E0-243 Term paper 2

Q (1) [15 + 5 + 5] In this question, you will design a **snoopy cache coherence** protocol like the ones we studied in the class. We had studied MSI and MESI protocols. Here, however, you have to draw the finite state machine (FSM) for a new protocol called M***C***SI. As usual, M, S and I denote modified, shared and invalid state, respectively. The new "C" state is like "S" state, except that it is introduced to aid in the cache-to-cache data transfer. For example, a block in "C" state can **only be read** by the local core, like "S" state. But a private cache with a block in "C" state **must respond with data (BusReply)**. Also, unlike "S" state, **at most only one** private copy of a cache block can be in "C" state at any given time. Remember that the goal of the C state in MCSI protocol is to **maximize cache-to-cache transfer**. Thus, you should design the FSM such that when there are one or more copies of a given block in private cache(s) in read-only state(s), then one of those copies of the block is in the "C" state, *as far as possible*. It's okay if sometimes there is no "C" copy but only "S" copy of a block, but this situation should be **minimized as far as possible**.

A. Draw the FSM for the MSCI coherence protocol. Draw by hand (no copy+paste business) and upload the pdfs on gradescope.
B. Do you need any additional hardware support for implementing the above protocol? Hint: Think of how "E" state in MESI protocol that we studied in the class was implemented.
C. Let us assume that your favorite application has a producer-consumer sharing pattern where there is one thread that produces a value by writing to shared variable while there are three consumer threads that consumes (read) the value written in the shared variable in a tight loop. Assuming cache-to-cache transfer are faster than bringing the data from LLC/memory to private caches, would you prefer your favorite application to run on a system with MESI or MCSI coherence protocol? Explain your choice.

Q (2) [25] In the MSI directory protocol that we studied in the class we made a big implicit assumption. We assumed that the directory is always able to hold information of all private caches. That will be true if the directory has number of entries equal to (or more) than the total number of cache blocks across ***all*** private caches in the systems and, also have the associativity of the directory to be at least equal to the ***sum*** of ways of all private caches.

However, such requirements are often too onerous to be implemented in real hardware. Therefore, on-chip directories are often designed as set-associative cache structures with limited capacity and associativity. It may thus, happen that an entry in directory needs to be evicted to make space for another entry. At the time of eviction, private cache (s) may still contain the cache block corresponding to the directory entry being evicted.

Please extend the state of table of MSI protocol (for both private cache and directory) to make it work in the presence of eviction of directory entries.

It's okay to copy+paste the state tables for the basic MSI protocol.

Hint: At the least, you would need to introduce a new event "Eviction" (column) in the directory's state table, and at least one new event/message for the state table of the private caches and a bunch of transient states.