Computer Architecture - CS2323
Lab 7
Report
Cache Simulator

Devraj ES22BTECH11011

Introduction

This lab involved the addition of a Data-Cache implementation on top of the RISCV Simulator of Lab 4. The code of the simulator was modified and added to in order to implement the Cache simulation.

Implementation

The following data structures have been used to simulate the cache:

- A struct data type to form the cache entries.

```
typedef struct entry{
    bool valid;
    bool dirty;
    int tag;
    vector<uint8_t> bytes;
}entry;
```

The vector ***bytes*** contains the data bytes of the memory address.

- A 2-D vector of the entry structures to form the actual cache.
- A 2-D vector of integers to keep track of which block of data to remove in case of a conflict.

When cache is enabled with a configuration file, the above data structures are reinitialized with the appropriate sizes and initial values (0's).

Storing data in the Cache

When cache is enabled, whenever a cache-miss occurs the required data is loaded from the given memory address using a function that return the data block of that memory address in the form of a vector of bytes.
These bytes are then stored into the bytes part of the entry structure at the appropriate index in the cache.

The valid bit for this entry is set to 1. If the miss was due to a write, the dirty bit is also set to 1 in case of the Write-back policy.

In case of a memory access spanning multiple blocks of memory, an error message is thrown.

Write Policies

In case of the Write-through policy, the writes operations are not mediated by the cache and hence, the data blocks are directly written to memory, i.e., the no-write-allocate policy is followed.

But for the Write-back policy, the data is written only to the cache with the dirty bit set to 1, and only when there is a cache-miss and the victim block is dirty will the data of that block be written back to the memory, i.e., the write-allocate policy is followed.

Also, for the Write-back policy, if the cache is invalidated, the dirty entries in the cache are written back to memory and their dirty bit is set to zero.

Replacement Policies

The program supports three replacement policies, namely, ***FIFO***, ***LRU*** and ***RANDOM***.

A 2-D vector of integers containing the indices of the data blocks at each index of the cache is maintained to keep track of the future victim block in case of a cache miss at that index of the cache.

For ***FIFO***, the 2-D vector acts as a vector of queues where the front of the queue is the first one in and hence, will be the next victim.

For ***LRU***, the implementation is the same as that of ***FIFO*** with the addition that in case of a cache-hit, the hit block will be moved to the front of the queue to be the next victim.

For ***RANDOM***, a random number between 0 and ***associativity***-1 is chosen and the block at that position in the hit index of the cache is removed.

Testing

The program was tested on multiple assembly codes involving reads and writes to memory.

The codes tested were:

- the codes provided in the Lab7 problem document
- the codes provided in Homework4 (Cache Experiments)

These codes were tested with different variations of cache properties and code parameters.

***NOTE*:**

- **All parts of the lab assignment are implemented in the code.**
- **Memory is reinitialized to zero every time a new file is loaded. To disable this, go to line 1378 of *simulator.cpp* .**