Name: Jahanvi Agrawal
RollNo. : IMT2019506
Email: Jahanvi.Agrawal@iiitb.org

ASSIGNMENT-3 REPORT

Note: The comparision table and observations are towards the end of this document.

Instructions to run the trace files:

Step1: Open the terminal and move into the folder named IMT2019506 which has both the .v files (with each being for each cache) and all the five trace files.

Step2: Now, for running a specific trace file, open the code and make sure to write the name of the trace file to be read, in the line where we are opening the requested trace file.
For example; For reading **swim.trace** file,

In case of DirectMapped Cache, open IMT2019506_dmCache.v and go to line 30 and write the trace file name there.
In case of SetAssociative Cache, open IMT2019506_setCache.v and go to line 43 and write the trace file name there.
It should look like: **file = $fopen("swim.trace","r");**

For the Direct Mapped Cache:
Step3: Run these commands on the terminal:

```
>iverilog -o test  IMT2019506_dmCache.v
>vvp test
```

For the 4-way Set Associative Cache:
Step3: Run these commands on the terminal:

```
>iverilog -o test  IMT2019506_setCache.v
>vvp test
```

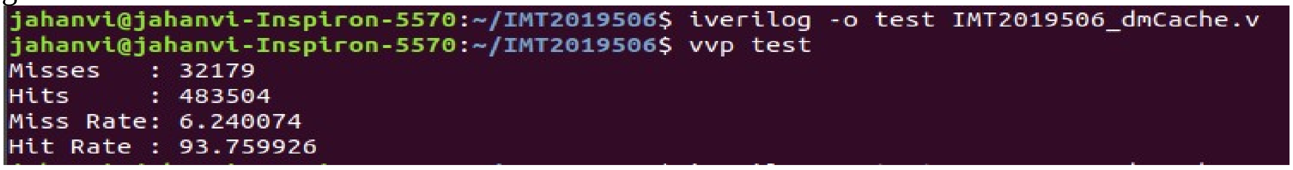
DIRECT MAPPED CACHE:

#IndexBits = 16
#TagBits = 14
#ByteOffsetBits = 2

Thus, cache has 2^(16) rows with 47bits in each row (1bit for valid bit + 14 tag bits + 32bits for data).

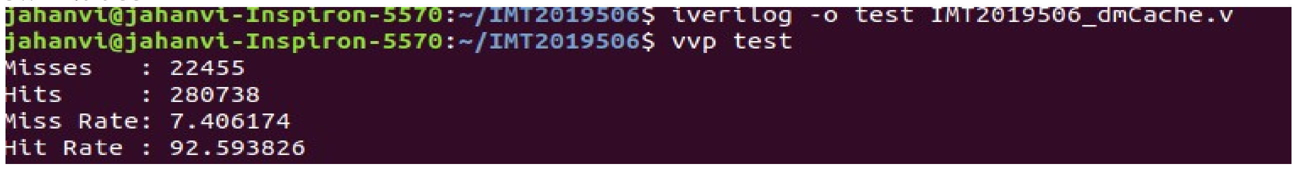
Following are the snapshots of the terminal for each trace file:

1. gcc.trace



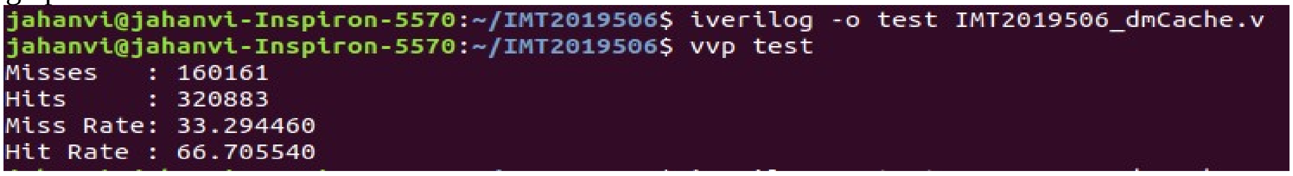
```
jahanvi@jahanvi-Inspiron-5570:~/IMT2019506$ iverilog -o test IMT2019506_dmCache.v
jahanvi@jahanvi-Inspiron-5570:~/IMT2019506$ vvp test
Misses    : 32179
Hits      : 483504
Miss Rate: 6.240074
Hit Rate : 93.759926
```

2. swim.trace



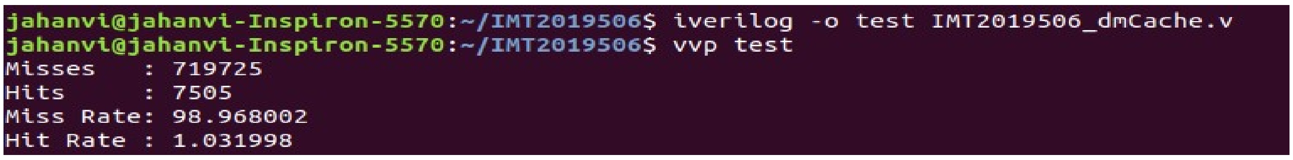
```
jahanvi@jahanvi-Inspiron-5570:~/IMT2019506$ iverilog -o test IMT2019506_dmCache.v
jahanvi@jahanvi-Inspiron-5570:~/IMT2019506$ vvp test
Misses    : 22455
Hits      : 280738
Miss Rate: 7.406174
Hit Rate : 92.593826
```

3. gzip.trace



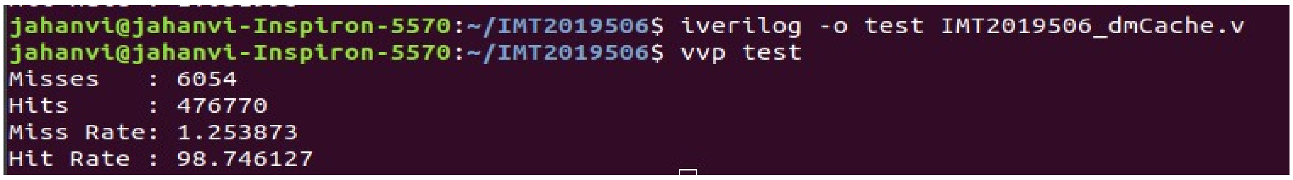
```
jahanvi@jahanvi-Inspiron-5570:~/IMT2019506$ iverilog -o test IMT2019506_dmCache.v
jahanvi@jahanvi-Inspiron-5570:~/IMT2019506$ vvp test
Misses    : 160161
Hits      : 320883
Miss Rate: 33.294460
Hit Rate : 66.705540
```

4. mcf.trace



```
jahanvi@jahanvi-Inspiron-5570:~/IMT2019506$ iverilog -o test IMT2019506_dmCache.v
jahanvi@jahanvi-Inspiron-5570:~/IMT2019506$ vvp test
Misses    : 719725
Hits      : 7505
Miss Rate: 98.968002
Hit Rate : 1.031998
```

5. twolf.trace



```
jahanvi@jahanvi-Inspiron-5570:~/IMT2019506$ iverilog -o test IMT2019506_dmCache.v
jahanvi@jahanvi-Inspiron-5570:~/IMT2019506$ vvp test
Misses    : 6054
Hits      : 476770
Miss Rate: 1.253873
Hit Rate : 98.746127
```

<u>4-WAY SET ASSOCIATIVE CACHE:</u>

#IndexBits = 14
#TagBits = 16
#ByteOffsetBits = 2

Thus, cache has 2^(14) rows with 196bits in each row (Each row has 4-ways with each way of 49bits (1bit for valid bit + 16 tag bits + 32bits for data))

<u>Least-Recently-Used Algorithm used:</u>

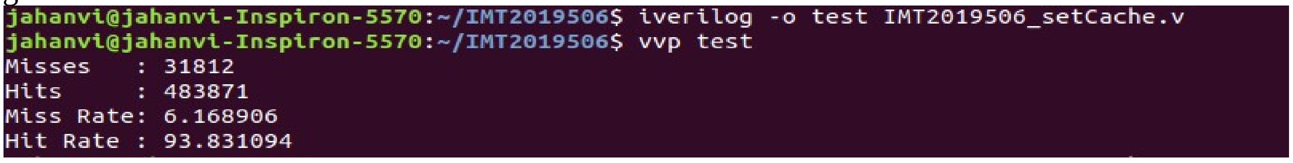
To implement eviction, I have implemented the following LRU algorithm.
(Indexing of Ways begins from 0)
I have made a reference 2D array (*recAccess*) of 2^(14) rows and 4 colums(for each way). This array stores the latest instruction number that accessed a particular way at a particular index of the cache.
So if instruction 5 accesses way-2 at index-0 of the cache, then in this 2D array *recAccess* I update reAccess[0][2] = 5; which means that way-2 at index-0 was latest accessed by instruction 5.
And if again any other instruction accesses the same way at the same index, the instruction number in the 2D array is updated.
Now, when we need to evict i.e., when all the 4 ways at that index are full, to get the least recently used way at that index, I refer to the 2D array row at that index location. The **minimum** is found out of all the 4 columns (or ways). By doing so, we are getting the least recently used way out of the four ways . After this, as I have got the least recently used way, I can evict this way in my cache and update the tag bits and data accordingly.
Example:
If an index row in the 2D array *recAccess* looks like:

5	1	4	6

Now, if we request instruction 7 which has the same index as the above row and as all the ways are full, we need to perform eviction in the *cache*. We take help of *recAccess* array. The minimum in this row is at Way-1 i.e. for that specific index location Way-1 is the least recently used way as out of all the other three ways, it was last used by instruction 1. So, we conclude that in the *cache,* at the required index, we need to evict Way-1 for instruction 7.
In the *cache,* at way-1 the tag bits and data is updated for instruction 7. Also, in the *recAccess* array, the instruction number at way-1 of that index is updated to 7.
And, we move on to the next instruction.

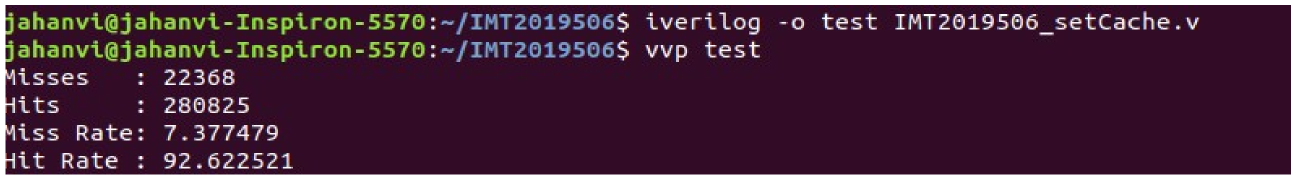
Following are the snapshots of the terminal for each trace file:

1. gcc.trace



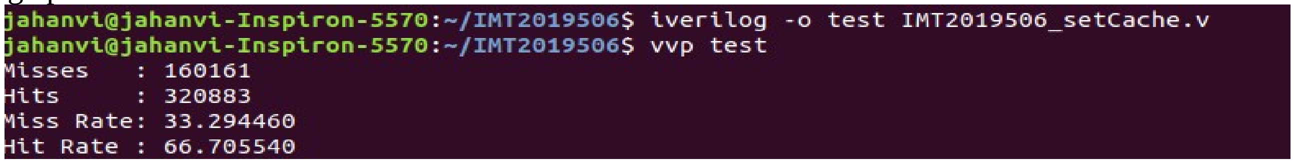
```
jahanvi@jahanvi-Inspiron-5570:~/IMT2019506$ iverilog -o test IMT2019506_setCache.v
jahanvi@jahanvi-Inspiron-5570:~/IMT2019506$ vvp test
Misses    : 31812
Hits      : 483871
Miss Rate: 6.168906
Hit Rate : 93.831094
```

2. swim.trace



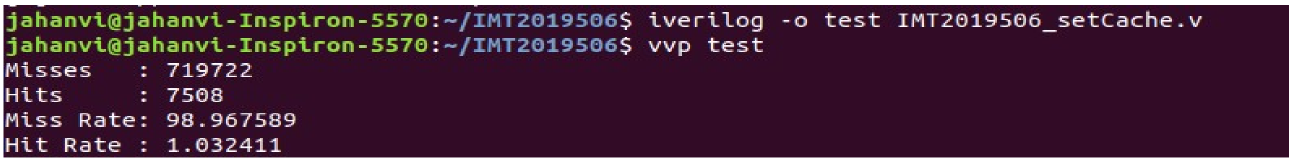
```
jahanvi@jahanvi-Inspiron-5570:~/IMT2019506$ iverilog -o test IMT2019506_setCache.v
jahanvi@jahanvi-Inspiron-5570:~/IMT2019506$ vvp test
Misses    : 22368
Hits      : 280825
Miss Rate: 7.377479
Hit Rate : 92.622521
```

3. gzip.trace



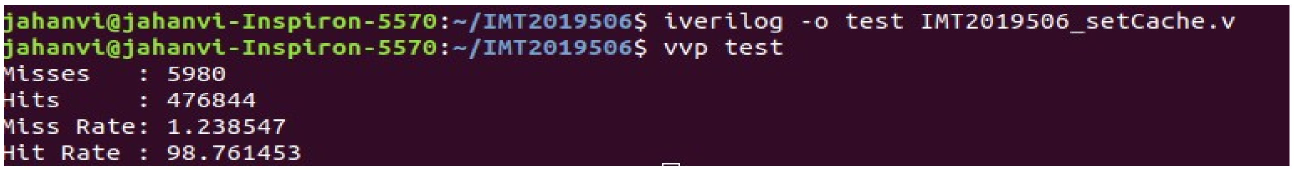
```
jahanvi@jahanvi-Inspiron-5570:~/IMT2019506$ iverilog -o test IMT2019506_setCache.v
jahanvi@jahanvi-Inspiron-5570:~/IMT2019506$ vvp test
Misses    : 160161
Hits      : 320883
Miss Rate: 33.294460
Hit Rate : 66.705540
```

4. mcf.trace



```
jahanvi@jahanvi-Inspiron-5570:~/IMT2019506$ iverilog -o test IMT2019506_setCache.v
jahanvi@jahanvi-Inspiron-5570:~/IMT2019506$ vvp test
Misses    : 719722
Hits      : 7508
Miss Rate: 98.967589
Hit Rate : 1.032411
```

5. twolf.trace



```
jahanvi@jahanvi-Inspiron-5570:~/IMT2019506$ iverilog -o test IMT2019506_setCache.v
jahanvi@jahanvi-Inspiron-5570:~/IMT2019506$ vvp test
Misses    : 5980
Hits      : 476844
Miss Rate: 1.238547
Hit Rate : 98.761453
```

COMPARISION and OBSERVATIONS:

For each cache, the following are the **HITS RATES** for each of the input trace files.

	DirectMapped Cache Hit Rates	**4-way SetAsscociative Cache Hit Rates**
gcc.trace	93.759926	93.831094
swim.trace	92.593826	92.622521
gzip.trace	66.705540	66.705540
mcf.trace	1.031998	1.032411
twolf.trace	98.746127	98.761453

Looking at this table, we can see that for a specific trace file:
Hit rate for the DirectMapped Cache < Hit rate for the 4-Way SetAssociative Cache
This happens cause the number of times we need to **evict** the cache memory is greater in the case of Direct Mapped Cache than the 4-Way SetAssociative Cache. Hence, the hit rate is lesser for the DirectMapped cache.

For example:

l 0x00080000
l 0x00040000
l 0x00000000
s 0x000C0000
s 0x00100000
s 0x00080000

All the above addresses are having the same index but different tag.
(Indexing of Ways begins from 0)

For the above set of instructions, in the *DirectMapped Cache*, the eviction is higher, as in the cache, we will have to **evict** as in when the next address comes in with the same index and it is not a tag match. Here, eviction is done 5 times at index-0 in the cache(i.e. for instructions 2,3,4,5 and 6). When the last instruction is passed 0x0008000, it is not in the cache, so it's not a hit. Hence, the **hit rate is lesser** in this cache.

Now, in the *4-Way Set Associative Cache*, the instruction-1 goes at way-0 of index-0 of the cache, the instruction-2 goes in way-1 of index-0, the instruction-3 goes in way-2 of index-0 and the instruction-4 goes in way-3 of index-0. Since, we had empty ways , there was **no need of eviction** for these instructions. For instruction 5, since all the ways are full now at index-0 and there is no tag match, using the LRU algorithm we get the least recently used way and store instruction-5 there. Here, when the last instruction is passed 0x0008000, this address is still in the cache in the way-0 at index-0, so it's a hit and hence, we get a **better hit rate** in this cache.