Modeling 101

Muhammad Mahmood, Easton Culver, Roman Millem, Octavio Villalaz

CECS525

Abstract

Memory is a large driving force in almost any form of computing, and as such there have been many methods devised to help streamline and improve memory as a whole. One of these methods is the use of cache memory, and it greatly improves the overall speed of the computer. However, there are more methods within cache memory that affect how it runs as well. Taking a look at four different sets of cache memory mapping, the hit rate will be recorded to determine how well each mapping method performs when judged to each other’s hit rates. The first will be direct mapped cache mapping, followed by fully associative mapping, then set associative, and finally 8-way set associative mapping. Each are recorded with a specific cache memory size, as well as a set number of memory locations to access.

Body

Direct mapped cache mapping is known for being one of the lowest lowest hit rate cache mappings that there is. This does tradeoff that it normally does not require a lot of cache memory to work. When given the 1 KB cache limit, and the 16 byte block size, the total amount of sets that were allowed was 64 sets within the cache memory. Similar to the 8 way set associative, it was created using an array of 64 separate sets within the rows, and then 2 different columns to account for the data as well as the tags. After initializing every last cache memory location to a value of empty, -1, it would then start going through each of the memory locations and see if the set they are supposed to be in is filled, if it isn’t then it counts it as a miss, but if it is a hit then it would count it as a hit. At the end the average hit rate of the direct mapped cache memory was around 28%.

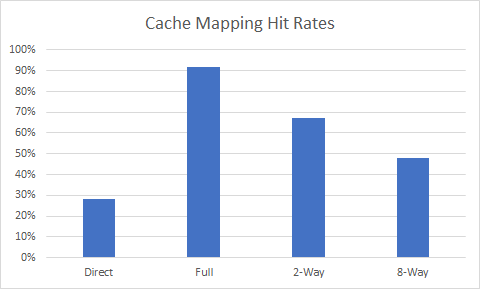
Fully associative cache mapping excels in having more hits and less misses, which up-front is amazing and hit ratio being the spec compared in the project it would prevail. Coming with the higher hit rate there is a sacrifice for the cost of this mapping method. Since any block of main memory can fit in any block of cache memory there is a lot of opportunity and cost wasted in searching. Further, in this mapping method there is no set bit for it looks for the association through the tag bit and has no specific block to go to. Using a 7 bit memory address and 1KB cache the hit rate was at 92% with 8 misses.

Set associative cache is a trade-off between direct-mapped and fully associative cache, it reduces conflict by providing N blocks in each set where the data is mapped. In a 2-way set associative the cache is grouped into sets and each set is composed of 2 blocks that will determine if the data can be stored, which produces a miss, or not, which produces a hit. Having a 1KB cache size and 16-byte blocks, meant that the cache will have 32 sets with each of those having 2 blocks of storage. Because a 7 bit address is used that meant the tag was composed of 2 bits, that represented in which block the data gets stored, the set of 5 bits and 1 bit for data. Testing it with a set of 100 numbers yielded a 67% hit ratio.

8 way set associative cache mapping allocated more lines per set in order to organize larger addresses more easily, and balance hit ratio with search time. When working with a 1KB cache with 16 byte block size, the cache would contain 8 sets. Each set contains 8 lines consisting of a valid bit, tag, set, and data block. To test hit ratio, this was simplified to an 8x8 array with the rows acting as sets and the columns acting as tags. Using 7 bit memory addresses, the most significant 3 bits are the tag bits, with the set bits being bits 2-4, and the data bit being the least significant bit. Each memory address is checked in the array, asking if cache[set][tag] is empty or full. Initially, the cache is filled with a value of -1 in each slot, signaling an invalid bit. If the position is found to be empty (a value of -1), the miss counter would increase and the position would be filled with the data (either 0 or 1). If the position was filled, the hit counter would increase. When tested with the set of 100 numbers used for all mapping methods the hit ratio was found to be 48%.

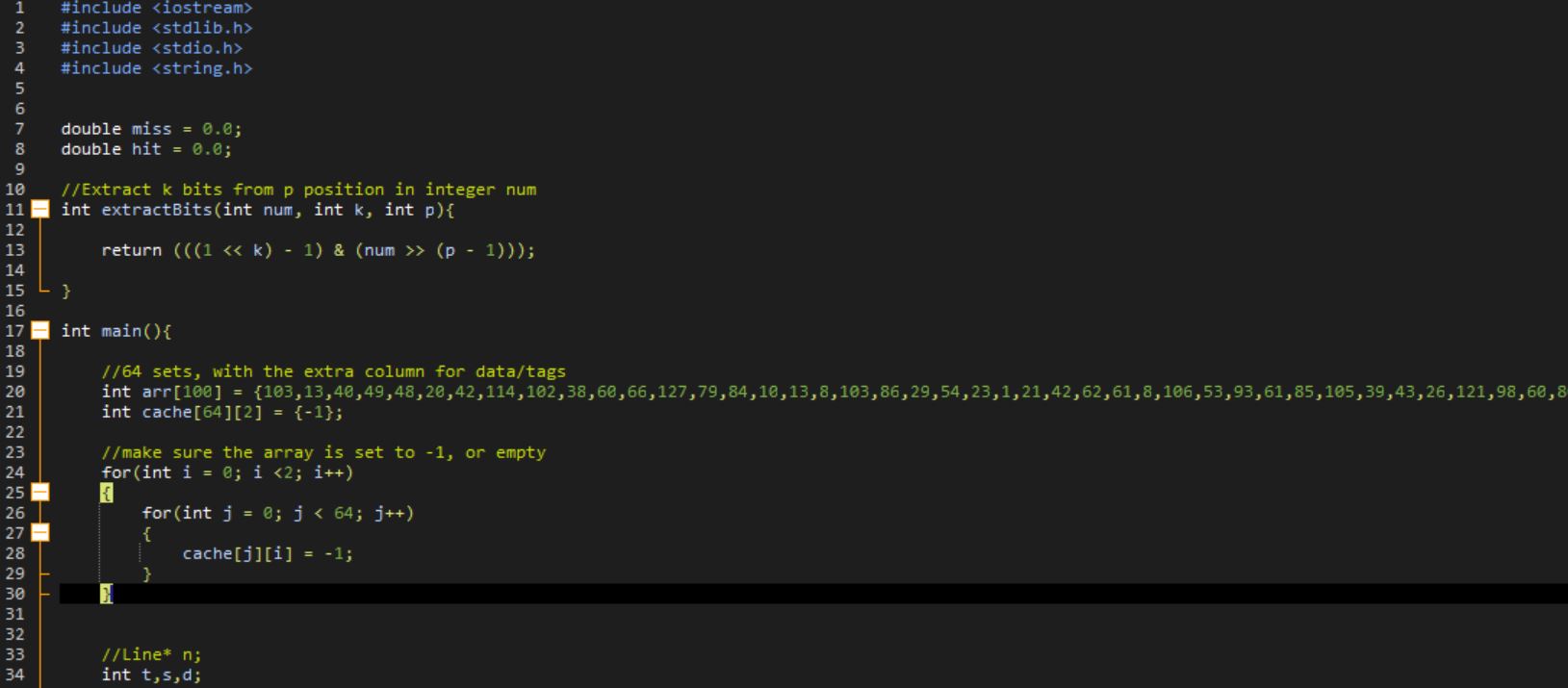
The figure below shows the resulting hit rates of the various cache mapping methods using the following list of memory addresses:

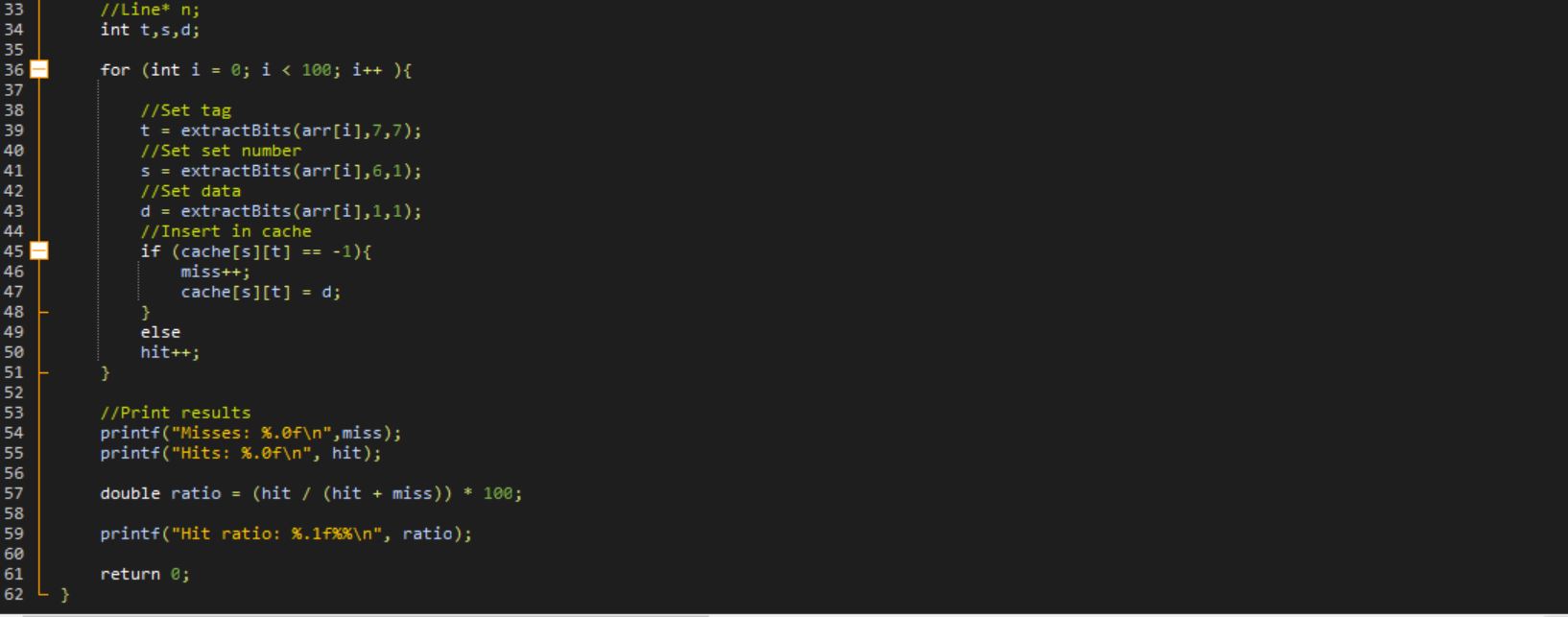
{103,13,40,49,48,20,42,114,102,38,60,66,127,79,84,10,13,8,103,86,29,54,23,1,21,42,62,61,8,106,53,93,61,85,105,39,43,26,121,98,60,80,65,35,73,19,6,58,44,33,60,12,40,18,80,60,22,121,11,71,108,98,33,66,94,96,113,119,5,46,15,26,81,82,59,61,38,69,57,79,43,12,82,67,28,107,43,123,28,37,4,58,122,62,64,36,20,88,0,113}



Source Code (Software)

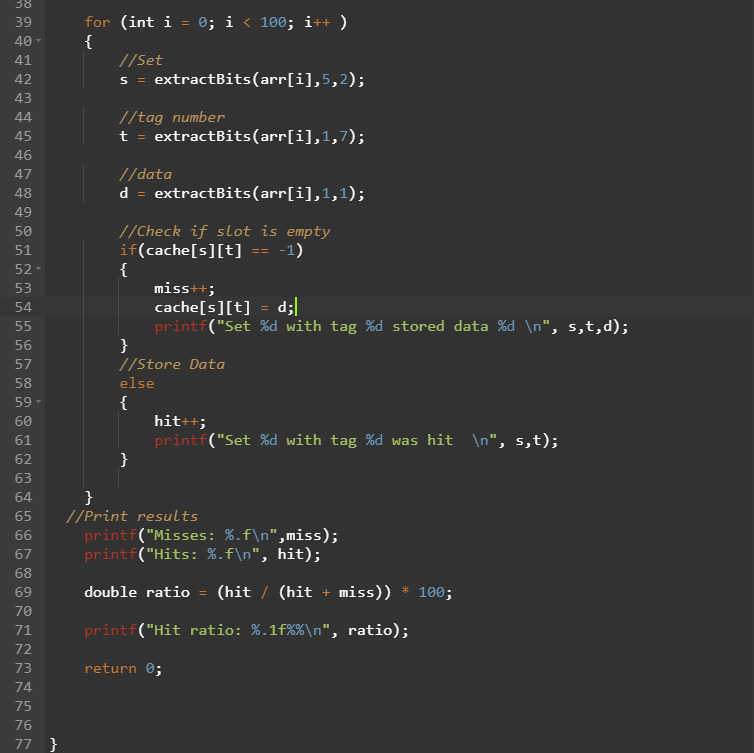
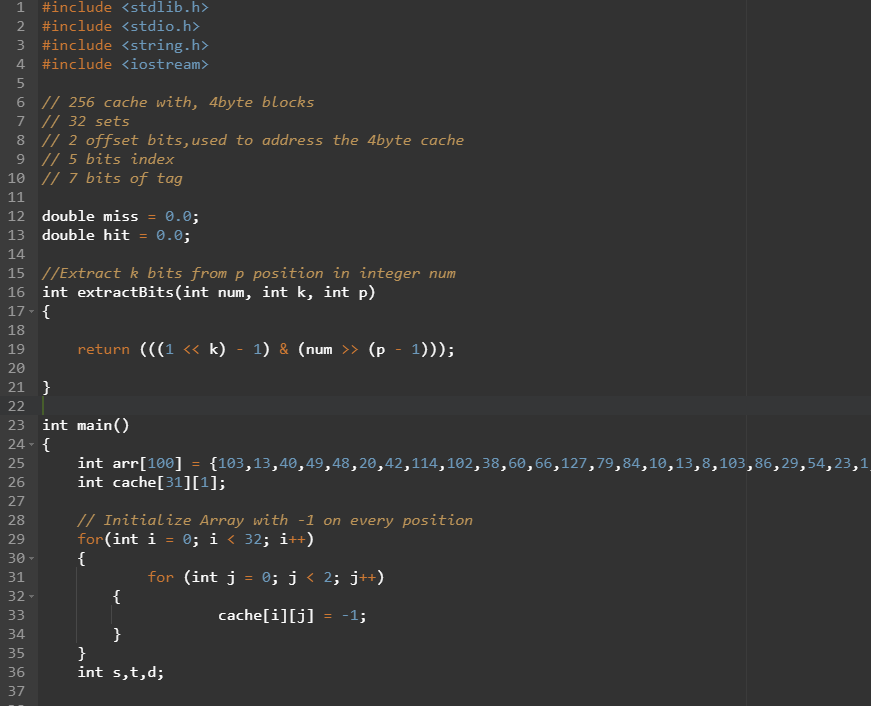
Direct Mapped:



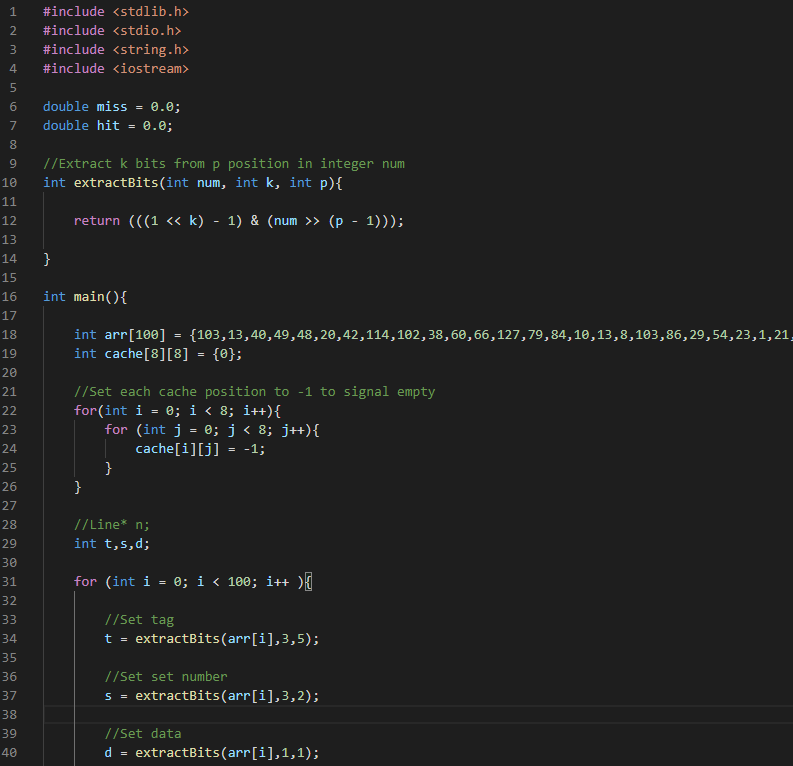


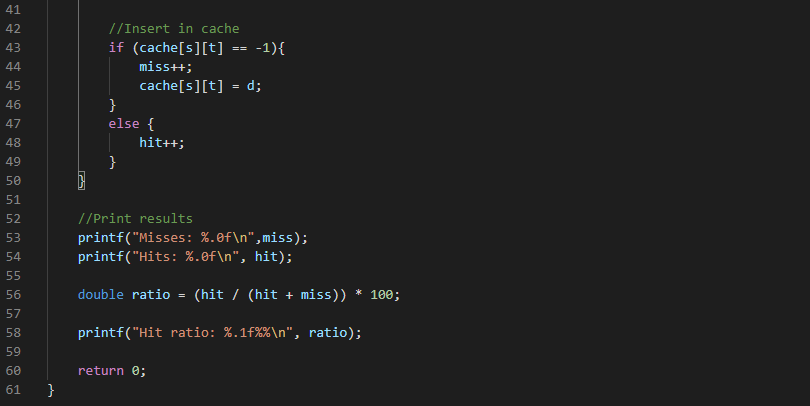
Fully Associative:

2 Way Set Associative



8 Way Set Associative:





Schematics (Hardware)

None.

Analysis

The fully associative cache mapping method was found to have the highest hit ratio. This makes sense since this method does not rely on sets. Each memory address can map to any location depending on the hash function. However, the fully associative method suffers from the longest search time since each position must be searched for each address. We chose not to test run time as we were not directly working with cache memory and instead working with arrays where each algorithm written does not exactly match how cache memory is mapped. Also, each method was written and run on a different system so run times would vary.

Conclusion

The main goal of this project was to examine the hit rates of 4 different types of cache mapping methods. Those methods are direct, fully associative, 2-way, and 8-way set associative cache mapping. The results are accurate with the expected results but hit rate is just one aspect of choosing a method for a processor. Modern day processors use a form of N-way set associative cache mapping as it is a compromise between hit rate and organization/search time. Our project showing an N-way set associative hit rate of around 50-70% shows that a hit rate of over 90% is not necessary for modern cache computing.

References

Demo Video: <https://youtu.be/F4imbIfnL_8>

<https://www.sciencedirect.com/topics/computer-science/set-associative-cache>