ECE/CS 250 – Recitation #12 – Prof. Bletsch
Starting the Cache Simulator project

Objective: In this recitation, you will write some test cases and foundational code for the cache simulator project.

Complete as much of this as you can during recitation. If you run out of time, please complete the rest at home.

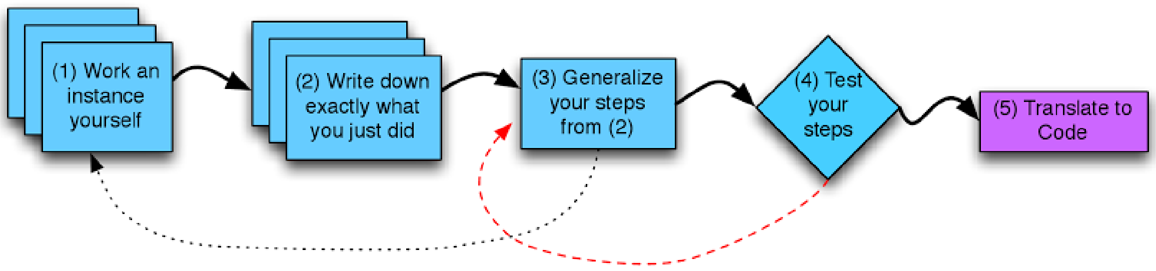
1. Getting the math right

The following steps aren't large; don't overthink them.

(a) How can we use bitshift operations to compute powers of 2? Test your approach in C.
(b) Implement and test the ones() function described in Homework 5.
(c) Write a C function that will give you the lower N bits of a provided 32-bit unsigned integer. Test it.
(d) Write a C function that will get rid of the lower N bits of a provided 32-bit unsigned integer. Test it.
(e) Write a C function that will replace the upper N bits of a 32-bit unsigned integer X with an input Y. Test it.

2. The virt2phys program structure

The virt2phys program isn't meant to be that long or complex, but it could be if you don't plan appropriately. Below is Prof. Hilton's recommended procedure for designing *any* software:


(1) Work an instance yourself
(2) Write down exactly what you just did
(3) Generalize your steps from (2)
(4) Test your steps
(5) Translate to Code

The "Hilton Method" for algorithm design

Do steps 1-4 for virt2phys program. Don't produce any code, just algorithm notes.

3. Generating cachesim tests

Based on the input and output formats described in the Homework 5 writeup, develop a test trace with at least 10 operations. Then, by hand, generate the corresponding expected simulator output for each of the following cache configurations:

- 4kB cache, 2-way, 8-byte blocks
- 4kB cache, 1-way, 8-byte blocks
- 4kB cache, 2-way, 32-byte blocks

Make sure that your trace file will generate hits, compulsory misses, and conflict misses.

4. Input parsing for cachesim

In C, develop code that can parse the trace file format and print the following information for each input line:

- A Boolean indicating if the line is a “store” operation
- The memory address involved
- The number of bytes in the operation
- For stores, the actual bytes being stored (don’t just parrot the hex string – parse it into an array of bytes). Note that you can use `fscanf` with the `%2hhx` specifier to help here.

5. Associativity for cachesim

Develop a data structure and accessor/mutator functions that can associate numeric tags with numeric records. This structure cannot be an array indexed by tag, as the tag could potentially be quite large. However, performance isn’t a huge concern, so this could be an array of tag/value pairs which are linearly searched. This code will come in handy when implementing associativity in your cache simulator. (Note: if this gets problematic, try using the Hilton Method described earlier.)