MOESI Invalidation Protocol Project Report

Herbert Aruya

***Abstract*—Report on my code base in which implements a cache simulator using the MOESI Invalidation protocol. This report will outline the thought process and problems faced before arriving to the finished solution.**

I. Introduction

FOR this program, the code base implements a solution for a proposed cache simulator. This does not in any way handle the hardware logistics that comes along with cache but more of a higher level abstraction to what it is. This simulation implements a MOESI Invalidation Protocol for the cache. Ontop of this there are two types of cache configurations implemented within this solution: Direct map single cache and LRU direct map double cache. As this is a simulation, the background processing is not visible to the user but there are statistics that are outputted by the program to get a bit of insight on what occured in these processes. These statistics include,

1) Cache-to-Cache Transfers
2) Cache-to-Memory Transfers
3) Dirty Writebacks
4) Final Configuration of Cache

II. Background

The MOESI Invalidation protocol, that is the main staple of this program, is an invalidation protocol used by the cache to manage cache, done by imploying the idea of states. MOESI itself stands for the states that are in this protocol; Modified, Owner, Exclusive, Shared and Invalid. The use of these states and interactions within help the cache remain stable as dirty writeback, write back to main memory, is a key issue in performance. By minimizing this dirty writeback, the computer becomes faster as a result. This protocol can be applied to cache in general. For this code base, there are two types of caches implemented direct mapping and least recently used. For both of these configurations, abstractions are simple but implementation is not. In the case of direct mapping, the cache only has one line, directly mapping to the index given to by the processor. But for the least recently used configuration, the processor needs more than one cache line, as one would just degenerate to the previously stated direct mapping. The benefit of this lies in the fact that an index of the cache is now able to hold more than one tag, decreasing the number of times needed to go to main memory.

III. Project Specification

As for my original plan with this project, it wasn't supposed to me as modular as it is now at its final implementation. Through research and further understanding of the project and problem set, the demand of modularity came as a realization to how many different independent components are working together to form a cache. This idea was further supported when I began to understand how each step of the process works and builds on top of the last step.

Professor: Chenyun Pan

IV. Problem Analysis/ Solution Design

Understanding the problem set is important; With implemlementing cache there comes a need for modularity to maximize the performance and scaleability of the solution. For this code base this approach is taken to its full extent as its modules interact extensively throughout the program. These modules include, but not limited to:

1) User-Interface
2) Processor Manager
3) Processor Bus
4) Data Collection
5) Cache
6) Processor
7) Protocol

The transparency of the innate heirarchy between modules shaped the path to the arrival to this solution as it is used to model the inheritance used within this program. Although only a couple, these parent-child relationships aid in the validity of the final solution. Each class, of course, is important to the final solution generated. To begin, the front end is used to generate a simple interactive user interface to set up the program. This along checks for the validity of files provided then starts up the process manager. The process manager in this case is used to feed the commands from different user given files to each processor. Remark that for the execution for these processors to remain valid, a processor bus is needed to direct signals between processors; This is also done and initialized within the process manager class. Now the remaining modules are simple, the processors execute one instruction at a time, generating signals if needed. The bus as said before facilitates signals between processors. And the cache, beloning in each processor class, is used to store addresses recently used by processor.

V. Evaluation

While programming, a few problems arose. The first of two problems came when designing the program, from the vocality of this report its not suprising that there were challenges stemming from modularity, it wasn't perfect. While trying to make the final program as cohesive as possible, a bit of coupling occured that couldn't have been avoided. When collecting data, the class had to intermingle with the other classes to properly and accurately collect data needed. But in the end, the problem

was minimized as much as possible while keeping the data with the same percentage of accuracy. The second problem arose from the misunderstanding of material, while trying to generate a solution for the extra credit confuguration, I as a programmer was confused. The final result is where I would personally like it to be but I tried my best all the way till the finsh line.