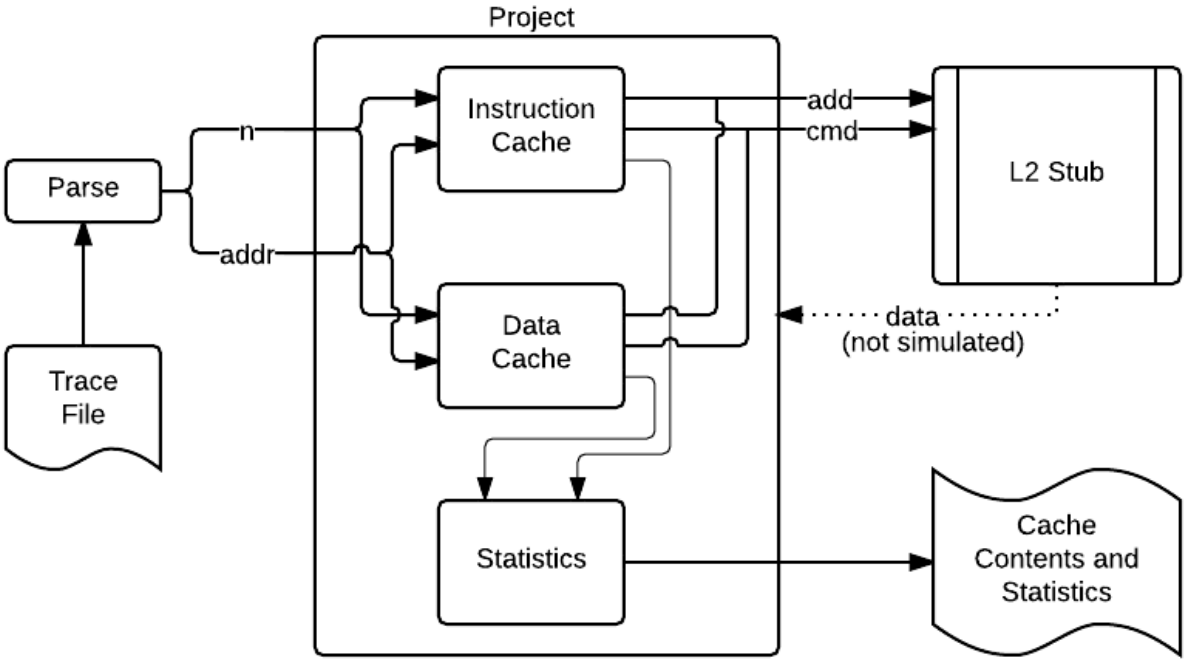
Eric Krause / Bradon Kanyid / Kevin Riedl / Andy Goetz - Fall 2012

# ECE 485/585 Final Project

## L1 Cache Simulation

| Version | Comments | Date |
|---|---|---|
| 1.0 | Initial Release | December 2, 2012 |

## Specifications:




This project functionally simulates a split instruction/data L1 cache for a 32-bit processor in a system with multiple processors. The system employs MESI protocol to ensure cache coherence. The instruction cache is 2-way set associative, consists of 16K sets, and has 64-byte lines. The data cache is 4-way set associative, consists of 16K sets, and has 64-byte lines. Both caches employ LRU replacement policy and are backed by an L2 cache (which is modeled as a stub in this simulation). Statistics regarding number of reads, writes, hits, and misses are generated, as well as a hit percentage rate. This simulation has a single-cycle interface between a processor and L1, and between L1 and L2. All processor reads and writes are a single byte.

The 32-bit addresses from the processor are broken down into the following fields:

- Bits 31:20 = 12-bit tag
- Bits 19:6 = 14-bit index
- Bits 5:0 = 6-bit byte offset

We specify the following method for interface with L2. A 26-bit address specifies a 64-byte cache line, and must be supplied with one of the following 2-bit cmd_out commands:

- 00: No Operation (ignore any input on address lines)
- 01: Read from L2
- 10: Write to L2
- 11: Read with intent to modify

In order to run, our simulation requires a trace file formatted using the protocol specified in the project description. A print command will output human-readable cache contents and statistics.

## Assumptions

In the course of designing our L1 cache, we were forced to make several assumptions regarding the CPU it was designed for.

- The cache hierarchy is inclusive. By making the L2 cache support an inclusive policy, the synchronization logic between the L1 and L2 cache is greatly simplified.
- The data cache is write-through. The L1 and L2 caches together are required to support memory writebacks. However, because the cache design also needs to support MESI, we decided to implement the L1 data cache as a simple write-through cache. Because the cache hierarchy uses an inclusive policy, evictions forced by MESI in the L2 cache will force an eviction in the L1 cache. If the L1 cache had a dirty line, the MESI eviction would force a writeback to main memory in the L1, greatly complicating our code.
- Cache contents (actual data) are irrelevant for this simulation. Thus, all byte offsets are ignored. Because we only stub out the processor, the next level cache, and our cache eviction policy is based entirely on memory addresses, there is no need to examine data values.
- All read and write operations reference single byte locations. In order to simplify the cache design, we assume that all reads and writes reference a single byte. This means we do not need to worry about supporting unaligned memory references.

## Testing

In order to test our implementation, we developed a set of test stimulus covering all of the corner cases. These tests are listed in the text file 'testplan'.

In addition to the testplan developed by our team, we also performed blackbox testing with another ECE485 team. We traded testbenches, and confirmed that both of our implementations produced the same hit ratios.