COSC 4310 Computer Architecture

Nicholas Wade

11/29/2021

Dr. Liu

Processor Comparison:

|  |  |  |  |
| --- | --- | --- | --- |
| CPU name | Intel Xeon Platinum 8358 | AMD EPYC 7443 | Intel Xeon E-2324G |
| Clock Rate | Max MHz: 3400  Nominal: 2600 | Max MHz: 4000  Nominal: 2850 | Max MHz: 4600  Nominal: 3100 |
| Enabled | 64 cores, 2 chips, 2 threads/core | 48 cores, 2 chips, 2 threads/core | 4 cores, 1 chip |
| # of cache levels | 3 | 3 | 3 |
| # of caches per each level | L1: 2 caches (I + D separate)  L2: 1 cache (I+D unified)  L3: 1 cache (I+D unified) | L1: 2 caches (I + D separate)  L2: 1 cache (I+D unified)  L3: 2 caches (I+D unified, shared/6 cores) | L1: 2 caches (I + D separate)  L2: 1 cache (I+D unified)  L3: 1 cache (I+D unified) |
| Cache size | L1: 32 KB I + 48 KB D  L2: 1.25 MB I+D  L3: 48 MB I+D | L1: 32 KB I + 32 KB D  L2: 512 KB I+D  L3: 128 MB I+D, 32 MB shared/6 cores | L1: 32 KB I + 48 KB D  L2: 512 KD I+D  L3: 8 MD I+D |
| Memory size | 1 TB (16 x 64 GB 2Px4 PC4-3200AA-R) | 2 TB (16 x 128 GB 4Rx4 PC4-3200V-L) | 64 GB (2 x 32 GB 2Rx8 PC4-3200AA-E) |
| Storage size | 1 x 4 TB PCIE NVME SSD | 1 x 960 GB M.2 SSD SATA | 70 GB on tmpfs |

Introduction of Cache Implementation:

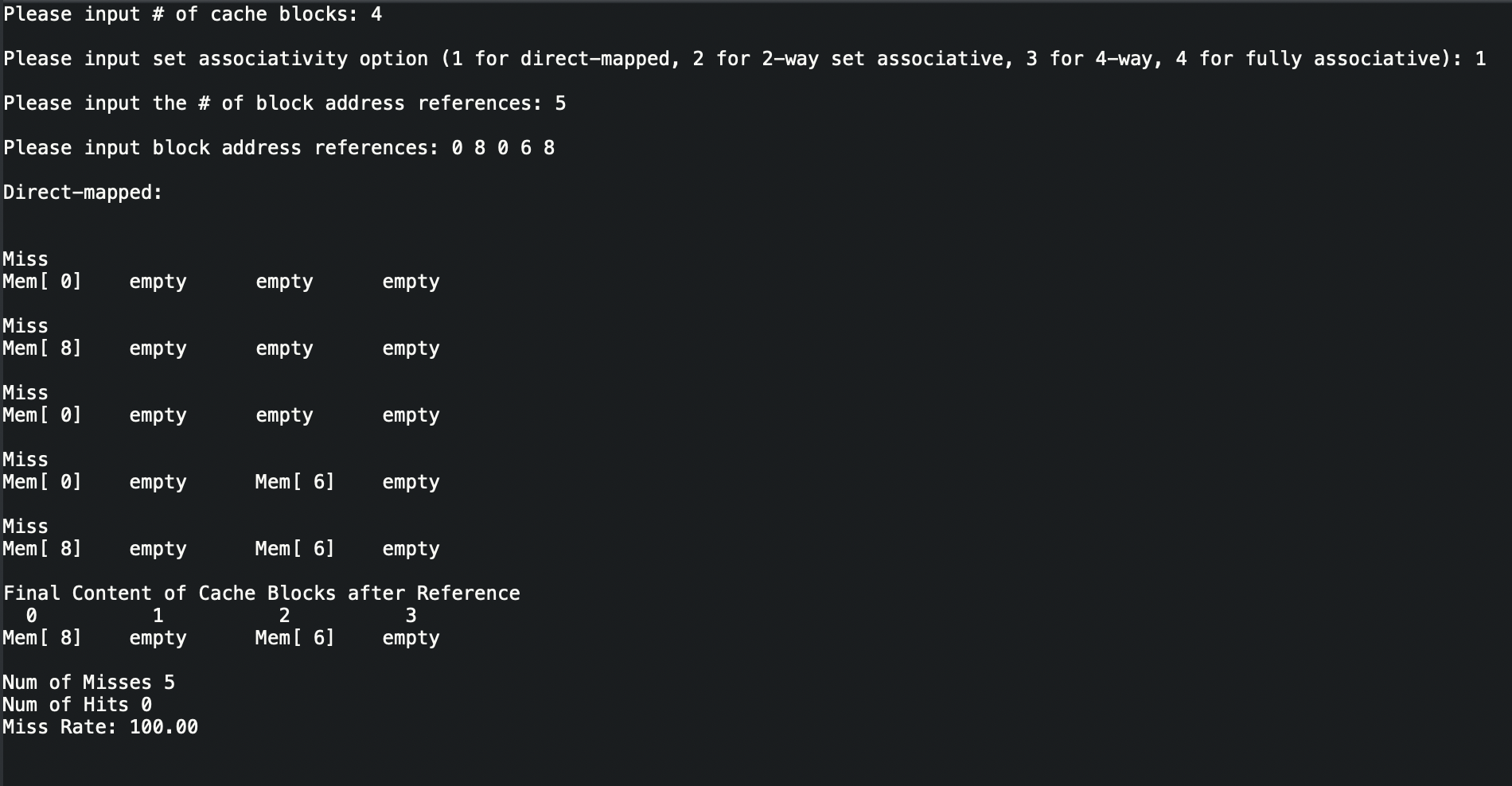
In order to implement my Simple Cache project, I made a program using java. It starts out by prompting the user the number of blocks in the cache, then was type of set associativity to use. After that it takes in the number of memory address references so that I could make arrays of appropriate size, then the user will input each memory address in base 10.

Once this information is taken in, the appropriate set associativity method will be called which is inside another class that is holding all of the set associative methods. Once the information is passed to the method, the method will process the data and give clear results of what the final content of the cache should be.

Testing Results:

When testing to make sure my output was correct, I used the cache and cache-solutions pdf as a constant. So, as you can see below, I have used the same input as those worksheets and each output is the same as on those worksheets, which is a good indicator that the methods are working correctly. Additionally, throughout designing this program I had tried multiple combinations of number of blocks, memory address values, etc. and it outputs correct cache content each time.

Direct-mapped:



2-Way Set Associative:

Text

Description automatically generated

4-Way Set Associative:

Text

Description automatically generated

Fully-Associative:

Text

Description automatically generated

Miss Rate Comparison Analysis:

|  |  |
| --- | --- |
| Set Associativity Option | Miss Rate |
| Direct-mapped | 100.00% |
| 2-Way Associative | 80.00% |
| 4-Way Associative | 100.00% |
| Fully-Associative | 60.00% |

As we can see from the results the direct-mapped associative option had a very high miss rate of 100% as well as 4-way associative because the cache has 4 blocks, thus it is basically the same as direct-mapped which makes sense that it would yield the same result. 2-way associative had the median rate of 80% which is slightly better than direct-mapped and 4-way associative. And the option that had the lowest miss rate was fully associative with a result of 60% miss rate which is a bit better than 2-way associative and much better than direct-mapped and 4-way associative.

Even though these are the results that I have yielded using these inputs, this is just one combination of inputs. In order to get a better idea of which methods have a better miss rate it would be better to test many different combinations for each method and comparing that data to previous results.