Lab 7 - Project Report

AI24BTECH11031 - Shivram S
AI23BTECH11022 - Dhadheechi Ravva

Contents

1 Introduction

As part of the CS2323 Computer Architecture course, we had to extend the RISC-V simulator developed in Lab 4 to support cache simulation. This report outlines the design of the cache simulator.

2 Usage

The project includes a `Makefile` which can be used to build the project and test it. To build, run:

```
$ make
```

This produces an executable called `riscv_sim` in the project directory.

The simulator can be run using the following command:

```
$ ./riscv_sim
```

3 Modifications to the Simulator

The `Simulator` struct had to be extended to support the cache simulator. Two fields - `cache` and `cache_config` were added to hold the cache and its configuration.

The `cache_enabled` field indicates whether the cache simulator is enabled. The `execution_in_progress` field indicates if the simulator is currently executing a program - this is necessary to ensure that the cache configuration is not changed during execution.

The implementation of load and store instructions were changed to use two new functions - `mem_read` and `mem_write`. These functions delegate memory access to the cache if it is enabled, otherwise they directly access the memory.

4 Design of The Cache

The simulator's cache consists of several **lines**. Each line consists of several **entries**, and each entry holds a block of memory.

The cache has a **replacement policy** which determines the block to be replaced in case of a conflict, and a **write-back policy**, which tells it how to handle writes.

The Cache struct holds all data related to the simulator's cache

```
typedef struct Cache {
    // Cache configuration
    size_t num_lines, block_size, associativity;
    enum {WRITEBACK, WRITETHROUGH} write_policy;
    enum {FIFO, LRU, RANDOM} replacement_policy;

    size_t hits, misses, writebacks; // Statistics

    size_t monotime; // For maintaining insert and access times
    uint8_t *mem; // Simulator memory
    CacheLine *lines; // The actual cache data
    FILE *output_file; // File for logging accesses
} Cache
```

The number of lines in the cache is given by num_lines, the number of entries in a line is given by associativity, and the number of bytes in a block is given by block_size.

Each cache entry has the format

```
typedef struct CacheEntry {
    int valid, dirty;
    uint64_t tag;
    union { uint64_t insert_time, access_time; };
    uint8_t *data;
} CacheEntry;
```

The valid bit determines if the entry is valid, and the dirty bit indicates that the entry contains changes not written to memory. The insert_time and access_time are used to select the entry to be replaced in the case of FIFO and LRU replacement policies respectively. The tag field holds the higher bits of the block's starting address.

The cache configuration is represented by a struct called CacheConfig, and is loaded from a file. The loading of the configuration is handled by the load_cache_config function.

```
typedef struct CacheConfig {
    size_t size, block_size, associativity,
    writeback_policy, replacement_policy;
} CacheConfig;
```

The logic of the cache itself is implemented in three functions.

- `cache_read` reads a specified number of bytes at an address.
- `cache_write` writes a specified number of bytes to an address.
- `cache_evict` is an internal function that selects blocks to be evicted from the cache.

In addition to this, there are several smaller functions for initializing the cache, to dump the cache to a file, and to invalidate the cache. The cache simulator also logs all memory access to a file.

5 Command Line Interface

All simulator commands are supported. The following cache-specific commands have been added:

- `cache_sim enable <config_file>`: Enables the cache simulator, using the configuration in `output_file`.
- `cache_sim disable`: DIsables the cache simulator.
- `cache_sim status`: Displays the status of the cache, along with the cache configuration (if cache is enabled).
- `cache_sim invalidate`: Invalidates all cache entries
- `cache_sim dump <output_file>` Writes valid cache entries to `output_file`.
- `cache_sim stats`: Displays cache statistics such as hit count, and hit rate.

6 Testing

The project contains several test cases in the `test/` directory. The shell script `test.sh` runs each test case and compares the cache output to the expected output in `expected.output`. In addition to this, the number of hits, misses, and writebacks were compared manually.

7 Future Scope

- Implementing a real-time graphical view of the cache to facilitate profiling of code.
- Adding ability to view the content of specific blocks in the cache.
- Adding more replacement policies.
- Showing the 'dirty' status separately for each word in a block.

8 File Structure

```
+-- Makefile
+-- report.pdf
| \-- main.tex // Source file for the report
+-- src
| +-- asm // Source code for the assembler
| | +-- emitter.c
| | +-- emitter.h
| | +-- lexer.c
| | +-- lexer.h
| | +-- parser.c
| | +-- parser.h
| | +-- tables.c
| | \-- tables.h
| +-- main.c
| +-- simulator.c // Source code for the simulator
| +-- simulator.h
| +-- cache.c // Source code for the cache simulator
| +-- cache.h
+-- test // Testcases
\-- test.sh // Automatic testing script
```