Laboratory 7

Objective

The objective of this laboratory is to design and model a write back cache and memory controller for the KURM data path. Your cache should implement a 64-word store and use a direct mapping method to access elements of the cache.

Each cache entry should contain a valid bit, a tag and four 16-bit data values. To access the cache, an address must be split into tag and index values. The index value is used to access an element in the cache array, while the tag value is compared with the tag value stored in the cache entry to determine if the correct value is stored in the cache entry. Because the cache contains words, the low bit of the address can be ignored, as long as memory is being accessed on a word boundary. You should use the cache size to determine how many of the remaining 15 bits are used for the tag and index values.

When a memory read is issued to your cache controller, it should determine immediately whether a hit has occurred and if the cache value is valid. The tag is used to determine if the memory address is contained in the cache. The valid bit indicates if the stored value is correct. Initially set to `0`, the valid bit will be set when a cache line is loaded from main memory or the CPU. If a hit occurs, cache contents should immediately be returned to the CPU. If a hit does not occur, then the CPU should be stalled, main memory accessed, the cache updated appropriately and the valid bit set before providing memory contents to the CPU. Note that, when a cache line is loaded from main memory, it may be necessary to store the existing cache contents prior to updating. Also note that the valid bit is always set after a read from the cache.

When a memory write is issued to your cache controller, it should determine immediately whether a hit has occurred. Again, the tag is used to determine if the right memory address is contained in the cache. If so, the cache is updated with the new value. If not, the cache is updated to reflect the new value after storing the cache contents to memory, if needed. Note that the valid bit is always set after a write to the cache.

Problem 1

Implement a behavioral cache controller using an array of bit vectors to implement the cache storage. The cache controller should be the KURM's only access point to memory. Effectively, address and data information flows from the CPU to the cache controller, which, in turn, generates address and data information for main memory. The main memory controller moves out of the KURM and into the cache controller.

Your cache controller should look to the KURM like a traditional memory, with data in, data out and address ports and read and write strobes to control access. The only addition is the hit signal used to stall the CPU when a cache miss occurs.

Your cache controller should also interface with your memory unit to allow reads due to cache misses and writes due to memory updates. Data in, data out, address and strobe signals should be provided to interface the cache with memory.

Problem 2

Integrate two copies of your cache controller into the KURM by replacing instruction and data memories with cache components. Do not attempt access to the same memory model with both cache controllers. It is sufficient to integrate the controllers into the KURM and continue to access unique instruction and data memories.

Problem 3

Develop a test program for your KURM CPU that tests the functionality of the cache. It should minimally test capabilities for accessing the cache and storing values in the cache, as well as for reading and writing from main memory.

Optional Problem 1

Implement a memory controller that will allow both the instruction and data cache memories to access a common memory module. Your controller should manage access conflicts and support write-back to memory.

Optional Problem 2

Implement a set-associative cache with four words per set.

Optional Homework

The objective of this homework is to evaluate different cache organizations using the *dinero* cache simulator and traces provided. The simulator and traces can be loaded from the web from

http://www.cs.wisc.edu/~markhill/DineroIV/

and installed on your home computer.

1. Set dinero as a unified cache, and use the -b32 and -a1 options. Vary the cache size as 1k, 2k, 4k, 8k, 16k and 32k and plot the number of misses for each of the three traces (cc1, spice, tex).
2. Set i8192 and d8192 and vary the block size as 8, 16, 32 and 64 using the write back (-a1) protocol. Generate a plot showing hit ratio versus block size.
3. Set i8192 and d8192 and vary the associativity as 1, 2, 4 and full using the write back (-a1) protocol. Generate a plot showing the hit ratio versus the associativity.
4. Same as 3 but use write-through.
5. Set i8192, d8192, -b32 and -a4 and vary the replacement policy using LRU, FIFO and random. Generate a plot showing the hit ratio versus the replacement policy.
6. Same as 5 but use full associativity.

Write up your results. Explain how the parameters varied in 1–6 affect overall performance.