Caches

In this Lab we will implement a Direct-Mapped Cache and a Two-Way Set-Associative Cache using LRU replacement policy.

For this Lab you must complete all the Exercises and Discussion Questions

Coding guidelines: You are only allowed to change the following files: CAU.bsv, DirectMappedCache.bsv, TwoWayCache.bsv and FixedReqTest.bsv. Modifications to other files will be overwritten during didit grading. You should provide answers to the discussion questions in discussion.txt.

1 Direct-Mapped Cache

Lecture 14 introduces how to construct a blocking Direct-Mapped Cache using a Cache-Array Unit (CAU) in Bluespec. The Lecture slides have all the important code components for a direct-map cache which you can assemble into a working cache and connect to a processor.

   Look at CacheTypes.bsv for some cache-related type declarations and useful functions. For example, the cache you will implement has 64 cache lines, therefore the type CacheIndex is a bit vector of 6 bits, and functions getIndex, getTag, getWordOffset help you extract cache-related types from byteAddresses.

Exercise 1 (20 Points): Implement Cache Array Unit (CAU) by filling the skeleton code in CAU.bsv.
Test your design by running make CAUTest; ./CAUTest

Exercise 2 (30 Points—only passes BeverenDirectMap test is 20 points): Implement a blocking direct-mapped cache by filling the skeleton code in DirectMappedCache.bsv using the CAU you implemented.
To test your design with Beveren test, run make BeverenDirectMap; ./BeverenDirectMap.
To compile your design with a Multi-cycle Processor, run make ProcessorDirectMap.
You can run a suite of tests on the processor using the test.sh by running ./test.sh or bash test.sh. You should pass the microtests (option 5) and the fullasmtests (option 6).
2 Two-way Set-Associative Cache

Exercise 3 (30 Points—only passes BeverenTwoWay test is 20 points): Implement a blocking two-way set-associative cache by filling the skeleton code in TwoWayCache.bsv using a Least-Recently-Used (LRU) cache replacement policy.

To test your design with Beveren test, run make BeverenTwoWay; ./BeverenTwoWay.
To compile your design with a Multi-cycle Processor, run make ProcessorTwoWay.
You can run a suite of tests on the processor using the test.sh by running ./test.sh or bash test.sh.
You should pass the microtests (option 5) and the fullasmtests (option 6).

3 Cache Hit and Miss

Exercise 4 (10 points): Implement getHitCount and getMissCount methods in DirectMappedCache.bsv and TwoWayCache.bsv. The getHitCount and getMissCount methods are a part of the Cache interface for getting hit and miss counts from the cache.

```haskell
interface Cache#(numeric type logNumLines);
  // methods for the processor to interact with the cache
  method Action req(MemReq req);
  method ActionValue#(Word) resp();
  // methods for the cache to interact with DRAM
  method ActionValue#(LineReq) lineReq;
  method Action lineResp(Line r);
  // methods for getting the cache hit and miss counts
  method Bit#(32) getHitCount;
  method Bit#(32) getMissCount;
endinterface
```

You should add a cache hit counter and a cache miss counter for each of the Direct-Mapped Cache and the Two-Way Set-Associative Cache. Each counter should be a 32-bit register, and each counter should only be incremented in one rule in your cache.
Test your design by running make HitMissTestDirectMap; ./HitMissTestDirectMap and make HitMissTestTwoWay; ./HitMissTestTwoWay.

Now that we have access to hit and miss counts of our cache, we can examine the cache more.
Fill in the test vectors in FixedReqTest.bsv to produce two tests that show you have a Two-Way Set-Associative Cache and that the replacement policy is LRU respectively. The tests will print the number of cache hits and misses for your cache. Solely from the hit and miss counts, you should be able to deduce the fact that your cache is a 2-way set associative cache using the first test vector and that your cache uses an LRU replacement policy from the second test vector.
Test your designs by running make FixedReqTest; ./FixedReqTest.

Discussion Question 1 (5 points): What is your test vector for testing your cache is 2-way set associative? How does your test of FixedReqTest.bsv shows that your cache is 2-way set associative?

Discussion Question 2 (5 points): What is your test vector for testing your cache uses LRU replacement policy? How does your test of FixedReqTest.bsv shows that your cache is implementing an LRU replacement policy?