Computer Architecture (COL 216)

II Semester 2019-2020

Assignment 12: Simulating Cache Memory

Submission Deadline: 21 June 2020

INDIVIDUAL (not group) ASSIGNMENT

[This assignment is intended ONLY for graduating students registered for the current semester. Other students (such as 2nd year B. Techs) should not submit the assignment. If you belong to this category, you have already been notified about it by the academic section.]

In this assignment you will develop a simulation software for cache memory. The simulated cache will have the functionality that was discussed in the cache memory lecture videos, but with a *new Replacement Policy* that works as follows. Each Cache Set is divided into 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

How is *priority* established? If a line is accessed again after the initial access that fetches it into the cache, it is promoted to the HIGH PRIORITY group. 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. Within a priority group, the Least Recently Used policy may be used to manage the lines.

Make any reasonable assumptions you need. The above priority scheme is just an outline; it may not be complete. Feel free to add other details to make this work.

Inputs to the software will be: (inputs to be taken from input file)

1. Cache size
2. Cache line/block size
3. Cache associativity
4. The value of T
5. A sequence of memory access requests: Memory address, R or W (for Read or Write), and Data (if Write)

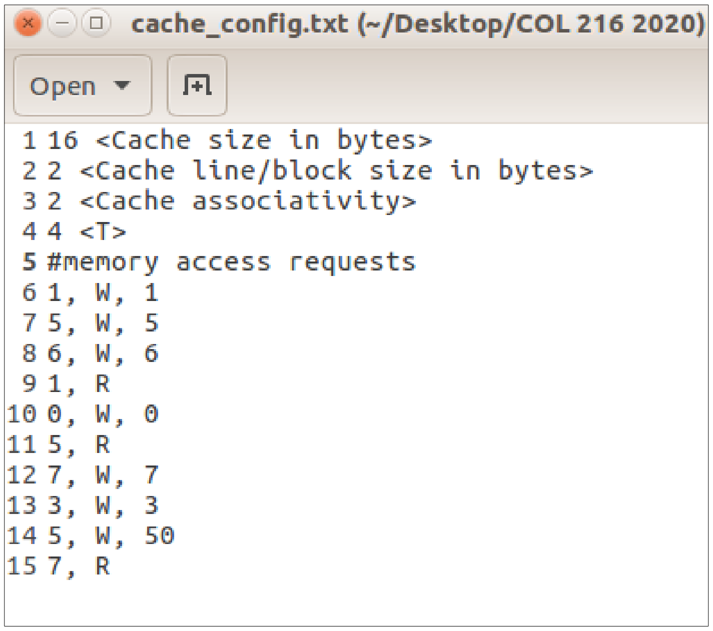
Outputs from the software will be: (output the content and cache statistics to output file)

1. The complete content of the cache (Data, Tag, and Control bits of every cache line) in each Way after the memory access sequence is complete.
2. Cache statistics:
   a. Number of Accesses
   b. Number of Reads, Read Hits, and Read Misses
   c. Number of Writes, Write Hits, and Write Misses
   d. Hit Ratio

Deliverables:

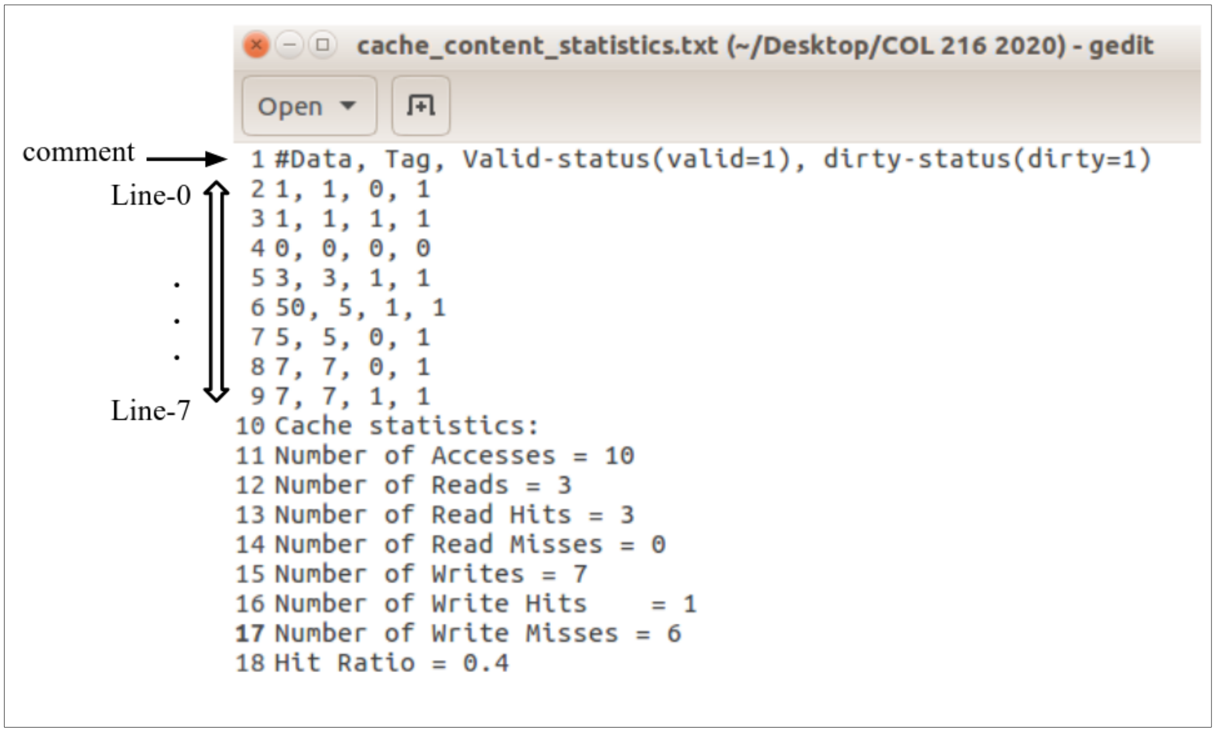
1. Software implementation of the cache described above
2. Document describing the design. In the document, discuss your implementation of the replacement policy. Are the High and Low priority groups fixed in size? That is, in an 8-way associative cache, are there 4 high-priority and 4 low-priority groups? Or some other break-up? Why? Can these sizes change at runtime?

Sample Input File:
Considered a very small cache size for demonstrating sample input and sample output.



```
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
```

Sample Output File:
Assumptions made :
1. Out of 2 ways for each set, one way is assigned to high priority group and second way is assigned to low priority group.
2. When valid status of the line is being modified to invald, the dirty status is left unmodified. When a line is being made invalid, modifying the dirty status is upto the implementation choice.


cache_content_statistics.txt (~/Desktop/COL 216 2020) - gedit
Open
comment
Line-0
.
.
.
Line-7

```
#Data, Tag, Valid-status(valid=1), dirty-status(dirty=1)
1, 1, 0, 1
1, 1, 1, 1
0, 0, 0, 0
3, 3, 1, 1
50, 5, 1, 1
5, 5, 0, 1
7, 7, 0, 1
7, 7, 1, 1
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    = 1
Number of Write Misses = 6
Hit Ratio = 0.4
```