NOVA SCHOOL OF SCIENCE & TECHNOLOGY

Arquitetura de Computadores 2021/22

Trabalho Individual 3 (Individual Assignment 2)

Due: May 19, 2022, 23:59

This homework is a continuation of the cache simulation proposed earlier in Ficha08. You can discuss generic problem issues with your colleagues, but the solution and the code writing should be strictly individual. All solutions will be automatically compared against each other and plagiarism cases will be punished in accordance with the regulations.

Your solutions are to be submitted to DI's Mooshak (http://mooshak.di.fct.unl.pt/~mooshak/). You are limited to 10 submissions (more will be ignored!).

You must submit a single file: **`cache.c`**.

1. Simulation of a Set-Associative Cache

Consider the set associative cache (in Portuguese: associativa por grupos) depicted in Fig. 1: the cache has a total of N sets, each with 4 lines; each line is 16 bytes long.

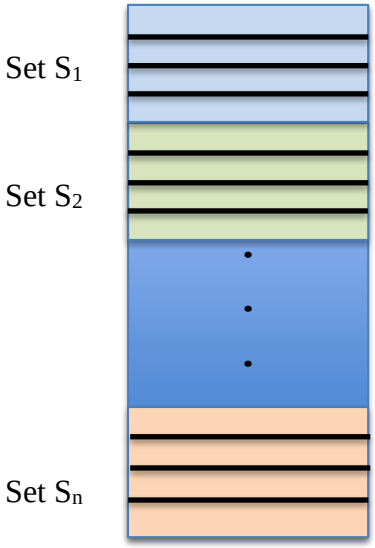

Figure 1. A set-associative cache

Now, you should download the TI3.zip file and "unzip" it: you will find a **`ti3`** directory, and under it, an **`src`** directory and, below it, one more directory, **`caoss`**. In that **`caoss`** directory you will find several files, including the one that interest us, named **`cache.c`**. which, as usual, is incomplete and has some **`TODO`**s here and there.

To understand the implementation, the first important thing is: the number of sets and the number of lines per set are passed as a program argument at runtime - that is, you execute the program, for example, as follows:

```
$ ./caoss examples/small.out 2 4
```

These arguments are handled in the **`cache.c`** code as follows: the first numeric argument (2 in this example) is the *number of sets* that the cache has, and will be stored in the variable **`number_sets`**; the second numeric argument (4 in this example) is the *number of lines per set*, and will be stored in the variable **`number_lines`**. Therefore, the cache being simulated (as a result of the command line used) in the example is a set-associative cache with 2 sets and a total of 8 lines, 4 per each set. As in Ficha08, the string "**`examples/small.out`**" identifies the program that is being used to access the main memory (and, as a consequence, the cache holds copies of the memory).

Another important information is how the cache and the main memory are kept coherent; in this exercise the memory is updated only when the cache line that holds updated (known as dirty) data needs to be evicted, to make room for a new block – this is called the *writeback* cache policy.

2. Implementing the Set-Associative Cache

To implement the set-associative cache, you must complete the code in **`cache.c`** of the following functions (NOTE: this list is **not** complete):

- **`get_block`** e **`get_offset`** – These will be exactly the same as those developed in Ficha08.
- **`get_set`** – given the block number (passed as an argument), compute the number of the set that holds (or will hold) that block.
- **`get_tag`** – compute the tag that will be used to identify the block when stored in a cache line.
- **`find_in_set`** — given the set number and the block tag, find if the block is in one of the set's lines. If found, return the line number, else return -1.
- **`find_in_cache`** — this is a complex function, because it handles many distinct cases; however, the TODOs here are simple :
    - the first TODO addresses the case where we want to store a block in a set, but all the lines are occupied. A victim (line) is selected and, if the line is dirty, we have to copy the line contents to memory.
    - the second TODO is where we load a new block in a line, so we have to fill the line's contents reading from memory, and we have to store the tag, valid, dirty and lru fields.

3. Creating and testing the executable program

To create an executable program "you" (that is, your shell) must be inside the **`ti3`** directory; then, you just need to execute **`make`**. That's all.
Here are some results for different numbers of *sets* and *lines-per-set*:

```
$ ./caoss examples/small.out 1 4
Cache stats:
        number of accesses =       20
        hits               =        8
        conflict misses    =        8
        compulsory misses  =        4
        effective writes   =       10
        hit ratio =  40.0%
```

```
$ ./caoss examples/small.out 2 4
Cache stats:
        number of accesses =       20
        hits               =       10
        conflict misses    =        2
        compulsory misses  =        8
        effective writes   =       10
        hit ratio =  50.0%

$ ./caoss examples/bzip.out 2 4
Cache stats:
        number of accesses =     8194
        hits               =     4345
        conflict misses    =     3841
        compulsory misses  =        8
        effective writes   =     2963
        hit ratio =  53.0%
```