Question 1(a)

Assume that the cache does not generate memory write traffic when modified blocks are evicted. This is because the cache uses write-through policy which updates the memory on every write.

Reads that hit in the cache do not generate memory traffic.

Assume that a write access generates a one-word write to memory. Memory traffic due to writes is $0.25 \times 1 \times 10^9$ words per second.

Since the cache allocates a block of two words on any miss, memory read traffic due to misses is $0.05 \times 2 \times 10^9 = 0.1 \times 10^9$ words per second.

Percentage of memory bandwidth used on average is $\frac{(0.1+0.25)\times 10^9}{10^9} = 35\ \%$.

Question 1(b)

Similar to Question 1(a), reads that hit in the cache do not generate memory traffic.

Similar to Question 1(a), the cache allocates a block of two words on any miss, memory read traffic due to misses is $0.05 \times 2 \times 10^9 = 0.1 \times 10^9$ words per second.

Since we are using a write-back cache now, writes that hit in the cache also do not generate memory traffic.

Assume that the cache is full so all cache misses result in a block being evicted from the cache. Since the cache uses a write-back policy, modified blocks generate memory write traffic when evicted. Memory write traffic due to misses is $0.05 \times 0.3 \times 2 \times 10^9 = 0.03 \times 10^9$ words per second.

Percentage of memory bandwidth used on average is $\frac{(0.1+0.03)\times 10^9}{10^9} = 13$ %.

Question 2(a)

Fraction of instructions that are not load/stores is $1 - 0.26 - 0.09 = 0.65$.

Average CPI for instructions that are not load/stores is $1 + (0.005 \times 50) = 1.25$. This is because 0.995 of the instruction fetches hit in the instruction cache and 0.005 of the instruction fetches miss in the instruction cache and incur a 50 cycle miss penalty. The base CPI is 1.

Loads

If the *data* cache is perfect, average CPI is 1.25 due to instruction cache misses.

Average stall time is $0 + (0.01 \times 50) = 0.5$ cycles. A cache read hit time of 1 cycle introduces no stall cycles.

Assume that the cache does not write to memory when modified blocks are evicted because the write-through policy is used.

Average CPI for load instructions is $1.25 + 0.5 = 1.75$.

Stores

If the *data* cache is perfect, average CPI is 1.25 due to instruction cache misses.

A store instruction writes data to both the data cache and memory. Average stall time is $1 + (0.01 \times 50) = 1.5$ cycles. A cache write hit time of 2 cycles introduces 1 stall cycle to the pipeline. Since the write-allocate policy is used, 50 stall cycles is incurred on a write miss. Since a write buffer decouples the memory writes from the pipeline, no additional stalls are incurred.

Average CPI for store instructions is $1.25 + 1.5 = 2.75$.

Final CPI is $(0.65 \times 1.25) + (0.26 \times 1.75) + (0.09 \times 2.75) = 1.515$.

Question 2(b)

Similar to Question 2(a), fraction of instructions that are not load/stores is 0.65. Average CPI for instructions that are not load/stores is 1.25. This is not affected by whether the cache is write-back or write-through.

Loads

If the *data* cache is perfect, average CPI is 1.25 due to instruction cache misses.

Assume that the time to read a block from memory into the cache cannot be overlapped with the time taken to write a modified block back to memory because the same bus is used.

For reading blocks from memory into the cache, average stall time is $0 + (0.01 \times 50) = 0.5$ cycles. A cache read hit time of 1 cycle introduces no stall cycles.

Assume that the cache is full so all misses result in a block being evicted from the cache. For writing modified blocks from the cache to memory, average stall time is $0.01 \times 0.5 \times 50 = 0.25$ cycles. This is because the data cache miss rate is 1%, 50% of the blocks are dirty and writing the block to memory takes 50 cycles. Note that in this case, there is no write buffer.

Average CPI for load instructions is $1.25 + 0.5 + 0.25 = 2$.

Stores

If the *data* cache is perfect, average CPI is 1.25 due to instruction cache misses.

A cache write hit time of 2 cycles introduces 1 stall cycle to the pipeline.

Average stall time due to cache miss penalty is $0.01 \times 50 = 0.5$ cycles. Since the write-allocate policy is used, 50 stall cycles is incurred on a write miss.

Average stall time for writing dirty blocks back to memory is $0.01 \times 0.5 \times 50 = 0.25$ cycles

Average CPI for store instructions is $1.25 + 1.5 + 0.25 = 3$.

Final CPI is $(0.65 \times 1.25) + (0.26 \times 2) + (0.09 \times 3) = 1.6025$.

Question 3(a)

No.

Assume that memory is byte addressable.

Since the L1 cache uses 4-byte blocks and there are 4 sets in the direct-mapped L1 cache, addresses must have at least 2 bits for block offset and 2 bits for set index.

Since the L2 cache uses 8-byte blocks and there are 8 sets in the direct-mapped L2 cache, addresses must have at least 3 bits for block offset and 3 bits for set index.

Assume that addresses are n-bits so there are $(n-6)$-bits for L2 tags.

Consider 3 addresses shown the table. Note that $n > 6$ but we only show 8 bits in the example.

Address in binary	L1 Tag	L1 Block Index	L2 Tag	L2 Block Index
0000 0000	0	0	0	0
0000 0100	0	1	0	0
0100 0000	4	0	1	0

Assuming that the cache is initially empty, consider the following sequence of memory accesses:

1. 0x00
2. 0x04
3. 0x40

When address 0x00 is accessed, a block is allocated in the L2 cache with tag 0 and index 0. A block is also allocated in the L1 cache with tag 0 and index 0.

When address 0x04 is accessed, it hits in the L2 cache since it is in the same L2 block as address 0x00. A block is allocated in the L1 cache with tag 0 and index 1.

When address 0x40 is accessed, a conflict miss occurs. The block in the L2 cache with tag 0 and index 0 is evicted and a block in the L2 cache with tag 1 and index 0 is allocated.

Similarly, the block in the L1 cache with tag 0 and index 0 is evicted and a block in the L1 cache with tag 4 and index 0 is allocated.

However, without any other mechanism to enforce inclusion, address 0x04 is still cached in the L1 cache but not in the L2 cache. This is because 0x04 and 0x00 have different L1 block index but the same L2 block index. Therefore, the L2 cache is not guaranteed to be inclusive of the L1 cache.

Question 3(b)

No.

Assume that memory is byte addressable.

Since the L1 cache uses 8-byte blocks and there are 4 sets in the 2-way associative L1 cache, addresses must have at least 3 bits for block offset and 2 bits for set index.

Since the L2 cache uses 8-byte blocks and there are 8 sets in the 2-way associative L2 cache, addresses must have at least 3 bits for block offset and 3 bits for set index.

Assume that addresses are n-bits so there are $(n-6)$-bits for L2 tags.

Consider 3 addresses shown the table. Note that $n > 6$ but we only show 8 bits in the example.

Address in binary	L1 Tag	L1 Block Index	L2 Tag	L2 Block Index
0000 0000	0	0	0	0
0100 0000	2	0	1	0
1100 0000	6	0	3	0

Assuming that the cache is initially empty, consider the following sequence of memory accesses:

1. 0x40
2. 0x00
3. 0x40
4. 0xc0

When address 0x40 is accessed, a block is allocated in the L2 with tag 1 and index 0. A block is also allocated in the L1 cache with tag 2 and index 0.

When address 0x00 is accessed, a block is allocated in the L2 cache with tag 0 and index 0. Since the L2 cache is 2-way set associative, the L1 cache can allocate this block without evicting the block that contains the address 0x40. The block that contains the address 0x40 is now the least recently used block in the L2 cache. A block is also allocated in the L1 cache with tag 0 and index 0. Since the L1 cache is 2-way set associative, the L1 cache can allocate this block without evicting the block that contains the address 0x40. The block that contains the address 0x40 is now the least recently used block in the L1 cache.

When address 0x40 is accessed, this is a hit in the L1 cache so the block that contains 0x00 is now least recently used in the L1 cache.

When address 0xc0 is accessed, a block is allocated in the L2 cache with tag 3 and index 0. Since the L2 cache is 2-way set associative and blocks with tags 0 and 1 are currently allocated in the L2 cache for set 0, a block must be evicted. Since the block containing the address 0x40 is the least recently used block in the L2 cache, the block that contains the address 0x40 is evicted from the L2 cache.

A block is also allocated in the L1 cache with tag 6 and index 0. Since the L1 cache is 2-way set associative and blocks with tag 0 and 2 are currently allocated in the L1 cache for set 0, a block must be evicted. Since the block that contains the address 0x00 is the least recently used block in the L1 cache, the block that contains the address 0x00 is evicted from the L1 cache.

Now, the address 0x40 is cached in the L1 cache but not in the L2 cache. Therefore, the L2 cache is not guaranteed to be inclusive of the L1 cache.