CSE 471: Computer Design & Organization
Assignment 2
Due: Tuesday, May 5

This assignment is meant to cement your understanding of modern cache subsystem design and the interaction between certain cache configuration parameter values and the effect they have on cache performance.

For this assignment work in teams of two. Work with someone you didn't work with on the last assignment.

Part I: The assignment

Dr. Chip D. Signer has been put in charge of designing a new chip for his employer. Dr. Signer wants to reverse engineer the first-level caches of a rival chipmaker in order to ensure that his chip performs better. He knows that his rival's caches use the LRU replacement policy, and that some of the caches have a victim cache, but that's it. In order to understand the complete design of the caches, Dr. Signer wants to determine:

- their block size (in bytes)
- their total size (in bytes)
- their associativity
- the size of the victim buffer (when there is one)

Your assignment is to come up with and implement a series of algorithms to determine the parameters of any "mystery" cache, which Dr. Signer can use to discover the configurations of his competitor's caches. We recommend following this order of inferring parameters: first block size, then cache size, then associativity/victim buffer size.

You should embed the code for your algorithms within a Python infrastructure, which we will provide. The data structure that is returned by *main()* in this infrastructure is a dictionary (i.e., a hash table) for the values of the cache that is being investigated by your algorithms. You should populate the dictionary with the discovered cache parameter values. When you're done, make sure that those values are there.

Your code should discover a cache's configuration by accessing it and evaluating the resulting behavior. You can access each cache only via the following interface:

- **access(address)** - queries the cache for the byte at the specified address. Returns True if the access is a hit and False for a miss. If a victim buffer is present, True indicates a hit in the cache *or* the victim buffer; False means the access missed in both. Note that this call updates the cache: on a miss the block containing the requested address is brought

into the cache; on a hit the LRU information is updated, the victim buffer (if present) may be updated, etc.

- **reset()** - reset the state of the cache so that all lines (including those of any victim buffer) are empty. The initial state of the cache is the same as after a call to **reset().**

We will be looking over your source code, so don't use any interface to the cache other than the two functions specified above. You can of course supply additional functions to help with debugging, such as printing, but your code will be tested against our version of the solution.

The caches all use a strict LRU policy to manage both lines within sets and entries in the victim buffer. Block sizes are the same for the cache and the victim buffer.

Aim to determine plausible cache and victim cache designs. The cache size (which does *not* count the size of the victim buffer), the associativity and the block size will always be powers of two. The victim buffer size may not always be a power of two, however. We won't test your code on extremely onerous corner cases, such as fully-associative caches with a victim buffer, or caches where the size of the victim buffer is equal to or larger than the cache itself.

Two Python files will help get your going. *Cache.py* contains the definition of the cache class. Feel free to examine its code to remind you how caches work. *discoverCacheParams.py* is the file in which your code should be embedded. Stub methods are already in place to tell you exactly where each section of your code should go. The constants *MAX_** at the top of *discoverCacheParams.py* list the maximum values for various cache parameters. The minimum value for each parameter should be obvious: it doesn't make sense to have 0 associativity, or a block size of 0 bytes, etc. Your algorithms needn't check or handle values outside these ranges.

Our solution added about 50 lines of Python to *discoverCacheParams.py*, and about half of these were for inferring victim buffer size.

Part II: The report

Your report should be a scaled-down version of a normal technical paper. Two pages of text should be enough to explain whatever you need to explain. Just include the following sections: (1) an introduction that simply provides in a nutshell the problem you are trying to solve, and (2) a section that describes at a high-level the algorithm in each of the methods you implement.

Learning to write concisely is one of the goals of an assignment like this. We are looking for a concise description of the techniques you used to discover the cache configuration parameter values. If your report is much longer than the recommended length, we may not read the whole thing.

Part III: The mechanics

Download the Python files you need for this assignment from the course website.

You can run the code for this assignment with the command **python discoverCacheParams.py** in your Windows, Mac or Linux command terminal. We will be using attu for this assignment, just as we did for the Homework 1.

You must fill in the missing function definitions in *discoverCacheParams.py* to infer the parameters of the mystery cache object provided to *main()*. You should submit your modified *discoverCacheParams.py* file to the Catalyst dropbox.

You should ensure that your code works for the different cache configurations provided in the comments at the bottom of *discoverCacheParams.py*. We will test your code on other cache configurations, however, so it's a good idea to come up with your own test cases as well.

We will discuss Python in discussion section, in case you are not familiar with it or need a refresher. Python is a scripting language, whose operations are fairly high-level, such as iterate, take the log, exponentiation, etc. and is therefore simpler than an imperative language, such as C or C++. The infrastructure we've given you is fairly straightforward, with clearly demarked sections (such as "discoverBlockSize"), and descriptive variable names.