# Homework 9

**Due** May 29 by 11:59pm **Points** 100 **Submitting** a file upload **File Types** tar, taz, gz, zip, txt, and pdf
**Available** after May 24 at 11:59pm

## Caching and Pipelining

Submit online via Canvas.

- For text and discussion, submit in txt or pdf
- For the code, submit as a tar, zip, or tgz in the appropriate source format for the language you choose

In this assignment, you will gain a deeper understanding of caching by exploring and analyzing how it works for real 32-bit X86-family address traces.

## Trace Files

In **this directory (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/hw9/traces/)** , you will find four trace files:

- sample.txt, containing just a few entries. Use this for Q1 below.
- long-trace.txt, with millions of entries. Use this for Q2 below.
- ver1.txt and ver2.txt, each with millions of entries. Use these for Q3 below.

Each trace contains a list of byte-by-byte memory accesses: a ``D'' or ``I'' (indicating ``data'' or ``instruction fetch'') followed by the address in hex. (For this assignment, you're not going to distinguish between D and I.... but since the data was free, I left it there.)

FYI: the big traces above were generated using valgrind to monitor actual C code running on x86, in 32-bit mode.

## Code Samples

You will need to write programs that read these trace files and simulate appropriate caching strategy, and collect statistics. Use a language you understand and which lets you manipulate the 32-bit addresses to extract things like tags and set indices.

In **this directory (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/hw9/code/)** , you will find sample programs in C, Java, and Python for reading data in from these text files. (If you need file IO examples in other languages, let the staff know.)

## Cache Parameters

In the problems below, simulate caches with 64 lines, each of which can hold a 16-byte block.

Where appropriate, use LRU for eviction.

These are unified caches, not spit caches.

Assume that the caches are all initially empty. For purposes of this exercise, consider a ``miss'' to be a memory touch did not find that byte in the cache.

When simulating caches, keep in mind that you don't need to keep track of each byte. Rather, for a cache line, you only need to keep track of whether it's valid and, if so, the tag of the block that lives there.

## Q1: Warm-up (20pts)

On the short sample.txt file, simulate four approaches to caching:

- direct-mapped
- 2-way set-associative
- 4-way set-associative
- fully associative

Do not proceed to the other questions unless your behavior matches **this sample output**---or unless you have a good reason why it doesn't (e.g., we have a bug).

## Q2: Diminishing Returns? (30pts)

As mentioned in class, the conventional wisdom says that although increases in associativity improves performance, the amount of improvement drops off rather quickly.

Using simulation, determine the hit rate and miss rate for the above schemes for the long-trace.txt file (generated from doing quicksort on an array of 512 random integers). Is the conventional wisdom right for this case?

## Q3: Cache-Friendly Code

Let's consider two ways to do a basic array operation. Here's one version:

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

#define HALF  (1 << 8)

main() {

  char *bigarray;
  unsigned int i,j, addr;


  bigarray = malloc(HALF*HALF);

  if (NULL == bigarray) {
    printf("Malloc failed!\n");
    exit(-1);
  } else {
    printf("allocated 0x%x by 0x%x bytes\n", HALF, HALF);
  }

  memset(bigarray,0,HALF*HALF);

  // the versions differ from here down
  printf("looping i, then j\n");

  for (i = 0; i < HALF; i++) {
    for (j = 0; j < HALF; j++) {
      addr = (i*HALF) + j;
      bigarray[addr]++;
    }
  }
  printf("done\n");
}
```

Here's the different part of a second version:

```
    printf("looping j, then i\n");

    for (j = 0; j < HALF; j++) {
      for (i = 0; i < HALF; i++) {
        addr = (i*HALF) + j;
        bigarray[addr]++;
      }
    }


    printf("done\n")
```

ver1.txt and ver2.txt contain address traces for the above programs.

Assume that we're using a 2-way set associative cache and that a cache miss costs 50 times more than a cache hit.

- If we consider the effects of caching alone, which version will be slower, and by what percentage? (Yes, you'll need to simulate here. Show your work :) (15pts)
- What's the reason for this difference? (5pts)

## Q4: Pipelining (30pts)

As discussed and demonstrated in class last week, the pipelined Y86 simulator can be rebuilt to disable all hazard protection.

- cd to where your simulator source is installed (from the **Y86 Tools** page)
- cd down to pipe/
- make he following change to the Makefile:

```
# Modify this line to indicate the default version to build

VERSION=broken
```

- then "make clean; make"
- you'll then have "psim" ready to run.

(There's also a pre-compiled version for OSX that *might* work on your Mac, at **psim-for-osx.zip (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/hw9/psim-for-osx.zip)**. Leave the "pipe.tcl" file in the same directory as "psim-broken")

As shown in last week's slides, invoke the simulator in GUI mode by typing "psim -g foo.yo"

If you build the right simulator, your window should have "pipe-broken.hcl" in its title.

Develop and demonstrate a data hazard and a control hazard for the naively pipelined Y86. (However, do not use ones we discussed in class.)

With annotated screenshots, explain what's happening, and how a properly pipelined Y86 would resolve these hazards.

**Homework 9 rubric**

| Criteria | Ratings | Pts |
|---|---|---|
| Q1. Attempt |  | 10 pts |
| Q1. Clarity, etc. |  | 10 pts |
| Q2. Running the simulations<br>view longer description |  | 10 pts |
| Q2. Analysis and answering the question |  | 20 pts |
| Q3. "Which version is slower?" - a reasonable answer given the data<br>view longer description |  | 10 pts |
| Q3. "Which version is slower?" - correctness<br>view longer description |  | 5 pts |
| Q3. What's the reason for this difference? |  | 5 pts |
| Q4. Control hazard<br>view longer description |  | 15 pts |
| Q4. Data hazard<br>view longer description |  | 15 pts |
|  |  | Total Points: 100 |