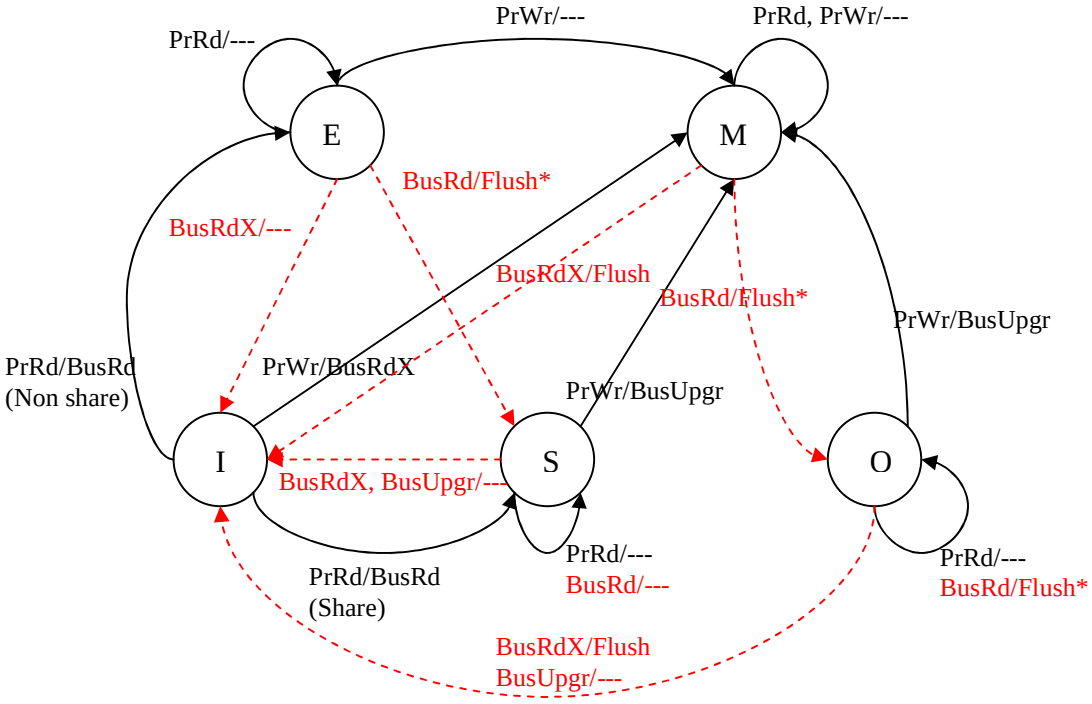
ECE4100/6100 Advanced Computer Architecture
Programming Assignment 5

Assigned: Nov 24, 2010 **Due by 12:35pm Dec 13 (Mon), 2010**

In this assignment, you will implement a **MOESI invalidation-based coherence protocol for a 4-way shared-memory multiprocessor** using High-level programming languages such as C, C++ or Java. Similar to previous project, this project must be done individually.

Problem Statement

MOESI contains one additional state "**O**wned" on top of the MESI protocol. The **O** state is implemented to facilitate cache-to-cache transfer when the cache line is modified by a processor P (in **M** state) and can directly be supplied by P. Similar to the S state, a line found in the O state in one processor will be present in at least one other processor's cache. Note that the memory will have a stale version of this line. In the MOESI state diagram below, we also introduce a new transaction called *BusUpgr* (Bus upgrade) to substitute BusRdX in some transitions where we can simply invalidate others' cache lines without incurring an actual data exclusive read. In this simulation, cache-to-cache transfers only take place during the action of **Flush*** in which the requested cache line is directly sent to the requester by the processor having the same line in E, M or O state without updating the memory. Note that all the other "Flush" transactions will update the memory and a request will receive the line from the memory. When all lines are in S state, a BusRd is put out by a processor, the line will be supplied from the memory.



Dealing with Dirty Writeback

What were not shown in the state diagram are the extra actions needed for dirty writebacks due to conflict misses. A cache line in the M or O state is inconsistent with the data in the main memory. Therefore, when such lines are being evicted, they need to update the memory to make it consistent upon the eviction.

Memory traces

There are four memory traces provided with this assignment. Each trace represents one memory access pattern of one processor in a four-way multiprocessor system. Each line in the trace contains one particular memory access issued by the processor. The format of each trace is shown as follows:

<time stamp> <Read or Write> <physical memory address>

In the second field, "0" is a Read while "1" is a Write. If should two memory accesses from different processors occur at the same cycle among traces (i.e. simultaneous requests at the same cycle), then simply assume the priority is always given to the one with the smallest processor ID. In other words, the higher processor ID request will be pushed one cycle after its immediate lower processor ID request.

Simulation Configurations

You need to conduct this work by using the following cache configuration. Each processor only contains a "single level cache". All caches employ write-allocate policy.

1. **16KB, 64-byte cache line, Direct-mapped cache for all processors.**
2. **32KB, 128-byte cache line, 2-way using LRU replacement policy for all processors.**

Outputs from the design

You need to show the following statistics for the entire traces for your check-off.

1. The total number of cache-to-cache transfers for each processor pair in the following format.

```
P0 cache transfers: <p0-p1>  <p0-p2>  <p0-p3>
P1 cache transfers: <p1-p0>  <p1-p2>  <p1-p3>
P2 cache transfers: <p2-p0>  <p2-p1>  <p2-p3>
P3 cache transfers: <p3-p0>  <p3-p1>  <p3-p2>
```

2. The total number of invalidations due to coherence (i.e. not including line replacement) in each processor in the following format.

```
P0 Invalidation: m=  o=   e=   s=   i=
P1 Invalidation: m=  o=   e=   s=   i=
P2 Invalidation: m=  o=   e=   s=   i=
P3 Invalidation: m=  o=   e=   s=   i=
```

3. The number of dirty writebacks from each processor. Note that, in this report, please write back all the dirty lines at the end of your simulations.

4. The number of lines in each state at the end of each simulation for each processor.

Design Strategy

1. You need to implement a cache simulator and initiate four of them during your simulation. Note that you do not need to simulate the actual data portion of the cache in this simulator. When you perform your MOESI, you have to conduct cycle-based simulation, in other words, maintain a global clock during your execution. If an event (from any processor) takes place in a particular cycle, then the state of the corresponding cache line needs to be updated according to MOESI.
2. A 2-way LRU simply tracks the latest used way to be the MRU, the other line must be the LRU one.
3. Making your cache configurable will be easier for varying the configuration.
4. Deriving the state diagram once yourself will substantially help you understand the protocol which is not really that complex as it seems.

How to receive the credit

This assignment does not require a check-off. But do provide give a Makefile or READ.txt to the TAs about how to compile and execute your code. Please upload a zip file containing your source code with detailed comments and a short report about your implementation.

Honor Code

This assignment must be done **individually**. Due to its complexity, you are encouraged to form a study group with your peers to discuss your design approaches. However, you are not allowed to copy others' codes. For those who violate the rule, both the originator and the copier will receive zero credit and will be **immediately** reported to the Dean of Students' Affair for further action.