problem statement, 8.0% of instructions cause a miss. This means that the processor must then access main memory. This incurs a cycle delay of ceil(70/0.66) = 107 cycles.

$CPI_B = 0.08 * 107 = 8.56$

Lastly, 36% of all instructions contribute to memory accesses. Similarly, 8.0% of these instructions will also cause a miss. This will subsequently increase the CPI as shown.

$CPI_C = 0.36 * 0.08 * 107 = 3.08$

Summing up all CPI parts we get the total CPI for P1:

$CPI_{P1} = CPI_A + CPI_B + CPI_C = 1 + 8.56 + 3.08 = 12.64$

Multiplying this by the cycle time gives us the average time needed to execute an instruction:

Average time (P1) = CPI_{P1} * Cycle time = 12.64 * 0.66 ns = 8.34 ns

---

The same methodology applies for CPI calculation for P2.

$CPI_A = 1$
$CPI_B = 0.06 * 78 = 4.68$
$CPI_C = 0.36 * 0.06 * 78 = 1.68$

$CPI_{P2} = 1 + 4.68 + 1.68 = 7.36$

Average time (P2) = CPI_{P2} * Cycle time = 7.36 * 0.90 ns = 6.63 ns

---

Since P2, on average, needs less time than P1 for an instruction, it is faster.

5.19.10 d

6.50 ns	9.85 cycles	Worse

The miss penalty must now be adjusted to account for the L2 cache. Now, if a miss occurs and the data is in L2, P1 would need ceil(5.62/0.66) = 9 cycles to get data to L1.

If the data is not in L2 (95% of the time), we need 107 additional cycles as computed previously. Total miss penalty will then be as follows:

Total Miss Penalty = Miss_{L2} + Miss_{L1} = 9 + (0.95 * 107) = 110.65

AMAT (P1) = 1 + (0.08 * 110.65) = 9.85 cycles

Comparing to 5.6.2 AMAT, which is 9.56 cycles, including an L2 leads to a worse AMAT.

5.19.10 e

Computation will be similar to 5.6.3, except now we account for the increased miss penalty in cycles from L2.

CPI_A = 1
CPI_B = 0.08 * 110.65 = 8.85
CPI_C = 0.36 * 0.08 * 110.65 = 3.19

CPI_{P1} = 1 + 8.85 + 3.19 = 13.04

5.19.10 f
Because the clock cycle time and percentage of memory instructions is the same for both versions of P1, it is sufficient to focus on AMAT. We want
AMAT with L2 < AMAT with L1 only
1 + 0.08[9 + m*107] < 9.56
This happens when m< .916.

5.19.10 g
We want P1's average time per instruction to be less than 6.63 ns. This means that we want
(CPI_P1 * 0.66) < 6.63. Thus, we need CPI_P1 < 10.05
CPI_P1 = AMAT_P1 + 0.36(AMAT_P1 – 1)
Thus, we want
AMAT_P1+ 0.36(AMAT_P1-1) < 10.05
This happens when AMAT_P1< 7.65.
Finally, we solve for
1+ 0.08[9+ m*107] < 7.65
and find that
m< 0.693
This miss rate can be at most 69.3%.