

**POLITEHNICA University of Bucharest,
Automatic Control and Computers Faculty
Department of Computer Science and Engineering**

<http://www.csit-sun.pub.ro>

DIGITAL COMPUTERS

Semestrial project – III-rd year

Leading teacher:

Teaching Assistant, Ph. D. Decebal Popescu

Students:

Calangiu Robert Aurelian CB 331

Deaconescu Adrian Răzvan CB 331

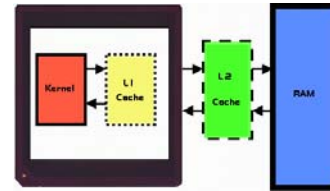
Ioana Marius Andrei CB 331

razvandeaconescu@home.ro



București 2004

Project Name: “Java Cache Simulator”



Abstract

The project consists of a Java visual application simulating a cache memory. The user selects a file containing references to the cache memory. These references are used to simulate data reading or writing to the cache memory or instruction fetch. Using the GUI the user can also choose the type of cache memory he/she wishes to use, the memory size, the mapping type and other options. The simulation results are the miss rate for each type of memory access both in numerical and percentage format.

Table of Contents

1. General Presentation
2. Mapping Methods
 - 2.1. Associative Mapping
 - 2.2. Direct Mapping
 - 2.3. Set Associative Mapping
3. Implementation
4. User Manual
5. Results & Conclusions
6. Future releases
7. Bibliography

1. General Presentation

The Cache Memory is a high speed, but little capacity memory. It is used as a buffer between the processor and the Main Memory. It occupies the second place in the memory hierarchy, after the processor registers. While it is slower than the registers, it has a far greater capacity. It is however faster than the Main Memory and has a little capacity in comparison.

As it can not have a great capacity, because it would occupy important space in the silicon chip, the best mapping methods must be used. Mapping is the process that transfers the Main Memory locations to the Cache Memory. If we use a good mapping method, the processor access time to the Main Memory would decrease visibly.

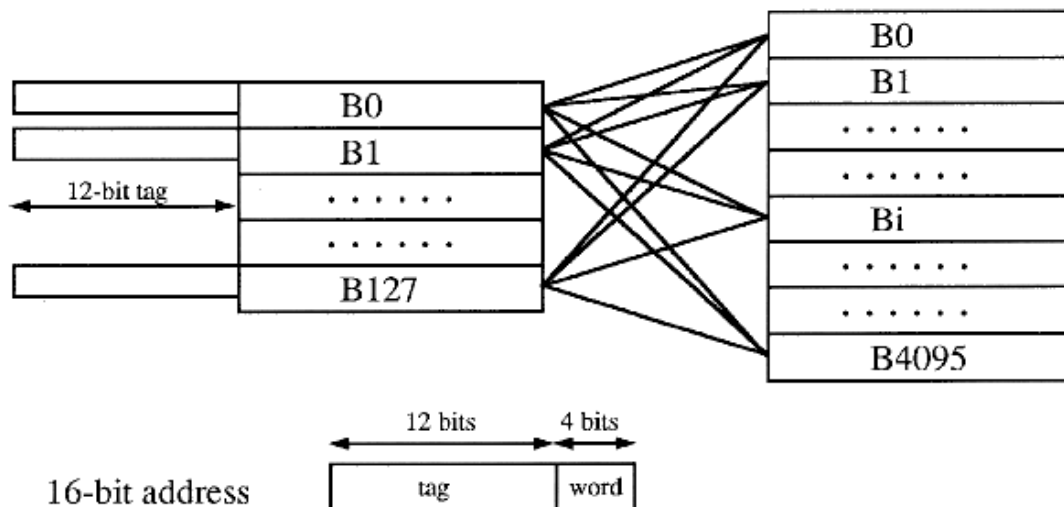
We use the hit arate and miss rate notions for measuring the performance of the Cache Memory and the mapping method that it is used. These notions represent the probability that a write/read request or instruction fetch the processor signals won't need to get all the way to the Main Memory as the Cache Memory already contains those data.

2. Mapping Methods

There are three generally used mapping methods: DirectMapping, Full Associative Mapping, Set Associative Mapping. These represent the methods through which the Main Memory locations are copied to the Cache Memory. For each particular type of mapping we don't normally replace a single memory location, in case of a miss, but a whole block of memory locations. This is done because a memory location that has been used is likely to be reused in the immediate future, as well as the near memory locations (from the same memory block). Generally, if we would create the graphical dependencies between the miss rate and the block dimension we would see that it decreases once we increase the block dimesnion, it would reach a relativ minimum and it would increase again. That shows us that there is an optimal block dimension that would be used for a Cache Memory mapping scheme.

2.1 Associative Mapping

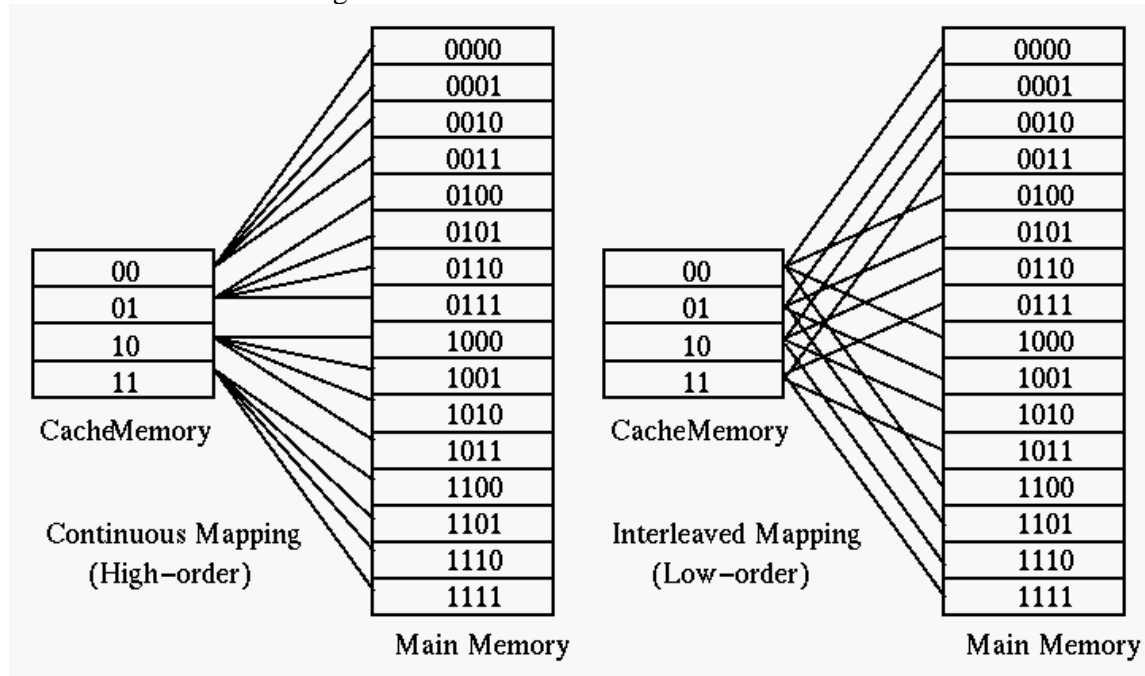
When using associative mapping the Main Memory can be mapped anywhere in the Cache Memory. Its advantage is flexibility, as a Main Memory block can be found anywhere in the Cache Memory. Its disadvantage is the fact that it is slow. In order to verify that a memory location is mapped in the Cache Memory we would have to check all the Cache Memory blocks to find out if it is there.



The above figure represents the graphical representation of an associative mapping method when using a 4096 blocks Main Memory and an 128 blocks Cache Memory.

2.2 Direct Mapping

When using direct mapping multiple blocks of memory can be mapped in the same Cache Memory block. When using a 4096 blocks Main Memory and an 128 blocks Cache Memory, we can map $4096/128 = 32$ Main Memory block in the same Cache Memory block. There are two direct mapping submethods: continuous mapping and interweaved mapping as we can see in the next figure:



When using continuous mapping, 32 consecutive Main Memory blocks (numbered 0 to 31) are mapped in the first Cache Memory block, the next 32 Main Memory blocks (32 to 63) are mapped in the second Cache Memory block, etc.

When using interweaved mapping, however, the first 128 consecutive Main Memory blocks are mapped, respectively, in the 128 Cache Memory blocks, so that in the first block we could have one of the blocks 0, 128, ..., 4064. The second block would contain either of the blocks 1, 129, 257, ..., 4065, and so on.

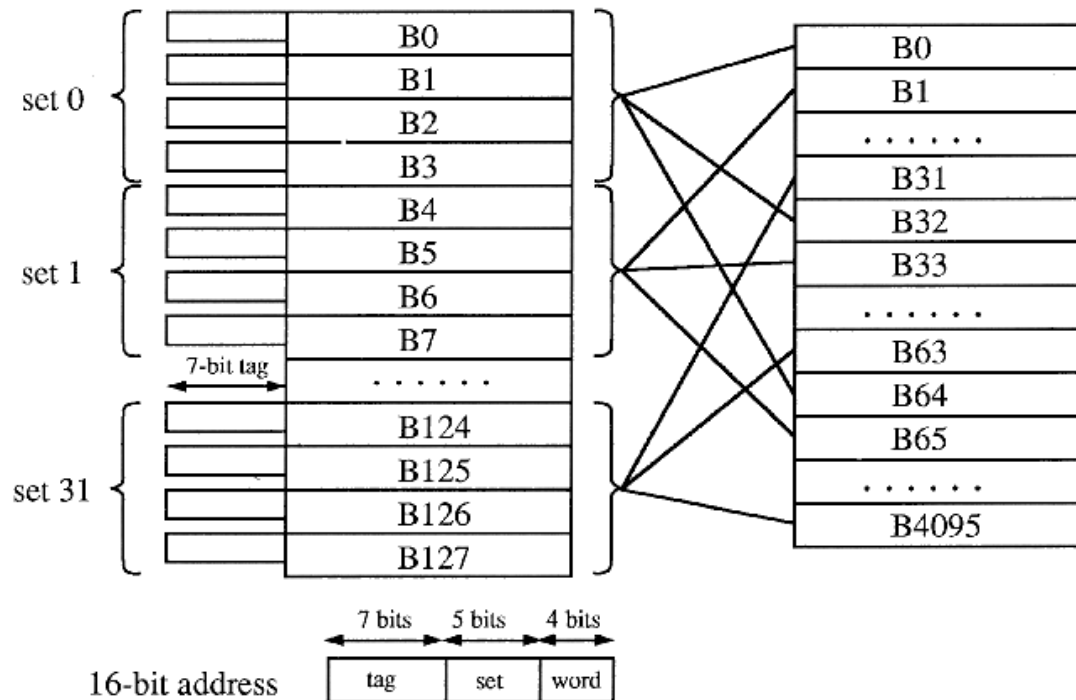
The advantage of using direct mapping is its great speed. Unlike associative mapping, the search for a matching block is very simple because we know precisely where the block can be found and we can see if it is there or not. The disadvantage is its lack of flexibility. For example if two Main Memory blocks are mapped in the same Cache Memory block and are repeatedly used (within a cycle) they constantly keep replacing each other in the Cache Memory Block even if the other blocks may be free.

2.3 Set Associative Mapping

Set associative mapping is a trade off between direct mapping and associative mapping. We “break” the Cache Memory in separate sets. Each set contains a given number of blocks. Thus a number of Main Memory blocks are mapped in other Cache Memory blocks (evidently an inferior number of Cache Memory blocks). The same as the direct mapping, we

can use continuous mapping or interweaved mapping. Direct mapping and associative mapping may be considered special cases of the set associative mapping.

-- Set-associative mapping



3. Implementation

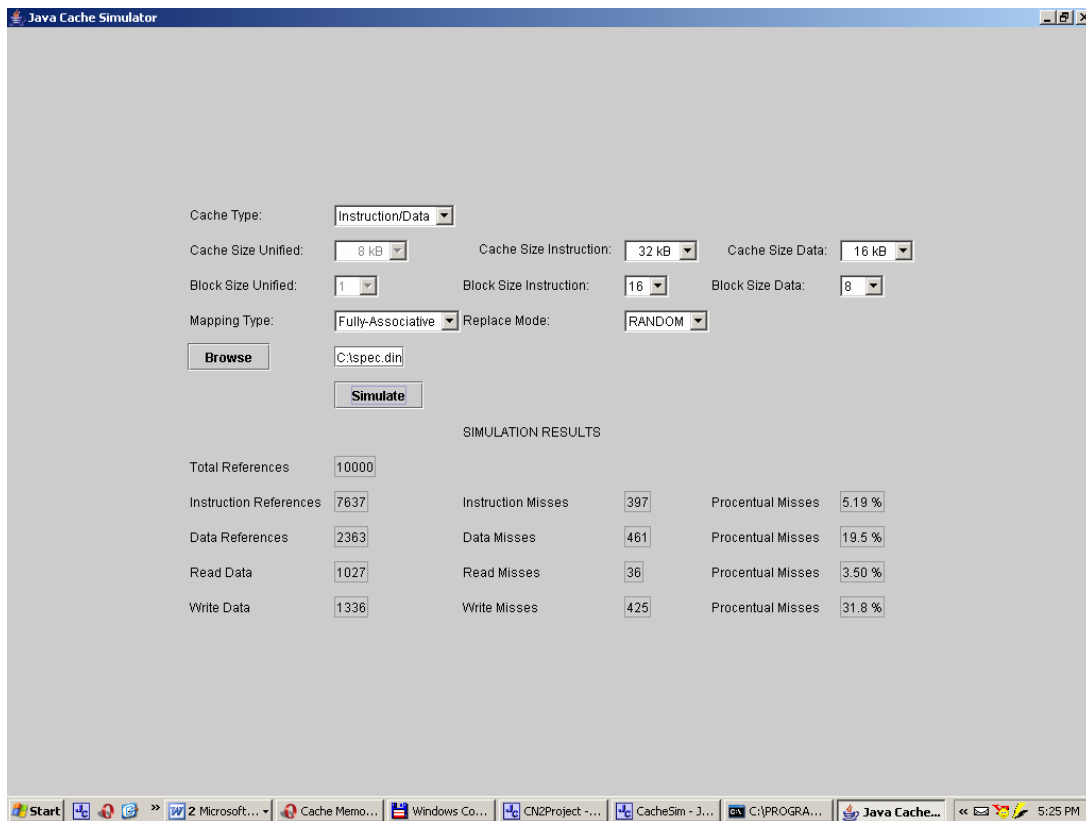
We have chosen the Java programming language for its great portability and reusability. The archive contains three Java files (CacheIO.java, CacheSimulator.java, JavaCacheSimulator.java) each handling a particular aspect of the project.

The CacheIO.java file contains the methods needed for reading the references within the file. The file spec069.din contains 10.000 references to be simulated. The CacheSimulator.java has the classes that basically implement all the different mapping methods and options. It interacts with the classes within the CacheIO.java to get the references from the input file. JavaCacheSimulator.java is the file containing classes for implementing the GUI that allows the user to selected different methods for cache simulation. It interacts with the CacheSimulator.java file to transmit the simulation parameters.

The archive contains a .bat file that can be used to create the .class files and launch the application on a Win32 machine. There is also a jar file that can be also used for launching the application.

4. User manual

When launching the application the user will be prompted with a visual interface that is shown in the next picture. The top of the interface is used to insert data and options and the bottom is used to print out the simulation results.



The top part contains a set of comboboxes, each with the following semnification:

- Cache Type – the Cahe Memory that we use
 - Unified – we use a single memory
 - Instruction/Data – we use two memories: one for reading/writing data and another for instrucion fetch
- Cache Size Unified, Cache Size Data, Cache Size Instruction – select the Cache Memory Capacity
- Block Size Unified, Block Size Data, Block Size Instruction – select the memory block capacity
- Mapping Type – select the mapping method: Full Associative, Direct, Set Associative
- Replace Mode – for the associative mapping the user can choose the desired block replacing method: LRU (Least Recently Used), FIFO (First In First Out) and Random

By using the browse button, the user selects the .din file containing the given references, as those that are sed by the DINERO cache simulator.

In the bottom part of the visual panel the user can find the simulation results by analyzing the input file. The information shown will present the number of references, number of instruction fetches and data, the miss rate both in numerical and procentual format.

5. Results & Conclusions

The archive contains the file spec069.din containing 10,000 references for access that are initiated by the processor towards the Main Memory, that will be used in the simulator. Due to the fact that the references number is quite small (we should use a file containing 1,000,000 references for trully representative conclusions), the user will see that the result won't change when altering the memory size and mapping method (even when using the 8 kB memory). This happens because the cache memory never gets full and the replacement algorithm is thus never used. However we can see clearly that the block dimension has a great impact on the miss rate.

A file that should be used for a representative simulation is presented in the next chapter. It wasn't used in the simulation due to its great dimension.

6. Future releases

In the future releases we hope for a more thorough testing and an expanding of the project. We would like to implement more options and to optimize the simulation algorithms. We would also like to represent the result in a graphical scale and also allow the tracing of certain diagrams that could strengthen the conclusions of the simulation.

7. Bibliography

DINERO man page : <http://www.ece.cmu.edu/~ece548/tools/dinero/man/dinero.htm>
<http://www.ee.duke.edu/~jab/ee252/CacheSimulatorTesting.html> (here you can find a simulation file containing over 7 million references)
<http://www.cs.utexas.edu/users/vin/Classes/CS352-Spring00/CS352/Project/Cache.Project.html>
<http://www.ece.ualberta.ca/~joly/cmpe382/ma2.html>