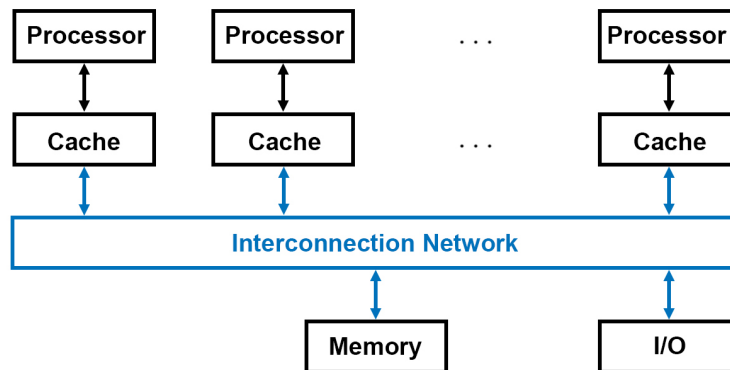




[Courseware](#) [Course Info](#) [Discussion](#) [Wiki](#) [Progress](#) [Discussion Guidelines](#) [Resources](#) [Exploring Engineering](#)
[Syllabus](#) [How to Use Jade](#)

Help

For all homework problems, assume a multiprocessor organization without an L2 cache:

[Show Discussion](#)[New Post](#)

The following additional details on the MESI protocol are provided for answering this homework question:

- The first CPU to read a block into its cache can mark it as *Exclusive*, since no other cache holds the block and the main memory has the same copy.
- If another CPU then reads the block, it can be provided by main memory (since it is up to date), but the first CPU must tell the second that it has a copy. Then both CPUs mark their copies as *Shared*. Now if either CPU wants to write the block, it must first tell the other to invalidate its copy.
- If no other CPU reads a block before a CPU with the *Exclusive* copy wants to write it, then it performs the write and marks the block as *Modified*, since it now owns the only up to date copy (main memory has a stale copy).
- When a CPU provides a *Modified* copy to another CPU that wants to read it, main memory is also updated, and both CPUs mark their copy as *Shared*.

With this additional background, consider a multiprocessor with two CPUs. Block A is in main memory, but in neither cache (in both, the cache location where block A would be held is marked *Invalid*). Select the correct state of the block in both CPU caches under each of the following scenarios.

Help

[Show Discussion](#)[New Post](#)

1 A. HOMEWORK (1/1 point)

Read by CPU 1, then write by CPU 1.

- ☐ CPU 1: *Invalid*, CPU 2: *Exclusive*
- ☐ CPU 1: *Exclusive*, CPU 2: *Invalid*
- ☐ CPU 1: *Exclusive*, CPU 2: *Shared*
- ☒ CPU 1: *Modified*, CPU 2: *Invalid* ✓

EXPLANATION

CPU 1 will obtain an *Exclusive* copy when it reads it since CPU 2 does not have a copy. When it writes to the block, it can mark it *Modified*. CPU 2 still has the location marked *Invalid*.

[Final Check](#)[Save](#)[Hide Answer](#)

You have used 1 of 2 submissions

[Show Discussion](#)[New Post](#)

1 B. CHECK YOUR UNDERSTANDING (1/1 point)

Read by CPU 2, then read by CPU 1.

- ☐ CPU 1: *Invalid*, CPU 2: *Shared*
- ☐ CPU 1: *Exclusive*, CPU 2: *Invalid*
- ☒ CPU 1: *Shared*, CPU 2: *Shared* ✓
- ☐ CPU 1: *Shared*, CPU 2: *Invalid*

EXPLANATION

CPU 2 will obtain an *Exclusive* copy when it reads it since CPU 1 does not have a copy. When CPU 1 reads a copy, it will receive the copy from main memory, but CPU 2 will inform it that it has a copy. Both CPUs will mark their copies *Shared*.

Help

Final Check

Save

Hide Answer

You have used 1 of 2 submissions

Show Discussion

 New Post**1 C. CHECK YOUR UNDERSTANDING** (1/1 point)

Read by CPU 1, then write by CPU 1, then read by CPU 2.

☐ CPU 1: *Invalid*, CPU 2: *Shared*☐ CPU 1: *Exclusive*, CPU 2: *Shared*☒ CPU 1: *Shared*, CPU 2: *Shared* ☐ CPU 1: *Invalid*, CPU 2: *Invalid***EXPLANATION**

CPU 1 will obtain an *Exclusive* copy when it reads it since CPU 2 does not have a copy. When it writes to the block, it can mark it *Modified*. Now when CPU 2 reads the block, CPU 1 provides it, main memory is updated, and both CPUs mark their copies as *Shared*.

Final Check

Save

Hide Answer

You have used 1 of 2 submissions

Show Discussion

 New Post**1 D. CHECK YOUR UNDERSTANDING** (1/1 point)

Read by CPU 1, then write by CPU 1, then write by CPU 2.

- ☐ CPU 1: *Exclusive*, CPU 2: *Invalid*
- ☐ CPU 1: *Exclusive*, CPU 2: *Shared*
- ☐ CPU 1: *Shared*, CPU 2: *Exclusive*
- ☒ CPU 1: *Invalid*, CPU 2: *Modified* ✓

EXPLANATION

CPU 1 will obtain an *Exclusive* copy when it reads it since CPU 2 does not have a copy. When it writes to the block, it can mark it *Modified*. Now consider the different outcomes above after CPU 2 writes the block. Since CPU 2 is the last to write the block, it will not have it as *Invalid*, which eliminates the first outcome. The second and third outcomes are contradictory, as one cache cannot have the block as *Exclusive* and another *Shared*.

The logical outcome is the fourth, which arises as follows: When CPU 2 misses when attempting to write the block, it sends a *Read + Invalidate* message on the shared interconnect. This message causes CPU 1 to provide its *Modified* copy to CPU 2 and mark its copy as *Invalid*. CPU 2 receives the block, performs the write, and marks its copy as *Modified*.

Why does CPU 1 need to provide the block if CPU 2 is going to overwrite it? Remember that a cache block may contain multiple words. CPU 1 may have overwritten word 1 in the block and CPU 2 may need to overwrite a word 2. With CPU 1 providing its copy to CPU 2, the modifications to both words 1 and 2 are preserved.

Final Check

Save

Hide Answer

You have used 1 of 2 submissions

Show Discussion

 New Post**2. HOMEWORK** (1/1 point)

Consider the scenario where two CPUs have *Shared* copies of a block in their caches and they *simultaneously* want to write it. They may then simultaneously invalidate each other's copy! One solution to this problem is to use a *multi-drop bus* as the shared interconnect. Here, we have a shared set of wires that can be used by only one of the CPUs at a time (similar to what we saw in the LC-3). A *bus arbiter* determines which of the two gets to use the bus if they both simultaneously request it. The CPU with lower priority will wait until the higher priority one has performed its actions, which could include invalidating its copy of a block, before retrying its cache access (which could cause it to take a different action).

Consider a multiprocessor with two CPUs connected to each other and memory by a multi-drop bus. Whenever both CPUs simultaneously request the use of the bus, CPU 1 is given priority. Both CPUs have the same block in their caches in state *Shared* and simultaneously attempt to write to the block.

What will be the final outcome after both CPUs perform their writes?

Help

- ☐ Both CPUs will have the block as *Invalid*.
- ☐ Both CPUs will have the block as *Exclusive*.
- ☐ CPU 1 will have the block in state *Modified* and CPU 2 in state *Invalid*.
- ☒ CPU 2 will have the block in state *Modified* and CPU 1 in state *Invalid*. ✓

EXPLANATION

Since CPU 1 has priority, it will use the bus to invalidate the copy of CPU 2 and then write to the block. CPU 2, now will miss in the cache and make a *Read + Invalidate* request on the bus. CPU 1 will provide the block and mark its copy as *Invalid*. CPU 2 will write to the block and mark it *Modified*.

Final Check

Save

Hide Answer

You have used 1 of 2 submissions

Show Discussion

 New Post

EdX offers interactive online classes and MOOCs from the world's best universities. Online courses from MITx, HarvardX, BerkeleyX, UTx and many other universities. Topics include biology, business, chemistry, computer science, economics, finance, electronics, engineering, food and nutrition, history, humanities, law, literature, math, medicine, music, philosophy, physics, science, statistics and more. EdX is a non-profit online initiative created by founding partners Harvard and MIT.

© 2015 edX Inc.

EdX, Open edX, and the edX and Open edX logos are registered trademarks or trademarks of edX Inc.

Terms of Service and Honor Code

Privacy Policy (Revised 10/22/2014)

**About edX**

About

News




Contact

FAQ

edX Blog

Donate to edX

Jobs at edX

Follow Us Facebook Twitter LinkedIn Google+ Tumblr Meetup Reddit Youtube