6.823 Computer Architecture	DUE: 5/8/2002
Group 18: Timothy Danford, Kilian Pohl, Bryan Russell	
Problem Set 6	*Instructors: Krste Asanovic, Srinivas Devadas*

Problem 3

(a)

No.	Current State	Event Received	Next State	Action
1	C-invalid	load	C-transient	load-request → home
2	C-invalid	store	C-transient	store-request → home
3	C-invalid	invalidate-request	C-invalid	invalidate-reply → home
4	C-invalid	shared-copy-request	C-invalid	shared-copy-reply → home
5	C-invalid	exclusive-copy-request	C-invalid	exclusive-copy-reply → home
6	C-shared	load	C-shared	processor reads cache
7	C-shared	store	C-transient	store-request → home
8	C-shared	replace	C-invalid	nothing
9	C-shared	invalidate-request	C-invalid	invalidate-reply → home
10	C-modified	load	C-modified	processor reads cache
11	C-modified	store	C-modified	processor writes cache
12	C-modified	replace	C-invalid	write-back → home
13	C-modified	shared-copy-request	C-shared	shared-copy-reply → home
14	C-modified	exclusive-copy-request	C-invalid	exclusive-copy-reply → home
15	C-transient	load-reply	C-shared	data → cache, processor reads cache
16	C-transient	store-reply	C-modified	data → cache, processor writes cache
17	C-transient	invalidate-request	C-transient	retry → home
18	C-transient	shared-copy-request	C-transient	retry → home
19	C-transient	exclusive-copy-request	C-transient	retry → home

(b)

No.	Current State	Message Received	Next State	Action
1	H-uncached	load-request	H-shared[{k}]	load-reply $\rightarrow$ k
2	H-uncached	store-request	H-modified[k]	store-reply $\rightarrow$ k
3	H-shared[S]	load-request	H-shared[S $\cup$ {k}]	load-reply $\rightarrow$ k
4	H-shared[S]	store-request	H-transient	invalidate-request $\rightarrow$ S
5	H-modified[m]	load-request	H-transient	shared-copy-request $\rightarrow$ m
6	H-modified[m]	store-request	H-transient	exclusive-copy-request $\rightarrow$ m
7	H-modified[m]	write-back	H-uncached	data $\rightarrow$ memory
8	H-transient	load-request	H-transient	retry $\rightarrow$ j
9	H-transient	store-request	H-transient	retry $\rightarrow$ j
10	H-transient	write-back	H-unchanged	data $\rightarrow$ memory
11	H-transient[count $>$ 1]	invalidate-reply	H-transient[–count]	nothing
12	H-transient[count $=$ 1]	invalidate-reply	H-modified[k]	store-reply $\rightarrow$ k
13	H-transient	shared-copy-reply	H-shared[{m,k}]	data $\rightarrow$ memory, load-reply $\rightarrow$ k
14	H-transient	exclusive-copy-reply	H-modified[k]	store-reply $\rightarrow$ k

(c) Suppose that as the home site sends the exclusive-copy-request message to a given site and then site encounters a replace command. The replace command simply results in sending the one-way write-back message to the home site and changing the state to C-invalid. Now, if the site encounters a load or store command before the original exclusive-copy-request arrives to the site, the site's state then changes to C-transient (and a load-request or store-request is sent to the home site). This is how the scenario in line 19 can be reached.

(d) If the network is non-FIFO, then we can get into an undesired state. Suppose that a cache m that is in state $C - modified$ wishes to *write back* (i.e. *replace*) and then *store* data. With the FIFO assumption, the cache switches to the $C - invalid$ state, sends a $store-request$ to the home site, switches to the $C-transient$ state, receives a $store-reply$, and then switches to state $C - modified$. The home site goes from $H - modified[m]$ to $H - uncached$ and then back to $H - modified[m]$. Without the FIFO assumption, the $store - request$ can get to the home site before the $write - back$ message. So, the home site will then change to $H - transient$ and send an $exclusive - copy - request$ back to the cache site. Since the cache site is in $C - transient$ state (after sending the $store - request$), it is waiting for a $store - reply$. However, when the cache site receives the $exclusive - copy - request$, the cache site returns a *retry* back to the home site. So, the home site will continue to *retry* the $exclusive-copy-request$. Now, when the $write-back$ message arrives to the home site, the state then changes to $H-uncached$. So, the end result is the home site having an incorrect state ($H - uncached$ instead of $H - modified$) and there will be the $exclusive-copy-request/retry$ message bouncing back and forth between the cache and home sites.

If, for some reason, there is a modification to the protocol such that, when a home directory gets a store-request from a cache that already owns the data, there is a way to simply pass along a "permission" message for the owning cache to go ahead and write to the address that already belongs to it:

If that addition is made to the protocol, there is still a problem.

Then the cache line can be written to and replaced (generating a write-back) twice, which would generate two different write-backs with two different values in transit to the same directory at the same time.

And because the network isn't FIFO, that would mean that the write-backs could appear in the wrong order, causing an incorrect value to be written in the home directory.