CSCI 424
Computer Architecture
Spring 2021

Homework #4

Due: April 8, 2021 via Blackboard by 11:00pm
Points: 50

Instructions:

a. Upload your version of *cache_424_w.py in* Blackboard. Please do not rename the file.
b. Submit a one-page report (PDF format) consisting of the output of your code and any other comments that you might want to include to clarify your logic and output.

1. **(Warm-up)** Below is a sequence of twelve 32-bit memory address references given as ***word addresses***.

   1, 18, 2, 3, 4, 20, 5, 21, 33, 34, 1, 4

   a. Assuming a direct-mapped cache with one-word blocks and a total size of 16 blocks, list if each reference is a hit or a miss assuming the cache is initially filled with word address 0, 1, 2, …15 memory data. Show the state of the cache after the last reference. Calculate the hit rate for this reference string. **(Correct answer: hit rate = 4/12)**

   b. Now, assuming a direct-mapped cache with two-word blocks and a total size of 8 blocks, list if each reference is a hit or a miss assuming the cache is initially empty. Show the state of the cache after the last reference. Calculate hit rate for this reference string. **(Correct answer: hit rate = 1/12)**

2. (35 pts) We will incrementally build a simple cache simulator in python. The goal is to understand the concepts studied in the class in more depth. We will start by building a simple direct-mapped cache simulator.

Clone the base code from https://github.com/adwaitjog/csci424-s21

There are two folders: Traces and hw4

In hw4, we have two python files: a) *sim_424.py* and *cache_424_w.py*. Read the python files carefully. The *sim_424.py* is the main cache simulator file.

cache_424_w.py is the class file, which contains cache class definition and empty functions. These functions are called appropriately in sim_424.py. You need to implement these functions – there is no need to change anything else in the code. In summary, you need to write the following four python functions.

a. **find_set** function takes an address and returns the set value (Note: for direct-mapped caches, number of ways = 1, and the number of sets = number of blocks in the cache).
b**. find_tag** function takes an address and returns the tag value.
c. **find** function takes an address and returns a bool (false for a cache miss, true for a cache hit). You can reuse **find_set** and **find_tag** functions to implement this **find** function.
d. **load** function takes an address and loads the appropriate data in the cache. Note that load function is only called in sim_424.py when there is cache miss. You can reuse the above implemented functions if you need them.

The documentation related to Traces folder is available in sim_424.py. The Traces folder essentially contain reference strings (second column). These reference strings are byte addresses (i.e., word address *times* 4) in decimal.

Example command line

python sim_424.py test.trace 16 1 8

where test.trace is a trace file stored in ../Traces
16 is the cache size in bytes (always a multiple of 4 as the number of bytes per word is 4)
1 is the number of ways (for direct-mapped caches (this homework), no need to change this parameter)
8 is the number of bytes per block (always a multiple of 4 as number of bytes per word is 4)

3. (15 pts) Verify that the hit rate obtained from the python cache simulator is the same as what you got from the calculation performed in Question 1. The related trace files for Q1 are hw4_a.trace and hw4_b.trace. Note that in hw4_a.trace, the first 16 lines are just there to fill the cache as per Question 1 requirement – do not consider them for the calculation for miss-rates. Reference outputs and some additional simple traces are also available in Github.

Some side notes:

1. The cache size (in bytes) is the product of: number of sets, number of ways, and block size (in bytes).

2. Each memory address (byte or word address) points to some data in the memory. This cache simulator essentially works only on addresses. The actual value of the data is not of interest to us in this cache simulator project.