INSTITUTO SUPERIOR TÉCNICO

Departamento de Engenharia Informática

Computer Organization

LEIC-A, LEIC-T

Second Lab Assignment: TLB Cache Simulator

Version 1.0

2025/2026

1 Introduction

Most modern architectures enforce the programs running on their CPUs to act on their own private address space. Instead of directly accessing physical memory addresses, programs access *virtual* addresses that are, with the cooperative effort of both the hardware and the operating system, translated to physical addresses on the fly. This technique was originally motivated in the late 1950s by the need to provide programs with bigger memory address spaces than the ones made available solely by the physical memory, making use of storage devices as alternative sources of (much slower) memory. However, that motivation has since shifted to providing efficient and safe sharing of memory among programs co-located on the same machine.

Programs use virtual addresses to access memory, and the operating system is responsible for allocating blocks of contiguous memory, or *pages*, to each program. The operating system stores these pages on a data structure called a *Page Table*, which it carefully maintains to keep track of which virtual memory addresses of which programs belong to which pages.

However, this translation process does not come for free. The Page Table kept by the operating system is itself stored in memory. This means that every instruction run by the CPU that acted upon a virtual address would trigger, at best, and before even running the instruction, a memory read operation from the Page Table.

To aid in this effort, modern architectures implement a Translation Lookaside Buffer (or TLB), a small but extremely fast cache responsible for caching translations. This hardware component aims to accelerate the translation process by bypassing the need of reading from the Page Table every time we want to translate a virtual address.

1.1 Objective

The goal of this assignment is to implement a TLB on top of an already developed memory translation simulator. The TLB should be composed of 2 levels of caches, all fully associative with a LRU eviction policy and a write-back write policy.

2 Development Environment

Here is a description of some of the code and assets provided:

`inputs/` Directory containing a list of example inputs that can be provided to the simulator.

`outputs/` Directory containing a list of outputs expected to be generated by the simulator if a certain input is provided. Notice the filenames of outputs maching the corresponding inputs. There are 2 types of output files: `.log` and `.out`. `.log` files contain both the content sent to `stdout` and `stderr`, for debugging purposes. `.out` files contain only `stdout` output. As will be mentioned later, all `stderr` output will be ignored when evaluating the project.

- `outputs/tlbsim-l1` Expected outputs specific to a version of the simulator if one implements only the L1 cache of the TLB.
- `outputs/tlbsim-l2` Expected outputs specific to a complete version of the simulator

`Makefile` Provided Makefile. Should **not** be modified.

`src` Provided source code.

- `src/constants.h` Important constants. Feel free to experiment by modifying the values, but the project will only be tested against the original values. Pay special attention to the constants pertaining to the TLB and the Page Table, as you will likely need them.

`src/log.h` Logging functions. You are expected to only use the `log_dbg` macro, which logs to `stderr` and is therefore ignored for evaluation purposes. You should **not** output to `stdout`, as it will tamper with the expected output.

`src/clock.h` Time tracking functions. You are expected make use of `void increment_time(time_ns_t dt)`, but not the others.

`src/memory.h` Contains both DRAM access and storage access functions. There's no need to concern yourself with these functions, as they are used only internally on the page table.

`src/page_table.h` A Page Table simulator. You are expected to make use of `pa_dram_t page_table_translate(va_t virtual_address, op_t op)` and `void write_back_tlb_entry(va_t virtual_address)`.

`src/tlb.h` The header file for the assignment. You are expected to implement both `pa_dram_t tlb_translate(va_t virtual_address, op_t op)` and `void tlb_invalidate(va_t virtual_page_number)`.

`src/main.c` Main file of the simulator. It expects a path to an instructions file to be provided as an argument. The instructions contains both read and write operations to memory in the format of `R <address>` and `W <address>`. Check the `inputs/` directory for some examples.

3 Assignment

You are expected to implement these functions on the `src/tlb.c` file:

- `pa_dram_t tlb_translate(va_t virtual_address, op_t op)`
- `void tlb_invalidate(va_t virtual_page_number)`

You are also responsible for updating the following variables:

- `uint64_t tlb_l1_hits`
- `uint64_t tlb_l1_misses`
- `uint64_t tlb_l1_invalidations`
- `uint64_t tlb_l2_hits`
- `uint64_t tlb_l2_misses`
- `uint64_t tlb_l2_invalidations`

Again, you should **not** output to `stdout`, as it will tamper with the expected output.

4 Testing the Simulator

As mentioned above, we provide some test inputs and the corresponding expected outputs. Here is how you can run them:

```
$ make
gcc -Wall -Wextra -O3 -c src/clock.c -o build/clock.o
gcc -Wall -Wextra -O3 -c src/main.c -o build/main.o
gcc -Wall -Wextra -O3 -c src/memory.c -o build/memory.o
gcc -Wall -Wextra -O3 -c src/page_table.c -o build/page_table.o
gcc -Wall -Wextra -O3 -c src/tlb.c -o build/tlb.o
gcc -Wall -Wextra -O3 build/clock.o build/main.o build/memory.o build/
    page_table.o build/tlb.o -o build/tlbsim
$ ./build/tlbsim ../inputs/expected_inputs_100.txt
* W 670574be
***** Page fault! *****
[0] W DRAM[0]
PTE found (VA=670574be VPN=67057 PA=14be)
[100] W DRAM[14be]
* W 67057c63
[100] R DRAM[0]
PTE found (VA=67057c63 VPN=67057 PA=1c63)
[200] W DRAM[1c63]
* R 6705748d
[200] R DRAM[0]
PTE found (VA=6705748d VPN=67057 PA=148d)
[300] R DRAM[148d]
[...]
```

You can pipe the `stdout` content to a file and check if it matches the expected output:

```
$ ./build/tlbsim ../inputs/expected_inputs_100.txt > /tmp/expected_inputs_100.
    out
$ diff -y ../outputs/tlbsim-l2/expected_inputs_100.out /tmp/expected_inputs_100
    .out
```

You can also pipe both `stdout` and `stderr` to a file and check if it matches the expected output:

```
$ ./build/tlbsim ../inputs/expected_inputs_100.txt > /tmp/expected_inputs_100.
    log 2>&1
$ diff -y ../outputs/tlbsim-l2/expected_inputs_100.log /tmp/expected_inputs_100
    .log
```

You can also use the `run_tlbsim_l1_tests.sh` to check your solution against the expected output of a solution for the first version of the project with only the L1 cache implemented (or `run_tlbsim_l2_tests.sh` for the complete version). These scripts will generate a `reports` directory, and 3 files per available input inside the `inputs` directory: a `{input}.out` file, `{input}.log`, and `{input}.diff`. While the first two contain the outputs of your current implementation against the specific `{input}` provided, the latter contains a diff against the expected output.

Finally, although we provide some test traces, you are strongly encouraged to design your own tests.

5 Report and Evaluation

Projects will be evaluated solely on the contents of the `tlb.c` file. All other files will be overwritten to match their original contents. Moreover, projects will be evaluated with automatic testing matching on the content sent to `stdout`. All content sent to `stderr` will be ignored.

Students must deliver a 2 page report describing their implementation along with the code. Be prepared to demonstrate your work with test patterns that show your simulator functioning correctly, or to run test patterns provided, or suggested, by the teacher.

Finally, it is strongly recommended to develop the requested tasks in the proposed sequence (first the L1 cache, and only then the L2 TLB cache). Make use of the inputs and outputs provided, testing your implementation before moving on to the next phase.