Question 5)

CT	CT	CT	CT	CI	CI	CO	CO

C = 16 bytes, E = 1, B = 4 bytes, S = 4 sets
b = 2, s = 2 (2 block offset bits, 2 set index bits)

Picture: don't forget dirty (write-back) and valid bits (always)

Read 0x01: miss, load set 0 (tag = 0000) with first 4 bytes of memory {0,1,2,3}, read value {1}, set valid bit for set 0
Write 0x07, 6: miss, load set 1 with {4,5,6,7} and overwrite byte 3, giving {4,5,6,6}, set dirty bit for set 1, set valid bit for set 1
Read 0x06: hit, read byte 2 of set 1 = read {6}
Write 0x06, 7: hit, write to byte 2 of set 1, giving {4,5,7,6}
Read 0x20: miss on set 0 (tag 0010 != tag 0000), evict set 0 (no need to write since dirty bit not set), load new set 0 {32, 33, 34, 35}, read byte 0 with value {32}

2 accesses were hits, 3 were misses

Question 6)
1) True
2) 8 sets

3) i) uses main memory efficiently by treating it as a cache for an address space (caching)
ii) simplifies memory management by providing each process with a uniform address space (management)
iii) protects address space of each process from corruption by other processes (protection)

4) Temporal locality: recently referenced items are likely to be referenced again in the near future
Spatial locality: items with nearby addresses tend to be referenced close together in time

6) One plausible example: After the first free (by A), the allocator may allocate the same block for some other request (by B). Now, A frees the block again, but B has no knowledge of this and believes it owns the block. Now, C makes a request and receives the block from the allocator. At this point, both B and C believe they own the block and a write by one will possibly corrupt the other's data. **A, B, and C are all entities in the same Process/Address Space (think different functions)

7) miss rate = 100%

8) i) and iii). These create a pointer ip, and assign the address a + 3 (or identically &a[3]) to it. ii) dereferences the value stored at a[0], adds 3 to it, and then assigns the result to ip.

Question 7)
1) On a write-hit in the cache, after updating the block being written to, write-through immediately writes the block to the next lower level of the cache hierarchy. Write-back defers the update to the lower level until the block is evicted, and simply sets a dirty bit indicating that its copy is newer than the lower level cache.
2) False
3) False

4) True
5) sizeof(struct test) = 32 bytes
Starting addresses: {0, 4, 6, 16, 24}
6) 16, 3, 11, 12345