|  |
| --- |
| **Portland state university** |
| Microprocessor System Design |
| **Project Report for Design and Simulation of a Split L1 Cache** |
|  |
| Neerja Bawaskar (PSU ID: 912362484)  Rahul Wagh (PSU ID: 967287161)  Ameya Deswandikar (PSU ID: 970606576) |
| 12/5/2012 |

Course Number: ECE585

Department: Electrical and Computer Engineering

Maseeh College of Engineering and Computer Science

Portland State University

|  |
| --- |
|  |

Contents

[Project Specification 2](#_Toc342487004)

[Design 3](#_Toc342487005)

[Assumptions 3](#_Toc342487006)

[Flowcharts 4](#_Toc342487007)

[Read 4](#_Toc342487008)

[Write 5](#_Toc342487009)

[Reset 6](#_Toc342487010)

[Invalidate 6](#_Toc342487011)

[Testing 7](#_Toc342487012)

[Test case design 7](#_Toc342487013)

[Trace FIle 1 8](#_Toc342487014)

[Trace FIle 2 9](#_Toc342487015)

[Trace FIle 3 10](#_Toc342487016)

[Trace FIle 4 11](#_Toc342487017)

[Implementation 12](#_Toc342487018)

[Choice of Language 12](#_Toc342487019)

[Development environmen**t** 12](#_Toc342487020)

[Class Diagram 13](#_Toc342487021)

[File Dependencies 14](#_Toc342487022)

[Error Handling 15](#_Toc342487023)

[Maintainability 15](#_Toc342487024)

[Repository 15](#_Toc342487025)

[Coding Standards 16](#_Toc342487026)

[Appendix 17](#_Toc342487027)

[Source code 17](#_Toc342487028)

[Trace file 17](#_Toc342487029)

[Simulation results 17](#_Toc342487030)

# Project Specification

Design and simulation of a split L1 cache backed by a unified L2 cache with MESI cache coherence protocol in a multiprocessor system.

Specifications:

|  |  |  |
| --- | --- | --- |
| **Parameter** | **L1 Data Cache** | **L1 Instruction Cache** |
| Associativity | Four-way | Two-way |
| Number of sets | 16K | 16K |
| Line size | 64 bytes | 64 bytes |

Both caches implement an LRU replacement policy and have a shared L2 cache.



Figure 1: L1 Cache external interfaces

# Design

The L1 cache is supported by a unified L2 cache. Since this is a multiprocessor system with MESI cache coherence mechanism, we maintain inclusivity with L2 to simplify the L1 cache design.

In order to maintain inclusivity, we implemented a write-through with write allocate policy on both L1 Data and Instruction caches. This implies that we need not implement the MESI protocol in the L1 cache. The cache coherence mechanism is handled in the lower level cache (L2), freeing up the L1 cache to serve the CPU. Design and simulation of the L2 cache is out of scope for this project. We tested the L1-L2 interface using a L2 stub: a minimal implementation of the cache interface specified below.

The figure below represents the public interfaces provided by the L1 Data and Instruction caches.



Figure 2: Public Interfaces

|  |  |
| --- | --- |
| **Interface** | **Description** |
| Read | Read data from the cache |
| Write | Write data to the cache |
| Reset | Clears the Valid and Tag bits and clears the data in the cache |
| Invalidate | Clears the Valid bit of the cache line |

## **ASSUMPTIONS**

* The last level in the memory hierarchy has to be Write Back in order to satisfy the given specification of write back with respect to memory.
* The memory is byte addressable.
* The maximum size of address is 32 bits.

## **Flowcharts**

### Read

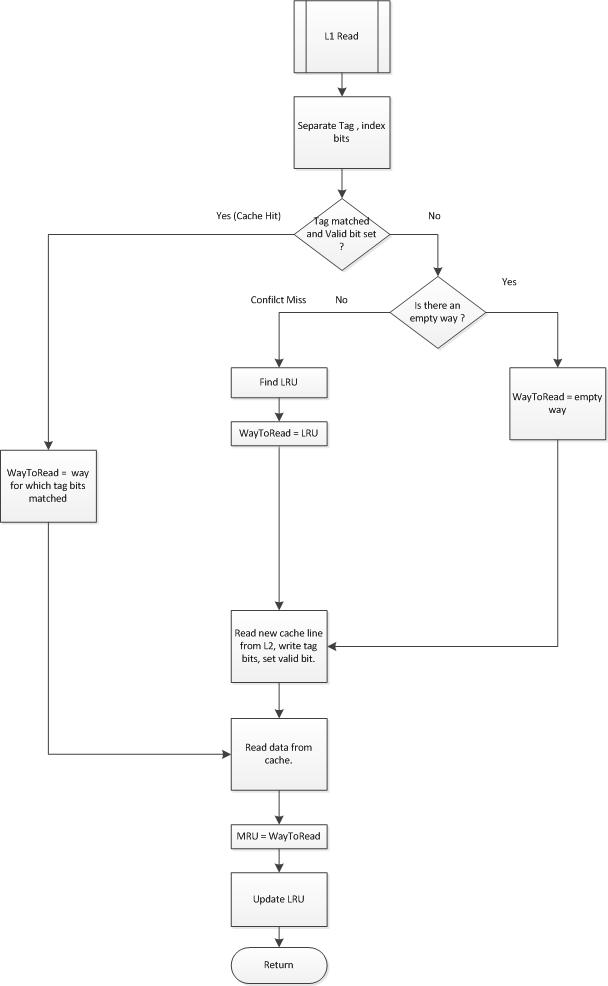


Figure 3: Flowchart for READ

### Write

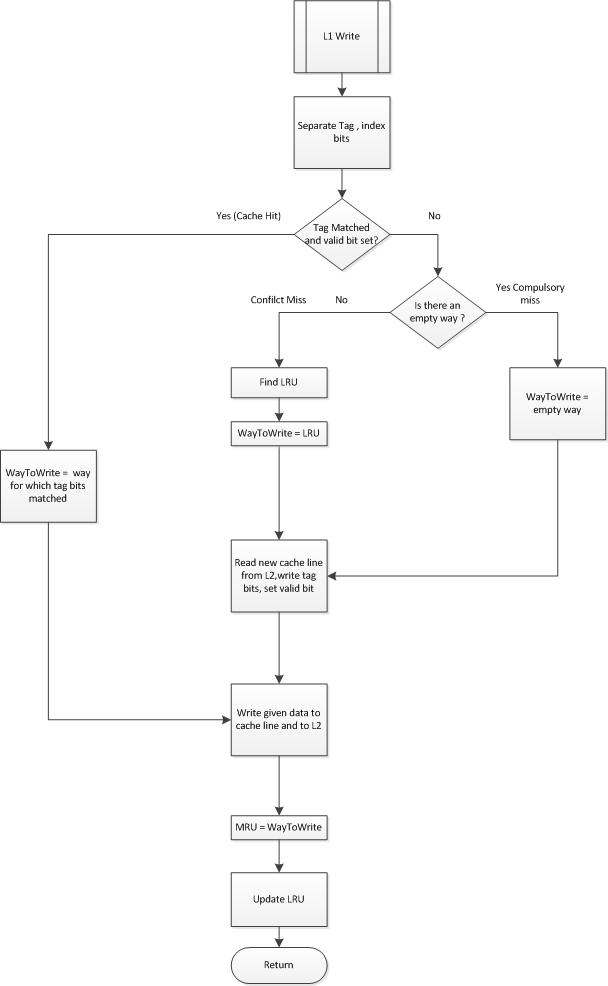


Figure 4: Flowchart for WRITE

### **Reset**

The Reset command clears the Valid bits and the Tag bits of each line in the cache. The LRU bits are set to a default value of 3 (i.e. least recently used). The data in each line of every set is also cleared.

### **Invalidate**

The Invalidate command makes the cache line invalid by clearing the Valid bit of the line. The reasons for invalidate could be due to an eviction in L2 cache (due to inclusivity) or due to the cache coherence protocol implementation (MESI). In case of a hit to a modified line, L2 cache will send an Invalidate to the L1.

# Testing

The trace file has the following format:

n address

where n is

|  |  |
| --- | --- |
| 0 | Read data request from L1-D cache |
| 1 | Write data request to L1-D cache |
| 2 | Instruction Fetch (Read request from L1-I cache) |
| 3 | Invalidate command from L2 |
| 8 | Clear the cache and reset all states |
| 9 | Print contents and state of the cache |

## **Test CASE DESIGN**

We designed test cases to validate the working of our cache in the following scenarios. The test cases were implemented using four different trace files.

1. Write hits to each line in the set. (L1 Data cache only).
2. Reads hits to each line in the set. (For both L1 Data and Instruction cache)
3. Eviction cases of all lines in the set. (For both L1 Data and Instruction cache)
4. Testing an Invalidate command to show how it affects a hit.
5. Testing a Cache hit for different bytes in the same line.
6. Cache hits to the set with at least one empty line
7. Reset and Print commands
8. Print the cache statistics when the entire trace file is read
9. Testing for invalid command inputs i.e. other than 0,1,2,3,8 and 9
10. Testing for addresses greater than 32-bits
11. Testing for whitespace and blank lines

### Trace FIle 1

1. Only writes to L1-D
2. Last set accessed
3. Invalidate command
4. Eviction cases for all lines in the set using LRU replacement policy
5. Cache write hit for different bytes in the same line
6. Write hits to all lines in the set

Table 1: Trace file 1 entries

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Command** | **Address** | **Tag Bits (12)** | **Index Bits (14)** | **Offset Bits (6)** | **Hit/Miss** | **Comments** |
| 1 | 0xFFFC0 | 0x000 | 0x3FFF | 0x0 | Miss | Way = 0 |
| 1 | 0xE00FFFC1 | 0xE00 | 0x3FFF | 0x1 | Miss | Way = 1 |
| 3 | 0xFFFC8 | 0x000 | 0x3FFF | 0x8 | --- | Invalidate Way 0 |
| 1 | 0xFFFC7 | 0x000 | 0x3FFF | 0x7 | Miss | Way = 0 |
| 1 | 0xD00FFFC0 | 0xD00 | 0x3FFF | 0x0 | Miss | Way = 2 |
| 1 | 0xC00FFFF7 | 0xC00 | 0x3FFF | 0x37 | Miss | Way = 3 |
| 1 | 0xB00FFFC7 | 0xB00 | 0x3FFF | 0x7 | Miss, Evict | LRU = 1 |
| 1 | 0xA00FFFC5 | 0xA00 | 0x3FFF | 0x5 | Miss, Evict | LRU = 0 |
| 1 | 0x900FFFD3 | 0x900 | 0x3FFF | 0x13 | Miss, Evict | LRU = 2 |
| 1 | 0x800FFFE2 | 0x800 | 0x3FFF | 0x22 | Miss, Evict | LRU = 3 |
| 1 | 0xB00FFFC7 | 0xB00 | 0x3FFF | 0x7 | Hit | Way = 1 |
| 1 | 0xA00FFFC5 | 0xA00 | 0x3FFF | 0x5 | Hit | Way = 0 |
| 1 | 0x900FFFD3 | 0x900 | 0x3FFF | 0x13 | Hit | Way = 2 |
| 1 | 0x800FFFE2 | 0x800 | 0x3FFF | 0x22 | Hit | Way = 3 |

For L1 Data Cache:

No. of reads = 0

No. of writes = 13

No. of misses = 9

No. of hits = 4

Hit Ratio = 0.3076

### Trace FIle 2

1. Only reads to both L1-D and L1-I
2. Hits to a cache line in the set which has at least one empty line.
3. Cache read hit for different bytes in the same line
4. Read hits to all lines in the set for both caches

Table 2: Trace file 2 entries

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Command** | **Address** | **Tag Bits (12)** | **Index Bits (14)** | **Offset Bits (6)** | **Hit/Miss** | **Comments** |
| 0 | 0xF000FFF0 | 0xF00 | 0x3FF | 0x30 | Miss | Way=0 |
| 0 | 0x E000FFF0 | 0xE00 | 0x3FF | 0x30 | Miss | Way=1 |
| 0 | 0x F000FFF7 | 0xF00 | 0x3FF | 0x37 | Hit | Way=0 |
| 0 | 0x D000FFF0 | 0xD00 | 0x3FF | 0x30 | Miss | Way=2 |
| 0 | 0x C000FFF0 | 0xC00 | 0x3FF | 0x30 | Miss | Way=3 |
| 0 | 0x B000FFF0 | 0xB00 | 0x3FF | 0x30 | Miss, Evict | LRU=1 |
| 0 | 0x A000FFF0 | 0xA00 | 0x3FF | 0x30 | Miss, Evict | LRU=0 |
| 0 | 0x 9000FFF0 | 0x900 | 0x3FF | 0x30 | Miss, Evict | LRU=2 |
| 0 | 0x 8000FFF0 | 0x800 | 0x3FF | 0x30 | Miss, Evict | LRU=3 |
| 0 | 0x B000FFF0 | 0xB00 | 0x3FF | 0x30 | Hit | Way=1 |
| 0 | 0x A000FFF0 | 0xA00 | 0x3FF | 0x30 | Hit | Way=0 |
| 0 | 0x 9000FFF0 | 0x900 | 0x3FF | 0x30 | Hit | Way=2 |
| 0 | 0x 8000FFF0 | 0x800 | 0x3FF | 0x30 | Hit | Way=3 |
| 2 | 0x F000FFF0 | 0xF00 | 0x3FF | 0x30 | Miss | Way=0 |
| 2 | 0x F000FFF1 | 0xF00 | 0x3FF | 0x31 | Hit | Way=0 |
| 3 | 0x F000FFF1 | 0xF00 | 0x3FF | 0x31 | --- | Invalidate Way=0 |
| 2 | 0x F000FFF1 | 0xF00 | 0x3FF | 0x31 | Miss | Way=0 |
| 2 | 0x E000FFF0 | 0xE00 | 0x3FF | 0x30 | Miss | Way=1 |
| 2 | 0x D000FFF0 | 0xD00 | 0x3FF | 0x30 | Miss, Evict | LRU=0 |
| 2 | 0x C000FFF0 | 0xC00 | 0x3FF | 0x30 | Miss, Evict | LRU=1 |
| 2 | 0x D000FFF0 | 0xD00 | 0x3FF | 0x30 | Hit | Way=0 |
| 2 | 0x C000FFF0 | 0xC00 | 0x3FF | 0x30 | Hit | Way =1 |

Table 3: Trace file 2 statistics

|  |  |  |
| --- | --- | --- |
| **Parameter** | **L1 Data Cache** | **L1 Instruction Cache** |
| Number of reads | 13 | 8 |
| Number of writes | 0 | 0 |
| Number of misses | 8 | 5 |
| Number of hits | 5 | 3 |
| Hit Ratio | 0.3846 | 0.375 |

### Trace FIle 3

1. Both Reads and Write to L1-D cache tested
2. Consecutive Read, Write, Invalidate and Read to the same line tested
3. Print command tested
4. Reset command tested
5. Printing statistics at the end of line

Table 4: Trace file 3 entries

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Command** | **Address** | **Tag Bits (12)** | **Index Bits (14)** | **Offset Bits (6)** | **Hit/Miss** | **Comments** |
| 1 | 0x4003C0 | 0x400 | 0xF | 0x0 | Miss | Way=0 |
| 0 | 0xF003C2 | 0xF00 | 0xF | 0x0 | Miss | Way=1 |
| 1 | 0xF003C4 | 0xF00 | 0xF | 0x4 | Hit | Way=1 |
| 3 | 0xF003C2 | 0xF00 | 0xF | 0x0 | --- | Invalidate Way1 |
| 0 | 0xF003C2 | 0xF00 | 0xF | 0x0 | Miss | Way=1 |
| 9 |  |  |  |  | Print | Print stats |
| 8 |  |  |  |  | Reset | Clear |
| 0 | 0xF003C2 | 0xF00 | 0xF | 0x0 | Miss | Way=0 |

For L1 Data Cache:

|  |  |  |
| --- | --- | --- |
| **Parameter** | **After '9' command** | **After end of file** |
| Number of reads | 2 | 1 |
| Number of writes | 2 | 0 |
| Number of misses | 3 | 1 |
| Number of hits | 1 | 0 |
| Hit Ratio | 0.25 | 0 |

### Trace FIle 4

1. Presence of an invalid command (other than 0,1,2,3,8 and 9)
2. Address greater than 32-bit - Extra bits starting from MSB truncated
3. Additional White spaces in the trace file

Table 5: Trace file 4 entries

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Command** | **Address** | **Tag Bits (12)** | **Index Bits (14)** | **Offset Bits (6)** | **Hit/Miss** | **Comments** |
| 7 | 0x 4003C0 | 0x400 | 0xF | 0x0 | --- | Bad Command |
| 0 | 0x F00F003C2 | 0xF00 | 0xF | 0x0 | Miss | Address greater than 32 bits |
| 2 | 0x F000FFF0 | 0xF00 | 0x3FF | 0x30 | Miss | White space tested |

# Implementation

This section deals with the implementation details of the simulator.

## **Choice of Language**

We chose C++ to implement the simulator as opposed to Verilog or C. Key driver behind this was the implementation of two caches: L1 Instruction and L1 Data and their interface to another cache. We modeled a generic cache using an abstract class. This provides a consistent interface for all caches. The L1 data and instruction caches implement this interface as a write through cache. Design of the L2 stub also became easier, since it minimally implements this common interface.

## **Development environment**

We used Microsoft Visual Studio C++ 2012 environment to design, develop and test our simulator. The project was built as an executable on the Windows Platform.

## **Class Diagram**



Figure 5: Class Diagram

## **File Dependencies**

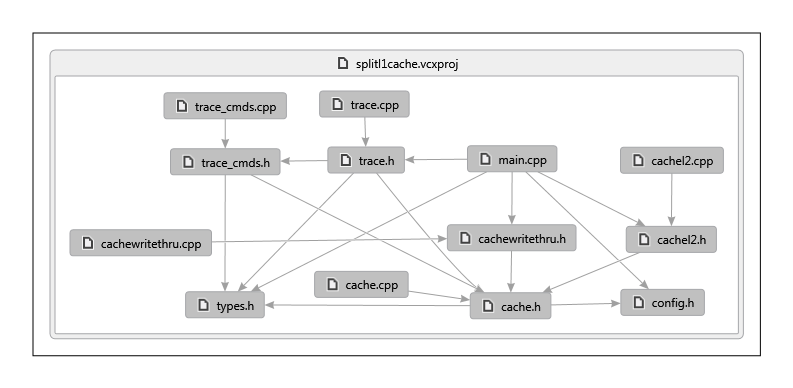


Figure 6 : File Dependency Graph

Table 6: File Organization

|  |  |  |
| --- | --- | --- |
| **File** | **Module** | **Description** |
| Types.h | Global | Contains type definitions for the entire project. |
| Config.h | Global | Configuration parameters such as Line size, number of sets etc |
| Cache.h | Cache | Public interface for class Cache |
| Cache.cpp | Cache | Implementation of class Cache |
| CacheWriteThru.h | CacheWriteThru | Public interface for class CacheWriteThru |
| CacheWriteThru.cpp | CacheWriteThru | Implementation of class CacheWriteThru |
| CacheL2.h | CacheL2 | Public interface for class CacheL2 |
| CacheL2.cpp | CacheL2 | Implementation of CacheL2 |
| Trace.h | Trace | Public interface for class Trace |
| Trace.cpp | Trace | Implementation of class Trace |
| Trace\_cmds.h | Trace | Trace command function headers |
| Trace\_cmds.cpp | Trace | Trace command function implementations |
| Main.cpp | Global | The integration of all modules |

## **Error Handling**

1. Arguments were checked before proceeding in any function.
2. Exceptions handling used to catch memory allocation failures for the new operator.
3. All pointers checked for NULL before dereferencing
4. All new lines / empty lines in trace file skipped
5. Invalid command line arguments checked
6. Invalid commands in the trace file tagged as bad commands and skipped.
7. Memory leaks checked using the *\_CrtDumpMemoryLeaks()* function of Microsoft Windows C runtime.

## **Maintainability**

The trace commands can be easily extended. To add new commands, the user should add his command to the enumeration in types.h, assign the proper number to the enumeration (command number). Then the user can define the implementation of his command in trace\_cmds.h / .cpp. Finally, he should add the enumeration and function pointer in the array trace\_commands in trace.h. This provides a modular way for adding commands without use of complex switch case based implementations. This also results in minimal code changes.

The class design provides an abstract base class which serves as an interface for any future caches to implement. This is generic enough for all kinds of caches. They could make use of this to interact with each other in a consistent fashion.

### Repository

We used Subversion for source control. The repository is hosted on Google Code and provides up to 5 GB space for hosting projects. All previous revisions can be easily obtained via the repository. It also contains an issue tracking system for managing defects.

### Coding Standards

Coding standards are used to

1. Organize code better
2. Eliminate possible hazards
3. Maintain consistency
4. Improve readability
5. Ease of maintenance
6. Achieve Portability

We used the C++ standard recommended by Google Inc. ([The Google C++ Style Guide](http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml)) Some features which picked up are:

1. Use of custom primitive data types: UINT8, UINT16 etc. (Portability)
2. One header file and cpp file per class. (Modularity)
3. Consistent naming schemes for Classes, variables, enums (Consistency, Readability)
4. Use of literal as first comparison argument e.g. (0 == my\_var) (Avoid possible pitfalls)
5. Use of const references whenever possible (Optimal implementation)

# APPENDIX

## SOURCE CODE

## TRACE FILE

## SIMULATION RESULTS