ECE 485/585
Fall 2019
Final Project Description

Your team is responsible for the simulation of the last level cache (LLC) for a new processor that can be used with up to three other processors in a shared memory configuration.

The cache has a total capacity of 16MB, uses 64-byte lines, and is 8-way set associative. It employs a write allocate policy and uses the MESI protocol to ensure cache coherence. The replacement policy is implemented with a pseudo-LRU scheme.

Although you do not need to model it, the processor's next higher level cache uses 64-byte lines and is 4-way set associative. It employs a write-once policy: the first write to a line is write through; subsequent writes to the same line are write-back. Your system *must maintain inclusivity.*

Model and simulate your cache in a hardware description language (HDL) such as Verilog or SystemVerilog or a programming language such as Python, C, or C++. Your simulation does not need to be clock accurate. If using an HDL, your design does not need to be synthesizable. You do not need to store data in the cache as the cache behavior (hits, misses, evictions, etc.) is independent of the data.

However, you must model communication between your LLC and the next higher level cache, bus operations that your LLC performs, snoop results that *your* LLC reports on the bus in response to snooping the simulated bus operations of other processors and their caches. Implement the following functions and use them as appropriate in your simulation. You can put the #defines in a separate include file so they can be used by your main code.

You can use whatever method you want to simulate the snoop result returned by *other* LLCs in response to your LLC's bus operations. But it must be repeatable and easily changed so that you can test that your simulated cache behaves properly for all return values.

```
/*
 * Bus Operation types
 */

#define READ            1     /* Bus Read */
#define WRITE           2     /* Bus Write */
#define INVALIDATE      3     /* Bus Invalidate */
#define RWIM            4     /* Bus Read With Intent to Modify */


/*
 * Snoop Result types
 */

#define NOHIT           0     /* No hit */
#define HIT             1     /* Hit */
#define HITM            2     /* Hit to modified line */

/*
 * L2 to L1 message types
 */

#define GETLINE         1     /* Request data for modified line in L1 */
#define SENDLINE        2     /* Send requested cache line to L1 */
#define INVALIDATELINE  3     /* Invalidate a line in L1 */
#define EVICTLINE       4     /* Evict a line from L1 */
// this is when L2's replacement policy causes eviction of a line that
// may be present in L1.   It could be done by a combination of GETLINE
// (if the line is potentially modified in L1) and INVALIDATELINE.

/*
Used to simulate a bus operation and to capture the snoop results of last
level caches of other processors
*/
void BusOperation(char BusOp, unsigned int Address, char *SnoopResult);
SnoopResult = GetSnoopResult(Address);
#ifndef SILENT
printf("BusOp: %d, Address: %h, Snoop Result: %d\n",*SnoopResult);
#endif
}


/*
Used to simulate the reporting of snoop results by other caches
*/
char GetSnoopResult(unsigned int Address);
{}
```

```
/*
Used to report the result of our snooping bus operations performed by other
caches
*/
void PutSnoopResult(unsigned int Address, char SnoopResult);
{
#ifndef SILENT
printf("SnoopResult: Address %h, SnoopResult: %d\n", Address,SnoopResult);
}
#endif


/*
Used to simulate communication to our upper level cache
*/
void MessageToCache(char Message, unsigned int Address);
{
#ifndef SILENT
printf("L2: %d %h\n",  Message, Address);
#endif
}
```

Output

Maintain and report the following key statistics of cache usage for each cache and print them upon completion of execution of each trace:

- Number of cache reads
- Number of cache writes
- Number of cache hits
- Number of cache misses
- Cache hit ratio

Modes

Your simulation must support at least two modes (without the need to recompile). In the first mode, your simulation displays only the required summary of usage statistics and responses to 9s in the trace file and nothing else. In the second mode, your simulation should display everything from the first mode but also display the bus operations, reported snoop results, and communication messages to the higher level cache described above. You may choose to have additional modes or conditional compilation that display debug information but these should not be turned on in the version that you demonstrate to me.

Your simulation should not require recompilation to change the name of the input trace file. Ideally, if you're writing your simulation in C, you should use command line arguments (e.g. argc, argv[]) to specify the mode and input file. If you're using Verilog, consider using the $value$plusargs function.

Testing

Testing is a crucial part of your project. Create an explicit plan for how you will ensure that all aspects of your model is operating correctly and include it in your final report. Describe each of the tests you perform on the model.

Project report

You must submit a report that includes a design specification that describes the interface to the cache module (to the next level in the memory hierarchy, and any shared buses) relevant internal design documentation, the source modules for your cache, any associated modules used in the validation of the cache (including the testbench if using Verilog), and your simulation results along with the usage statistics.

Make (and document) reasonable assumptions about the processor and memory subsystem and their interfaces. The report should justify these assumptions as well as any design decisions you make. Be sure to state any assumptions about the L1 cache as well.

Traces

Your testbench must read events from a text file of the following format. You should not make any assumptions about alignment of memory addresses. You can assume that memory references do not cross cache line boundaries.

```
n address
```

Where n is

0 read request from L1 data cache
1 write request from L1 data cache
2 read request from L1 instruction cache
3 snooped invalidate command
4 snooped read request
5 snooped write request
6 snooped read with intent to modify requestnal
8 clear the cache and reset all state
9 print contents and state of each valid cache line (allow subsequent trace activity)

The address will be a hex value. For example:

```
2 408ed4
0 10019d94
2 408ed8
1 10019d88
2 408edc
```

When printing the contents and state of the cache use a *concise but readable* form that shows only the valid lines in the cache along with any state information.

Grading

The project is worth 100 points. Your grade will be based upon:

- External specification
- Completeness of the solution (adherence to the requirements above)
- Correctness of the solution
- Quality and readability of the project report (e.g. specifications, design decisions)
- Validity of design decisions
- Quality of implementation
- Structure and clarity of design
- Readability
- Maintainability
- Testing
- Presentation of results