

COSC 4310

Lizbeth Trujillo

November 2022

Dr. Liu

## Simple Cache: Output

### Direct Mapped

```
15         int numCacheBlocks = scanner.nextInt();
16         //calculate bit offset
17         int offset = (int) (Math.log(numCacheBlocks) / Math.log(2));
18
Run: Cache x
/Library/Java/JavaVirtualMachines/amazon-corretto-11.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/
Name: Lizbeth Trujillo

Enter the number of cache blocks:
5
Select Replacement Policy:
1: LRU ( Least Recently Used )
1
Enter a sequence of addresses to use in the cache:
0 8 0 6 8
Choose cache Associativity:
1: Direct-Mapped 2: Set Associative 3: Fully Associative
1
Direct Mapped
[ MEM[0] ]
[ MEM[8] ]
[ MEM[0] ]
[ MEM[0] , MEM[6] ]
[ MEM[8] , MEM[6] ]

Hit/Miss ratio: 0/5

Run  TODO  Problems  Debug  Terminal  Build
All files are up-to-date (moments ago)
```

### 2-Way Associative:

```
53         }
54         break;
55     case 3:
Run: Cache x
/Library/Java/JavaVirtualMachines/amazon-corretto-11.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Con
Name: Lizbeth Trujillo

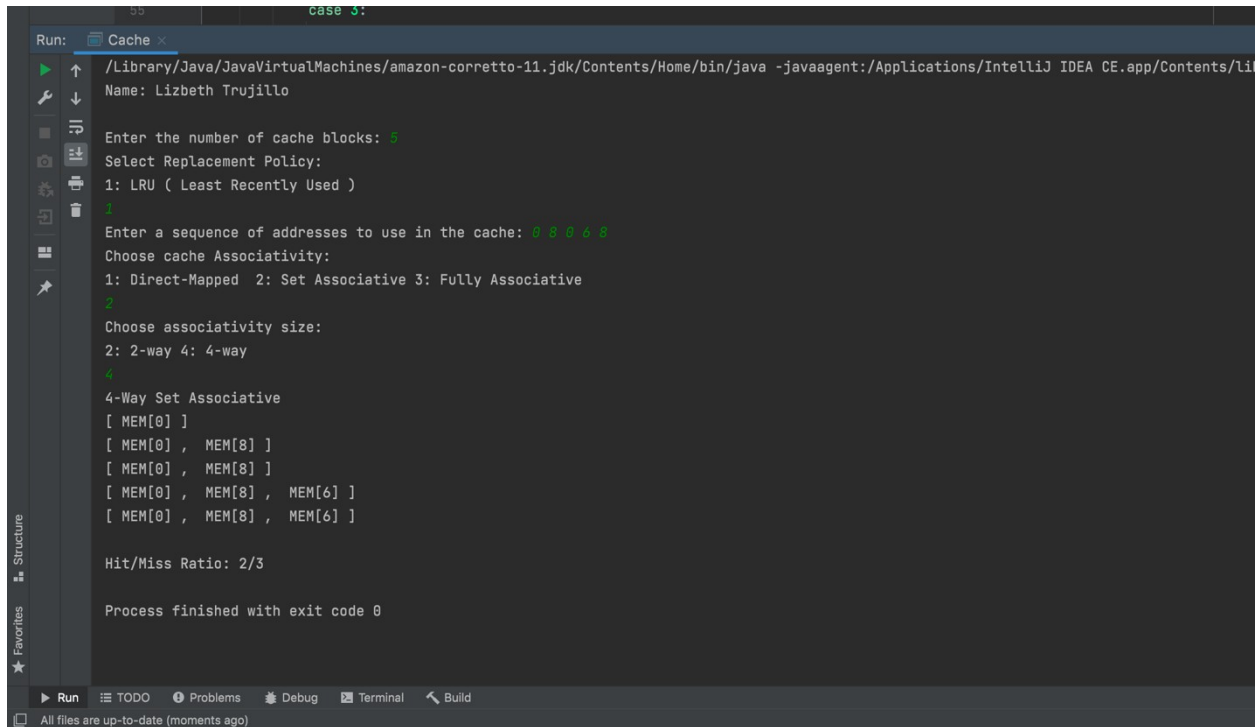
Enter the number of cache blocks: 5
Select Replacement Policy:
1: LRU ( Least Recently Used )
1
Enter a sequence of addresses to use in the cache: 0 8 0 6 8
Choose cache Associativity:
1: Direct-Mapped 2: Set Associative 3: Fully Associative
2
Choose associativity size:
2: 2-way 4: 4-way
2
2-Way Set Associative
[ MEM[0] ]
[ MEM[0] , MEM[8] ]
[ MEM[0] , MEM[8] ]
[ MEM[0] , MEM[6] ]
[ MEM[8] , MEM[6] ]

Hit/Miss Ratio: 1/4

Process finished with exit code 0
|

Run  TODO  Problems  Debug  Terminal  Build
Build completed successfully in 1 sec, 628 ms (moments ago)
```

## 4-Way Associative:



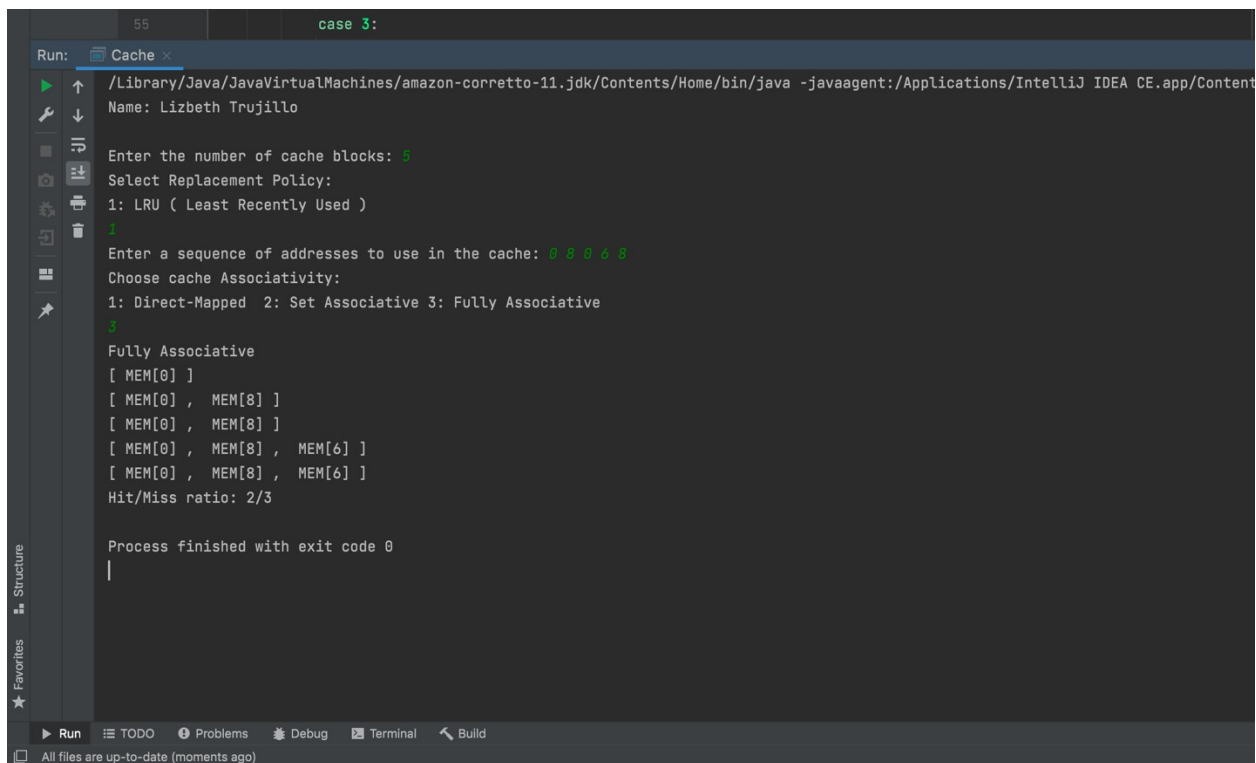
```
Run: Cache x
/Library/Java/JavaVirtualMachines/amazon-corretto-11.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/Li
Name: Lizbeth Trujillo

Enter the number of cache blocks: 5
Select Replacement Policy:
1: LRU ( Least Recently Used )
1
Enter a sequence of addresses to use in the cache: 8 8 8 6 8
Choose cache Associativity:
1: Direct-Mapped 2: Set Associative 3: Fully Associative
2
Choose associativity size:
2: 2-way 4: 4-way
4
4-Way Set Associative
[ MEM[0] ]
[ MEM[0] , MEM[8] ]
[ MEM[0] , MEM[8] ]
[ MEM[0] , MEM[8] , MEM[6] ]
[ MEM[0] , MEM[8] , MEM[6] ]

Hit/Miss Ratio: 2/3

Process finished with exit code 0
```

## Fully Associative:



```
Run: Cache x
/Library/Java/JavaVirtualMachines/amazon-corretto-11.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Content
Name: Lizbeth Trujillo

Enter the number of cache blocks: 5
Select Replacement Policy:
1: LRU ( Least Recently Used )
1
Enter a sequence of addresses to use in the cache: 8 8 8 6 8
Choose cache Associativity:
1: Direct-Mapped 2: Set Associative 3: Fully Associative
3
Fully Associative
[ MEM[0] ]
[ MEM[0] , MEM[8] ]
[ MEM[0] , MEM[8] ]
[ MEM[0] , MEM[8] , MEM[6] ]
[ MEM[0] , MEM[8] , MEM[6] ]
Hit/Miss ratio: 2/3

Process finished with exit code 0
```

### Miss Rate Comparison Analysis:

From the results of the different cache methods, it is clear that Direct Mapped Caches are most likely to return the most misses. Since Direct Map caches only allow one spot per unique n-bit offset value, there is a lot of replacement taking place. The 2-Way associative cache was slightly better since there is an additional storage spot per unique 1 bit offset. Since the example input was small, I couldn't really see a big difference between the fully associative and the 4-way set output. However, when tested with larger inputs, the 4-way cache had fewer misses in comparison to the full. This is because the fully associative set had a limited space of  $\text{offset}^2$ .