COL216 – Computer Architecture
Assignment 12
Simulating Cache Memory
(Manoj Kumar, cs5180411@cse.iitd.ac.in)

AIM :

In this assignment, I developed a simulation software for cache memory. The simulated cache will have the functionality that the cache is devided in Cache Sets, and each cache set is devided in two groups :
1. One group contains the HIGH PRIORITY lines of the set.
2. The other group contains the LOW PRIORITY lines of the set.

Priority Rule for a Block :

1. If a block/line is accessed first time, it will take place in LOW priority group.
2. If a line/block is accessed again after the initial access that fatches in into the cache, it is promoted to the HIGH PRIORITY group.
3. If a line is not accessed for sufficiently long (T cache accesses) after being moved to the HIGH PRIORITY group, it is moved to the LOW PRIORITY group.

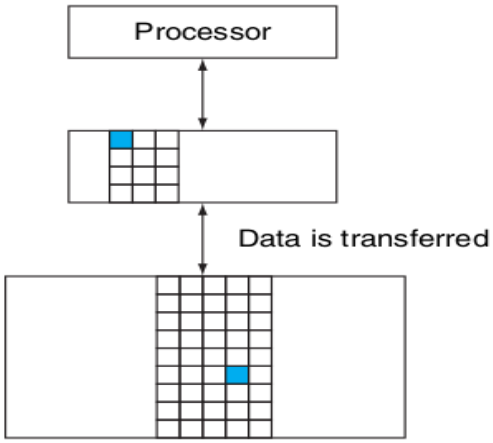
LOCALITY CONCEPT :

The principle of locality states that programs access a relatively small portion of their address space at any instant of time. There are two different types of locality:

1. **Temporal locality (locality in time):** The principle stating that if a data location is referenced then it will tend to be referenced again soon. For example, most programs contain loops, so instructions and data are likely to be accessed repeatedly, showing high amounts of temporal locality.
2. **Spatial locality (locality in space):** The locality principle stating that if a data location is referenced, data locations with nearby addresses will tend to be referenced soon. Since instructions are normally accessed sequentially, programs also show high spatial locality. For example, sequential accesses to elements of an array or a record will naturally have high degrees of spatial locality.

CACHE CONCEPT :

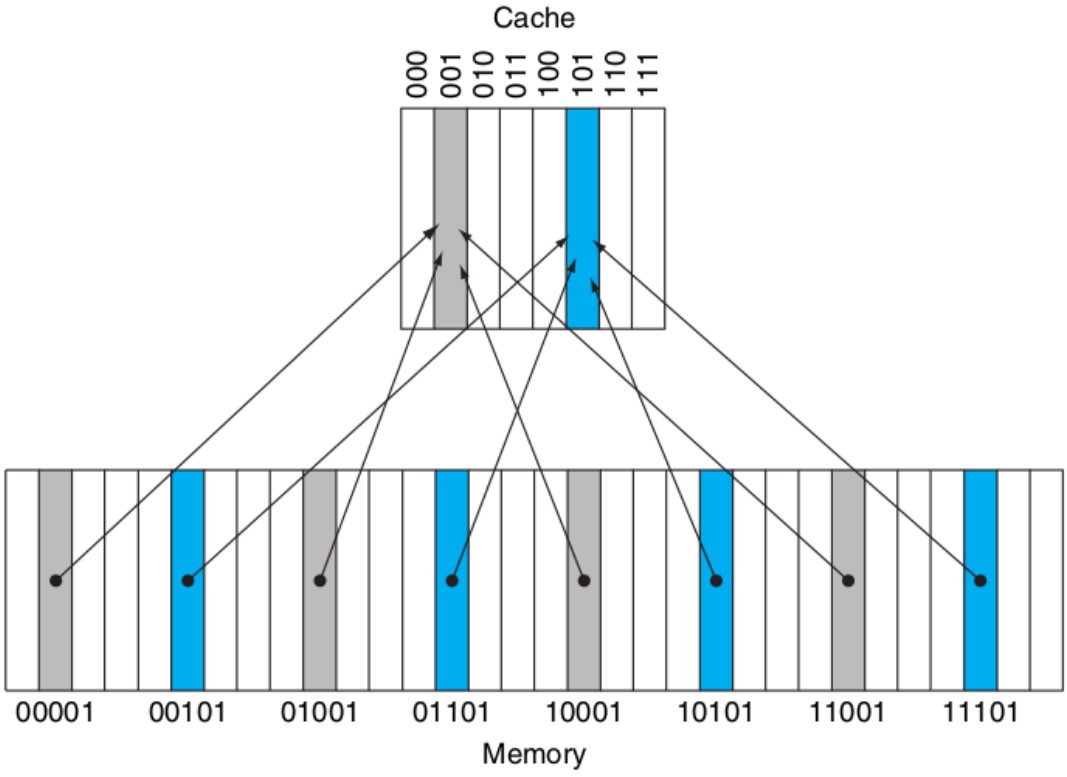
Here is the basic structure of memory heirarchy. Every pair of levels in the memory hierarchy can be though of as having an upper and lower level. With each level, the unit of information that is present or not is called a block or a line. Usually we transfer an entire block when we copy something between levels.



COL216 – Computer Architecture
Assignment 12
Simulating Cache Memory
(Manoj Kumar, cs5180411@cse.iitd.ac.in)

HIT : If the data requested by the processor appears in some block in the upper level, this is called a **hit**.
MISS : If the data requested by the processor is not found in any block in the upper level, this is called a **miss**.

Excess of main memory is much slower than cache. The probability of a HIT increases with the total number of blocks in cache memory. But size of cache memory decreases the speed of excessing cache memory also.
A direct-mapped cache with eight entries showing the addresses of memory words between 0 and 31 that map to the same cache locations, is showen below.



COL216 – Computer Architecture
Assignment 12
Simulating Cache Memory
(Manoj Kumar, cs5180411@cse.iitd.ac.in)

ASSUMPTIONS FOR THIS MODEL :

1. The main memory is byte addressable.
2. A block is a single entity which can be accessed.
3. Reading and writing a block happens at block granularity. Cache is accessed at block size granularity.
4. The given data should be of block size. It means data cannot be greater than total bytes in a block.
5. When a block is accessed first time, it belogs to the LOW priority group, but when it is accessed again, it is promoted to the HIGH priority group.
6. If a HIGH Proiority block is not accessed for T cache accesses, after being moved to the HIGH priority group, it is moved to the LOW priority group.
7. The number of blocks in LOW and HIGH priority groups in a set is dynamic. i.e. HIGH and LOW priority groups are not fixed in size. The number of blocks in HIGH and LOW priority groups depends on the cache configuration. Although Total number of blocks in a set remains constant.
8. *Least Recently Used (LRU)* policy within groups is followed.
9. *Write Back* procedure is followed to make changes in main memory.

REPLACEMENT POLICY :

Least Recently Used (LRU) policy within groups has been used to replace the least recently used block with newly accessed block.
When a block is accessed that is not present in cache, we go for main memory and write that block in cache. If there is a invalid block present in cache, we replace that invalid block with this newly arrived data and set this new block in LOW priority.
If there is no invalid block in the perticular set, we remove the least recently used block from LOW priority group and insert the newly used block at that place.
If the size of LOW priority group is zero, than we apply LRU on HIGH priority group and erase the least recently used block from HIGH priority, and insert the newly used block in LOW priority group.

Note :

1. A cache is **dirty** if the value of a block in cache and main memory are different or it is dirty if it is modified in cache but not in Main memory.
2. A cache is **valid** if it cantains valid tag and data.

COL216 – Computer Architecture
Assignment 12
Simulating Cache Memory
(Manoj Kumar, cs5180411@cse.iitd.ac.in)

MEMORY UPDATE :

Write Back policy has been used in memory update. It means we update a perticular cache block to main memory when some other block replaces the block from cache.
If we access a block for the first time, we get a MISS, we simply copy that block from Main memory and store it in the cache. Initially it is not dirty. Now in the next instruction, if value of the block is changed, we change it in the cache, not in the memory as it occurs a HIT. At this time, the block is dirty because we modify the perticular block in cache not in main memory.
Now if a new block access occurs and we have to erase this block to make space for the newly arrived block. Then before removing the block, we update the data value in one layer uper memory(in our assignment, main memory). This process is called Write Back.

Advantage of Write Back :

- Writes occur at the speed of the cache memory
- Multiple writes within a block require only one write to main memory
- As a result uses less memory bandwidth

INSTRUCTION TYPE :

Addr, type, Data

Here I assume that *Addr* can be upto 32 bits long integer. We separate these 32 bits in *tag*, *set address* and *offset*. *tag* bits determine the perticular block in a corresponding set. *set address* bits determine the corresponding set. and the *offset* bits determine the byte addresses that indicate same block.
Let's understand this with the below given test case.

1. Here cache size is 16 bytes and block size is 2 bytes. Then total blocks in cache = 16/2 = 8
2. Cache associativity is 2, it means each set contains 2 blocks. Total num of sets = 8/2 = 4
3. Let the sets be *set 0, set 1, set 2* and *set 3*.
4. number of *offset* bits = $\log_2$(block size) = $\log_2(2) = 1$
5. number of *set address* bits = $\log_2$(total num of sets) = $\log_2(4) = 2$
6. rest are *tag* bits.

tag **Set address** **Offset bits**

THROW ERRORS:

Program will throw error if any of the below listed conditions happens :

1. Cache Size, block Size and set associativity should follow the following constraint :

   *Cache Size = k*Block Size * Set Associativity*

   where k is an integer value (total number of sets in cache)

2. Data is given for a block. So given *data* should not exceed the block size.

   *Maximum bits in Data = Block Size*8*

COL216 – Computer Architecture
Assignment 12
Simulating Cache Memory

(Manoj Kumar, cs5180411@cse.iitd.ac.in)

$$OR$$
$$Data < 2^{Block\ Size*8}$$

3. Block size and set associativity should be a power of 2 and block size should be less than 8 bytes (64 bit).

$$Block\ size = 2^{A}$$
$$Set\ associativity = 2^{B}$$
$$Cache\ Size = k*2^{(A+B)}$$

4. Block Size and set Associativity cannot be zero either.
5. Block Size should be less than 9 (upto 64 bytes).

IMPLEMENTATION DETAILS :

1. Hashmap *m* has been used for main memory.

- Check whether the inputs are valid or not, if not then return.
- Find number of offset bits, set address bits and tag bits
- get maximum value a block can store (to check the valid access)
- For each access :
  - separate address, type of access, tag and data
  - check validity of data
  - check whether it is a HIT
    - update last access time and data (if it is a write).
    - If it is a read access, then read the data from cache
  - if it is a MISS
    - if it is a ‘R’ access
      - read data value from memory (create valid random value and store in main memory)
    - create a block that is to be inserted
    - insert that block in LOW priority of respective set using LRU policy within groups
    - Write Back : update the erased cache block in main memory.
  - Update the LOW and HIGH priority groups of each set with total number of accesses.
  - Print the set configuration
- Print Cache Statistics.

COL216 – Computer Architecture
Assignment 12
Simulating Cache Memory
(Manoj Kumar, cs5180411@cse.iitd.ac.in)

Sample Test Case :

```
16 <Cache size in bytes>
2 <Cache line/block size in bytes>
2 <Cache associativity>
4 <T>
#memory access requests
1, W, 1
5, W, 5
6, W, 6
1, R
0, W, 0
5, R
7, W, 7
3, W, 3
5, W, 50
7, R
```

EXPLANATION :
The initial configuration of all the sets are :

```
------------------------------------------------------------------
set 0
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
set 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
set 2
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
set 3
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
```

COL216 – Computer Architecture
Assignment 12
Simulating Cache Memory

(Manoj Kumar, cs5180411@cse.iitd.ac.in)

Now first Instruction is :
1, W, 1
address : 1 = 00000000000000000000000000000001
offset(binary) = 1
tag(binary) = 0
set address(binary) = 0
This indicates *set 0* with tag =0 and data =1.
Since, both blocks are *invalid,* so it is a MISS and we store *tag* and *data* values to one of the two blocks of *set0.*
Since, value of the block with tag bit 0 and set address 0 can be anything in main memory, so at the end, we need to modify this block in main memory, so it is in **dirty** state.
Now the new sets configuration is :

Next

```
------------------------------------------------------------------
set 0
LOW tag(binary) : 0 data(int) : 1 lastupdate : 1 valid : 1 dirty : 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
set 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
set 2
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
set 3
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
```

instruction is :
5, W, 5
address : 5 = 00000000000000000000000000000101
offset(binary) : 1
tag(binary) : 0
set address(binary) : 10
This indicates *set 2* with *tag = 0* and *data* = 5
Since, both blocks of *set 2* are invalid, so we store *tag* and *data* values to one of the two blocks of *set 2.*
Since, value of the block with tag bit 0 and set address 2 can be anything in main memory, so at the end, we need to modify this block in main memory, so it is in **dirty** state.
Since, both valid blocks are not yet accessed again, so both are in LOW priority group.
Now the new sets configuration is :

COL216 – Computer Architecture
Assignment 12
Simulating Cache Memory

(Manoj Kumar, cs5180411@cse.iitd.ac.in)


```
------------------------------------------------------------------
set 0
LOW tag(binary) : 0 data(int) : 1 lastupdate : 1 valid : 1 dirty : 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
set 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
set 2
LOW tag(binary) : 0 data(int) : 5 lastupdate : 2 valid : 1 dirty : 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
set 3
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
```

Next Instruction is :
6, W, 6
address : 6 = 00000000000000000000000000000110
offset(binary) : 0
tag(binary) : 0
set address(binary) : 11
This indicates *set 3* with *tag = 0* and *data* = 6
Since, both blocks of *set 3* are invalid, so we store *tag* and *data* values to one of the two blocks of *set 3.*
Since, value of the block with tag bit 0 and set address 3 can be anything in main memory, so at the end, we need to modify this block in main memory, so it is in **dirty** state.

Since, all valid blocks are not yet accessed again, so all are in LOW priority group.
Now the new sets configuration is :

```
------------------------------------------------------------------
set 0
LOW tag(binary) : 0 data(int) : 1 lastupdate : 1 valid : 1 dirty : 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
set 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
set 2
LOW tag(binary) : 0 data(int) : 5 lastupdate : 2 valid : 1 dirty : 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
set 3
LOW tag(binary) : 0 data(int) : 6 lastupdate : 3 valid : 1 dirty : 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
```

COL216 – Computer Architecture
Assignment 12
Simulating Cache Memory

(Manoj Kumar, cs5180411@cse.iitd.ac.in)

Next Instruction is :
1, R
address : 1 = 00000000000000000000000000000001
offset(binary) : 1
tag(binary) : 0
set address(binary) : 00
This indicates *set 0* with *tag = 0*
Since, the block corresponding to *set 0* and *tag 0* is already present in cache. So we will read the block data that is 1 and promote this block to high priority.
Now the new sets configuration is :

```
------------------------------------------------------------------------
set 0
HIGH tag(binary) : 0 data(int) : 1 lastupdate : 4 valid : 1 dirty : 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------------
set 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------------
set 2
LOW tag(binary) : 0 data(int) : 5 lastupdate : 2 valid : 1 dirty : 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------------
set 3
LOW tag(binary) : 0 data(int) : 6 lastupdate : 3 valid : 1 dirty : 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------------
```

Next Instruction is :
0, W, 0
address : 0 = 00000000000000000000000000000000
offset(binary) : 0
tag(binary) : 0
set address(binary) : 00
This indicates *set 0* with *tag = 0* and *data* = 0
Since, the block corresponding to *set 0* and *tag 0* is already present in cache. So we will write the block data to be 0 and promote this block to high priority(since it is on HIGH priority already so it will remain in the same group).

Now the new sets configuration is :

COL216 – Computer Architecture
Assignment 12
Simulating Cache Memory

(Manoj Kumar, cs5180411@cse.iitd.ac.in)


```
------------------------------------------------------------------
set 0
HIGH tag(binary) : 0 data(int) : 0 lastupdate : 5 valid : 1 dirty : 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
set 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
set 2
LOW tag(binary) : 0 data(int) : 5 lastupdate : 2 valid : 1 dirty : 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
set 3
LOW tag(binary) : 0 data(int) : 6 lastupdate : 3 valid : 1 dirty : 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
```

Next Instruction is :
5, R
address : 5 = 00000000000000000000000000000101
offset(binary) : 1
tag(binary) : 0
set address(binary) : 10
This indicates *set 2* with *tag = 0*
Since, the block corresponding to *set 2* and *tag 0* is already present in cache. So we will read the block data that is 5 and promote this block to high priority.
Now the new sets configuration is :

```
------------------------------------------------------------------
set 0
HIGH tag(binary) : 0 data(int) : 0 lastupdate : 5 valid : 1 dirty : 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
set 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
set 2
HIGH tag(binary) : 0 data(int) : 5 lastupdate : 6 valid : 1 dirty : 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
set 3
LOW tag(binary) : 0 data(int) : 6 lastupdate : 3 valid : 1 dirty : 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
```

Next Instruction is :
7, W, 7
address : 7 = 00000000000000000000000000000111
offset(binary) : 1
tag(binary) : 0
set address(binary) : 11

COL216 – Computer Architecture
Assignment 12
Simulating Cache Memory

(Manoj Kumar, cs5180411@cse.iitd.ac.in)

This indicates *set 3* with *tag = 0* and *data* = 7

Since, the block corresponding to *set 3* and *tag 0* is already present in cache. So we will write the block data to be 7 and promote this block to high priority

Now the new sets configuration is :

```
------------------------------------------------------------------
set 0
HIGH tag(binary) : 0 data(int) : 0 lastupdate : 5 valid : 1 dirty : 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
set 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
set 2
HIGH tag(binary) : 0 data(int) : 5 lastupdate : 6 valid : 1 dirty : 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
set 3
HIGH tag(binary) : 0 data(int) : 7 lastupdate : 7 valid : 1 dirty : 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
```

Next Instruction is :

3, W, 3

address : 3 = 00000000000000000000000000000011

offset(binary) : 1

tag(binary) : 0

set address(binary) : 01

This indicates *set 1* with *tag = 0* and *data* = 3

Since, both blocks of *set 1* are invalid, so we store *tag* and *data* values to one of the two blocks of *set 1*.

Since, value of the block with tag bit 0 and set address 1 can be anything in main memory, so at the end, we need to modify this block in main memory, so it is in **dirty** state.

Now the new sets configuration is :

```
------------------------------------------------------------------
set 0
HIGH tag(binary) : 0 data(int) : 0 lastupdate : 5 valid : 1 dirty : 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
set 1
LOW tag(binary) : 0 data(int) : 3 lastupdate : 8 valid : 1 dirty : 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
set 2
HIGH tag(binary) : 0 data(int) : 5 lastupdate : 6 valid : 1 dirty : 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
set 3
HIGH tag(binary) : 0 data(int) : 7 lastupdate : 7 valid : 1 dirty : 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
```

COL216 – Computer Architecture
Assignment 12
Simulating Cache Memory

(Manoj Kumar, cs5180411@cse.iitd.ac.in)

Next Instruction is :
5, W, 50
address : 5 = 00000000000000000000000000000101
offset(binary) : 1
tag(binary) : 0
set address(binary) : 10

This indicates *set 2* with *tag = 0* and *data* = 50

Since, the block corresponding to *set 2* and *tag 0* is already present in cache. So we will write the block data to be 7 and promote this block to high priority.
Also, in *set 0*, block with tag 0 has not been accessed for a sufficient time, so it moves to LOW priority group.

Now the new sets configuration is :

```
------------------------------------------------------------------
set 0
LOW tag(binary) : 0 data(int) : 0 lastupdate : 5 valid : 1 dirty : 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
set 1
LOW tag(binary) : 0 data(int) : 3 lastupdate : 8 valid : 1 dirty : 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
set 2
HIGH tag(binary) : 0 data(int) : 50 lastupdate : 9 valid : 1 dirty : 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
set 3
HIGH tag(binary) : 0 data(int) : 7 lastupdate : 7 valid : 1 dirty : 1
LOW tag(binary) : 0 data(int) : 0 lastupdate : 0 valid : 0 dirty : 0
------------------------------------------------------------------
```

and same prcedure will be followed for next instruction.

Cache Statistics:

```
Cache Statistics :
Number of accesses : 10
Number of reads : 3
Number of read hits : 3
Number of read misses : 0
Number of Writes : 7
Number of Write Hits : 3
Number of Write Misses : 4
total hits : 6
total miss : 4
Hit ratio : 0.6
```

COL216 – Computer Architecture
Assignment 12
Simulating Cache Memory

(Manoj Kumar, cs5180411@cse.iitd.ac.in)

Submitted By :
Manoj Kumar 2018CS50411

Thanks to Professor Preeti Ranjan Panda, all TAs and staff members
I really enjoyed the course. I got to know a lot of things about hardware and computer architecture :)