Portland State University

Design and Simulation Of Unified L2 Cache

Benjamin Huntsman, Michael Walton, and Suresh Krishna Sagar Gorthi

12/09/2013

Table of Contents

1. Introduction

This report provides a quick summary and overview of the Cache System designed as part of ECE 485/585 course. It also discusses the requirements, design process, test plan and results. The project requirements list all key product features and define all of the choices and assumptions made in the development of the L1 cache. The test cases chosen and how they show completeness of the product are included in the test plans. Finally, the results indicate the advantage of having an L1 cache backed up by a L2 cache.

The product is an L1 cache system implemented in a 32-bit processor. We simulated the design in System Verilog using various functions. All functions are integrated and testing is done on the integrated module. We analysed the performance using trace file, while maintaining and reporting cache usage statistics.

1.1. Using Cache for Performance

Improving the performance of digital computer systems has always been a challenge for system architects. Specifically, much design work has focused on decreasing the time, or number of clock cycles, a given CPU must spend communicating with the relatively large main memory, or RAM. Most modern microprocessors, include a small internal first level cache memory (L1) to increase system performance. Cache memories are fast memory storage devices that utilize the principle of locality of reference to improve CPU read-from-memory efficiency and, therefore, overall system performance. Whenever the CPU accesses the main memory for code or data, additional bytes "surrounding" the byte(s) being fetched are brought into the cache in the form of a cache line. The principle of locality of reference predicts that the CPU will in all probability use the additional bytes subsequent to the use of the data brought in, and quite possibly a multiple number of times. Multiple uses of the same code or data occur during program loops, for example. These subsequent accesses will be "hits" in the relatively small and fast cache, and will therefore speed up execution because each "hit" reduces by one the number of CPU accesses to the relatively large and slow main memory. In the event of a "miss" in the cache, the CPU must access the main memory for its required code or data and the cache is loaded with a new cache line of memory that "surrounds" this required code or data for potential subsequent use by the CPU. While loading a new cache line into the cache, the “Least Recently Used” (LRU) eviction policy is used. LRU uses one or more bits of the item to indicate which item in a particular index is the oldest, and thus the least likely to be used again

in the near future. Thus, when the processor needs to add a new item into that index line, it knows which of the items in the index can be safely replaced.

To further decrease memory latency and increase system performance, we can back the L1 cache by a small L2 cache. Both L1 and L2 cache maintain inclusivity property. This improves the performance of the system by having a back-up for the L1 cache.

2. Project Requirements/ Design Specifications

The following are the requirements specified in the project description. We are designing the system for a 32-bit processor, implying that both data and address buses are to be of 32-bit size. L1 instruction cache is a two way set associative and consists of 16K sets and 64 byte lines. L1 data cache is a four way set associative and consists of 16K sets of 64 byte lines. The L1 data cache is write through using write allocate. Both caches employ LRU replacement policy and are backed by a shared L2 cache. Both L1 and L2 have inclusivity property.

2.1. Reporting Statistics

The following are the key reporting statistics that are required to be printed 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

2.2. Assumptions

As part of the design, few assumptions were made. All effort was made to infer from the requirements and minimize any assumptions. However, the following assumptions were made:

LRU :

Here the LRU algorithm we use first puts all the values set to zero. When a line is accessed, its value is again set to '0' and all the others are incremented by '1'. Finally when the cache is full, the line with the highest LRU value set is evicted. If two lines have same LRU value, then the line first comes is evicted first and set to 0.

0 0 0 0 0 0 0 0

0 1 1 1 1 1 1 1 (when line 0 is accessed)

1 0 2 2 2 2 2 2 (when line 1 is accessed)

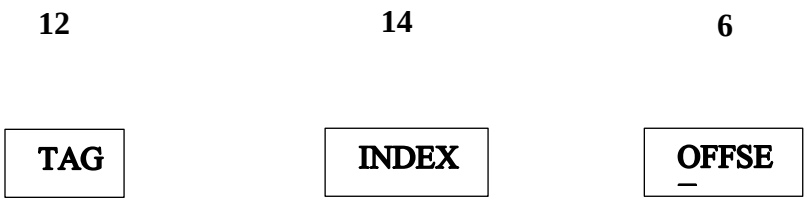
2 1 0 3 3 3 3 3 (when line 2 is accessed)

- L1 cache will be passing address in to L2 in the form of L2 address.
- The counter for LRU is of 3 bits.

L1 Cache

L1 cache receives a memory reference from CPU. When there is a CPU request and if it is a hit in the L1 cache the L1 satisfies the request. When there is a CPU request and if it is a miss in the L1 cache, it sends an L1 miss signal to the L2 cache. If it is a hit in the L2 cache lines are copied to L1 cache and it satisfies the CPU request. If it is a miss in L2 cache, then the line is copied from the DRAM and through L2 it is copied into L1 cache.

For the L2 cache design we have,



CONCLUSION:

Thus an L2 cache is designed and simulated. Designing a cache system has helped us understand the functioning of various sub-systems in a typical microprocessor system

Attachments:

1. **Code**
2. **Program Algorithm**