Lab 5 Report

Answers to the Prelab questions from Section 6

1. **Why are transient states necessary?**

   Transient states are necessary in computer systems to handle intermediate or temporary conditions during data transfer or processing. They help ensure data consistency, synchronization, and proper communication between different components.

2. **Why does a coherence protocol use stalls?**

   A coherence protocol uses stalls to temporarily pause or delay certain operations in order to maintain data coherence and prevent conflicts. Stalls allow the protocol to coordinate access to shared data and ensure that all cores have a consistent view of memory.

3. **What is deadlock and how can we avoid it?**

   Deadlock is a situation where multiple processes or threads are unable to proceed because each is waiting for a resource that is held by another. Deadlocks can be avoided by implementing strategies such as resource allocation algorithms, deadlock detection algorithms, and resource scheduling techniques.

4. **What is the functionality of `Put-Ack` (i.e., `WB-Ack`) messages? They are used as a response to which message types?**

   `Put-Ack` (`WB-Ack`) messages are used to acknowledge the successful completion of a write-back operation in a cache coherence protocol. They are sent as a response to `Put` (WB) messages that initiate write-backs from caches to the main memory.

5. **What determines who is the sender of a data reply to the requesting core? Which are the possible options?**

   The sender of a data reply to a requesting core is typically determined by the request type and the current cache situation in the cache coherence system.

   Possible options include the cache on other node, the directory, or the main memory.

6. **What is the difference between a `Put-Ack` and an Inv-Ack message?**

   The main difference between a `Put-Ack` and an `Inv-Ack` message lies in their purpose and usage.

   - `Put-Ack` messages are used to acknowledge the successful completion of a write-back operation.
   - `Inv-Ack` messages are used to acknowledge the invalidation of a cache line in a cache coherence protocol.

Answers to all the questions from Section 5

1. **How does the FSM know it has received the last `Inv_Ack` reply for a TBE entry?**

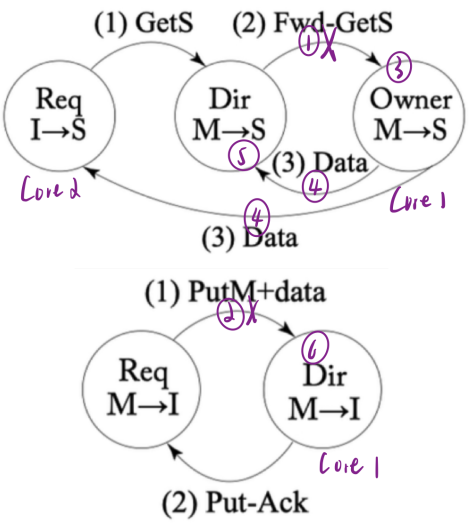
   The Directory maintains a record of all individuals sharing a specific block and sends an `AckCount` that corresponds to the number of sharers. By receiving this information, the cache controller can determine the total number of sharers and anticipate the number of `InvAcks` to expect. With this knowledge, the Finite State Machine (FSM) can ascertain whether it has received the final `InvAck` reply for a Transaction Block Entry (TBE).

2. **How is it possible to receive a `PUTM` request from a NonOwner core? Please provide a set of events that will lead to this scenario.**

   Please refer to the diagram below for the sequence of operations:

   1. Initially, Core 1 has a block in `M` state and the directory is in `M` state and thus does not have the block. Core2 makes a `GetS` request to Directory.
   2. Directory send the `Fwd-GetS` request to the owner, Core 1. The Directory adds the previous owner and the current requestor to the list of sharers and clears the owner. The Directory transitions from `M` to `MS_D` as it waits for the data from Core 1.
   3. Meanwhile, Core 1 receives a Replacement Request for the block, so it issues a write-back (`PutM` +Data) to the Directory.
   4. Core 1 then receives the `Fwd-GetS` request while it is in `MI_A`, and sends the requested block to the Directory and the Core 2.

5. The `Fwd-GetS` reply from Core 1 arrives, so now the Directory can complete the transition from `MS_D` to Shared.
6. The Directory then receives the PutM+Data message, but at this point Core 1 is no longer the owner.



3. **Why do we need to differentiate between a `PUTS` and a `PUTS-Last request`?**

   `PUTS` is sent to the directory cache controller when its cache block in `S` state has more than one sharer. In this case, the cache controller must keep its cache block because other sharers still exist.

   `PUTS-Last` is sent to the directory cache controller when there is only one sharer or none at all. In this case, the directory can invalidate it's cache block.

   Thus, we need to differentiate between `PUTS` and `PUTS-Last` so that the directory cache controller can know when to invalidate it's cache block.

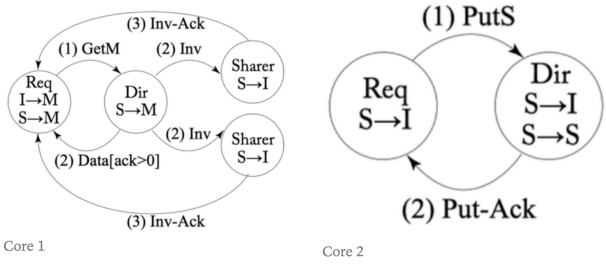
4. **How is it possible to receive a `PUTS-Last` request for a block in modified state in the directory? Please provide a set of events that will make this possible.**

   a. Core 1 issues a `GETM` request and goes to transient state

   b. Core 2 issues `PUTS-Last` request and goes to transient state

   c. Directory receives Core 1's `GETM` first, goes to `M` state, and sent invalidation as it's previously in `S` state

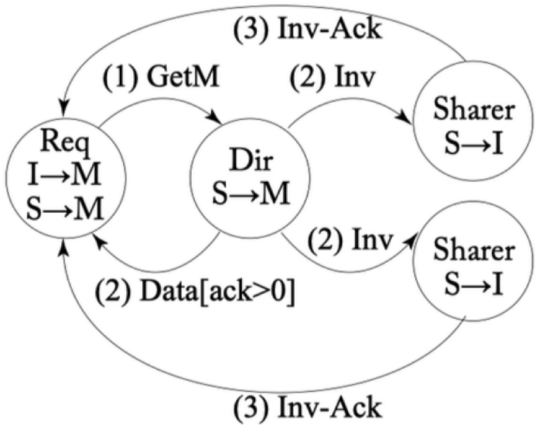
   d. Core 2's `PUTS-Last` request finally arrives at Directory, which is now in modified state because of last step.



5. **Why is it not possible to get an Invalidation request for a cache block in a Modified state? Please explain.**

   Invalidation request is only sent to the cache block in `S` state. And is only happening when the directory is in `S` state.

   For cache block in `M` state, it can't receive invalidation request because it's not in `S` state. And the directory must also be `M` state.



6. **Why is it not possible for the cache controller to get a `Fwd-GetS` request for a block in `SI_A` state? Please explain.**

   When the Directory receives a `GETS` request, it will send a `Fwd-GetS` to the owner of the block. Since the block is in `SI_A` state, it is not the owner and therefore it will be impossible for its cache controller to receive a `Fwd-GetS`.

7. **Was your verification testing exhaustive? How can you ensure that the random tester has exercised all possible transitions (i.e., all `{ State, Event }` pairs)?**

   While the system can detect unexpected transitions, it does not provide a signal indicating whether all transitions have been encountered at some point.

   As a result, achieving "exhaustive" testing of the system is not possible. Only an "extensive" one is possible. To ensure comprehensive coverage of transition states, we conducted progressive testing by increasing the number of cores, instructions (loads), and decreasing cache size. With more cores communicating and increased traffic from additional instructions, the likelihood of uncovering previously unseen transitions and associated errors was heightened. This approach allowed for a more extensive testing of the system, although not reaching complete exhaustiveness.

   When errors were encountered, we reviewed and addressed those transitions.

Table documenting all your protocol modifications (e.g., new transient states) with respect to Tables 8.1 and 8.2

	Load	Ifetch	Store	Replacement	Fwd GetS	Fwd GetM	Inv	WB Ack	Inv Ack	Inv Ack all	Data from Dir No Acks	Data from Dir Ack Cnt	Data from Dir Ack Cnt Last	Data from Owner	
I	v as i p m / IS D	v as i p m / IS D	v am i p m / IM AD												I
S	r m	r m	v x am p m / SM AD	v bs / SI A			fi cc h o / I								S
M	r m	r m	s m	v x bm / MI A	de e o / S	e cc h o / I									M
IS D	z	z	z	z			z				u w rx n / S			u w rx n / S	IS D
IM AD	z	z	z	z	z	z			q n		u sx w n / M	u q n / IM A	u sx w n / M	u sx w n / M	IM AD
IM A	z	z	z	z	z	z			q n	w sx n / M					IM A
SM AD	r m	r m	z	z	z	z	fi o / IM AD		q n		sx w n / M	q n / SM A	sx w n / M		SM AD
SM A	r m	r m	z	z	z	z			q n	q sx w n / M					SM A
SI A	z	z	z	z			fi o / II A	h w cc o / I							SI A
MI A	z	z	z	z	de e o / SI A	e o / II A		h w cc o / I							MI A
II A	z	z	z	z				cc w h o / I							II A
	Load	Ifetch	Store	Replacement	Fwd GetS	Fwd GetM	Inv	WB Ack	Inv Ack	Inv Ack all	Data from Dir No Acks	Data from Dir Ack Cnt	Data from Dir Ack Cnt Last	Data from Owner	

Difference with Table 8.1 is highlighted in the table above

	GETM	GETS	PUTM	PUTM Owner	PUTM NotOwner	PUTS NotLast	PUTS Last	PUT Ack	Memory Data	Memory Ack	Data	
I	e qf i / IM MD	qf i / IS MD			a i	a i	a i					I
M	f e i	f ars2 aos c i / MS AD		lq pl c i / MI A	a i	a i	a i					M
S	ds2 fwm cs e i / M	ds ars2 i			k a i	k a i	k a i / I					S
MS AD	z	z			k a i	k a i	k a i				lqi pl2 pr / MS A	MS AD
MS A	z	z		z	z	z	z			qm / S		MS A
MI A	z	z		z	z	z	z			la qm / I		MI A
IS MD	z	z		z	z	z	z		d ars1 qm / S			IS MD
IM MD	z	z		z	z	z	z		d qm / M			IM MD
	GETM	GETS	PUTM	PUTM Owner	PUTM NotOwner	PUTS NotLast	PUTS Last	PUT Ack	Memory Data	Memory Ack	Data	

Difference with Table 8.2 is highlighted in the table above

Simple table	Ifetch	Data_from_Dir_Ack_Cnt_Last
I	Send GetS to Dir/IS_D	
IS_D	Stall	
IM_AD	Stall	-/M
IM_A	Stall	
S	Hit	
SM_AD	Hit	-/M
SM_A	Hit	
M	Hit	
MI_A	Stall	
SI_A	Stall	

Simple table	Functionality
MS_AD	Modified -> Shared, waiting for data, after getting data, waiting for ack from memory (`MS_A`)
MS_A	Modified -> Shared, waiting for ack from memory
MI_A	Modified -> Invalid, waiting for ack from memory
IS_MD	Invalid -> Shared, waiting for data from memory
IM_MD	Invalid -> Modified, waiting for data from memory

Distribution of Work

We split work evenly