Programming Assignment 2

TLB Simulator

Due: July 12, 2023

1 Introduction

This project hopes to get you better acquainted to the concept of paging.

2 Given Materials

On Brightspace under **Content/Assignments/Assignment 2** you'll see three files:

- tbl.c
- tlb_project_address_generation.py
- makefile_tlb

`tlb.c` contains code to help get you started on making your TLB simulator. It has a basic parser that reads in the binary input files necessary to test out your simulator. More details on the binary files later.

`tlb_project_address_generation.py` is a handy python script which you can edit to construct your own binary paging tests. You can stick to the tests I provide you in the **Content** folder, but I will be using my own secret tests to check correctness, so it might be a good idea to create your own.

`makefile_tlb` has a basic compilation line which will output the executable binary `tlb`. You should rename the file to `makefile` (Brightspace wouldn't let me upload another makefile to the Assignments folder). You can run the makefile by typing `make` into the directory with both the makefile and c file. You may change the makefile if you so choose to, but **do not** change the executable name; I will expect the binary to be named `tlb` when I go to test your shell.

3 Requirements

You will build a simple TLB simulator with a few replacement algorithms. Assumptions about the system:

- We're using a **16 bit address space**
- There **cannot** be more than 255 TLB entries
- All pages will be **uniform in size**; you do not have to support superpages
- A page **cannot** be greater than 255 bytes
- Main memory is large enough to store the **entirety of physical memory**

You are tasked with determining the number of Page Table Entries, and figuring out how to index into a TLB. Chapters 18 and 19 of *Three Easy Pieces* should prove very useful.

3.1 Inputs

3.1.1 Binary Files

The format of the binary files used for input is as follows:

- The first byte is the page size
- The second byte is the TLB size
- The next (Page Table Entries × 2) bytes are the Physical Page Numbers. The mappings are in sequential order of their Virtual Page Table counterparts.
- Finally, the remaining bytes are the Virtual Addresses to be translated. As it is a 16-bit address space, each address is 2 bytes.

3.1.2 Command line arguments

The format for command line arguments are as follows:

```
./tlb <binary input> <replacement algorithm code> <random seed>
```

If I wanted to run my simulator on a given input with Random replacement I would enter:

```
./tlb 30_small_addr_range 1 20
```

The codes for the three replacement strategies are: 0 for FIFO, 1 for Random, 2 for Corbato's Clock.

3.2 Replacement Algorithms

You will implement 3 replacement algorithms:

- FIFO
- Random
- Corbato's Clock

3.2.1 FIFO (First In First Out)

As simple as it sounds. Eject the oldest page in the TLB.

3.2.2 Random

Select a random number within the TLB entry bounds and choose that line for eviction. **Make sure to input a seed number into the command line to test correctness on subsequent runs.**

3.2.3 Corbato's Clock

You will implement Corbato's Clock algorithm as described in *Three Easy Pieces* (here on p.12) Although the authors specify that the clock hand can start at any arbitrary index, you will make it so the clock hand will always begin at index 0—this makes it so I can check correctness.

3.3 Output

You will output the Virtual and Physical address for every translation performed to the command line using this format

```
printf("VA:%x — PA:%x\n", VIRTUAL_ADDRESS, TRANSLATED_PHYSICAL_ADDRESS);
```

You will also output the number of page faults, TLB hits, and TLB misses to the command line using this format:

```
printf("Page Faults: %d\nTLB hits: %d\nTlb misses: %d\n",
        PAGE_FAULTS,
        TLB_HITS,
        TLB_MISSES);
```

<< Note: You'll find both of these lines commented out in the skeleton code. >>

4 Evaluation

I will not be going out of my way to test every edge-case possible for your simulator, but I will make sure that it functions identically to mine for a handful of tests. I will pipe your output to a file and run a `diff` command with my own. You can do the same with the sample output and sample binaries I provided you on Brightspace to check your own work (All tests with Random replacement use `seed` = 10).

I also expect to see comments throughout your source code. Your comments do not need to be overly detailed, but they should provide a high level explanation as to why certain design choices were made.

If I can't compile your code, you will receive a 0 for the project. Compilation will give you a baseline of 15 points. I have five secret binaries I will be using to test your code; the first of which is a rather simple check and will be worth 10 points. Each subsequent binary will be worth a total of 20 points (meaning if all output is correct you'll receive 5 bonus points for the assignment). Full marks will be given to students who have an identical output to my own, but partial credit will be given to those who don't.

5 Submission

As stated previously, make sure I can run your makefile and that the binary is named `tlb`. Submit your code and makefile in a `tar.gz` zipped file with the following naming format <email_username>_tlb.tar.gz. If I were submitting my file would be named as follows: jraskin3_tlb.tar.gz

(You can compress a file by running `tar -czvf name-of-archive.tar.gz shell.c makefile`)
Do not submit the test binaries! I only want your code and a way to compile it.