COM3610, HW3
TLB / Cache Simulator

1 Logistics

This is an individual project. Your submission will be tested on the COM3610 VM / WSL platform.

2 Overview

This lab will help you understand the impact that the TLB (and caches in general) have on performance. You will write a small C program (about 200-300 lines) that simulates the behavior of a TLB.

3 Downloading the assignment

Download `tlbsim-handout.tar` from Canvas. Start by copying `tlbsim-handout.tar` to a Linux directory in which you plan to do your work. Then give the command

```
linux> tar xvf tlbsim-handout.tar
```

This will create a directory called `tlbsim-handout` that contains a number of files. You will be modifying `csim.c`. To compile, type:

```
linux> make clean
linux> make csim
```

WARNING: Do not let the Windows WinZip program open up your `.tar` file (many Web browsers are set to do this automatically). Instead, save the file to your Linux directory and use the Linux `tar` program to extract the files. In general, for this class you should NEVER use any platform other than Linux to modify your files. Doing so can cause loss of data (and important work!).

4 Description

4.1 Reference Trace Files

The `traces` subdirectory of the handout directory contains a collection of *reference trace files* that we will use to evaluate the correctness of the TLB simulator.

The *operation* field of each operation denotes the type of memory access:
"L": The CPU wants to load data from this VPN (virtual page number).
"S": The CPU wants to store data in this VPN.
"M" The CPU wants to modify data in this VPN (i.e., a data load followed by a data store).

The traces have the following form:

```
 M 0421c7f0,4
 L 04f6b868,8
 S 7ff0005c8,8
```

The format of each M, L, and S line is:

```
[leading space] operation [space] address,size
```

The *address* field specifies a 64-bit VPN. The *size* field specifies the number of bytes accessed by the operation. Since the goal of this assignment is to track hits and misses in the TLB, the size field is not relevant for this assignment.

4.2 Writing the TLB Simulator

You will write a cache simulator in `csim.c` that simulates the hit/miss behavior of the TLB on this trace, and outputs the total number of hits, misses, and evictions.

I have provided you with the binary executable of a *reference cache simulator*, called `csim-ref`, that simulates the behavior of a TLB with arbitrary size and associativity on a trace file. It uses the LRU (least-recently used) replacement policy when choosing which cache line to evict.

The reference simulator takes the following command-line arguments:

```
Usage: ./csim-ref [-hv] -s <s> -E <E> -b <b> -t <tracefile>
```

`-h`: Optional help flag that prints usage info

- `-v`: Optional verbose flag that displays trace info
- `-s <s>`: Number of set index bits ($S = 2^s$ is the number of sets)
- `-E <E>`: Associativity (number of lines per set)
- `-b <b>`: Number of block bits ($B = 2^b$ is the block size)
- `-t <tracefile>`: Name of the trace to replay

The command-line arguments are based on the notation (*s*, *E*, and *b*) from Section 6.4.1 of the CS:APP2e textbook. For example:

```
linux> ./csim-ref -s 4 -E 1 -b 4 -t traces/yi.trace
hits:4 misses:5 evictions:3
```

The same example in verbose mode:

```
linux> ./csim-ref -v -s 4 -E 1 -b 4 -t traces/yi.trace
L 10,1 miss
M 20,1 miss hit
L 22,1 hit
S 18,1 hit
L 110,1 miss eviction
L 210,1 miss eviction M
12,1 miss eviction hit
hits:4 misses:5
evictions:3
```

Your job is to fill in the `csim.c` file so that it takes the same command line arguments and produces the identical output as the reference simulator. Notice that this file is almost completely empty. You'll need to write it from scratch.

Programming Rules

- Your `csim.c` file must compile without warnings in order to receive credit.
- Your simulator must work correctly for arbitrary *s*, *E*, and *b*. This means that you will need to allocate storage for your simulator's data structures using the `malloc` function. Type "man malloc" for information about this function.

To receive credit, you must call the function `printSummary`, with the total number of hits, misses, and evictions, at the end of your `main` function:

```
printSummary(hit_count, miss_count, eviction_count);
```

5 Evaluation

This section describes how your work will be evaluated. The full score for this lab is 34 points:

- Correctness: 27 Points
- Style: 7 Points

I will run your TLB simulator using different cache parameters and traces. There are eight test cases, each worth 3 points, except for the last case, which is worth 6 points:

```
linux> ./csim -s 1 -E 1 -b 1 -t traces/yi2.trace
linux> ./csim -s 4 -E 2 -b 4 -t traces/yi.trace
linux> ./csim -s 2 -E 1 -b 4 -t traces/dave.trace
linux> ./csim -s 2 -E 1 -b 3 -t traces/trans.trace
linux> ./csim -s 2 -E 2 -b 3 -t traces/trans.trace
linux> ./csim -s 2 -E 4 -b 3 -t traces/trans.trace
linux> ./csim -s 5 -E 1 -b 5 -t traces/trans.trace
linux> ./csim -s 5 -E 1 -b 5 -t traces/long.trace
```

You can use the reference simulator `csim-ref` to obtain the correct answer for each of these test cases. During debugging, use the `-v` option for a detailed record of each hit and miss.

For each test case, outputting the correct number of TLB hits, misses and evictions will give you full credit for that test case. Each of your reported number of hits, misses and evictions is worth 1/3 of the credit for that test case. That is, if a particular test case is worth 3 points, and your simulator outputs the correct number of hits and misses, but reports the wrong number of evictions, then you will earn 2 points.

5.1 Evaluation for Style

There are 7 points for coding style. These will be assigned manually by me.

6 Working on the Lab

6.1 Working on Part A

I have provided you with an autograding program, called `test-csim`, that tests the correctness of your submission.

For each test, it shows the number of points you earned, the TLB parameters, the input trace file, and a comparison of the results from your simulator and the reference simulator.

```
linux> make
linux> ./test-
csim
                        Your simulator     Reference simulator

Points (s,E,b)    Hits Misses Evicts      Hits Misses Evicts
 3 (1,1,1)           9       8       6       9       8       6 traces/yi2.trace
 3 (4,2,4)           4       5       2       4       5       2 traces/yi.trace
 3 (2,1,4)           2       3       1       2       3         1 traces/dave.trace
 3 (2,1,3)         167      71      67     167      71      67 traces/trans.trace
 3 (2,2,3)         201      37      29     201      37      29 traces/trans.trace
 3 (2,4,3)         212      26      10     212      26      10 traces/trans.trace
 3 (5,1,5)         231       7       0     231       7       0 traces/trans.trace
 6 (5,1,5)      265189   21775   21743  265189   21775   21743 traces/long.trace
```

Here are some hints and suggestions:

- Do your initial debugging on the small traces, such as `traces/dave.trace`.
- The reference simulator takes an optional `-v` argument that enables verbose output, displaying the hits, misses, and evictions that occur as a result of each TLB lookup. You are not required to implement this feature in your `csim.c` code, but we strongly recommend that you do so. It will help you debug by allowing you to directly compare the behavior of your simulator with the reference simulator on the reference trace files.
- We recommend that you use the `getopt` function to parse your command line arguments. You'll need the following header files:

  ```
  #include <getopt.h>
  #include <stdlib.h>
  #include <unistd.h>
  ```

  See "`man 3 getopt`" for details.

- Each data load (L) or store (S) operation can cause at most one TLB miss. The data modify operation (M) is treated as a load followed by a store to the same address. Thus, an M operation can result in two TLB hits, or a miss and a hit plus a possible eviction.

6.2 Putting it all Together

I have provided you with a ***driver program***, called `./driver.py`, that performs a complete evaluation of your simulator and transpose code. This is the same program that I will use to evaluate your handins. The driver uses `test-csim` to evaluate your simulator. Then it prints a summary of your results and the points you have earned.

To run the driver, type:

```
linux> ./driver.py
```

7 Handing in Your Work

To hand-in this assignment, upload `csim.c` to Canvas.