CS5222 Advanced Computer Architecture

Assignment 3 – Cache Simulator

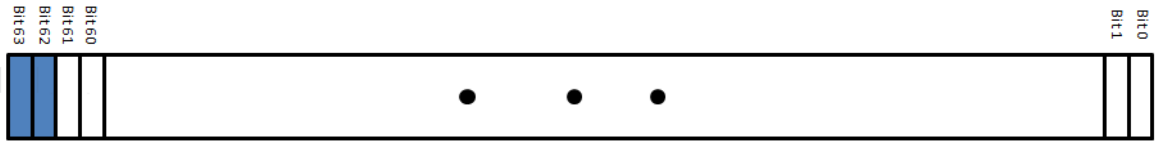
Due Saturday, 31 October 2015, 5pm

This is an individual programming assignment. All standard rules on plagiarism in SoC and NUS apply. The aim of the assignment is to produce a cache simulator that incorporates some of the concepts we have dealt with in class.

First, we need address traces. The following links is for two address traces that I have generated. One is from "ls" and the other is partially from "gzip". Download them. They are binary files that has been compressed by bzip2. So use bunzip2 to recover the binary (which is very big).

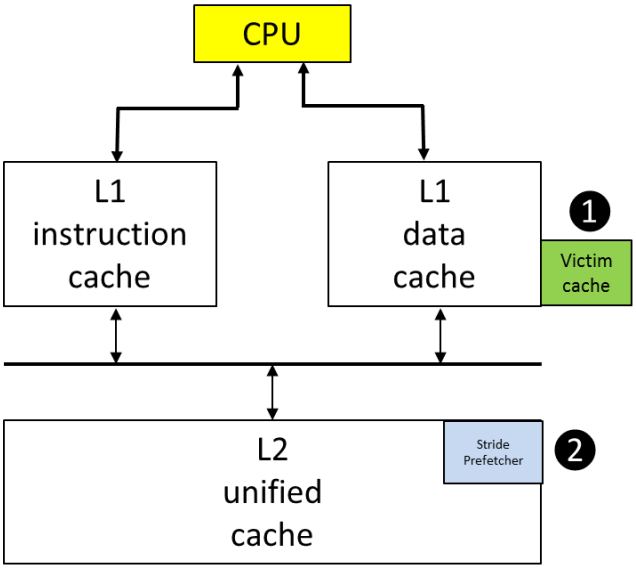
1. https://www.dropbox.com/s/uze1255nwzy8omo/ls.trace.bz2?dl=0
2. https://www.dropbox.com/s/whzh1tsx4u2eai0/gz.trace.bz2?dl=0

Each file is just an array of 64-bit "`unsigned long long`" in C. Just use "`fread`" (in C) and read in the addresses either one at a time or a "chunk", i.e., as an array, at a time. (Do remember to use x86 Unix byte ordering.) Each 64-bit byte-address also encodes information whether it is an instruction fetch, a result of a load instruction or a store instruction. This is how it is encoded (after noticing that the highest 2 bits are always 0):



Bits 62 and 63 encodes the type of reference. If Bit 63 = 0 and Bit 62 = 0, then it is an instruction fetch. If Bit 63 = 0 and Bit 62 = 1, then it is the address of a load instruction. If Bit 63 = 1 and Bit 62 = 0, then it is the address of a store instruction. Bear in mind that as address bits, they are both always 0 – the two bits as encoded are *not* part of the actual address and must be removed before it is used as such. Also, the address is to a byte in (virtual) memory.

Your task is to write a cache simulator that will simulate the following memory hierarchy:



1. The cache parameters are as follows:
   - L1 instruction cache: 32Kbyte[1], 64 byte blocks, 8-way set associative.
   - L1 data cache: 32Kbyte, 64 byte blocks, 8-way set associative, write-back, write allocate.
   - L2 unified cache: 256Kbyte, 64 byte blocks, 8-way set associative, write-through, write allocate.

   Do not that while there is a separate path to the L1 caches, access to L2 is via a single bus. There is therefore a need to implement a simple queue to interleave the requests from both L1 caches to L2.

2. For all caches, implement both (as a compile option) tree based and bit based pseudo-LRU.
3. Implement a 8-block fully associative victim cache for the L1 data cache.
4. Implement the table based stride prefetcher for L2.
5. Please state clearly any assumptions you made.
6. Your simulator should report summary statistics (as detail as possible) of each element in the hierarchy. Basic statistics would include read hit/miss rates, write hit/miss rates, number of compulsory misses, (if possible) conflict and capacity misses. (Think about how to derive the latter.)
7. Run your simulator over the traces I have supplied. In particular, you will get extra credit for putting in counters to try to characterize (i.e., find out interesting behavior of) the traces.
8. Submit your source code as well as a writeup of your implementation in a zip file to the IVLE submission folder for this assignment.
9. Enjoy!

[1] The capacity of the cache refers to the total amount of data bytes that it can hold. This number does not include tags, valid bits, LRU bits and other overhead.