ECE411 MP3 FINAL REPORT

THE JAGERMEISTERS
Ali Bhujwala, Vivek Britto, Anuj Pasricha

1. Introduction

The purpose of this machine problem is to develop a fully-functional LC-3b processor. The primary focus before ECE411 was to understand the design of a processor that functions correctly. Through ECE411, the primary focus has been to not only ensure correctness in a processor, but also to ensure that the processor performs well. This documents presents in detail the approach we took to improve performance of the LC-3b processor. In addition to the required 5-stage pipeline, 2-way set associative L1 data and instruction caches, a unified L2 cache, hazard detection, and data forwarding, we implement 6 advanced design options - branch target buffer, victim cache, local branch history table, multi-cycle L2 cache, 4-way set associative L2 cache, and true Least Recently Used (LRU) replacement policy for L2 and victim caches.

Processor design is hard and interesting. We will go into detail why that holds true as we go into implementation and testing detail for each advanced design option implemented in the following sections. In section 2, we provide an overview of the project, presenting our initial motivations for pursuing the advanced design options that we did. In section 3, we dive into the technical details of our project and perform a thorough performance analysis for each advanced design option implemented. We present additional observations in section 4 and we conclude by presenting important takeaways about our implementation and processor design, in general, in section 5.

2. Project Overview

We learned about how changing cache dimensions and criterion influenced overall processor performance through charts of real-world processors and benchmarks. However, our team wanted to see first-hand how cache parameters really affected performance. Therefore, we decided to focus on cache-intensive advanced design options.

The division of labor within our team was very straightforward. Due to Anuj's and Ali's failed attempts at getting the branch target buffer and local branch history table to work, Anuj and Ali worked alongside Vivek in developing the advanced cache options. Anuj was in charge of expanding the L2 cache. Ali was in charge of implementing true LRU for victim and L2 caches. Vivek implemented the victim cache and with the assistance of Anuj and Ali, debugged the entire pipeline using Checkpoint 3 code.

The major thing we realized is how hard it is to implement even a 5-stage pipeline with advanced performance features. Time management and conceptual understanding aside, it requires careful, logical thinking to go from the design stage to the implementation stage. Anuj and Ali have a very strong conceptual understanding of branch target buffer

and local branch history table, however it was difficult for them to understand how all the components integrated well with the pipeline. However, our most notable achievement was boosting frequency by almost 25MHz by implementing the multi-cycle L2 cache option. We did not realize we could get such a massive frequency gain from such a minor addition.

3. Design Description

a. *Overview*

In addition to the basic design as described in the MP3 design document, we decided to implement a victim cache, a 4-way set associative L2 cache, true LRU for victim and L2 caches, and a multi-cycle L2 cache. We go into detailed descriptions of each advanced design option below.

Due to the time crunch, we only tested our final processor using the Checkpoint 3 test code. We did not develop our own test code because we found the variety of instructions in CP3 test code to be ample.

b. *Advanced Design Options*

i. Option 1: Victim Cache

1. Design

   A victim cache serves as an intermediary cache connected to the L1 data cache. We can think of the victim cache as a recycling cache, wherein discarded data from the L1 data cache is stored, in case it will be needed in the future - the price of such access in terms of performance won't be very high as opposed to fetching from the L2 cache or even main memory. When something is ejected from (in our case) L1 data, it is stored in a fully-associative, 4 entry victim cache. If a memory access does not find a value in L1 data, it first looks in the victim cache. If it matches a tag there, it swaps the value there with the value to be ejected from the L1 data cache. If nothing is found in the victim cache, the memory is fetched straight from L2 into L1 data (not to the victim cache). Our victim cache used true LRU for replacement.

   We decided to connect the victim cache to our L1 data cache because no ejections occur from the L1 instruction cache. We could have connected the victim cache to our L2 cache, however that would mean that the victim cache could also contain instructions at any point in time, which as we stated before, would be a waste of space in the victim cache.

2. Testing

We ran into several issues when testing our victim cache with the CP3 test code. First, we need to ensure that the old value in the victim cache is written back to L1 data in case of a swap, not the new value just stored (swapped). We solved this issue by buffering previous values and storing on next cycle.

Second, we need to keep track of dirty values between L1 data and the victim cache, as values going from victim to L1 CAN be dirty (unlike any other situation of getting values from a lower cache level). We solved this issue by passing dirty values, along with data, between L1 data and victim caches.

3. Performance Analysis

   Addition of the victim cache led to a speedup of 1.255x as the run time for CP3 code decreased from 1.096ms to 0.873ms. However, we noted that the maximum frequency decreased from 87.7MHz to 78.78MHz. Note that this may be caused due to the fact that the processor has to write to another cache (the victim cache) instead of simply evicting the line from the L1 data cache - this could result in additional computational cycles, thereby reducing overall frequency.

   In the future, we would like to try to isolate the performance of the victim cache because with how we currently measure run time, the speedup of 1.255x is caused by the addition of both the victim cache and the 4-way L2 cache (+true LRU). We do not think that the expanded L2 cache plays a major role in this performance spike, however we are not completely certain.

ii. <u>Option 2: 4-way set associative L2 cache</u>

1. Design

   This was the easiest and most hassle-free part to implement. We simply added two additional ways to each set of the L2 cache, hence making it a 4-way set associative L2 cache. It uses the same connections as the 2-way set associative L2 cache, however it includes additional slots in the data, tag, valid bit, dirty bit, and LRU arrays to support all 4 ways.

2. Testing

   Like with any cache testing, we ran the CP3 test code on our design and verified the correctness of the design. In addition, we looked that individual cache reads and writes to see whether they were being performed correctly by seeing the mem_wdata and

mem_rdata values. Each write and read corresponded correctly with our state machine and how a cache should generally function.

3. Performance Analysis
   Tested individually, this option provided a performance boost of 3MHz, which wasn't significant at all. It lead to a marginal decrease in runtime due to the decreased likelihood of the cache line being evicted from the cache (as a result of increased space in the cache). We would like to conduct more rigorous tests in the future, isolating the 4-way L2 test from the victim cache test. Due to poor time management on behalf of the group, we were not able to conduct unit tests for each added advanced design feature. As the table in the appendix shows, we combined tests for victim cache and 4-way L2 (+true LRU).

iii. Option 3: True LRU for Victim and L2 caches

1. Design
   For L2 cache and victim cache, we implemented a true LRU scheme. For each set of the cache, an LRU "queue" was created, keeping track of the order of previous accesses. If a particular way in a set was accessed, the appropriate way number in that set's stack was taken from its current position and placed at the bottom of the queue. We did not run into any major issues during the implementation and testing processes.
2. Testing
   We tested for correct by implementing true LRU for both caches and running the CP3 code. Our testing produced the correct register outputs. However, in order to see whether true LRU really worked, we tracked the LRU bits on the waveform. We found that the LRU bits were set at the correct positions on the waveform for both caches, i.e., when, for example a new line was brought into a cache, thereby making the last accessed line least recently used.
3. Performance Analysis
   This option did not have a significant impact on performance. As stated above, we tested the victim cache and the 4-way L2 with LRU in the same trial, so we were not able to distinguish between the performance impact of the two. Previous tests showed that increasing the number of ways results in a 3MHz improvement in performance and we believe that the rest of the frequency decline is caused by the victim cache because it increases the critical path for the TRAP instruction. The true LRU policy does not slow

down or speed up the line eviction process - it is simply a way to keep track of the least recently used line.

iv. Option 4: Multi-cycle L2 cache

1. Design

   This advanced design option was implemented to reduce the critical path length by reducing the maximum register to register length. A buffer was created at the exit of the cache arbiter to hold mem_address and mem_wdata until the next cycle, and the L2 cache controller was set to also delay the fetching of data by one cycle upon receipt of a mem_read or mem_write request. We did not run into many design issues with this option. Adding an extra delay state in out L2 cache state machine and creating the additional registers did the trick. We kept missing one or the other, but when we realized both factors were necessary, it worked as intended.

2. Testing

   There was no real way to test this. After implementing all the other options, our maximum frequency was below the 100MHz threshold. After implementing this option, the frequency shot up, but the design also stayed functionally correct, as described previously. Therefore, this option worked to our advantage. We also noted state transitions for the L2 cache to the delay state and from the delay state to the next functional state. These transitions behaved as expected.

3. Performance Analysis

   It was interesting to note that even though the maximum frequency increased from 78.78MHz to 101.58MHz, our run time for CP3 code increased from 0.873ms to 0.895ms. We realize that this increase in run time was caused by the delay state we added to improve frequency.

4. Additional Observations

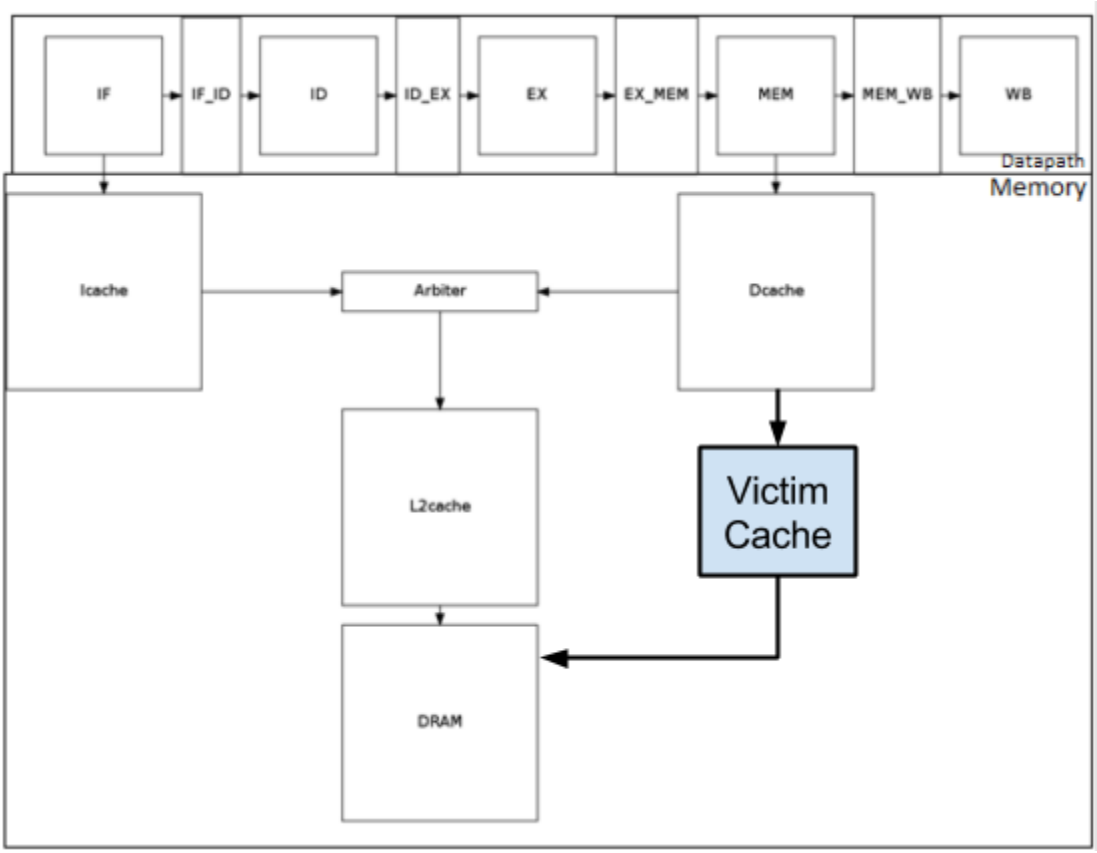
Before implementing multi-cycle L2, we were trying alternative ways to boost the maximum frequency. One of the things we tried was reducing the number of sets in the L2 cache. This reduction boosted the frequency by 3MHz, but not too much. We were blindly experimenting with cache dimensions at this point, when another group suggested implementing the multi-cycle L2 cache, which resulted in a massive performance boost.

5. Conclusion

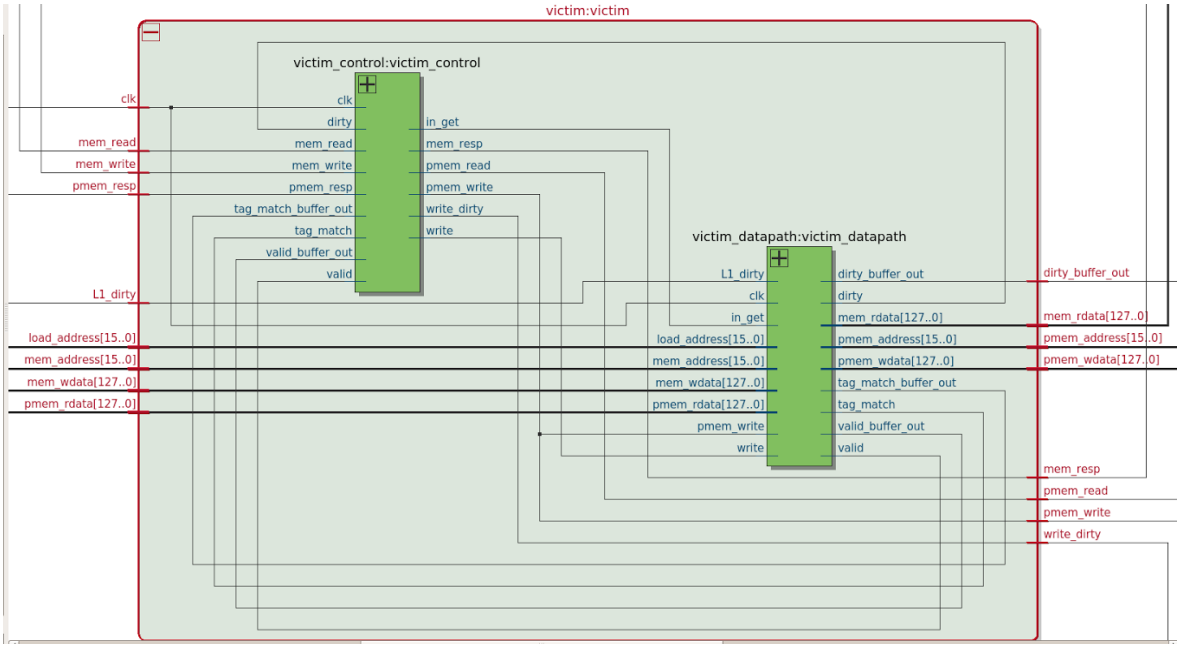
To conclude, increasing the number of ways in a cache boosts runtime, but decreases frequency. Additionally, introducing a multi-cycle cache boosts frequency at the expense

of runtime. We set out to gain a solid understanding of how cache parameters and different types of caches impact processor performance. We have successfully shown how additional ways in the L2 cache and the victim cache reduced runtime and we have also shown how making L2 multi-cycle boosts maximum frequency, at the cost of runtime. Everything we implemented was fairly standard, based on how such advanced design options are actually implemented. There was some debate over whether the victim cache should connect to the L2 cache or the L1 data cache, however we settled on the latter option for reasons mentioned before. It was also very fascinating to note how boosting processor performance is only a matter of tweaking parameters and finding the good balance between frequency and runtime.
However, the biggest takeaway for us from this assignment was realizing how hard, but interesting, processor design is. Pipeline complexity grows dramatically as we add more features, so coming up with a good organization scheme would serve us well in the future.

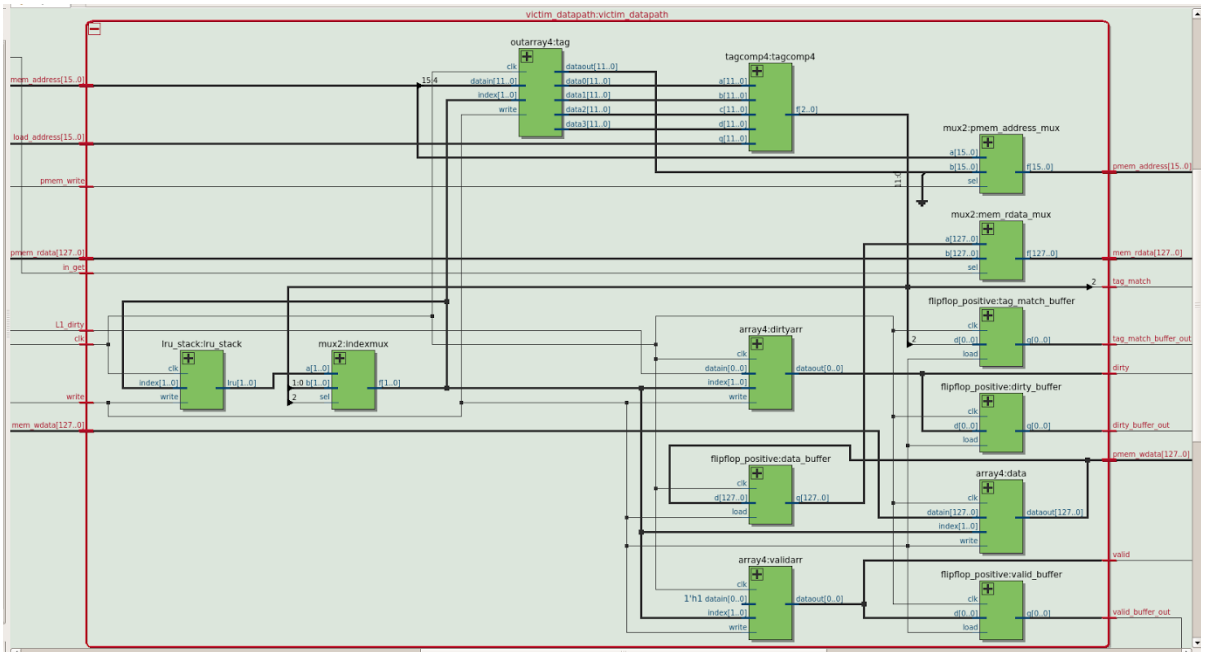
DIAGRAMS:



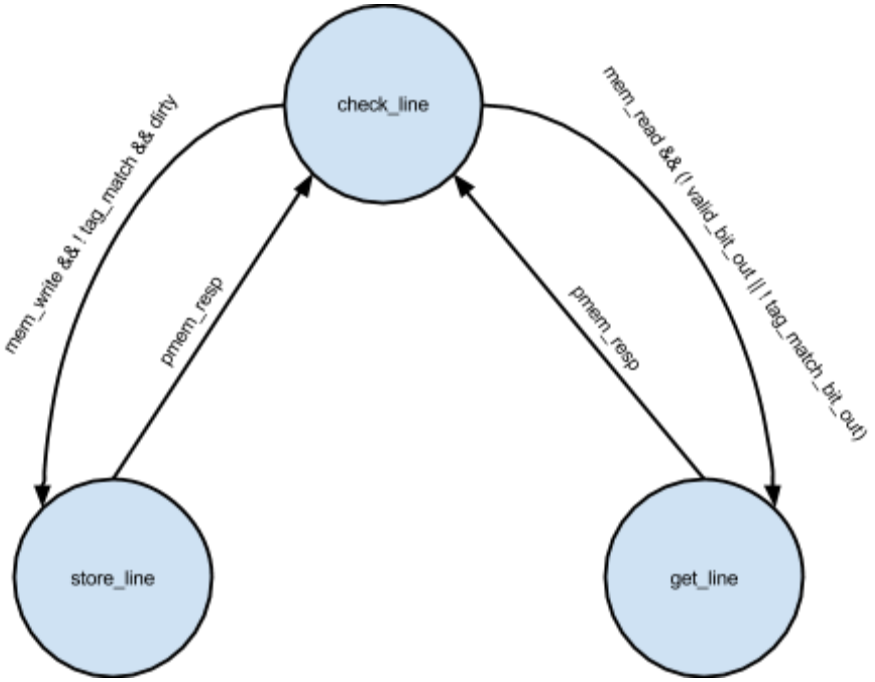
Modified Block Diagram (our high level contribution to the overall design - all other options were specific to existing blocks)



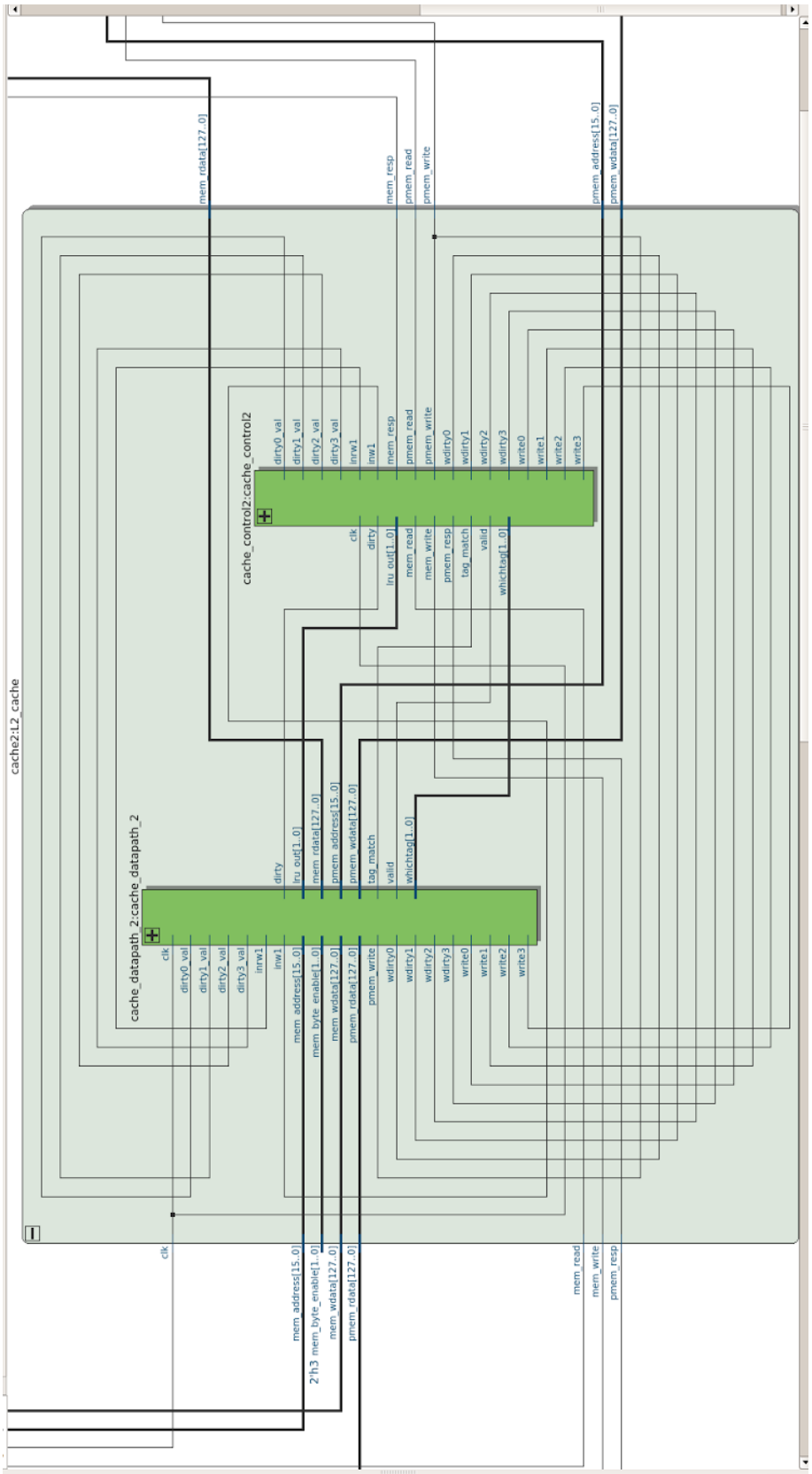
Victim cache high level layout



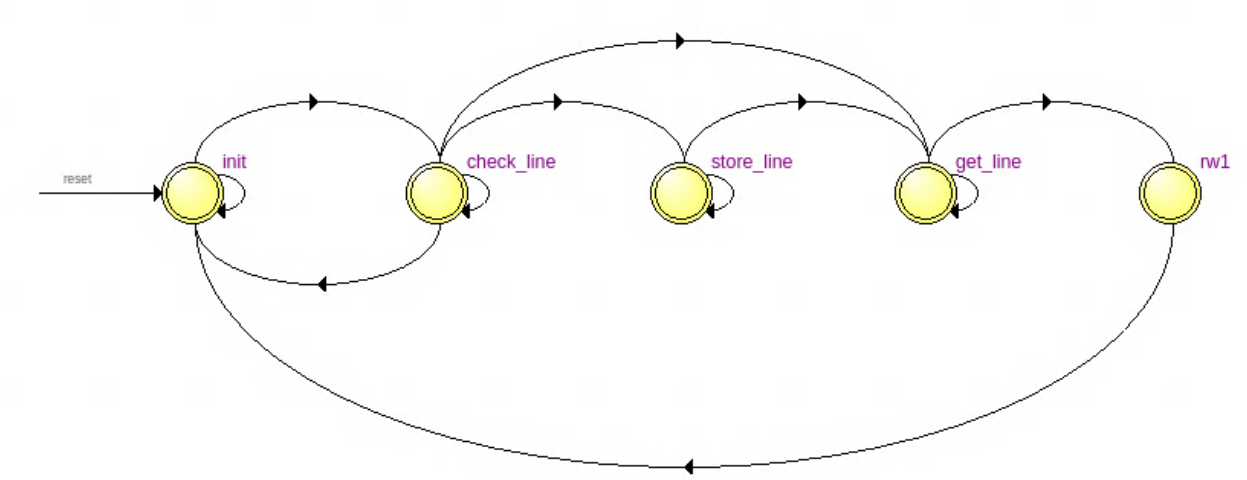
Victim cache (true LRU) datapath



State machine for victim cache



4-way L2 cache (multicycle with true LRU)



State machine for L2 cache (note the extra state ‘init’ used for multi-cycle L2)

	Source State	Destination State	Condition
1	check_line	store_line	(dirty).(!mem_read).(mem_write).(!always1) + (dirty).(mem_read).(!always1)
2	check_line	init	(always1)
3	check_line	get_line	(!dirty).(!mem_read).(mem_write).(!always1) + (!dirty).(mem_read).(!always1)
4	check_line	check_line	(!mem_read).(!mem_write).(!always1)
5	get_line	rw1	(pmem_resp)
6	get_line	get_line	(!pmem_resp)
7	init	init	(!mem_read).(!mem_write)
8	init	check_line	(!mem_read).(mem_write) + (mem_read)
9	rw1	init	
10	store_line	store_line	(!pmem_resp)
11	store_line	get_line	(pmem_resp)

State transition table for L2 cache

Testing Point	Run time for CP3 code	Maximum frequency
after cp2	n/a	106.75MHz
after cp3	1.096 ms	87.7MHz
After 4 way LRU L2 and victim cache	0.873 ms	78.78 MHz
After Multi-cycle L2	0.895 ms	101.58 MHz