North Carolina State University

Department of Electrical and Computer Engineering

ECE463/563: Fall 2019

Project1B: Cache and Memory Hierarchy Design

Design Specifications

Multi-level Cache & Decoupled Sector Cache

Part B Submission Deadline

October 3, 2019

11:59PM

Ground rules

1. All students must work alone. The project scope is reduced (but still substantial) for ECE 463 students, as detailed in this specification.
2. Sharing of code between students is considered cheating and will receive appropriate action in accordance with University policy. The TAs will scan source code (from current and past semesters) through various tools available to us for detecting cheating. Source code that is flagged by these tools like MOSS will be dealt with severely.
3. A Wolfware message board is provided for posting questions, discussing and debating issues, and making clarifications. It is an essential aspect of the project and communal learning. Students must not abuse the message board, whether inadvertently or intentionally. Above all, never post actual code on the message (unless permitted by the TAs/instructor). When in doubt about posting something of a potentially sensitive nature, email the TAs and instructor to clear the doubt.
4. You must do all your work in the C/C++ or Java languages. Exceptions must be approved. The C language is fine to use, as that is what many students are trained in. Basic C++ extensions to the C language (e.g., classes instead of structs) are encouraged (but by no means required) because it enables more straightforward code reuse.
5. Use of the Grendel environment is required. This is the platform where the TA will compile and test your simulator. Please test your simulator on Grendel machines before submission.

CAUTION: If you develop your simulator on another platform, get it working on that platform, and then try to port it over to Grendel at the last minute, you may encounter major problems. Porting is not as quick and easy as you think unless you are an excellent programmer. Worse, malicious bugs can be hidden until you port the code to a different platform, which is an unpleasant surprise close to the deadline. So, keep this in mind.

Project Overview

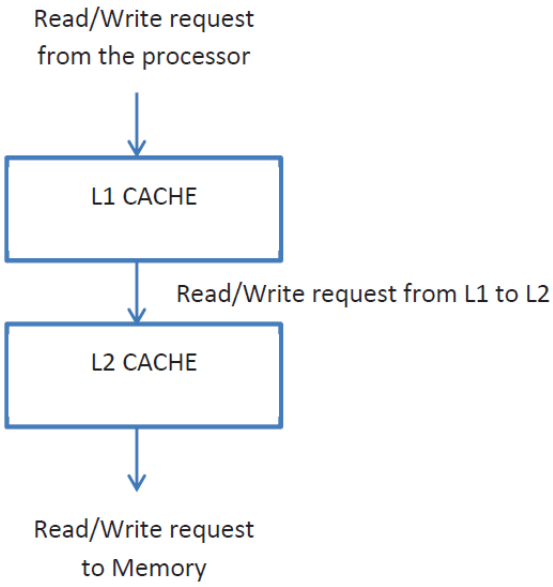
In this project, you will implement a flexible cache and memory hierarchy simulator and use it to study the performance of memory hierarchies using the SPEC benchmarks.

This project is divided into two parts. Both Part A and Part B are to be submitted separately. In Part A, you had designed a generic L1 cache simulator module with some configurable parameters.

This cache module can be instantiated (used) as an L1 cache, an L2 cache, or an L3 cache, and so on. Since it can be used at any level of the memory hierarchy, it will be referred to generically as CACHE throughout this specification. In Part B, you will design a flexible two-level memory hierarchy simulator with certain modifications to the L2 cache using the CACHE module designed in Part- A. These modifications are only for ECE563 students. ECE463 students will have to design a 2-level cache simulator only with basic cache module.

Both simulators will take an input in a standard format which describes the read/write requests from the processor. Simulator output is also expected to be in a standard format as explained in further sections.

1. Part B: Two Level Memory Hierarchy



1.1. Modeling the memory hierarchy

In this part, the simulator will create two instances of the CACHE designed in Part A. The simulator will read the trace file and assign the requests to L1 cache as before. Now, L1 cache will send read/write requests to L2 cache. L2 cache will interact with the memory.

L1 and L2 cache will keep track of their own counters for reads, writes, hits, misses etc. At the end of the simulation, program will print final state of both the caches and raw statistics for both caches.

Both L1 and L2 can be configured in terms of block size (however for this project both L1 and L2 will always have the same blocksize), cache size, associativity, etc. These parameters will be provided from command-line.

1.2. Configurable Parameters

CACHE should be configurable in terms of supporting any cache size, associativity, and block size, specified at the beginning of simulation

- BLOCKSIZE: Number of bytes in a block.
- L1_SIZE: Total bytes of data storage for L1 Cache.
- L1_ASSOC: Associativity of L1 cache (ASSOC=1 is a direct-mapped cache)

- L2_SIZE: Total bytes of data storage for L2 Cache
- L2_ASSOC: Associativity of L2 cache
- L2_DATA_BLOCKS: Number of data blocks in a sector (‘1’ for 463 students)
- L2_ADDR_TAGS: Number of address tags pointing to a sector (‘1’ for 463 students)
- Trace_file: Input Trace file (gcc_trace, etc.)

1.2. Configurable Policies

Apart from the configurable parameters, the CACHE can be configured in terms of policies. Corresponding policies will be specified at the beginning of the simulation.

1.2.1 Replacement Policy

All students (ECE463 and ECE563) need to implement the LRU (Least Recently Used) replacement policy, as discussed in the class.

1.2.2 Write Policy

All students (ECE463 and ECE563) need to implement WBWA (write-back + write-allocate) write policy.

1.2.2.1 Write-Back Write –Allocate (WBWA)

A write updates the corresponding block in CACHE, making the block dirty. It does not update the next level in the memory hierarchy (next level CACHE or the main memory) at that time. If a dirty block is evicted from CACHE, a “Writeback” is performed and the entire block will be sent to the next level in the memory hierarchy. A write that misses in CACHE will cause a new block to be allocated in CACHE. Therefore, both write misses and read misses cause blocks to be allocated in CACHE.

1.3. Modeling the CACHE

This simulator can model various instances of CACHE in a memory hierarchy. For Part A, the simulator will model only a single level memory hierarchy. CACHE receives a read or write request from the higher level (CPU). Only situation where

CACHE must interact with the next level below it (main memory) is when the read or write request misses or writebacks in the CACHE. CACHE always allocates a new block of data when a read request is missed. But a write-miss may or may not cause a new block to be allocated in the CACHE. This depends on the write policy.

Allocation of a new block

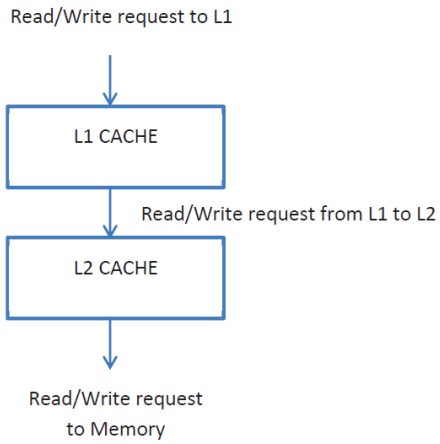
Think of one of the above scenarios in which CACHE needs to allocate a new block X. The allocation of requested block X is actually a two-step process. The two steps must be performed in the following order.

1. *Make space for the requested block X*. If there is at least one invalid (free) block in the set, then there is already space for the requested block X and no further action is required. (Go to step 2). To be correct during the simulation, make sure you initialize the LRU counter for all the data blocks in an index. So, if there are more than one invalid blocks, place the fetched block in a position depending upon the LRU counter and not in any one of them. On the other hand, if all blocks in the set are valid, then a block must be singled out for eviction, according to the replacement policy. For WBWA policy, if this evicted block is dirty, then a write-back of the evicted block must be issued to the next level of the memory hierarchy. For this part of the project, you do not need to do anything in such scenario.

2. *Bring in the requested block X*. Issue a read of the requested block X to the next level of the memory hierarchy and put the requested block X in the appropriate place in the set (determined in step 1).

Allocating a block: Sending requests to next level in the memory hierarchy

Your simulator must be capable of modeling one or more instances of CACHE to form an overall memory hierarchy, as shown in figure below



CACHE receives a read or write request from whatever is above it in the memory hierarchy (either the CPU or another cache). The only situation where CACHE must interact with the next level below it (either another CACHE or main memory) is when the read or write request misses or writeback in CACHE. When the read or write request misses in CACHE, CACHE must “allocate” the requested block so that the read or write can be performed.

To summarize, when allocating a block, CACHE issues a write request (only if there is a victim block and it is dirty) followed by a read request, both to the next level of the memory hierarchy. Note that each of these two requests could themselves miss in the next level of the memory hierarchy (if the next level is another CACHE), causing a cascade of requests in subsequent levels.

2. ECE563 Students: Modify the basic cache module to implement Decoupled Sector Cache

Student enrolled in ECE563 must additionally modify the basic cache module to implement Decoupled Sector Cache.

2.1 Decoupled Sector cache can also work as a basic cache

If the number of address tags and data blocks for a sector are both set to '1', as given for ECE463, then your decoupled sector cache module should work as a basic cache. You would think it for yourself why so after understanding the decoupled sector cache.

2.2 Understanding Decoupled Sector cache

A (N=2, P=8) decoupled sector cache is shown in the diagram below:

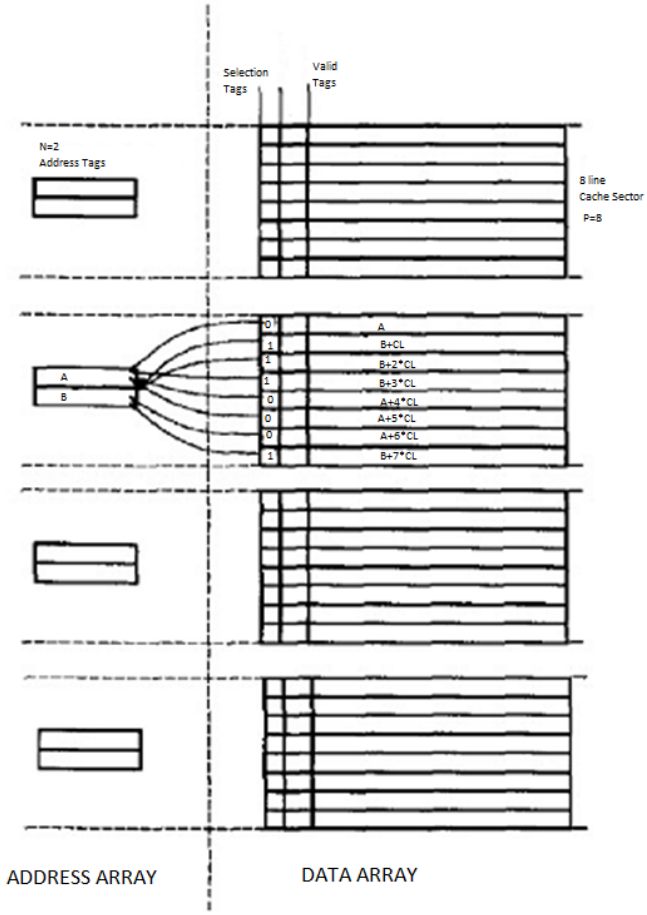


Fig.: (2,8) Decoupled Sector Cache

The above figure shows an example of decoupled sector cache, where the number of address tags in a sector are 2 and the number of data blocks in a sector are 8. In a basic cache module, there is a corresponding address tag for each of the data block. Whereas, in decoupled sector cache, there are N address tags for P data blocks. Please note that a sector can only have contiguous data blocks, each having an offset of CL(Cache line size/Data block size). However, they can belong to different N address tags, stored in the corresponding sector of address array.

Now, if you think in terms of hardware, you would find that the address array and data array are different structures as there is not an one-to-one correspondence. Here in the simulator, there is no data. However, as the selection and valid tags should be there for each of the data block and you need to simulate as it works in hardware, you need to have different structures for both. It is not shown in the picture above, but you also need to maintain dirty bits.

In sector cache, there are two type of misses:

a. Sector Misses: The referenced block is missing and no block in the same memory sector is alive.

   Sector cache would correspond to the misses, which would occur if the line size was the sector size.

b. Cache Block misses: The referenced block is missing, but some other block in that memory sector is alive.

Cache hit would be determined from an address, as shown below:

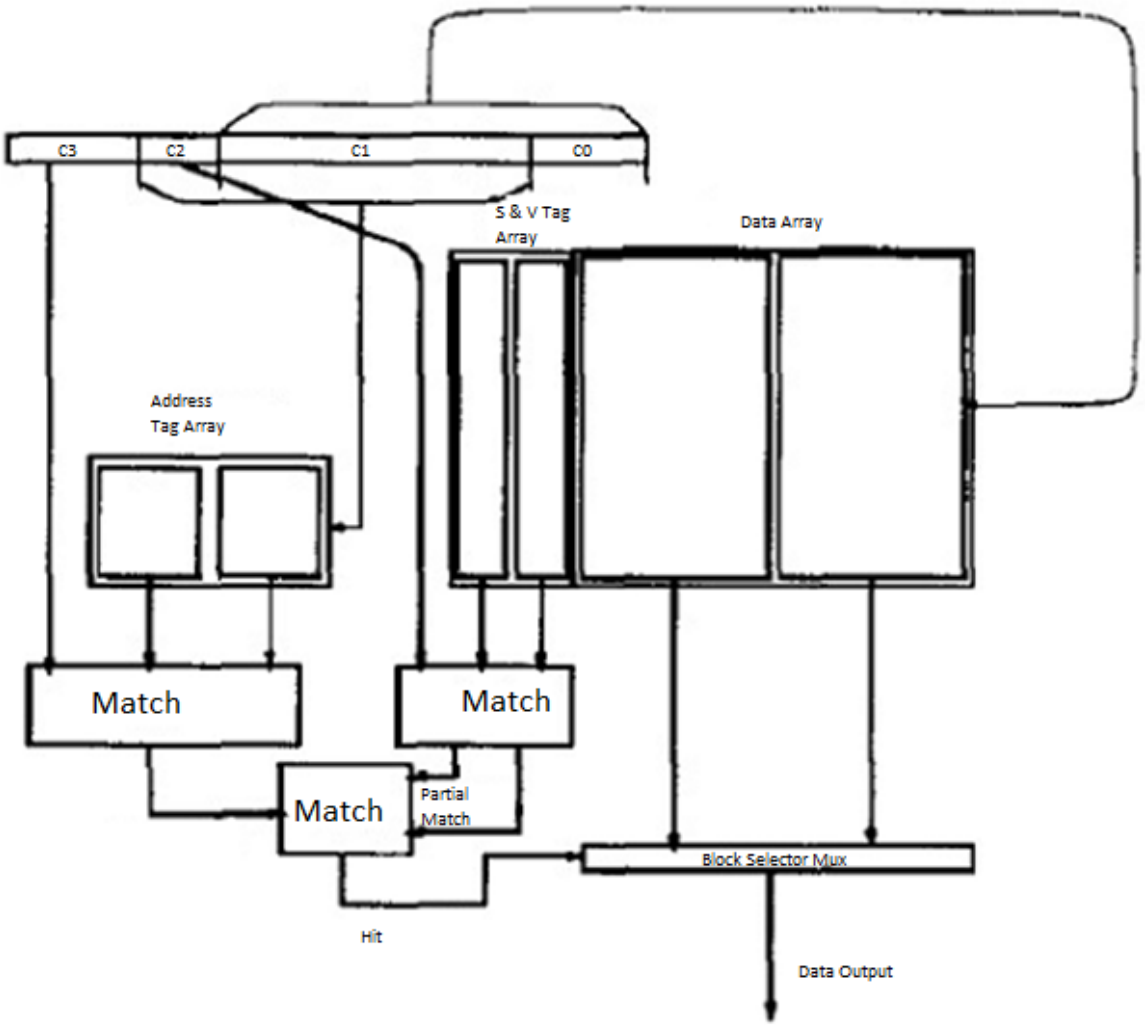


Fig: Determining Cache Hit or Miss

An address is divided into 5 parts, unlike 3 part in a basic cache module,

a. Block offset bits(few of the least significant bits depending upon blocksize) (not shown in the figure above – Above address is after removal of block offset bits)
b. C0 = $\log_2$P number of bits, where P=number of data blocks in a sector
   It would determine which block the address belongs to in a sector.
c. C1 = $\log_2$(CacheSize/(BlockSize*P*Assoc)) number of bits
   It would be your index for the sets in the cache.
d. C2 = $\log_2$N number of bits
   It would determine which of the N address tags corresponds to the address.
e. C3 = (32 – BlockOffsetBits – C0 – C1 – C2) number of bits
   It would be the tag, which would be stored and used for comparison.

As shown in the figure above, a hit condition would be determined if and only if:

1. There exists an address tag(C3) in one of the ways of set with index C1 and position C2 of the address sector.
2. The block at C0 position in the data sector is valid and its selection tag points at the position in the address sector, where its address tag is located.
3. If both these points are satisfied, then it is a hit.

When all the sectors' address tags are filled and you want to introduce another address to a particular sector, then you will have to evict an address and all the corresponding data blocks from that sector. Make sure to make all the respective valid bits to '0'. Once you evict all the data blocks, you can introduce the new address and the corresponding data block, as explained above.

3. Configurations to be test

ECE 463: L1 cache

L1 cache + L2 cache

ECE 563: L1 cache

L1 cache + L2 cache

L1 cache + L2 sector cache

4. Running the simulator

Example:

For 463 students and basic cache for 563 students,

$ ./sim_cache 64 1024 2 65536 8 1 1 gcc_trace.txt

For 563 students,

$ ./sim_cache 64 1024 2 65536 8 8 2 gcc_trace.txt

Your simulator must print outputs to the console (i.e., to the screen). This way, when a TA runs your simulator, he/she can simply redirect the output of your simulator to a file for validating the results.

Validation

Same requirement as Part A also applies for Part B.

5. Printing Instructions

For simplicity, keep it the same with Project 1A.

6. Raw measurements

For Part B you need to gather following statistics from the simulation.

a. number of L1 reads
b. number of L1 read misses
c. number of L1 writes
d. number of L1 write misses
e. L1 miss rate = MRL1 = (L1 read misses + L1 write misses)/(L1 reads + L1 writes)
f. number of L2 read
g. number of L2 read miss
h. number of L2 write
i. number of L2 write miss
j. number of L2 sector misses
k. number of L2 cache block misses
l. L2 miss rate = MRL2 = (from standpoint of stalling the CPU) = L2 write misses / L2 writes
m. number of l2 writeback to main memory
n. total memory access
   (with L2, should match: L2 read misses + L2 write misses + writeback from L2)
   (without L2, should match: L1 read misses + L1 write misses)

The simulator will print out the memory hierarchy configuration, the statistics and the status of both caches at the end of the simulation to output console in a specified format.

8. Notes

From the simulation results, you can analyze the performance of various single level cache memories.

8.1 Average Access Time (AAT)

It's the average time it takes for the memory hierarchy to service a single read/write request from the processor. For memory hierarchy with only L1 and L2 cache, AAT can be computed using following equations.

Average Access Time = HTL1 + (MRL1 *(HTL2+MRL2*Miss PenaltyL2))

Note: HTL1 , HTL2 and Miss PenaltyL2 can be obtained from the course website for this project.

8.2 Simulation Run time

Same requirements as Part A applies.

9. Experiments and Report

You need to submit a combined report for part A and part B. You will perform experiments with your simulator. You will present, analyze, and discuss the results in a written report.

You need to perform following experiments for each benchmark trace. (gcc_trace, perl_trace, go_trace, vortex_trace)

9.1.1 Explore the effect of following parameters on overall performance of cache

Using your simulator, explore the performance of different memory hierarchy design by making graphs for the following parameters:

For ECE 463 and ECE 563 students:

1. L1 Cache size vs. miss rate (For different associativities) [Without L2]
2. Associativity vs. miss rate
3. L2 Cache size vs. miss rate (keep L1 size constant)

For ECE 563 students:

1. Vary size of N address tags in decoupled sector cache vs miss rate (Keep L1 constant, P constant)
2. Vary size of P data blocks in decoupled sector cache vs miss rate (Keep L1 constant, N constant)

9.1.2 Thoroughly explore the design space and discuss noteworthy trends

Using your simulator, explore the memory hierarchy design space and collect result (raw measurements and AAT) for each configuration that you evaluate. Only evaluate configurations that fit within the Area Budget posted on the project website (Area ≤Area Budget). In your written report, include following.

a. Include results for all the configurations that you evaluate, using tables and graphs. Please present results in a way that reveals noteworthy trends as different parameters are varied. Graphs are best for displaying trends.
b. Discuss noteworthy trends in terms of the influence of different parameters on raw measurements and AAT.

9.1.3 Find best memory hierarchy configuration

From your thorough design space exploration, find the Best Memory Hierarchy for AAT (for Area $\leqslant$ Area Budget) for each trace. In your written report, include following.

a. Highlight the configuration that gives the lowest AAT.
b. Closely examine the configuration, its raw measurements, etc., and from this closer examination explain why it gives the lowest AAT compared to all other configurations.

What are unique aspects of this configuration that enable it to achieve the lowest AAT?

9.1.4 Compare and contrast different benchmarks

Are the Best Memory Hierarchies very similar or very different for different benchmarks? Can you conclude anything about the different benchmarks based on their Best Memory Hierarchies (e.g., is anything revealed regarding the degree of compulsory misses, capacity misses, or conflict misses)?

9.1.5 Advantages of Decoupled Sector Cache

What are the advantages of Decoupled Sector Cache?

Does implementing decoupled sector cache give you any benefits over same size basic cache module?

In this project, why we implemented decoupled sector cache over L2 cache and not L1 cache? Give reasons behind your answers.

10. References

The decoupled sector cache implementation is done by referring to the following paper.

> **Decoupled Sectored Caches: conciliating low tag implementation cost and low miss ratio by Andre Seznec**
>
> https://ieeexplore.ieee.org/document/288133

You can study this paper for diving deeper into Sector Cache implementation. You would see various versions in this paper. The one used in this project is the set-associative address and data array. In the paper, the associativity for the address array and data array can be different. However, in this project, we are implementing with same associativity of address and data array to keep it simple!

11. What to submit via Wolfware

You must hand in a single zip file called <unity-id>_project1B that is no more than 1MB in size. Notify the TA beforehand if you have special space requirements. However, a zip file of 1MB should be sufficient for any code.

Your <unity-id>_project1B must contain only the following (any deviation from the following requirements may delay grading your project and may result in point deductions, late penalties,etc.):

1. Source code. You must include the commented source code for the simulator program itself. You may use any number of .c/cpp/.cc/.h files.
2. Makefile.
3. Project Report. Should contain the data, graphs and analysis.

Below is an example showing how to create <unity-id>_project1B from an Eos or Grendel Linux machine. Suppose you have a bunch of source code files (*.cc, *.h), the Makefile, and your project report(report.doc).

```
$ tar  -zcvf  <unity-id>_project1B *.cc *.h Makefile report.docx
```

As TA will unzip and grade your code electronically using scripts your zip file must have all files inside directly (not in a folder inside the zip). So, keep this in mind if you create zip from GUI program in windows.

12. Grading Policy

ECE 463 students are encouraged to challenge the requirements of ECE 563 students, but not mandatory.

Definition of “project works”:

ALL raw measurements match

The final contents of the cache match

There will be no credit for a run if it does not work according to the definition above.

Item	Points(ECE463)	Points(ECE563)

Substantial Simulator can be compiled and run		20	20
L1, L2 cache works	Validation Runs	15	5
	Mystery Runs	15	5
L1 cache, L2 decoupled sector cache works	Validation Runs	+10	10
	Mystery Runs		10
Project Report		20	20
Total		**70 + 10**	**70**

Various deductions (out of 70 points):

-1 point for every 2 hours late, according to Wolfware timestamp.

TIP: Submit whatever you have completed by the deadline just to be safe. If, after the deadline, you want to continue working on the project to get more features working and/or finish experiments, you may submit a second version of your project late. If you choose to do this, the TA will grade both the ontime version and the late version (*late penalty applied to the late version*), and take the maximum score of the two. Hopefully you will complete everything by the deadline, however.

Up to -10 points for not complying with specific procedures. Follow all procedures very carefully to avoid penalties. Complete the **SUBMISSION CHECKLIST** to make sure you have met all requirements.

Cheating: Source code that is flagged by tools available to us will be dealt with according to University Policy. This includes a 0 for the project and other disciplinary actions.

Appendix

Design Guidelines

Here are some guidelines to get you started with design of your CACHE simulator. In this simulator you need to simulate only the **tags** for each cache block, and you don't need to simulate any kind of data-transfer to and from CACHE. You can use C/C++ or JAVA languages for your code. In this guide, use of C/C++ languages is assumed.

The guidelines provided here are just for your reference. It's not mandatory to follow these guidelines in your project. As long as your simulator output matches with the validation runs, you are good.

In your program, CACHE can be represented as a data structure implemented with a C **struct** or a C++ **class**. Use of C++ is not necessary but is recommended as it simplifies the development process. As we are designing a generic CACHE which can be used at any level in memory hierarchy, the implementation of CACHE should be independent of the position in memory hierarchy.

For this, the CACHE data structure needs to be a node in a linked list. Each CACHE will have access to its immediate next level in memory hierarchy. This can be implemented as a pointer to the CACHE data structure inside CACHE. For CACHE just above the main memory, this pointer will simply be NULL.

In this way, your simulator will only give read/write requests to the first level CACHE (just below the CPU) using its member functions. This CACHE will forward the requests to next level if needed using the **nextLevel** pointer.

CACHE will keep track of number of reads, writes, misses, write-backs, memory accesses etc. using some counter variables. These counter variables will be incremented during the simulation as needed.

In a simulator run, it will first initialize the CACHE data structures using command line arguments. Then it will start reading the trace file and issue the read/write requests to the first level (L1 CACHE) and L1 CACHE will increment its counters and updates its tags and if needed, it will issue read/write requests to L2 CACHE (if it exists). Similarly L2 CACHE will increment its counters and updates its tags and so on. At the end of simulation when all requests from the trace file are completed, simulator will read the counter values and display raw measurements and contents of each CACHE.

```
class CACHE
{
private:
/* AddCACHE data members
add variables for parameters like
size, block size, associativity,
write and replacement policies
dynamic Array for tag storage,
all counter variables
and other variables needed
*/

//pointer to the next CACHE in memory hierarchy
CACHE *nextLevel;
public:
//CACHE member functions
bool readFromAddress(unsigned int add);
bool writeToAddress(unsigned int add);
//functions to add more
//functionality in your CACHE
```