Project: Memory architecture in MIPS

Project: Memory architecture in MIPS

Objectives

The purpose of this assignment is to expose you to some of the tradeoffs in cache design and illustrate the effects of cache design on performance. This is a pair programming project; please work with your current partner on this project. **The project is due by 5:00pm on Wednesday, 14 November 2012. No late projects will be accepted.**

Background: Memory system architecture

For this project, we will be simulating the existence of a multilevel cache system using the MARS MIPS simulator. Our simulated system is structured as follows:

- There are only 4 general purpose registers available for storing data: $t0-$t3. **The rest of the registers are off-limits for this project**.

- There is an L1 cache with a size of 4 words. The cache starts at address 0x10010000 and ends (inclusive) at address 0x1001001c. Data is stored in every other word (i.e., at 0x10010004, 0x1001000c, 0x10010014, 0x1001001c). The remaining words store the tag (and any other overhead information) associated with the data. So, for instance, the tag associated with the data stored at 0x10010004 is stored at 0x10010000. (Thus the full cache size is 8 words, but 4 words go to overhead, so the data size is 4 words.) This cache is fully-associative.
- There is an L2 cache with a data size of 16 words and a total size (data + overhead) of 32 words. The cache starts at address 0x10010020 and ends (inclusive) at address 0x1001009c. As with the L1 cache, every other word stores data, and the remaining words store tags and other overhead. This cache is direct-mapped.
- The remaining memory addresses (0x100100a0 through 0x100101fc) correspond to RAM.

We'll assume that the caches use *inclusion*, which means that if something is present in the L1 cache then it's also present in the L2 cache. We'll also assume that writes propagate backward immediately, i.e. when a value in one of the caches is changed, it needs to change at each level cache at which it appears (and in RAM as well). (This is the *write-through* strategy your book discusses.)

Each cache employs the least recently used (LRU) strategy for cache replacement.

For this project, we'll only be concerned with the data cache---you can assume that the instruction cache is present, operating, and optimized.

Your task

Below is a series of scenarios of cache usage. Your task is to design one MIPS program for each task that fulfills that scenario exactly. In other words, your programs should utilize the registers, caches, and RAM exactly in the manner described.

The scenarios are as follows:

1. The miss rate of the L1 cache is less than 10%.
2. Prefetching improves the memory performance of the program by at least 10%.
3. The miss rate of the L2 cache is less than 5%.

Thus you should have 3 separate programs at the end of this project.

Helpful hints

Since we are simulating the caches here, you will have to include code that loads data into each cache at appropriate times. You may use the $s registers for this task.

You will need to calculate some offset version of the RAM memory address to store as the tag associated with a piece of data. For the fully-associative cache, the offset is the base address shifted right by 2 (to convert from byte address to word address). For the direct-mapped cache, things are a bit more complicated. You still need to shift right by 2 to convert from byte address to word address. Because there are 16 total data slots, you need 4 bits to determine the index; these are the lower 4 bits after shifting right. The actual location of the data, however, will be at byte offset 4 from (base address + index * 8). (Confused yet? Draw a picture and you'll see what's happening here.) Thus, here are the algorithms for calculating indices, tags, and storage addresses:

- Index: Shift the RAM memory address right by 2. AND the shifted address with 0x0000000F.
- Tag: Shift the RAM memory address right by 6.
- Actual storage address: Shift the index left by 3 (multiplies by 8). Store the tag at that address. Store the data at that address + 4.

You'll probably want to write some helper functions to do things like compute the index, compute the tag, compute the storage address, determine whether a cache slot is full, etc. since these are tasks you'll repeat often.

I found it easiest to write some code to preload data into the simulated RAM before writing the rest of the code. Here is my code; feel free to borrow and modify it.

```
# Load data into RAM
li $s0, 0x10010080
li $s1, 5
li $s2, 500
LOOP: sw $s1, 0($s0)
      addi $s0, $s0, 4
      addi $s1, $s1, 5
      bne $s1, $s2, LOOP
```

Evaluation

This project is worth 30 points. Each program for each scenario is worth 10 points, and will be evaluated on the following criteria:

- **Cache correctness (3 points).** This means that at each stage in the program, the cache (and registers, and RAM) contain the correct data, tags, and other identifying information.
- **Cache usage (4 points).** This means that the program uses the caches, registers, and RAM correctly as described in the scenario, and that the miss rates, when specified, are met.

- **Cache policy (3 points).** This means that data is placed in the correct slot and replaced at appropriate times. It also means that inclusion and write-through are implemented appropriately.

What to turn in

Turn in all of your MIPS programs in a single ZIP file. Make sure your name and your partner's name (where applicable) appear on all of your program files!

Grading summary

Participants	33
Submitted	15
Needs grading	0
Due date	Wednesday, 14 November 2012, 5:00 PM
Time remaining	Assignment is due

View/grade all submissions