**Table of Contents for ease**

Contents

[Introduction 2](#_Toc120545335)

[Line Representation 2](#_Toc120545336)

[Cache Representation 2](#_Toc120545337)

[Summarized Simulation Flow 2](#_Toc120545338)

[In-Depth Simulation Flow 2](#_Toc120545339)

[Description of Tests 3](#_Toc120545340)

[Table of Test Parameters 5](#_Toc120545341)

[Results 9](#_Toc120545342)

[Comparing Cache Sizes 10](#_Toc120545343)

[Comparing Associativity 11](#_Toc120545344)

[Comparing Replacement Methods 14](#_Toc120545345)

[Conclusion 14](#_Toc120545346)

[Appendix 15](#_Toc120545347)

# Introduction

This cache simulator was written in C++ utilizing Visual Studio Code. There are two cpp files and one header file. The main in this program simply calls the Cache class which holds all relevant functions to run this simulation.

## Line Representation

The data structure used to simulate a line in memory is a struct entitled Line. Each Line has 3 variables: line, tag, and counter. The line variable represents the decimal value of the line or set number depending on whether the associativity is Direct-Mapped or N-Way Set Associative. The tag variable represents the decimal value of the tag. The counter variable keeps track of a value depending on the replacement method: LRU or FIFO. In this simulation, offset is ignored and thus not stored.

## Cache Representation

The data structure used to simulate the cache memory is entitled cacheMem. It is a vector of pointers to different Lines.

## Summarized Simulation Flow

Inside Cache is multiple functions that simulate a cache. The program will initialize and calculate parameters, read the file of addresses, turn each address into binary, search for that tag in the existing cache. If the tag if found, we increment total and hits (counter depending on replacement method). If tag is not found, we still increment total and decide which line will be replace by the new Line. This process is repeated for each address in the file. Hit rate is calculated and then results including parameters are printed to the console.

## In-Depth Simulation Flow

From main, a Cache function initParams is called.

void initParams(int lineSize, int cacheSize, int linesPerSet, bool fifoReplace);

[fifoReplace == 0 -> LRU replacement fifoReplace == 1 -> FIFO replacement]

In initParams, the respective class members are assigned to the passed-in parameters. The function also calculates the number of lines (numLines), number of sets (numSets), address widths (offsetWidth, linewidth, and tagWidth). LineWidth is only calculated if the cache is simulating Direct-Mapped or N-Way Set Associative; otherwise, it is 0. For N-Way Set Associative, lineWidth may be thought of as the set width.

Next, a private function, readFile is called from within initParams. readFile reads the file line by line. When the address is retrieved from each line, another function called hexToBinary is called.

hexToBinary: converts our hexadecimal address to binary.

After hexToBinary, we are back inside readFile. Our newly returned binary address is then chopped up by our previously calculated address widths. Another function called search is called.

search (tag, line) :

Our total counter is incremented.

If our cacheMem size is 0, we create a Line newLine with the passed in line and tag and total as variables line, tag, and counter respectively. Next, if the cache is fully associative, cacheMem pushes back newline; otherwise, cacheMem is created with a size of numLines and newline is added at the correct position—line\*linesPerSet.

If our cacheMem size is not 0, we must search for the tag utilizing line or set if given. First two variables are created. A boolean entitled found and an int entitled startIndex.

int startIndex = line \* linesPerSet;

Utilizing startIndex, we may traverse the vector efficiently with a for loop regardless of associativity. The for-loop traverses from 0 to linesPerSet.

Meaning if the simulation is:

fully-associative linesPerSet == numLines == size of vector

direct-mapped linesPerSet == 1

N-Way Set Associative linesPerSet == N

Then if cacheMem size is greater than startIndex+1, a line exists at startIndex+1, and the tags match, we increment hits and switch found to 1. If our replacement method is LRU, we update counter. So, if a hit occurs or cacheMem size is less than or equal to startIndex+1, we break from the for-loop.

If we do not find tag in the cache, the program calls replace.

replace (tag, line) :

if our cache is fully-associative and not full yet, we simply push back a newLine containing the proper tag, line, and counter variables.

If our cache is direct-mapped, we simply replace the existing line at the correct line (index) position with newLine.

Otherwise, we do a similar strategy used in search.

startIndex is declared the same. Another int replaceIndex is set to startIndex. And int smallestCounter is set to the counter at replaceIndex.

Inside a for loop from 0 to linesPerSet, we find the smallest counter and therefore, replaceIndex. Then once the smallest counter is found, we replace the existing Line at replaceIndex with newLine.

Back inside readFile, We continue this entire process for each address in the file.

Back inside initParams, we calculate hitRate.

Back inside main, where parameters and hit rate are printed to the console.

# Description of Tests

The function to begin a simulation is as follows:

void initParams(int lineSize, int cacheSize, int linesPerSet, bool fifoReplace);

Inside main, I have structured my tests in 3 loops. (cout statements and a testNum variable have been omitted)

Fully-Associative:

    for (int i = 4; i < 7; i++){//2^4 , 2^5, 2^7

        for(int j = 10; j < 15; j++){//2^10 , 2^11, 2^12, 2^13, 2^14

            cache.initParams(pow(2, i), pow(2, j), pow(2, j)/pow(2, i), 0);

            cache.initParams(pow(2, i), pow(2, j), pow(2, j)/pow(2, i), 1);

        }

    }

Direct-Mapped:

    for (int i = 4; i < 7; i++){//2^4 , 2^5, 2^7

        for(int j = 10; j < 15; j++){//2^10 , 2^11, 2^12, 2^13, 2^14

            cache.initParams(pow(2, i), pow(2, j), 1, 0);

            //replacement method doesnt matter for DM

        }

    }

N-Way Set Associative

for (int h = 1; h < 7; h++) {

        for (int i = 4; i < 7; i++) {//2^4 , 2^5, 2^7

            for(int j = 10; j < 15; j++){//2^10 , 2^11, 2^12, 2^13, 2^14

                if (pow(2, j) / pow(2, i) > pow(2, h)){

                    cache.initParams(pow(2, i), pow(2, j), pow(2, h), 0);

                    cache.initParams(pow(2, i), pow(2, j), pow(2, h), 1);

                }

            }

        }

    }

if (pow(2, j) / pow(2, i) > pow(2, h))

This if loop located inside the N-Way Set Associative test loops is extremely important. It is checking that the number of lines > lines per set. This is important because if the number of lines is <= lines per set, then the number of sets is <= 1.

These loops significantly reduced the amount of time of having to type test cases. Since values are a positive power of 2, loops were easy to implement.

The values chosen for line size were 16, 32, and 64.

The values chosen for cache size were 1024, 2048, 4096, 8192, and 16384.

The linesPerSet is decided by the type of associativity.

For fully-associative the linesPerSet is simply the number of lines (cache size / line size).

For Direct-Mapped, the linesPerSet is always 1.

For N-Way Set Associative, linesPerSet equals N.

The replacement variable is either 0 or 1 (except for Direct-Mapped where it does not matter), indicating LRU or FIFO respectively.

These parameters were chosen in order to provide a broad range of possible simulations including similar values seen in class. In total, there were 205 tests.

[**Click to skip to results**](#_Results)

## Table of Test Parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Number** | **lineSize** | **cacheSize** | **LinesPerSet** | **fifoReplace** |
| ====FA Examples==== |  |  |  |  |
| 0 | 16 | 1024 | 64 | 0 |
| 1 | 16 | 1024 | 64 | 1 |
| 2 | 16 | 2048 | 128 | 0 |
| 3 | 16 | 2048 | 128 | 1 |
| 4 | 16 | 4096 | 256 | 0 |
| 5 | 16 | 4096 | 256 | 1 |
| 6 | 16 | 8192 | 512 | 0 |
| 7 | 16 | 8192 | 512 | 1 |
| 8 | 16 | 16384 | 1024 | 0 |
| 9 | 16 | 16384 | 1024 | 1 |
| 10 | 32 | 1024 | 32 | 0 |
| 11 | 32 | 1024 | 32 | 1 |
| 12 | 32 | 2048 | 64 | 0 |
| 13 | 32 | 2048 | 64 | 1 |
| 14 | 32 | 4096 | 128 | 0 |
| 15 | 32 | 4096 | 128 | 1 |
| 16 | 32 | 8192 | 256 | 0 |
| 17 | 32 | 8192 | 256 | 1 |
| 18 | 32 | 16384 | 512 | 0 |
| 19 | 32 | 16384 | 512 | 1 |
| 20 | 64 | 1024 | 16 | 0 |
| 21 | 64 | 1024 | 16 | 1 |
| 22 | 64 | 2048 | 32 | 0 |
| 23 | 64 | 2048 | 32 | 1 |
| 24 | 64 | 4096 | 64 | 0 |
| 25 | 64 | 4096 | 64 | 1 |
| 26 | 64 | 8192 | 128 | 0 |
| 27 | 64 | 8192 | 128 | 1 |
| 28 | 64 | 16384 | 256 | 0 |
| 29 | 64 | 16384 | 256 | 1 |
| ====DM Examples==== |  |  |  |  |
| 30 | 16 | 1024 | 1 | 0 |
| 31 | 16 | 2048 | 1 | 0 |
| 32 | 16 | 4096 | 1 | 0 |
| 33 | 16 | 8192 | 1 | 0 |
| 34 | 16 | 16384 | 1 | 0 |
| 35 | 32 | 1024 | 1 | 0 |
| 36 | 32 | 2048 | 1 | 0 |
| 37 | 32 | 4096 | 1 | 0 |
| 38 | 32 | 8192 | 1 | 0 |
| 39 | 32 | 16384 | 1 | 0 |
| 40 | 64 | 1024 | 1 | 0 |
| 41 | 64 | 2048 | 1 | 0 |
| 42 | 64 | 4096 | 1 | 0 |
| 43 | 64 | 8192 | 1 | 0 |
| 44 | 64 | 16384 | 1 | 0 |
| ===N-Way SA Examples=== |  |  |  |  |
| 45 | 16 | 1024 | 2 | 0 |
| 46 | 16 | 1024 | 2 | 1 |
| 47 | 16 | 2048 | 2 | 0 |
| 48 | 16 | 2048 | 2 | 1 |
| 49 | 16 | 4096 | 2 | 0 |
| 50 | 16 | 4096 | 2 | 1 |
| 51 | 16 | 8192 | 2 | 0 |
| 52 | 16 | 8192 | 2 | 1 |
| 53 | 16 | 16384 | 2 | 0 |
| 54 | 16 | 16384 | 2 | 1 |
| 55 | 32 | 1024 | 2 | 0 |
| 56 | 32 | 1024 | 2 | 1 |
| 57 | 32 | 2048 | 2 | 0 |
| 58 | 32 | 2048 | 2 | 1 |
| 59 | 32 | 4096 | 2 | 0 |
| 60 | 32 | 4096 | 2 | 1 |
| 61 | 32 | 8192 | 2 | 0 |
| 62 | 32 | 8192 | 2 | 1 |
| 63 | 32 | 16384 | 2 | 0 |
| 64 | 32 | 16384 | 2 | 1 |
| 65 | 64 | 1024 | 2 | 0 |
| 66 | 64 | 1024 | 2 | 1 |
| 67 | 64 | 2048 | 2 | 0 |
| 68 | 64 | 2048 | 2 | 1 |
| 69 | 64 | 4096 | 2 | 0 |
| 70 | 64 | 4096 | 2 | 1 |
| 71 | 64 | 8192 | 2 | 0 |
| 72 | 64 | 8192 | 2 | 1 |
| 73 | 64 | 16384 | 2 | 0 |
| 74 | 64 | 16384 | 2 | 1 |
| 75 | 16 | 1024 | 4 | 0 |
| 76 | 16 | 1024 | 4 | 1 |
| 77 | 16 | 2048 | 4 | 0 |
| 78 | 16 | 2048 | 4 | 1 |
| 79 | 16 | 4096 | 4 | 0 |
| 80 | 16 | 4096 | 4 | 1 |
| 81 | 16 | 8192 | 4 | 0 |
| 82 | 16 | 8192 | 4 | 1 |
| 83 | 16 | 16384 | 4 | 0 |
| 84 | 16 | 16384 | 4 | 1 |
| 85 | 32 | 1024 | 4 | 0 |
| 86 | 32 | 1024 | 4 | 1 |
| 87 | 32 | 2048 | 4 | 0 |
| 88 | 32 | 2048 | 4 | 1 |
| 89 | 32 | 4096 | 4 | 0 |
| 90 | 32 | 4096 | 4 | 1 |
| 91 | 32 | 8192 | 4 | 0 |
| 92 | 32 | 8192 | 4 | 1 |
| 93 | 32 | 16384 | 4 | 0 |
| 94 | 32 | 16384 | 4 | 1 |
| 95 | 64 | 1024 | 4 | 0 |
| 96 | 64 | 1024 | 4 | 1 |
| 97 | 64 | 2048 | 4 | 0 |
| 98 | 64 | 2048 | 4 | 1 |
| 99 | 64 | 4096 | 4 | 0 |
| 100 | 64 | 4096 | 4 | 1 |
| 101 | 64 | 8192 | 4 | 0 |
| 102 | 64 | 8192 | 4 | 1 |
| 103 | 64 | 16384 | 4 | 0 |
| 104 | 64 | 16384 | 4 | 1 |
| 105 | 16 | 1024 | 8 | 0 |
| 106 | 16 | 1024 | 8 | 1 |
| 107 | 16 | 2048 | 8 | 0 |
| 108 | 16 | 2048 | 8 | 1 |
| 109 | 16 | 4096 | 8 | 0 |
| 110 | 16 | 4096 | 8 | 1 |
| 111 | 16 | 8192 | 8 | 0 |
| 112 | 16 | 8192 | 8 | 1 |
| 113 | 16 | 16384 | 8 | 0 |
| 114 | 16 | 16384 | 8 | 1 |
| 115 | 32 | 1024 | 8 | 0 |
| 116 | 32 | 1024 | 8 | 1 |
| 117 | 32 | 2048 | 8 | 0 |
| 118 | 32 | 2048 | 8 | 1 |
| 119 | 32 | 4096 | 8 | 0 |
| 120 | 32 | 4096 | 8 | 1 |
| 121 | 32 | 8192 | 8 | 0 |
| 122 | 32 | 8192 | 8 | 1 |
| 123 | 32 | 16384 | 8 | 0 |
| 124 | 32 | 16384 | 8 | 1 |
| 125 | 64 | 1024 | 8 | 0 |
| 126 | 64 | 1024 | 8 | 1 |
| 127 | 64 | 2048 | 8 | 0 |
| 128 | 64 | 2048 | 8 | 1 |
| 129 | 64 | 4096 | 8 | 0 |
| 130 | 64 | 4096 | 8 | 1 |
| 131 | 64 | 8192 | 8 | 0 |
| 132 | 64 | 8192 | 8 | 1 |
| 133 | 64 | 16384 | 8 | 0 |
| 134 | 64 | 16384 | 8 | 1 |
| 135 | 16 | 1024 | 16 | 0 |
| 136 | 16 | 1024 | 16 | 1 |
| 137 | 16 | 2048 | 16 | 0 |
| 138 | 16 | 2048 | 16 | 1 |
| 139 | 16 | 4096 | 16 | 0 |
| 140 | 16 | 4096 | 16 | 1 |
| 141 | 16 | 8192 | 16 | 0 |
| 142 | 16 | 8192 | 16 | 1 |
| 143 | 16 | 16384 | 16 | 0 |
| 144 | 16 | 16384 | 16 | 1 |
| 145 | 32 | 1024 | 16 | 0 |
| 146 | 32 | 1024 | 16 | 1 |
| 147 | 32 | 2048 | 16 | 0 |
| 148 | 32 | 2048 | 16 | 1 |
| 149 | 32 | 4096 | 16 | 0 |
| 150 | 32 | 4096 | 16 | 1 |
| 151 | 32 | 8192 | 16 | 0 |
| 152 | 32 | 8192 | 16 | 1 |
| 153 | 32 | 16384 | 16 | 0 |
| 154 | 32 | 16384 | 16 | 1 |
| 155 | 64 | 2048 | 16 | 0 |
| 156 | 64 | 2048 | 16 | 1 |
| 157 | 64 | 4096 | 16 | 0 |
| 158 | 64 | 4096 | 16 | 1 |
| 159 | 64 | 8192 | 16 | 0 |
| 160 | 64 | 8192 | 16 | 1 |
| 161 | 64 | 16384 | 16 | 0 |
| 162 | 64 | 16384 | 16 | 1 |
| 163 | 16 | 1024 | 32 | 0 |
| 164 | 16 | 1024 | 32 | 1 |
| 165 | 16 | 2048 | 32 | 0 |
| 166 | 16 | 2048 | 32 | 1 |
| 167 | 16 | 4096 | 32 | 0 |
| 168 | 16 | 4096 | 32 | 1 |
| 169 | 16 | 8192 | 32 | 0 |
| 170 | 16 | 8192 | 32 | 1 |
| 171 | 16 | 16384 | 32 | 0 |
| 172 | 16 | 16384 | 32 | 1 |
| 173 | 32 | 2048 | 32 | 0 |
| 174 | 32 | 2048 | 32 | 1 |
| 175 | 32 | 4096 | 32 | 0 |
| 176 | 32 | 4096 | 32 | 1 |
| 177 | 32 | 8192 | 32 | 0 |
| 178 | 32 | 8192 | 32 | 1 |
| 179 | 32 | 16384 | 32 | 0 |
| 180 | 32 | 16384 | 32 | 1 |
| 181 | 64 | 4096 | 32 | 0 |
| 182 | 64 | 4096 | 32 | 1 |
| 183 | 64 | 8192 | 32 | 0 |
| 184 | 64 | 8192 | 32 | 1 |
| 185 | 64 | 16384 | 32 | 0 |
| 186 | 64 | 16384 | 32 | 1 |
| 187 | 16 | 2048 | 64 | 0 |
| 188 | 16 | 2048 | 64 | 1 |
| 189 | 16 | 4096 | 64 | 0 |
| 190 | 16 | 4096 | 64 | 1 |
| 191 | 16 | 8192 | 64 | 0 |
| 192 | 16 | 8192 | 64 | 1 |
| 193 | 16 | 16384 | 64 | 0 |
| 194 | 16 | 16384 | 64 | 1 |
| 195 | 32 | 4096 | 64 | 0 |
| 196 | 32 | 4096 | 64 | 1 |
| 197 | 32 | 8192 | 64 | 0 |
| 198 | 32 | 8192 | 64 | 1 |
| 199 | 32 | 16384 | 64 | 0 |
| 200 | 32 | 16384 | 64 | 1 |
| 201 | 64 | 8192 | 64 | 0 |
| 202 | 64 | 8192 | 64 | 1 |
| 203 | 64 | 16384 | 64 | 0 |
| 204 | 64 | 16384 | 64 | 1 |

[**Click to skip above Table of Test Parameters**](#_Table_of_Test)

# Results

Below will be a few graphs including the respective table data. The tables will be color coded by dark red (worst hit-rate) to dark green (best hit-rate).

[**Click to skip to conclusion**](#_Conclusion)

The following three sections of the results:

[Comparing Cache Sizes](#_Comparing_Cache_Sizes)

[Comparing Associativity](#_Comparing_Associativity)

[Comparing Replacement Methods](#_Comparing_Replacement_Methods)

## Comparing Cache Sizes

Respective table data: color coded by entire table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1024 | 2048 | 4096 | 8192 | 16384 |
| Fully-Associative (FIFO) | 0.921935333 | 0.956631667 | 0.973475333 | 0.980101 | 0.983069 |
| Fully Associative (LRU) | 0.934507333 | 0.964133667 | 0.977635667 | 0.9822277 | 0.984143 |
| Direct-Mapped | 0.840462667 | 0.892939333 | 0.945053333 | 0.962547 | 0.976561333 |
| 2-Way (FIFO) | 0.895324667 | 0.923067 | 0.961920667 | 0.9766297 | 0.981693667 |
| 4-Way (FIFO) | 0.917450333 | 0.948759333 | 0.967436333 | 0.9791927 | 0.982590667 |
| 8-Way (FIFO) | 0.920465333 | 0.955666333 | 0.972399667 | 0.9796457 | 0.982887 |
| 16-Way (FIFO) | 0.9260645 | 0.955788 | 0.972813 | 0.9798533 | 0.982976333 |
| 32-Way (FIFO) | 0.924713 | 0.957307 | 0.973293667 | 0.979998 | 0.982976333 |
| 64-Way (FIFO) |  | 0.954941 | 0.9710115 | 0.980099 | 0.983013667 |
| 2-Way (LRU) | 0.901941 | 0.929914333 | 0.965809 | 0.9780687 | 0.982149333 |
| 4-Way (LRU) | 0.927545667 | 0.955990667 | 0.972321 | 0.9805953 | 0.982407333 |
| 8-Way (LRU) | 0.932213333 | 0.963182333 | 0.976432 | 0.980609 | 0.981763333 |
| 16-Way (LRU) | 0.937674 | 0.963805333 | 0.977004667 | 0.9802733 | 0.981149333 |
| 32-Way (LRU) | 0.937977 | 0.9646985 | 0.977127333 | 0.9800317 | 0.980589667 |
| 64-Way (LRU) |  | 0.961946 | 0.9742575 | 0.9798397 | 0.980208333 |

This table data is color coded so that the lowest values are dark red, and the higher values are dark green. Values in the middle are lighter reds and greens along with white. This color analysis was applied to the entire table in order to reveal quickly the trends among the different cache sizes.

As depicted by the three images above, we can see that as cache size increases, so does hit rate. We can also see the direct mapped always performs the worst. Fully associative with LRU replacement always performs the best and it sometimes followed closesly by fully-associative with FIFO. The comparisons of other parameters will be expanded upon further.

## Comparing Associativity

Respective table data

|  |  |
| --- | --- |
| Direct-Mapped | 0.923512733 |
| 2-way | 0.9496518 |
| 4-way | 0.961428933 |
| 8-way | 0.9645264 |
| Fully-Associative | 0.965785967 |
| 16-way | 0.968159536 |
| 32-way | 0.972031292 |
| 64-way | 0.976494833 |

The two images above are the averaged hit rates based on associativity. Because it is averaged, it appears that 64-way performs best overall; however, this is because several 64-way performers were the omitted tests (see [description of tests](#_Description_of_Tests)) . Shown below will be the results grouped by line size and graphed according to cache size.

A picture containing chart

Description automatically generated

These tables were color coded by column rather than the entire table. This is to show the best and worst hit rate by associativity inside a certain cache size.

Just glancing briefly at the color coded tables, informs us that fully-associative (LRU) is the most successful type of associativity. It also informs us that direct-mapped performs the worst.

## Comparing Replacement Methods

Respective table data , color coded by column.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1024 | 2048 | 4096 | 8192 | 16384 |
| FIFO | 0.917658861 | 0.950308619 | 0.970335738 | 0.979359905 | 0.98274381 |
| LRU | 0.928643056 | 0.957667262 | 0.974369595 | 0.980235048 | 0.981772905 |

The two images above depict the averages of replacement methods. It is clear that LRU is the best method on average. Although not pictured, it was shown in the results that as line size increased, the effectiveness of LRU advanced past FIFO for larger cache sizes. For 16 sized lines, FIFO performed best at 8192 and 16384 cache size. When line size was increased to 32 and 64 bytes, LRU surpassed FIFO at cache size of 8192. The difference between LRU and FIFO at 16384 decreased when stepping from 32 byte to 64 byte line sizes. This indicates that as cache and line size increase, LRU becomes increasingly better than FIFO.

# Conclusion

As shown in the results, the larger the cache size, the better the hit rate. This is due to the simple fact that the cache can hold more lines. More lines equate to more chances of an address already being present without having to replace an address more frequently if cache size was smaller.

In each graph, it is evident that Direct-Mapped, on average, has worse hit rates. This is due to the fact that addresses with the same line number can only be mapped to one specific spot in the cache. This leads to often replacements. It is also clear that fully associative (specifically LRU) achieves the highest hit rates.

It is shown that typically, LRU replacement method works best. Although LRU is not always perfect, it is a good guess as to which address is less essential based on previous access rates. It seems as though LRU becomes the best method as cache size is increased. This makes sense because with smaller caches, different replacement methods will be less effective due to the small size of space. As cache sizes grow, it is more obvious what different replacement methods will do.

# Appendix

Table of every result

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test** | **Line Size** | **Cache Size** | **Associativity** | **Replace** | **Hit Rate** |
| 0 | 16 | 1024 | Fully-Associative | LRU | 0.937882 |
| 1 | 16 | 1024 | Fully-Associative | FIFO | 0.925344 |
| 2 | 16 | 2048 | Fully-Associative | LRU | 0.9622 |
| 3 | 16 | 2048 | Fully-Associative | FIFO | 0.954965 |
| 4 | 16 | 4096 | Fully-Associative | LRU | 0.970234 |
| 5 | 16 | 4096 | Fully-Associative | FIFO | 0.965713 |
| 6 | 16 | 8192 | Fully-Associative | LRU | 0.973662 |
| 7 | 16 | 8192 | Fully-Associative | FIFO | 0.971479 |
| 8 | 16 | 16384 | Fully-Associative | LRU | 0.975648 |
| 9 | 16 | 16384 | Fully-Associative | FIFO | 0.974296 |
| 10 | 32 | 1024 | Fully-Associative | LRU | 0.93828 |
| 11 | 32 | 1024 | Fully-Associative | FIFO | 0.926218 |
| 12 | 32 | 2048 | Fully-Associative | LRU | 0.967067 |
| 13 | 32 | 2048 | Fully-Associative | FIFO | 0.95938 |
| 14 | 32 | 4096 | Fully-Associative | LRU | 0.980065 |
| 15 | 32 | 4096 | Fully-Associative | FIFO | 0.976451 |
| 16 | 32 | 8192 | Fully-Associative | LRU | 0.983884 |
| 17 | 32 | 8192 | Fully-Associative | FIFO | 0.981915 |
| 18 | 32 | 16384 | Fully-Associative | LRU | 0.985883 |
| 19 | 32 | 16384 | Fully-Associative | FIFO | 0.984826 |
| 20 | 64 | 1024 | Fully-Associative | LRU | 0.92736 |
| 21 | 64 | 1024 | Fully-Associative | FIFO | 0.914244 |
| 22 | 64 | 2048 | Fully-Associative | LRU | 0.963134 |
| 23 | 64 | 2048 | Fully-Associative | FIFO | 0.95555 |
| 24 | 64 | 4096 | Fully-Associative | LRU | 0.982608 |
| 25 | 64 | 4096 | Fully-Associative | FIFO | 0.978262 |
| 26 | 64 | 8192 | Fully-Associative | LRU | 0.989137 |
| 27 | 64 | 8192 | Fully-Associative | FIFO | 0.986909 |
| 28 | 64 | 16384 | Fully-Associative | LRU | 0.990898 |
| 29 | 64 | 16384 | Fully-Associative | FIFO | 0.990085 |
| 30 | 16 | 1024 | Direct-Mapped | / | 0.841527 |
| 31 | 16 | 2048 | Direct-Mapped | / | 0.890857 |
| 32 | 16 | 4096 | Direct-Mapped | / | 0.938687 |
| 33 | 16 | 8192 | Direct-Mapped | / | 0.958387 |
| 34 | 16 | 16384 | Direct-Mapped | / | 0.969084 |
| 35 | 32 | 1024 | Direct-Mapped | / | 0.848234 |
| 36 | 32 | 2048 | Direct-Mapped | / | 0.898393 |
| 37 | 32 | 4096 | Direct-Mapped | / | 0.948445 |
| 38 | 32 | 8192 | Direct-Mapped | / | 0.965519 |
| 39 | 32 | 16384 | Direct-Mapped | / | 0.978677 |
| 40 | 64 | 1024 | Direct-Mapped | / | 0.831627 |
| 41 | 64 | 2048 | Direct-Mapped | / | 0.889568 |
| 42 | 64 | 4096 | Direct-Mapped | / | 0.948028 |
| 43 | 64 | 8192 | Direct-Mapped | / | 0.963735 |
| 44 | 64 | 16384 | Direct-Mapped | / | 0.981923 |
| 45 | 16 | 1024 | 2-way | LRU | 0.90909 |
| 46 | 16 | 1024 | 2-way | FIFO | 0.902479 |
| 47 | 16 | 2048 | 2-way | LRU | 0.935259 |
| 48 | 16 | 2048 | 2-way | FIFO | 0.928861 |
| 49 | 16 | 4096 | 2-way | LRU | 0.961728 |
| 50 | 16 | 4096 | 2-way | FIFO | 0.957804 |
| 51 | 16 | 8192 | 2-way | LRU | 0.970542 |
| 52 | 16 | 8192 | 2-way | FIFO | 0.969163 |
| 53 | 16 | 16384 | 2-way | LRU | 0.973329 |
| 54 | 16 | 16384 | 2-way | FIFO | 0.97323 |
| 55 | 32 | 1024 | 2-way | LRU | 0.905395 |
| 56 | 32 | 1024 | 2-way | FIFO | 0.898707 |
| 57 | 32 | 2048 | 2-way | LRU | 0.932567 |
| 58 | 32 | 2048 | 2-way | FIFO | 0.92541 |
| 59 | 32 | 4096 | 2-way | LRU | 0.967825 |
| 60 | 32 | 4096 | 2-way | FIFO | 0.964234 |
| 61 | 32 | 8192 | 2-way | LRU | 0.9798 |
| 62 | 32 | 8192 | 2-way | FIFO | 0.978431 |
| 63 | 32 | 16384 | 2-way | LRU | 0.984087 |
| 64 | 32 | 16384 | 2-way | FIFO | 0.983585 |
| 65 | 64 | 1024 | 2-way | LRU | 0.891338 |
| 66 | 64 | 1024 | 2-way | FIFO | 0.884788 |
| 67 | 64 | 2048 | 2-way | LRU | 0.921917 |
| 68 | 64 | 2048 | 2-way | FIFO | 0.91493 |
| 69 | 64 | 4096 | 2-way | LRU | 0.967874 |
| 70 | 64 | 4096 | 2-way | FIFO | 0.963724 |
| 71 | 64 | 8192 | 2-way | LRU | 0.983864 |
| 72 | 64 | 8192 | 2-way | FIFO | 0.982295 |
| 73 | 64 | 16384 | 2-way | LRU | 0.989032 |
| 74 | 64 | 16384 | 2-way | FIFO | 0.988266 |
| 75 | 16 | 1024 | 4-way | LRU | 0.932887 |
| 76 | 16 | 1024 | 4-way | FIFO | 0.922906 |
| 77 | 16 | 2048 | 4-way | LRU | 0.958085 |
| 78 | 16 | 2048 | 4-way | FIFO | 0.950931 |
| 79 | 16 | 4096 | 4-way | LRU | 0.968027 |
| 80 | 16 | 4096 | 4-way | FIFO | 0.963862 |
| 81 | 16 | 8192 | 4-way | LRU | 0.971471 |
| 82 | 16 | 8192 | 4-way | FIFO | 0.970854 |
| 83 | 16 | 16384 | 4-way | LRU | 0.972877 |
| 84 | 16 | 16384 | 4-way | FIFO | 0.973889 |
| 85 | 32 | 1024 | 4-way | LRU | 0.929371 |
| 86 | 32 | 1024 | 4-way | FIFO | 0.919621 |
| 87 | 32 | 2048 | 4-way | LRU | 0.957385 |
| 88 | 32 | 2048 | 4-way | FIFO | 0.950299 |
| 89 | 32 | 4096 | 4-way | LRU | 0.975807 |
| 90 | 32 | 4096 | 4-way | FIFO | 0.971756 |
| 91 | 32 | 8192 | 4-way | LRU | 0.982668 |
| 92 | 32 | 8192 | 4-way | FIFO | 0.981132 |
| 93 | 32 | 16384 | 4-way | LRU | 0.984219 |
| 94 | 32 | 16384 | 4-way | FIFO | 0.984349 |
| 95 | 64 | 1024 | 4-way | LRU | 0.920379 |
| 96 | 64 | 1024 | 4-way | FIFO | 0.909824 |
| 97 | 64 | 2048 | 4-way | LRU | 0.952502 |
| 98 | 64 | 2048 | 4-way | FIFO | 0.945048 |
| 99 | 64 | 4096 | 4-way | LRU | 0.973129 |
| 100 | 64 | 4096 | 4-way | FIFO | 0.966691 |
| 101 | 64 | 8192 | 4-way | LRU | 0.987647 |
| 102 | 64 | 8192 | 4-way | FIFO | 0.985592 |
| 103 | 64 | 16384 | 4-way | LRU | 0.990126 |
| 104 | 64 | 16384 | 4-way | FIFO | 0.989534 |
| 105 | 16 | 1024 | 8-way | LRU | 0.935505 |
| 106 | 16 | 1024 | 8-way | FIFO | 0.924068 |
| 107 | 16 | 2048 | 8-way | LRU | 0.961426 |
| 108 | 16 | 2048 | 8-way | FIFO | 0.953481 |
| 109 | 16 | 4096 | 8-way | LRU | 0.969068 |
| 110 | 16 | 4096 | 8-way | FIFO | 0.965213 |
| 111 | 16 | 8192 | 8-way | LRU | 0.970837 |
| 112 | 16 | 8192 | 8-way | FIFO | 0.971223 |
| 113 | 16 | 16384 | 8-way | LRU | 0.971787 |
| 114 | 16 | 16384 | 8-way | FIFO | 0.974124 |
| 115 | 32 | 1024 | 8-way | LRU | 0.936678 |
| 116 | 32 | 1024 | 8-way | FIFO | 0.926377 |
| 117 | 32 | 2048 | 8-way | LRU | 0.965582 |
| 118 | 32 | 2048 | 8-way | FIFO | 0.958575 |
| 119 | 32 | 4096 | 8-way | LRU | 0.978995 |
| 120 | 32 | 4096 | 8-way | FIFO | 0.975091 |
| 121 | 32 | 8192 | 8-way | LRU | 0.982695 |
| 122 | 32 | 8192 | 8-way | FIFO | 0.981558 |
| 123 | 32 | 16384 | 8-way | LRU | 0.983593 |
| 124 | 32 | 16384 | 8-way | FIFO | 0.984642 |
| 125 | 64 | 1024 | 8-way | LRU | 0.924457 |
| 126 | 64 | 1024 | 8-way | FIFO | 0.910951 |
| 127 | 64 | 2048 | 8-way | LRU | 0.962539 |
| 128 | 64 | 2048 | 8-way | FIFO | 0.954943 |
| 129 | 64 | 4096 | 8-way | LRU | 0.981233 |
| 130 | 64 | 4096 | 8-way | FIFO | 0.976895 |
| 131 | 64 | 8192 | 8-way | LRU | 0.988295 |
| 132 | 64 | 8192 | 8-way | FIFO | 0.986156 |
| 133 | 64 | 16384 | 8-way | LRU | 0.98991 |
| 134 | 64 | 16384 | 8-way | FIFO | 0.989895 |
| 135 | 16 | 1024 | 16-way | LRU | 0.937504 |
| 136 | 16 | 1024 | 16-way | FIFO | 0.925028 |
| 137 | 16 | 2048 | 16-way | LRU | 0.962167 |
| 138 | 16 | 2048 | 16-way | FIFO | 0.953813 |
| 139 | 16 | 4096 | 16-way | LRU | 0.969119 |
| 140 | 16 | 4096 | 16-way | FIFO | 0.965456 |
| 141 | 16 | 8192 | 16-way | LRU | 0.970088 |
| 142 | 16 | 8192 | 16-way | FIFO | 0.971356 |
| 143 | 16 | 16384 | 16-way | LRU | 0.970866 |
| 144 | 16 | 16384 | 16-way | FIFO | 0.97416 |
| 145 | 32 | 1024 | 16-way | LRU | 0.937844 |
| 146 | 32 | 1024 | 16-way | FIFO | 0.927101 |
| 147 | 32 | 2048 | 16-way | LRU | 0.966582 |
| 148 | 32 | 2048 | 16-way | FIFO | 0.958889 |
| 149 | 32 | 4096 | 16-way | LRU | 0.979732 |
| 150 | 32 | 4096 | 16-way | FIFO | 0.975648 |
| 151 | 32 | 8192 | 16-way | LRU | 0.982431 |
| 152 | 32 | 8192 | 16-way | FIFO | 0.981683 |
| 153 | 32 | 16384 | 16-way | LRU | 0.983048 |
| 154 | 32 | 16384 | 16-way | FIFO | 0.984725 |
| 155 | 64 | 2048 | 16-way | LRU | 0.962667 |
| 156 | 64 | 2048 | 16-way | FIFO | 0.954662 |
| 157 | 64 | 4096 | 16-way | LRU | 0.982163 |
| 158 | 64 | 4096 | 16-way | FIFO | 0.977335 |
| 159 | 64 | 8192 | 16-way | LRU | 0.988301 |
| 160 | 64 | 8192 | 16-way | FIFO | 0.986521 |
| 161 | 64 | 16384 | 16-way | LRU | 0.989534 |
| 162 | 64 | 16384 | 16-way | FIFO | 0.990044 |
| 163 | 16 | 1024 | 32-way | LRU | 0.937977 |
| 164 | 16 | 1024 | 32-way | FIFO | 0.924713 |
| 165 | 16 | 2048 | 32-way | LRU | 0.961814 |
| 166 | 16 | 2048 | 32-way | FIFO | 0.954348 |
| 167 | 16 | 4096 | 32-way | LRU | 0.968837 |
| 168 | 16 | 4096 | 32-way | FIFO | 0.965698 |
| 169 | 16 | 8192 | 32-way | LRU | 0.969462 |
| 170 | 16 | 8192 | 32-way | FIFO | 0.971432 |
| 171 | 16 | 16384 | 32-way | LRU | 0.9701 |
| 172 | 16 | 16384 | 32-way | FIFO | 0.974191 |
| 173 | 32 | 2048 | 32-way | LRU | 0.967583 |
| 174 | 32 | 2048 | 32-way | FIFO | 0.960266 |
| 175 | 32 | 4096 | 32-way | LRU | 0.980129 |
| 176 | 32 | 4096 | 32-way | FIFO | 0.976131 |
| 177 | 32 | 8192 | 32-way | LRU | 0.982125 |
| 178 | 32 | 8192 | 32-way | FIFO | 0.981845 |
| 179 | 32 | 16384 | 32-way | LRU | 0.982532 |
| 180 | 32 | 16384 | 32-way | FIFO | 0.984692 |
| 181 | 64 | 4096 | 32-way | LRU | 0.982416 |
| 182 | 64 | 4096 | 32-way | FIFO | 0.978052 |
| 183 | 64 | 8192 | 32-way | LRU | 0.988508 |
| 184 | 64 | 8192 | 32-way | FIFO | 0.986717 |
| 185 | 64 | 16384 | 32-way | LRU | 0.989137 |
| 186 | 64 | 16384 | 32-way | FIFO | 0.990046 |
| 187 | 16 | 2048 | 64-way | LRU | 0.961946 |
| 188 | 16 | 2048 | 64-way | FIFO | 0.954941 |
| 189 | 16 | 4096 | 64-way | LRU | 0.96856 |
| 190 | 16 | 4096 | 64-way | FIFO | 0.965675 |
| 191 | 16 | 8192 | 64-way | LRU | 0.969136 |
| 192 | 16 | 8192 | 64-way | FIFO | 0.971523 |
| 193 | 16 | 16384 | 64-way | LRU | 0.969462 |
| 194 | 16 | 16384 | 64-way | FIFO | 0.97425 |
| 195 | 32 | 4096 | 64-way | LRU | 0.979955 |
| 196 | 32 | 4096 | 64-way | FIFO | 0.976348 |
| 197 | 32 | 8192 | 64-way | LRU | 0.981882 |
| 198 | 32 | 8192 | 64-way | FIFO | 0.981958 |
| 199 | 32 | 16384 | 64-way | LRU | 0.982129 |
| 200 | 32 | 16384 | 64-way | FIFO | 0.984805 |
| 201 | 64 | 8192 | 64-way | LRU | 0.988501 |
| 202 | 64 | 8192 | 64-way | FIFO | 0.986816 |
| 203 | 64 | 16384 | 64-way | LRU | 0.989034 |
| 204 | 64 | 16384 | 64-way | FIFO | 0.989986 |