<u>Contributors:</u>

1) *Rohit Mogli → IMT2021503*
2) *Sunny Kaushik → IMT2021007*
3) *Vadlamudi Karthikeya→ IMT2021504*

Keywords:

4) <u>Block Size</u>: Caches are divided into blocks, which may be of various sizes, the number of blocks is usually in the power of 2, here in our project we have created caches with varying block sizes like 1, 2, 4, 8, 16.

5) <u>Associativity</u>: In a set-associative cache, there are a fixed number of locations (called a set) in which a given address may be stored. **The number of locations in each set** is associated with the cache.

6) <u>Index</u>: The index cache is a portion of the machine's memory allocated to caching the data of specific terms sent to the Content component during index actions. Using the index cache speeds up indexing because writing to memory is quicker than writing to disk.

7) <u>Tag</u>: Every line stores a tag, which is certain upper bits of the physical address. Hence, by looking at the tag (after having zeroed in on which cache line to look at), we can decide whether the request was a hit or a miss.

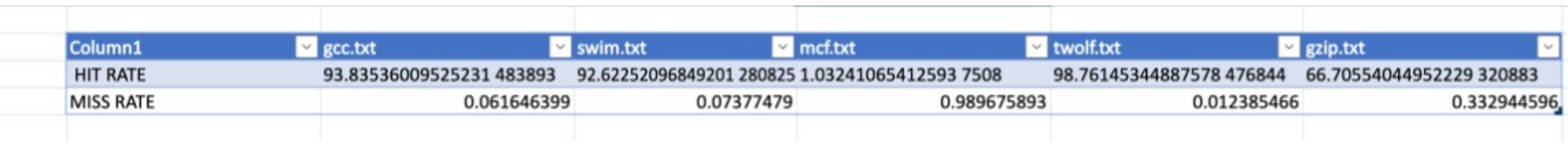
How to run the code:

Terminal:
—> Run the command “**g++ file_name.cpp**”
—> Then run “**./a.out**”

Visual Studio :
—> Open the file
“IMT2021007_IMT2021503_IMT2021504_(a/b/c/d).py”
in visual studio and simply right-click and choose to run option or go to the rightmost top corner and click the arrow icon.

Assignment Description:

Q1) a) We need 2 to the power 15 (215) lines.**



Column1	gcc.txt	swim.txt	mcf.txt	twolf.txt	gzip.txt
HIT RATE	93.83536009525231 483893	92.62252096849201 280825	1.03241065412593 7508	98.76145344887578 476844	66.70554044952229 320883
MISS RATE	0.061646399	0.07377479	0.989675893	0.012385466	0.332944596

—>We are implementing cache which is a special storage space for temporary files that makes a device, browser, or app run faster and more efficiently.

—>We create a two-dimensional array one maintaining the number of lines and the other maintaining number of ways. We can find the number of the lines using the formula(size of cache/block size * the number of ways).Log base 2 of (number of lines) is the number of the index bits, As the cache is four-way we can say that the number of bits for byte offset is 2 and as the number of bits is 32 number of bits for the tag is (32-(index bits +byte offset)).

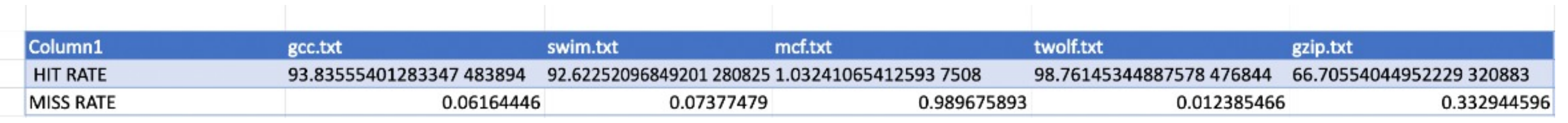
—>We take input from the file and separate the required address from each line using slicing.

—>We get the required tag bits, and index bits from the sliced address and now we match the index to the line number and insert the tag into the ways in order of LRU(least recently used) which removes the element which was least recently used when it comes to the situation of overwriting away.

—>Increase the count value if there is a hit (tag match and index match) for some other input value.

—>And we finally print the hit rate and the miss rate.

b)



Column1	gcc.txt	swim.txt	mcf.txt	twolf.txt	gzip.txt
HIT RATE	93.83555401283347 483894	92.62252096849201 280825	1.03241065412593 7508	98.76145344887578 476844	66.70554044952229 320883
MISS RATE	0.06164446	0.07377479	0.989675893	0.012385466	0.332944596

—>We are implementing cache which is a special storage space for temporary files that makes a device, browser, or app run faster and more efficiently.

—>We create a two-dimensional array one maintaining the number of lines and the other maintaining number of ways. We can find the number of the lines using the formula(size of cache/block size * the number of ways).Log base 2 of (number of lines) is the number of the index bits, As the cache is four-way we can say that the number of bits for byte offset is 2 and as the number of bits is 32 number of bits for the tag is (32-(index bits +byte offset)).

—>We take input from the file and separate the required address from each line using slicing.

—>We get the required tag bits, and index bits from the sliced address and now we match the index to the line number and insert the tag into the ways in order of LRU(least recently used) which removes the element which was least recently used when it comes to the situation of overwriting away.

—>Increase the count value if there is a hit (tag match and index match) for some other input value.

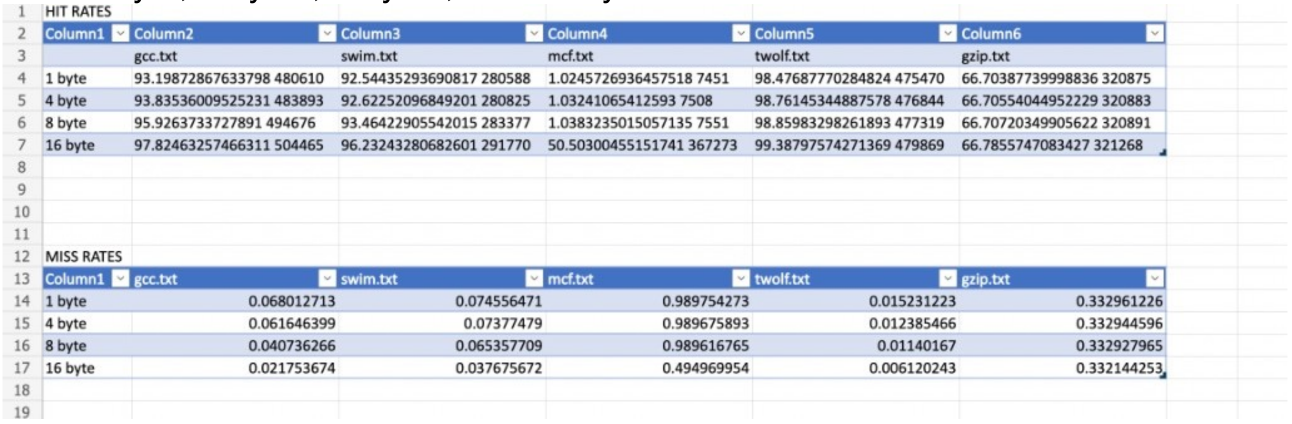
—>And we finally print the hit rate and the miss rate.

c)
—>In this part keeping the cache size 512 kB we are varying block size from 1 byte, 4 bytes, 8 bytes, and 16 bytes.

—>Increasing the block size means we are storing more amount of data in individual blocks.

—>Although this increases the hit rate but there are some disadvantages.
The larger the block size, the less the number of entries in the cache, and the more the competition between program data for these entries!
The larger the block size, the more time it takes to fetch this block size from memory.

Below is the table depicting the hit rates variation when we varied the block size from 1 byte, 4 bytes, 8 bytes, and 16 bytes.



HIT RATES

Column1	Column2	Column3	Column4	Column5	Column6
	gcc.txt	swim.txt	mcf.txt	twolf.txt	gzip.txt
1 byte	93.19872867633798 480610	92.54435293690817 280588	1.0245726936457518 7451	98.47687770284824 475470	66.70387739998836 320875
4 byte	93.83536009525231 483893	92.62252096849201 280825	1.03241065412593 7508	98.76145344887578 476844	66.70554044952229 320883
8 byte	95.9263733727891 494676	93.46422905542015 283377	1.0383235015057135 7551	98.85983298261893 477319	66.70720349905622 320891
16 byte	97.82463257466311 504465	96.23243280682601 291770	50.50300455151741 367273	99.38797574271369 479869	66.7855747083427 321268

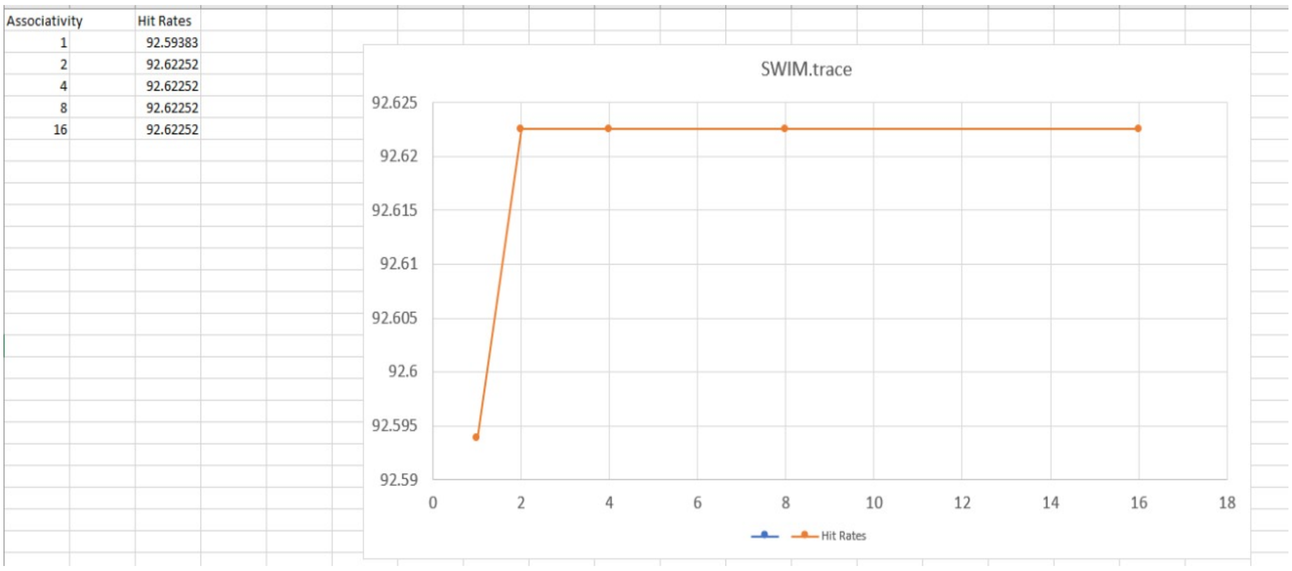
MISS RATES

Column1	gcc.txt	swim.txt	mcf.txt	twolf.txt	gzip.txt
1 byte	0.068012713	0.074556471	0.989754273	0.015231223	0.332961226
4 byte	0.061646399	0.07377479	0.989675893	0.012385466	0.332944596
8 byte	0.040736266	0.065357709	0.989616765	0.01140167	0.332927965
16 byte	0.021753674	0.037675672	0.494969954	0.006120243	0.332144253

d)
—> In this part we are varying the associativity from 1, 2, 4, 8, and 16 wherein according to the associativity, we will change tag and index.

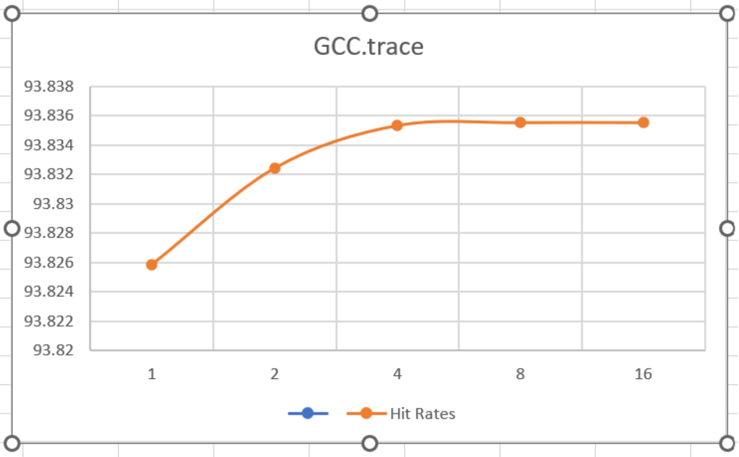
—> Associativity is the size of these sets, or, in other words, how many different cache lines each data block can be mapped to. Higher associativity allows for more efficient utilisation of cache but also increases the cost.

Graphs show the increase in hit rates as we increase the associativity from 1,2, 4, 8, and 16 in different test cases.

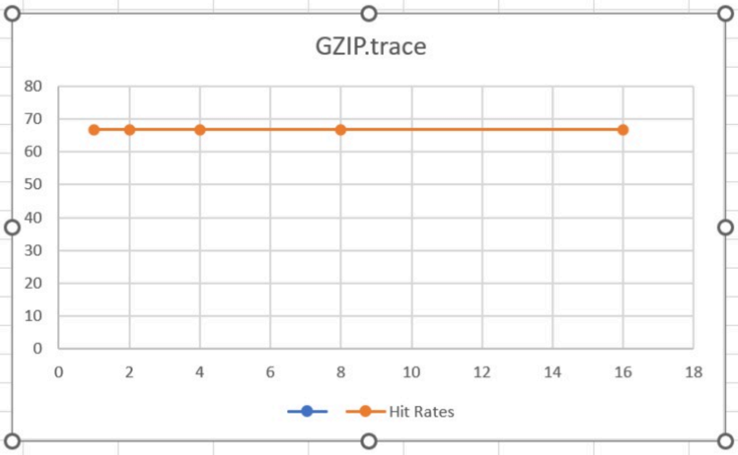


Associativity	Hit Rates
1	92.59383
2	92.62252
4	92.62252
8	92.62252
16	92.62252

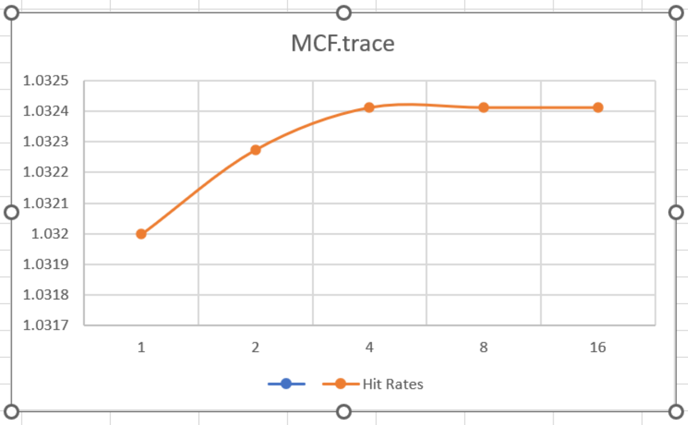
Associativity	Hit Rates
1	93.82586
2	93.83245
4	93.83536
8	93.83555
16	93.83555



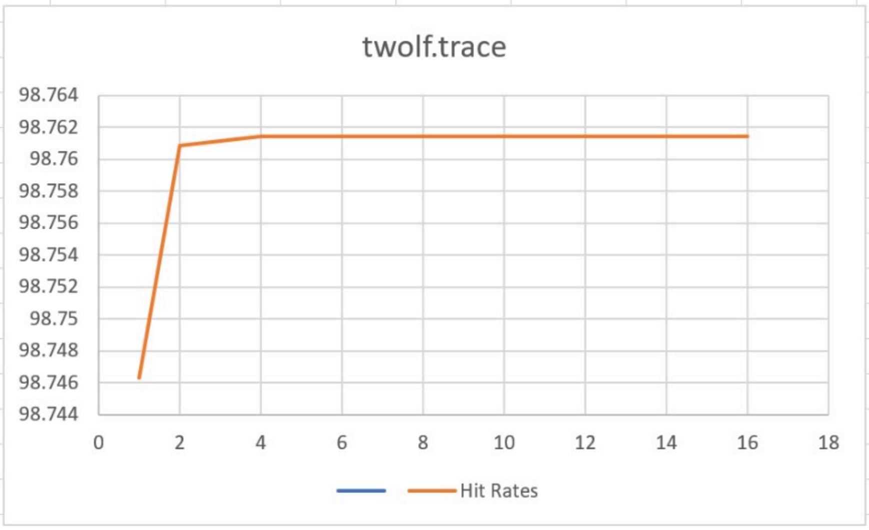
Associativity	Hit Rates
1	66.70554
2	66.70554
4	66.70554
8	66.70554
16	66.70554



Associativity	Hit Rates
1	1.031998
2	1.032273
4	1.032411
8	1.032411
16	1.032411



Associativity	Hit Rates
1	98.7463
2	98.7608
4	98.7615
8	98.7615
16	98.7615



Therefore we can conclude from the above that if we increase the associativity increases the hit rate.