Lab 3: Blocking Cache

Asma Ansari (ara89) & Nathan Vogt (ncv7)

I. Introduction

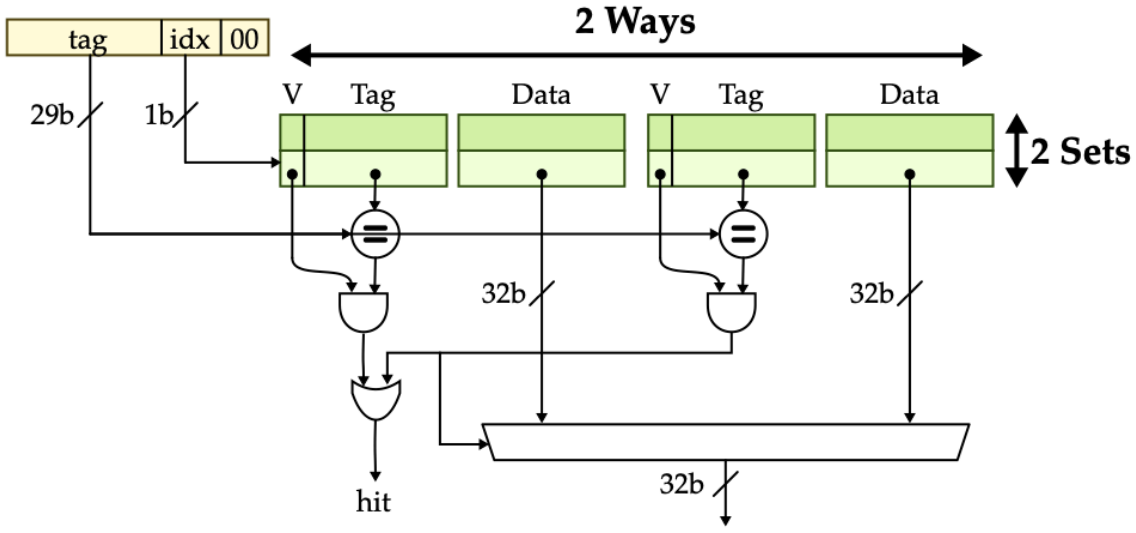
Memory accesses tend to be incredibly lengthy compared to other modules within a data path. Caches can decrease the time for memory accesses by storing frequently accessed addresses and data. Spatial and temporal locality are key features that caches utilize to maximize cache hits. There are three types of caches: direct-mapped, multi-way set-associative, and fully associative. This baseline design for this lab is a direct-mapped cache while the alternative design is a two-way set-associative cache. Direct-mapped caches are limited by the index associated with a particular address. The two-way set-associative cache allows multiple tags to be stored at one address. Furthermore, multi-level caches utilize different cache designs to speed up data accesses that are "closer" to the processor. Thus, this lab's objective is to illustrate the benefits of using more associative caches (though fully associative caches do not necessarily equate to quick data access times).

II. Alternative Design

The alternative design is a two-way set-associative cache. We didn't have enough time to implement the alternative design for this lab, so we will discuss how we would have approached the implementation of this cache design. The primary difference between the baseline and alternative design is the setup of the tag and data arrays. In the baseline design, the cache could reference one cache line with a given index. The alternative design will maintain the same cache capacity but will allow two cache lines to be accessed at the same index. This means the number of sets will be halved, which reduces the number of bits in the index by 1 bit.

In the datapath, we would create duplicate tag and data arrays to store values for each way in each set. We would also add a register file to keep track of the LRU value in each set. During read/write operations, both ways of each set will be checked for a match, and the LRU bit for that way will be updated upon a match.

Figure 1: New datapath unit for a two-way set associative cache.



III. Testing Strategy

Category	Tests
Directed Test Cases	● WRITE_INIT ● WRITE_HIT ● WRITE_MISS ● READ_HIT ● READ_MISS ● EVICT
Randomized Test Cases	● Read/write commands using random data values.
Direct-Mapped Cache	● Read/write to the same cache line. ● Cause conflict misses. ● Multiple writes to the same location
Set-Associative	● Tests whether a set can be written to twice to populate both ways. ● Check replacement policy (Least Recently Used).
Banking	● Checks whether address bits are assigned correctly (tag, index, bank, and offset bits).

Table 1: Test table outlining what was being verified by the test cases.

Initially, we were given the directed and banking test cases to verify our baseline design. From there, we added some corner cases, randomized data value testing, and tests for each cache design. One corner case we came up with is the scenario in which the direct-mapped cache is underutilized and causes unnecessary evictions. These test cases allowed us to test each specific transaction thoroughly before moving on to implementing a different transaction or the alternative design. Once we verified that the transactions were all correct and that our baseline design was functional, we could move on to the alternative design and even begin optimizing it. Incremental testing allowed us to pinpoint where our design was failing such that we could focus on specific areas of the datapath and/or control unit.

The test cases specifically for the set-associative cache seeked to verify its eviction and replacement policy. A key aspect of this cache design is the two “ways” in each set: one index can access both “Way 0” and “Way 1”. It is up to the rest of the hardware to determine the tag match and set the Least Recently Used (LRU) bit during cache transactions. Therefore, we designed our set-associative cache tests such that they wrote two cache lines at the same index and read from them to verify that both cache lines were being stored in the same set. Moreover, the tests to check for the LRU replacement policy filled at least one set within the cache completely and then tried to access data that was not in the cache at that time. Thus, one of the cache lines would need to be evicted which would be either whichever cache line is older and/or has not been accessed most recently.

All of the tests were set up with a black-box design with inputs passed in and an expected output listed. This simplified our test development process since we could reuse the same commands to check for different behaviors. Ultimately, we are trying to read/write/evict cache lines and words so depending on the cache design, the inputs and expected outputs were changed accordingly. We supplemented this by utilizing the provided traces to quickly check inputs and outputs and have a more simple illustration of which test functions are failing. Furthermore, any tests we developed were run on the FL model first so that we can ensure our tests are not incorrect.

IV. Evaluation

Given that we did not complete the alternative design, the discussion of the alternative design's performance will be hypothetical but rooted in what we learned in class. From Topic 4, we learned that increasing associativity and/or the cache capacity usually increases the hit rate. Additionally, maintaining the same cache capacity while increasing the associativity can decrease the hit rate until it matches that of a direct-mapped cache with a larger capacity.

Average Memory Access Latency (AMAL) Analysis

The average memory access latency (AMAL) is an important performance metric for caches since it considers the hit latency, miss rate, and penalties for misses. Since both cache designs have the same capacity and cache line size, the increased associativity of our alternative design has a lower miss rate compared to the baseline design. The alternative design specifically lowers conflict misses since each index maps to one set that can hold up to two cache lines. Therefore, data accesses with high spatial and temporal locality benefit from a more associative cache design versus a direct-mapped cache which would have to evict the cache line each time.

Despite this improvement in the AMAL, the hit latency worsens with our alternative design. Tag matching takes longer since the system must check multiple locations in the set to find a tag match (if any). Essentially, confirming a cache hit or miss takes longer as associativity increases. However, the overall performance of the set-associative cache is likely better since there are fewer misses. Misses require the system to check main memory which takes significantly longer than checking the cache. Ultimately, direct-mapped caches will access main memory more often, meaning the cache's performance will take the miss penalty more often as well.

Area & Energy Analysis

Direct-mapped caches are relatively straightforward with their replacement policies, essentially evicting cache lines every time there is a conflict miss. Increasing the number of ways requires more nuanced replacement policies and handling of these policies within the hardware. We used the Least Recently Used (LRU) replacement policy which is represented as a bit associated with each way within a set. Every time a new address maps to a set that is full, this replacement policy determines which cache line gets evicted first.

In our theoretical design discussed above, we used a regfile to keep track of these LRU bits which adds an additional module to the overall area of the design. Furthermore, we would duplicate the tag and data arrays such that each array would represent Way 0 and Way 1 for each set. Notably, keeping track of the LRU bits and updating them accordingly worsens the access latency because there are more operations occurring during one data access (regardless of whether the transaction is a hit or miss).

Furthermore, this increase in area impacts the power consumption of each cache transaction within the two-way set-associative cache. More hardware to support more operations within one access consumes more power to support the additional logic gates. Therefore, there is a tradeoff between power consumption and performance with the cache design.

V. Conclusion

We conclude that the two-way set-associative cache performs significantly better than the direct-mapped cache due to a decrease in miss rate. The miss rate would be approximately the same as a direct-mapped cache that is two times the size as the two-way set-associative cache in our lab. Programs are continuously accessing data from memory to execute tasks properly, so having frequently used data readily available is crucial to speeding up the program itself. However, it is important to note that direct-mapped caches have their own use cases: if a designer is aiming for power optimization, using a direct-mapped cache is one way to achieve that while also providing some method to speed up memory accesses.

VI. Work Distribution

Nathan worked more extensively on this lab compared to Asma. We agreed to split up the work such that we can focus on optimizing Lab 2 and Lab 3 simultaneously in preparation for Lab 4. Unfortunately, due to timing

constraints, Nathan and I were unable to complete the alternative design for this lab. Asma completed the lab report and worked on setting up the data path as well as some initial control signals in the control unit. Nathan implemented the majority of the FSM along with any additional modules and wires required for the datapath.