CS2323 - Lab 7 - RISC V Cache Simulator

Shiven Bajpai - AI24BTECH11030

November 11, 2024

Contents

1 Preface **2**

2 Implementation details **2**
2.1 Overview of Changes 2
2.2 Cache Implementation 2

3 Testing **3**

A Appendices **4**
A.1 Build instructions 4
A.2 Guide on how to use the simulator 4
A.3 Ways that this simulator can be improved 5
A.4 Project folder structure 6

1 Preface

The objective of this lab project was to extend the RISC-V simulator made in Lab 4 by adding the ability to simulate caches. Since my simulator already has a GUI, I took the liberty to make a window that allows you to see the cache contents.

The Appendices of these report are just updated versions of those from the previous report

2 Implementation details

2.1 Overview of Changes

For implementation I essentially just created a new `struct Memory` type that encapsulated all the data and made a few accompanying functions to encapsulate all the complexity. This abstraction provided a nice API for the rest of the project that handled everything from reads/writes to cache config parsing and cache dumps. All the code for this is in `memory.c`

The rest of the backend code only got minor changes where I changed memory accesses to use certain functions rather than directly reading/writing to the simulated memory buffer. Besides this there were a plethora of minor changes here and there to connect everything up.

The frontend saw significant additions to add parsing for the all new commands and displaying the new Cache pane which replaces both the registers and the stack or memory (whichever is visible) altogether.

2.2 Cache Implementation

The way the cache itself is implemented is that there is a single buffer in memory made up of multiple "blocks" made up of one byte for flags (Valid and Dirty), 8 bytes for Tag, <block size> bytes that store the data and then 8 more bytes for a timestamp if required by the replacement policy.

The 8 bytes for a Tag may seem wasteful, but implementation wise It was easier to just have a field that could fit a `uint64_t` and store the full address of the first byte of the block there. Any place where the tag needs to be shown, It gets shifted right to be the right size just before being displayed/output.

The `Memory` struct encapsulates a pointer to the main memory buffer, the aforementioned cache buffer, and 3 other structs: `CacheMasks`, `CacheStats`, `CacheConfig`.

`CacheConfig` contains all information about the cache from the config file passed when cache is enabled. It also has some other derived information like the pointer to the trace file, number of cache sets, etc. Everytime the backend is reset, the main control logic passes a new CacheConfig (possibly the same as last time) based on which the backend sets up its cache in memory.

`CacheMasks` Holds some useful precomputed bit masks and offsets that get used in many places in the code.

`CacheStats` Contains all statistical information about the Cache like access count, hit rate, etc.

Every time the memory is accessed, if cache is enabled then the read/write utility functions use the information in the `CacheMasks` and `CacheConfig` to find the data in the cache if present,

Update the cache and memory as necessary, and then updated the `CacheStats` as applicable.

The frontend reads directly from `CacheStats` to display stats.

The cache invalidation and dumping commands are trivial loop operations.

3 Testing

For testing, I used a few handwritten programs to test out the different Replacement Policies and to ensure that all the read and write instructions were working properly, and that write-backs were being handled correctly. Additionally I ran a random subset of the programs from Homework 4 and compared the stats reported by my simulator to those I got from RIPES to further test my statistics reporting.

A Appendices

A.1 Build instructions

To build this project, the development libraries for ncurses need to be installed using the command:

```
sudo apt install libncurses-dev
```

Then build the project by running `make`
The binary is generated in `/bin`

A.2 Guide on how to use the simulator

After running the simulator, the UI loads up in the terminal. After this all operations are done using a set of commands:

`load <filename>`
Attempts to load a file of source code. The path is assumed to be relative unless it is a full path.

`run`
Runs the code from the current line until the end or next breakpoint. Executes about 5 instructions per second. Keyboard Shortcut: F5

`step`
Runs one instruction, Keyboard Shortcut: F8

Tip: Holding down F8 runs the program a lot faster than `run` does

`stop`
Stops program execution, only meaningful after `run`. Keyboard Shortcut: F6

`reset`
Resets the simulator to the state it was in when the program was loaded, but does not remove any breakpoints and avoids recompilation

`regs`
Shows the registers pane, Hiding any other panes that are open in its place.

`show-stack`
Shows the stack-trace pane, Closing any other panes that are open in its place. If the stack-trace is already being shown then it does nothing

`mem <address>`
Shows the memory pane,, Hiding any other panes that are open in its place. The `<count>` argument from the problem statement is not implemented as the ability to scroll on the memory pane makes it redundant.

`break <line>`
Inserts a breakpoint at the specified line number, or removes it if already set. Line number must be as displayed in the code pane.

`cache_sim enable <config_file>`

Loads the configuration of the cache from the specified file. Enables the Cache if the configuration is successfully parsed and validated. Will cause any loaded file to be reset.

`cache_sim disable`
Disables the Cache. Will cause any loaded file to be reset.

`cache_sim invalidate`
Sets the Valid bit of all Cache lines to 0 If cache is enabled. If cache is disabled, returns an error does nothing.

`cache_sim status`
`cache_sim stats`
Shows the Cache pane with the active configuration, Data in cache and stats of cache.

`cache_sim dump <filename>`
Will dump information about current valid cache lines to the specified file.

`exit`
Closes the simulator. Keyboard Shortcut: F1

Always use the `exit` command, trying to exit using Ctrl + C will cause problems as the terminal may get misconfigured and text may not render properly. If this occurs, you have to either restart the terminal or run the `reset` terminal command.

You can scroll in the code, memory, cache and stack panes.

A.3 Ways that this simulator can be improved

There are several ways in which this simulator can be significantly improved, some of them dont even require significant changes. These are changes that I would've made if I had more time:

- Adding something resembling debugging symbols so that the stacktrace can show the arguments of each function call, displayed in the correct type, would make the stack trace a lot more useful.
- Adding pipelining to the simulator would be relatively simple. In code it would simply involve chopping up the step function according to the different stages and then ordering these stages in reverse order. Additionally some stalling/forwarding logic will be required.
- An indicator could be added at the bottom of the memory pane to show the current Stack size and percentage of memory being used
- ELF file parsing - Just to learn how ELF files are structured.
- Making a GUI instead of a TUI - Just to learn how to make GUIs
- Adding the "up-arrow to restore last command" functionality commonly seen in terminals would be very time-saving
- There is a major flaw in that I never implemented horizontal scrolling for the Cache pane due to a lack of time. This should probably be fixed ASAP
- `cache_sim` is a very lengthy prefix. A shorter alias should be added.

A.4 Project folder structure

```
/
 +-- bin
 +-- build
 +-- src
 |   +-- assembler             (files from previous Lab assignment)
 |   |   +-- assembler.c
 |   |   +-- assembler.h
 |   |   +-- index.c
 |   |   +-- index.h
 |   |   +-- translator.c
 |   |   +-- translator.h
 |   |   +-- vec.c
 |   |   +-- vec.h
 |   +-- backend               (implementation of the simulator)
 |   |   +-- backend.c
 |   |   +-- backend.h
 |   |   +-- memory.c          (implementation of cache)
 |   |   +-- memory.h
 |   |   +-- stacktrace.c
 |   |   +-- stacktrace.h
 |   +-- frontend              (ncurses frontend)
 |   |   +-- frontend.c
 |   |   +-- frontend.h
 |   +-- main.c                (main loop, initialization, memory management)
 |   +-- globals.c             (some globals)
 |   +-- globals.h
 +-- report
 |   +-- report.tex
 |   +-- report.pdf
 +-- .gitignore
 +-- LICENSE
 +-- Makefile
 +-- report.pdf
 +-- README.md
```