Cache Memory Simulator Documentation

Project By –
Ansh Arora
Roll No. 2019022
Section A
Group 3

Contents:

Abstract

A cache is a hardware or software component that stores data so that future requests for the data can be served faster. This data stored in the cache can be the result of an earlier computation or a copy of data stored elsewhere.
Most CPUs make use of such caches so that it is able to operate at a faster rate than what the primary memory, the RAM, allows it to. Modern day PCs, such as the Skylake microarchitecture from Intel makes use of 3 levels of cache. The Skylake features three levels of cache L1, L2 and L3 of 64KB, 256KB and 2MB per core.
The functioning of a cache allows the speeding up of processes, since the most used processes are kept in the cache memory, thus keeping in accordance with the property of temporal locality. Another thing that these caches make use of is the fact that they are

divided into multiple cache lines which individually are blocks which house data. This classification into blocks being used as the atomic unit of the cache is in accordance with the property of spatial locality. Thus, the cache serves a very important purpose.

Introduction to the software

The cache simulator that I have developed in Java aims at replicating the basic level functioning of a cache. The simulator features a simple user interface, where it first gives the user a choice on which type of cache functioning the user would like to simulate, giving the 3 options of Direct Mapping, Associative Memory and n-Ways Set Associative Memory. On selecting the primary cache choice, the user is taken into the cache instruction input part of the software, where the user enters instructions on read and write in the given instruction format, and the changes asked for are made in the L1 cache created on the user's request. Option for multiple instruction input and displaying the whole cache have also been included.

Cache Specifications

The Cache Simulator has been coded in JAVA and makes use of Classes for maintenance of Caches.

The cache takes in input data from the user for Cache Size(S) and size of one block(B), and thus deriving the number of cache lines (CL=S/B).
In this documentation, I have taken the running example that the single level cache made is a 16KB cache, with a Block Size of 32 Bytes, thus allowing for 512 cache lines in the single cache.

S = 16KB
CL = 512
B = 32 Bytes

In addition, the n-Way Set Associative Memory Cache has been developed as an 8-Way Cache, thus allowing for 64 sets.

All three kinds of Cache feature all the basic operations of the cache – lookup, read, write, rewrite, evict – functions each suited for their particular class of cache based on the selected Cache access method. The eviction method used is FIFO.

Operation Modes

The first step to operate this simulator is to decide the Size of the Cache and input it in KB, followed by the size of a single Block and input it in Bytes.

Based on the data entered, the simulator quickly calculates the data it needs for the further incoming instructions and shows the details for the 32-bit cache system in a single line neatly.

1) Associative Memory (Fully Associative)

The Fully Associative Cache works on the principal that all bits other than the offset bits of the 32-bit instruction are used as the tag. The tag is searched on every single element of the cache until a Cache Hit is found, otherwise it is considered to be a Cache Miss and the operations asked for proceed as requested by the user.

In the simulator, to go into the Associative Memory Access mode, type down 1 as your choice and press enter. The user is taken into the selected mode and will be now be prompted to enter the instruction.

```
THE CACHE SIMULATOR

32-bit System Cache details -> Size = 16KB  Single Block Size = 32 Bytes  Number of Cache Lines = 512

1. Associative Memory Cache (Fully Associative Cache)
2. Direct Mapping Cache
3. n-Way Set Associative Memory Cache (8-way)
Choice: 1
Fully Associative Cache Simulator
Enter Instruction(<read/write> <32 bit instruction> <data>):
```

The instructions have to be entered in the format-

<read/write> <32bit instruction> <data(if write)>

For example –

Write operation-
write 00000000000000001010100110000000 hello

```
Fully Associative Cache Simulator
Enter Instruction(<read/write> <32 bit instruction> <data>):
write 00000000000000001010100110000000 hello
Cache MISS
New Data Written at CL #0
Do you want to print out whole cache?(Yes=1 No=2):
```

Since no data existed in the cache for the given tag, new data is written after a Cache MISS. This is prompted along with the line number in which data has been written.

Read Operation-
read 00000000000000001010100110000000
We use the operation from the previous example on the cache to read in the stored data from the cache.

```
Do you want to print out whole cache?(Yes=1 No=2): 2
More instructions after this?(Yes=1 No=2): 1
Enter Instruction(<read/write> <32 bit instruction> <data>):
read 00000000000000001010100110000000
Cache HIT
hello
Do you want to print out whole cache?(Yes=1 No=2):
```

Since data with the tag already existed in the cache, we have a cache HIT and the data stored in that particular cache line is printed out.

Other than this, the feature to display out the whole cache has also been given to the user. They can choose to do so by entering 1 for Yes when prompted, at the end of each instruction.

```
Do you want to print out whole cache?(Yes=1 No=2): 1
00000000000000000010101001100 hello
00000000000000000010101011100 CO
00000000000000000010101110000 Ansh
More instructions after this?(Yes=1 No=2):
```

2) Direct Mapping

In this cache access method is based on the LSB 9 bits of the 32-bit instruction bits after removing the offset bits. This index is used to perform operations on the indexed block.

The basic instruction format remains the same as the previous one, with the only change observed in printing out the full cache.

As can be seen here in the example, data was written in cache lines indexed by #3(0b11) and #5(0b101). A read example has also been shown, where the simulator first looks up for the index place in the cache and then checks if the cache matches or not. Based on that, it displays whether there has been a hit or a miss. The whole cache on being printed is 512 lines, which depicts which lines are currently empty while the ones which have been written on show the tag values and the data they store.

```
THE CACHE SIMULATOR

32-bit System Cache details -> Size = 16KB  Single Block Size = 32 Bytes  Number of Cache Lines = 512

1. Associative Memory Cache (Fully Associative Cache)
2. Direct Mapping Cache
3. n-Way Set Associative Memory Cache (8-way)
Choice: 2
Direct Mapping Cache Simulator
Enter Instruction(<read/write> <32 bit instruction> <data>):
write 00000000000000000000000001100000 Hi!
Cache MISS
New Data Written at CL #3
Do you want to print out whole cache?(Yes=1 No=2)
2
More instructions after this?(Yes=1 No=2)
1
Enter Instruction(<read/write> <32 bit instruction> <data>):
write 00000000000000000000000010100000 Hello!
Cache MISS
New Data Written at CL #5
Do you want to print out whole cache?(Yes=1 No=2)
2
More instructions after this?(Yes=1 No=2)
1
Enter Instruction(<read/write> <32 bit instruction> <data>):
read 00000000000000000000000001100000
Cache HIT
Hi!
Do you want to print out whole cache?(Yes=1 No=2)
1
null null
null null
null null
000000000000000000 Hi!
null null
000000000000000000 Hello!
null null
null null
```

3) N-Ways Associative Memory (8-Way)

The N-Way Associative Memory Cache Access approach is an intermediate of both the preceding approaches. Here, the cache has been divided into 512/8=64 sets, and each set holds 8 blocks. The set is accessed using the LSB 6 bits of the 32-bit instruction excluding the offset. After that, there is a search within the set found, where all 8 elements are searched for a matching 21-bit tag. If a match is found, there is a cache MISS, otherwise there is a cache HIT. The instruction syntax remains the same from the last two parts, with changes in how the whole cache is printed out.

```
THE CACHE SIMULATOR

32-bit System Cache details -> Size = 16KB  Single Block Size = 32 Bytes  Number of Cache Lines = 512

1. Associative Memory Cache (Fully Associative Cache)
2. Direct Mapping Cache
3. n-Way Set Associative Memory Cache (8-way)
Choice: 3
n-Way Set Associative Memory Cache Simulator
Enter Instruction(<read/write> <32 bit instruction> <data>):
write 00000000000000000000000011000000 Hello!
Cache MISS
New data written at set #6 way #0
Do you want to print out whole cache?(Yes=1 No=2): 2
More instructions after this?(Yes=1 No=2): 1
Enter Instruction(<read/write> <32 bit instruction> <data>):
write 10000000000000000000000011000000 Hi!
Cache MISS
New data written at set #6 way #1
Do you want to print out whole cache?(Yes=1 No=2): 2
More instructions after this?(Yes=1 No=2): 1
Enter Instruction(<read/write> <32 bit instruction> <data>):
write 00000000000000000000000000000000 Ansh
Cache MISS
New data written at set #0 way #0
Do you want to print out whole cache?(Yes=1 No=2): 2
More instructions after this?(Yes=1 No=2): 1
Enter Instruction(<read/write> <32 bit instruction> <data>):
write 00000000000000000000000000000000 CO
Cache HIT
Do you want to print out whole cache?(Yes=1 No=2): 1
{00000000000000000000 CO , }
{}
{}
{}
{}
{}
{00000000000000000000 Hello! , 10000000000000000000 Hi! , }
{}
{}
{}
```

In the example here, I have written down in two ways within the same set of a cache, and one outside the cache. The UI shows where exactly the new data has been written. Here, I have also shown an example of a rewrite, where a preexisting cache line is hit and then rewritten over. The sets in the Cache here are shown in the form of {}, with the empty ones having no elements within them while the ones with elements have their ways showed as separated by commas.
