ECE 6913A Computer Architecture
Lab 3: Cache Simulator

Due: 12/8/2021 11:55PM

In this lab assignment you will implement a two-level (L1 and L2) cache simulator in C++. The cache simulator will take several parameters describing the cache (block size, associativity, etc.) along with a memory access trace file for an input program.

Cache Design

- Read Miss: on a read miss, the cache issues a read request for the data from the lower level of the cache. Once the data is returned, it is placed in an empty way, if one exists, or data in one of the ways is evicted to create room for the new data.
    - The ways of the cache are numbered from {0,1, 2, …, W-1} for a W-way cache. If an empty way exists, data is placed in lowest numbered empty way.
    - Eviction is performed based on a round-robin policy. Each way has a counter that is initialized to 0, counts to W-1 and loops back to zero. The current value of the counter indicates the Way from which data is to be evicted. The counter is incremented by 1 after an eviction.
- Write Hit: both the L1 and L2 caches are write-back caches.
- Write Miss: both the L1 and L2 caches are write no-allocate caches. On a write miss, the write request is forwarded to the lower level of the cache.
- Inclusive: the L1 and L2 caches are inclusive. This is the design we assumed in class, it means that all data in the L1 is also included in the L2, or that L1's data is a subset of L2. If you're interested, you can read about exclusive and non-inclusive caching design alternatives. They complicate the design but can offer higher performance (please don't implement them in this lab, though!).

Configuration File (config.txt):

The parameters of the L1 and L2 caches are specified in a configuration file. The format of the configuration file is as follows.

- Block size: Specifies the block size for the cache in bytes. This should always be a non-negative power of 2 (i.e., 1, 2, 4, 8, etc.).
- Associativity: Specifies the associativity of the cache. A value of "1" implies a direct-mapped cache, while a "0" value implies the cache is fully associative. Should always be a non-negative power of 2.
- Cache size: Specifies the total size of the cache data array in KiB.

An example config file is provided below. It specifies a 16KiB direct-mapped L1 cache with 8-byte blocks, and a 32KiB 4-way set associative L2 cache with 16-byte blocks

Sample configuration file for L1 and L2 Cache:

```
L1:
8
1
16
L2:
16
4
32
```

The skeleton code provided reads the config file and initializes variables for the L1 and L2 cache parameters.

Trace File (trace.txt):

Your simulator will need to take as input a trace file that will be used to compute the output statistics. The trace file will specify all the data memory accesses that occur in the sample program. Each line in the trace file will specify a new memory reference. Each line in the trace cache will therefore have the following two fields:

- Access Type: A single character indicating whether the access is a read ('R') or a write ('W').
- Address: A 32-bit integer (in unsigned hexidecimal format) specifying the memory address that is being accessed.

Fields on the same line are separated by a single space.

The skeleton code provided reads the trace file one line at a time in order. After each access, your code should emulate the impact of the access on the cache hierarchy.

Simulator Output:

For each cache access, your simulator must output whether the access caused a read or write hit or miss in the L1 and L2 caches, or, in the L2 cache, if it was not accessed. Each event is coded with a number, as shown below.

0: No Access
1: Read Hit
2: Read Miss
3: Write Hit
4: Write Miss

For example, if a read access Misses in the L1 cache but hits in the L2 cache, your simulator would output:

2 1

where the first number corresponds to the L1 cache event and the second to the L2 cache event.

(Note: when there is a read miss in L1 and read hit in L2, the L1 cache might have to evict some data to make room for the data returned by the L2 cache. If the evicted data is dirty, this will result in a write access to L2. If the write access to L2 results in a hit no data will be evicted/replaced. If it results in a miss, the data will be forwarded to main memory without changing the L2 cache state since the L2 cache is a write-no-allocate cache.)

What you need to submit:

- Your source code
- A readme.txt with instructions on how to compile your code.
- Please use the Makefile provided.