1. (a) Cache coherence is a key feature in chip multiprocessor design because it allows multiple cores to have the same view of memory. This is important since it allows for the maximum amount of coordination and collaboration between cores, which in turn, lets programmers obtain a maximal performance from parallelising programs.

   Cache coherency is important because it avoids the need for each write to memory to be written *all the way* to memory (which can take many hundreds of CPU cycles in the worst case), but instead lets memory be written to a fast cache (which can take one cycle, or only a few cycles) and written later, while maintaining the semantics of the computation on the accessing core and other cores. This also applies when reading values from memory.

   (b) Modified means that a core has written to the memory location, but the write has not been propagated through to main memory.

   Shared means that this cache holds the up-to-date value of a memory location, and at least one other cache holds the same value.

   Invalid means that the value held for the memory location has been updated by either another CPU core or a DMA operation since the memory location was loaded.

   (c) i. This is what the MESI protocol does. The advantage of having the 'exclusive' state is that the core does not need to send out an invalidate broadcast on the system bus if it writes to the value.

   ii. This is an optimisation since one core could evict the cache line from its cache and not have to write the line back to memory, since the other cache still retains the updated value, and this cache would go to state M or O depending on if there was more than one core sharing the value. This is the MOSEI protocol (Owned).

   (d)

Cycle	States	Explanation
1.	EII	Core one has loaded x and is the only core to have done so
2.	SSI	Both core one and two have loaded x; they're both in the shared state
3.	SSI	Add does nothing to memory
4.	SSI	Add does nothing to memory
5.	MII	Core one writes to the cache, and invalidates core two.
6.	OIS	Core three loads the value, and receives it directly from core one, which becomes an owner
7.	OIS	Add does nothing
8.	IIM	Core three's write invalidates core one. Core one may write back to memory now if this is not optimised out.
9.	IMI	Core two writes its value to memory, and invalidates core three, which may write back.

   (e) Since the threads do not synchronise their memory accesses, the actual order that their instructions are executed is not defined at run-

time. If two threads read the same value from memory, they then both add one to it, and then write back one after the other, then the location will only be incremented once. The memory location could be incremented by either one two or three. (Include example run in exam for each case)

(f) Transactional memory would require bits to indicate whether the cache line is in the readset or writeset of the transaction. This can be done with two extra bits (one for each set). These bits are then used at the end of transactions (assuming lazy conflict detection) to determine whether it is safe to commit the transaction.