

Assignment 2

Cache Implementation

Dr. Nguyen Hua Phung

March 2021

1 Student Outcomes

After completing this assignment, students will be able to

- explain how a cache works
- apply AVL tree to search in a simple cache

2 Introduction

In a real computer machine, the speed of the main memory is very low in comparison with the speed of its processor. To improve the performance, a fast cache memory is used to reduce the processor's time waiting when accessing instructions and data in main memory. The cache stores copies of the data from the main memory so that future requests for that data can be served faster.

2.1 How it works

When the processor firstly references a memory slot in the main memory, the content of the memory slot together with its address are put into the cache. Subsequently, when the processor references the memory slot again, the desired contents are read directly from the fast cache instead of the slow main memory.

2.2 Cache replacement Policies

As the size of a cache is limited, when the cache is full and a new data is inserted into the cache for future requests, an existing data in the cache must be selected by the cache replacement policy and it is removed to have place for the new data. There are many cache replacement policies, some of which are listed as below:

- FIFO (First In First Out): the oldest data is selected

- LIFO (Last In First Out): the newest data is selected
- LRU (Least Recently Used): the least recently used data is selected
- MSU (Most Recently Used): the most recently used data is selected
- LFU (Least Frequently Used): the least frequently used data is selected
- RR (Random Replacement): a random data is selected

3 Assignment Requirements

3.1 Descriptions

This assignment requires to implement a cache memory for data accessing (read/write).

3.1.1 Cache interface

The assignment must be implemented as a class `Cache` with the following methods:

- **Data* read(int addr):** returns the data stored at the address **addr** if the **addr** is kept in the cache, otherwise, return `NULL`. The type `Data` is described later.
- **Elem* put(int addr, Data* data):** puts **addr** and **data** into the cache and returns the element which is remove out of the cache if any, or `NULL` otherwise.
- **Elem* write(int addr, Data* cont):** searches if the **addr** is in the cache. if it is, this method replaces the existing value associated with the **addr** with the **cont**. If it is not, this method puts the **addr** and **cont** into the cache. This method returns the element removed out of the cache if any, otherwise, returns `NULL`. Note that the **cont** together with its address **addr** are put into the cache, not into the main memory, so the data in the cache now is not the copy of the data in the memory anymore. Therefore, the field **sync** of the element in the cache must be set to false.
- **void print():** prints the value of the elements in the cache in the descending order of the element's living time. Each element must be printed by the method **print** of class `Elem`.
- **void preOrder():** prints the value of the elements in the cache in the preorder of the AVL tree which must be used to search an address in the cache.
- **void inOrder():** prints the value of the elements in the cache in the inorder of the AVL tree

When a new element is put into the **full** cache (in method **put** and **write**), the **FIFO** replacement policy is applied to remove an existing data in the cache. Note that the size of the cache is given in the definition of **MAXSIZE** in file "main.h". To perform a searching in the cache, an AVL tree with the address as a key must be implemented.

3.1.2 Explanation of methods

When the processor would like to read data at address **addr**, it will call method **read(addr)** of class Cache. If the address **addr** together with its content are in the cache, the content is returned. Otherwise, NULL is returned and the processor will read data **cont** from main memory and call **put(addr,cont)** to put item (addr,cont) into the cache. When the processor would like to write data **cont** into address **addr**, it will call method **write(addr,cont)**. This method will modify the content of the address **addr** if any or put a new item into the cache whose **sync** is false. When method **put** or **write** returns an item as the cache is full, the processor will check **sync** of the item. if it is false, the processor will write the data of the item into the main memory.

The other methods of class Cache are just used to check your implementation.

3.1.3 Data Input and Output

The following classes, which are defined in file "main.h", are used as data input and output:

- class Float: used for a float data
- class Int: used for an integer data
- class Bool: used for a boolean data
- class Address: used for an address content
- class Data: used to represent any data
- class Elem: used to represent an element in the cache which includes 3 main fields:
 - **addr**: keeps the address of the memory slot
 - **data**: keeps the content of the memory slot
 - **sync**: shows whether the content in the cache is the same as the content of the corresponding memory slot in the main memory.

3.2 Instructions

To complete this assignment, students must

- Download the initial code, unzip it

- There are 4 files: main.cpp, main.h, Cache.cpp and Cache.h. You MUST NOT modify files main.cpp and main.h.
- Modify files Cache.h and Cache.cpp to implement the cache memory.
- Make sure that there is only one **include** directive in file Cache.h that is `#include "main.h"` and also one **include** in file Cache.cpp that is `#include "Cache.h"`. No more include directive is allowed in these files.

4 Submission

Files Cache.h and Cache.cpp are required to submit as attachments before the deadline is given in the link "Assignment 2 Submission". There are some simple testcases used to check your solution to make sure your solution can be compiled and run. You can submit as many as you like but just the last submission is marked. As the system cannot response too many submissions in the same time, you should submit as soon as possible. You will take your own risk if you submit on the time of deadline. After the deadline, you cannot submit anymore.

5 Plagiarism

You must complete this assignment by yourself and you must prevent your solution from being stolen by someone else. Otherwise, you will be punished by the university rule for plagiarism.