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?

      The cache block size in bytes is:
      Block offset = 4 bits (offset) -> 2^4 = 16 bytes per block

      The cache block size in words is:
      16 bytes per block/4 = 4 words

      The cache block size in double words is:
      16 bytes per block/8 = 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.]*

      (If direct-mapped) The number of sets this cache has is:
      Index = 4 bits -> 2^4/1 = 16/1 = 16 cache sets
      (If fully-associative) The number of sets this cache has is:
      Index = 4 bits -> 2^4/16 = 16/16 = 1 cache set

   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.
      # of Blocks = 4 bits (index) -> 2^4 = 16 blocks
      Tag bits = 32 – index – offset = 32 – 4 – 4 = 24 bits
      Meta bits = tag + valid = 24 + 1 = 25 bits
      LRU bits for Two-way = 1/set * # blocks = 1/2 * 32 = 16 bits
      LRU bits for Four-way = 2/set * # blocks = 2/4 * 64 = 32 bits
      Above LRU calculation for 2-way uses 1 bit per set to track which two lines are least recently used, whereas 4-way uses 2 bits per set to encode LRU out of 4 options.

Type	# Blocks	Data bits	Meta bits
Direct	16	16B * 8 * 16 = 2048	25 * 16 = 400
Two-way	2 * 16 = 32	16B * 8 * 32 = 4096	25 * 32 = 800
Four-way	4 * 16 = 64	16B * 8 * 64 = 8192	25 * 64 = 1600

d. Simulate the direct-mapped and four-way set associative cache with respect to the following series of memory accesses. In the table below, indicate whether each memory access was a hit or a miss and 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	Miss
`0x1001EFA4`	Miss	Miss
`0x1001FEA8`	Miss	Hit
`0x100100A0`	Miss	Miss
`0x100100B0`	Miss	Miss
`0x100100C0`	Miss	Miss
`0x10011FA1`	Miss	Miss
`0x1001EEA2`	Miss	Miss
`0x1001EFAF`	Miss	Miss
`0x100100A2`	Miss	Hit

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

Hit rate of Direct Mapped: 0%
Hit rate of 4-way Set Associative: 20%

2. Cache Write Nuances
   a. ZyBooks (Textbook) 5.19.6 all (a-c)
      a. **Between L1 and L2:**
         Since L1 is write through, all store operations go to L2. So, we need a write buffer to absorb write traffic and reduce stalls.
         Between L2 and Main Memory:
         Additionally, since L2 is write back, dirty blocks evicted from L2 are written to the memory. This is why we need a write back buffer to queue these.

      b. For L1 Write-Miss, firstly we have no block being loaded into L1 since it is non-write allocate. Then the write is sent directly to L2.
         Now, if the L2 has the block, then the block gets updated. If L2 does not have the block, then this will result in a miss and due to the write-allocate policy, L2 will allocate the block, fetch it from memory, and then the write gets applied.
         In the end, if the evicted L2 block is dirty, it must be written back to memory before it is replaced.

      c. **If L1 Write-Miss:**
         Then there is no data loaded into L1. The write is then forwarded to L2. Since L2 has write-allocate, if it misses, it will fetch the block from memory then allocate in L2. Then it writes to the newly allocated block.
         If L2 needs to evict a block, it checks if it is dirty, then writes it back to memory. If it is not dirty, it simply discards the block.
         If L1 Read-Miss:
         The requested block is forwarded to L2.
         If L2 is a hit, then it will move the block to L1, and evict from L2. If L1 needs to be evicted, it will move the block back to L2 (and if L2 needs to evict, then we follow the L2 write-back rules).
         If L2 is a miss, then a fetch from memory occurs into L1. L2 stays the same.

   b. ZyBooks (Textbook) 5.19.7 all (a-b)
      a. Read:
         1000 * 0.3% = 3 misses, meaning 3 * 64 = 192 bytes are read
         250 * 2% = 5 misses, meaning 5 * 64 = 320 bytes read
         100 * 2% = 2 misses, meaning 128 bytes read
         Write:
         100 writes total means 100 * 64 = 6400 bytes written
         Total:
         Read = 192 + 320 + 128 = 640 bytes
         Writes = 6400 bytes
         Assuming a CPI of 2 is needed, for 1000 instructions = 2000 cycles

Therefore, the Minimum Read bandwidth is 640/2000 = 0.32 bytes/cycle and the Minimum Write bandwidth is 6400/2000 = 3.2 bytes/cycle.

b. Instruction cache = 192 bytes read
Data read misses = 5 * 64 = 320 bytes read
Write misses = 2 * 64 = 128 bytes read (since it is write allocate on 2 misses)
Since loaded blocks are 3 for Instruction cache, 5 for Data read misses and 2 for Write misses, that totals up to 10 blocks.
30% dirty means 0.3 * 10 = 3 dirty blocks. Then 3 * 64 = 192 bytes written.
Total:
Read = 192 + 320 + 128 = 640 bytes
Write = 192 bytes
Assuming the need for a CPI of 2, for 1000 instructions, this gives us 2000 cycles as the total (same as part a). Therefore, the Minimum Read bandwidth is 640/2000 = 0.32 bytes/cycle and the Minimum Write bandwidth is 192/2000 = 0.096 bytes/cycle.

3. Cache Hierarchy Performance
   a. ZyBooks (Textbook) 5.19.10 all (a-g)
      a. P1:
      1/0.66*(10^-9) = 1.52 GHz
      P2:
      1/0.90*(10^-9) = 1.11 GHz

      b. P1:
      Average Memory Access Time = 0.66 + (8% * 70) = 6.26ns
      P2:
      Average Memory Access Time = 0.90 + (6% * 70) = 5.10ns

      c. P1:
      6.26/0.66 = 9.48 cycles
      Stall CPI = 36% * 9.48 = 3.41
      Total CPI = 1 + 3.41 = 4.41
      P2:
      5.10/0.9 = 5.67 cycles
      Stall CPI = 36% * 5.67 = 2.04
      Total CPI = 1 + 2.04 = 3.04
      Therefore, given the above, P2 is faster since it has the lower CPI.

      d. P1 with L2:
      Average Memory Access Time = 0.66 + (8% * (5.62 + (95% * 70))) = 6.43ns

Therefore, since P1 with L2 is 6.43ns versus 6.26ns without L2, this shows that it is better without the L2 cache.

e. 6.43/0.66 = 9.74 cycles
Stall CPI = 36% * 9.74 = 3.51
Total CPI = 1 + 3.51 = 4.51

f.

We want average memory access time with L2 to be less than average memory access time without L2.

So,

$$0.66 + 0.08 \times (5.62 + x \times 70) < 6.26$$

$$0.08 \times (5.62 + 70x) < 5.6$$

$$5.62 + 70x < 70$$

$$70x < 64.38$$

$$x < 0.92 \,//$$

g.

average memory access time for P1 with L2:

Stall CPI = 3.04 − 1.0 = 2.04) converted to cycles

$$\frac{2.04}{0.36} = 5.667 \text{ cycles}$$

$$5.67 \times 0.66\text{ns} = 3.742 \text{ ns}$$

∴ Solving for L2 miss rate:

$$0.66 \times 0.08 \times (5.62 + x \times 70) < 3.74$$

$$0.08 \times (5.62 + x \times 70) < 3.08$$

$$5.62 + x \times 70 < 38.5$$

$$70x < 32.88$$

$$x < 0.47 \,//$$