CSC 322 Systems Programming Spring 2022
Lab Assignment L3: Understanding Cache Memories
Due: Friday, November 11

1 Goal

This lab will help you understand the impact that cache memories can have on the performance of your C programs. You will write a small C program that simulates the behavior of a cache memory. This is an individual project.

2 Introduction

This lab provides you a package `cachelab.tar`, which is in Brightspsace. It has the following files:

- `cachelab.c`: a template for the cache lab. Currently it has a function `printResult`.
- `cachelab.h`: a header file declaring global variables and functions.
- `cachelab_readme.txt`: a readme file explaining special instructions and/or implementation of extra algorithms
- `addressX`: input files with addresses. There are three address files.

Once you download the package, put it in your working directory, and enter the command: `tar xvf cachelab.tar`. Check whether you have these files. This lab works at any host of the department (`pi`, `altair`, or any other hosts). Remember that before you submit, put your ID in the filename so that your files are distinguished from others.

A. How to Start the Cache Lab

First, the simulator **should run on Linux machine at *cs.oswego.edu*** since it will use a specific library which works only on Unix/Linux (and macOS as well). To start your cache simulator, you will enter the following command. The arguments are described in `Table 1`.

```
./cachelab -m <m> -s <s> -e <e> -b <b> -i <file> -r <option>
```

`Table 1. Arguments`

Option	Description
-m	The size of address used in the cache (bit)
-s	The number of index bits ($S = 2^s$ is the number of sets)
-e	The number of line bits ($E = 2^e$ is the number of lines; associativity)
-b	The size of block bits ($B = 2^b$ is the block size)
-i	A set of addresses (file name)
-r	Page Replacement Algorithm – FIFO/Optimal/LRU

For this lab, you could assume that all the arguments are provided and thus you do not have to check any missing arguments. `Figure 1` shows how to build a direct mapped cache which has 64-bit address (thus $m = 64$). There are 16 sets (because s = 4), and the size of block is 16 bytes (thus $b = 4$). Note that direct-mapped cache has only 1 line per set (thus $e = 0$).

```
jwlee@altair~ > gcc cachelab-jwlee.c -o cachelab
jwlee@altair~ > ./cachelab -m 64 -s 4 -e 0 -b 4 -i address01 -r fifo
10 M
. . . (omitted)
```

Figure 1. Start the cache lab

B. How to Run the Cache Lab

Once the simulator starts, it will read an input file, which consists of cache addresses, for simplicity. Note that address is a hexadecimal. The input file name is given "`address#`" `Figure 2` shows the address in the input file. In the package, there are three address files.

```
jwlee@altair~ > cat address01
10
20
22
18
E10
210
12
jwlee@altair~ >
```

Figure 2. Example of *address.txt*

The simulator reads each address and print out whether it is in a cache (*hit*) or not (*miss*), as shown in `Figure 3`. Cache *hit* prints **H**, whereas cache *miss* prints **M**. At the end of the output, it presents statistics: the number of *hits* and *misses*, *miss rate*, and *total running time*. The formula required for total running time will be specified in the next section. In order to print the statistical results, use the declared function, `printResult` in the `cachelab.c`.

```
jwlee@altair~ > ./cachelab -m 64 -s 4 -e 0 -b 4 -i address01 -r lru
10 M
20 M
22 H
18 H
E10 M
210 M
12 M
[result] hits: 2 misses: 5 miss rate: 71% total running time: 507 cycle
jwlee@altair~>
```

Figure 3. Output after running the simulator

When a cache has to be replaced, the simulator uses *LRU* (**L**east-**R**ecently **U**sed) policy. For the *LRU* policy, review the supplementary document.

C. Running Time

Running time is calculated by the following formula:

$$\textbf{\textit{Total Running Time}} = \textbf{\textit{the Number of Code}} \times \textbf{\textit{Average Access Time}}$$

Average Access Time stands for average time spent until we get a code from memory or cache. When the code is in cache, it is said "cache hit" and there is no need to access memory. In this case, the access time is equal to *cache hit time*. When it is not located in cache, it is said "cache miss" and then memory should be accessed to find it in memory. In this case, penalty (we say *miss penalty*) is incurred. In order to formalize Average Access Time, we average the *cache hit time* and *miss penalty* over the *miss rate* which presents how frequently cache miss happens. Therefore, the *Average Access Time* is calculated by the following formula:

$$\textbf{\textit{Average Access Time}} = \textbf{\textit{Cache Hit Time}} + \textbf{\textit{Miss Rate}} \times \textbf{\textit{Miss Penalty}}$$

In In this lab, we assume that the *cache hit time* is 1 cycle, and *miss penalty* is 100 cycles. They are defined in `cachelab.h`, therefore you can use the defined terms: `HIT_TIME` and `MISS_PENALTY` respectively. Consider the sample codes shown in `Figure 3`. Since there are 2 hits and 5 misses, the miss rate is 71%. Note that it is 71.42…% and rounded down for printing by integer representation. Therefore, the Average Access time is obtained:

$$\textbf{\textit{Average Access Time}} = 1 + (0.7142\ldots) * 100 = 72.428\ldots \cong 72$$

Getting all together, the Total Running Time is calculated:

$$\textbf{\textit{Total Running Time}} = 7 \times 72.428\ldots \cong 507$$

D. Page Replacement

In this lab, three page-replacement algorithms will be applied - ***fifo***, ***optimal***, and ***lru*** as shown in `Table 2`.

`Table 2. Options for Argument of Page Replacement`

Option	Description
fifo	Running ***First-in First-out*** algorithm (optional)
optimal	Running ***optimal*** algorithm (optional)
lru	Running ***Least Recently Used*** algorithm (required)

For more detailed, review the supplementary document for the cachelab. The *lru* algorithm is required and should be implemented. Those who want to do extra work may optionally implement other two algorithms – *fifo* and *optimal*, which may give you up to 10 credits totally.

E. Extra Credit Work1: FIFO Algorithm (4 points)

With the option of *fifo*, the cache simulator runs **F**irst-**I**n **F**irst-**O**ut algorithm for page replacement. The supplementary document gives an example of this algorithm. Students those who implement it will get 4 extra points.

F. Extra Credit Work2: Optimal Algorithm (6 points)

With the option of *optimal*, the cache simulator runs *optimal* algorithm for page replacement. The supplementary document gives an example of this algorithm. Students those who implement it will get 6 extra points.

3 Requirements

A. Developing environment

The program should be **implemented in C only**. The program in another language will not be graded. And the program should be executable in CS Linux machine. The program which cannot be compiled or executed in the CS Linux machine will be considered as incomplete program and will lose most points.

B. Handling exceptions

In this lab, we only care one exception – missing options. With a missing option, the cache cannot be built, therefore the simulator must catch this exception. **The program, which fails to detect any missing options, will lose points**.

C. Readability of source code

The source code should be easy to read with **good indentation** and **proper number of comments**.

D. Creating a readme file

The program should be submitted along with a readme file which has information such as your ID, name, etc. It also may provide special instructions to run this program if exists. You may indicate which extra work has been done if you did in the readme file. The source file and readme file should be named following the format below. Before you submit your files, put your login ID so that your file can be distinguished from others.

```
cachlab-your_loginID.c
cachlab-your_loginID_readme.txt
```

E. Submitting by due

The program should be submitted in Brightspace within the three weeks after the project is posted. All submission **by 11:59 PM** on that day will be accepted without any penalty. On the due date, Brightspace may be suffering of too much network traffics and be unstable. There is no excuse about the issue, therefore you are strongly recommended to submit earlier than the due date.

4 Grading

A. Grading criteria

The lab is assigned **30** points, which is worth 15% of the final grade. It will be graded by evaluating the requirement. Any missing and unsatisfiable criteria will take off points. The tentative and brief criteria are below.

- Compilation: **5** points
- Execution: **25** points
- Extra 1(fifo): **4** points
- Extra 2(optimal): **6** points

B. Late penalty

Late submission will take off **10% per week** after due date. **Thus, submission after 10 weeks will not be accepted in any circumstances**.

5 Academic Integrity

Any dishonest behaviors will not be tolerated in this class. Any form of plagiarism and cheating will be dealt with according to the guidelines on the Academic Integrity Policy on-line at http://www.oswego.edu/integrity. For more information about university policies, see the following online catalog at:

http://catalog.oswego.edu/content.php?catoid=2&navoid=47#stat_inte_inte

Student who is against the honor code will not have any credits in this project.