Cache Lab

Main idea:

In this lab, you will learn how a cache uses the address generated by the CPU to calculate the tag, set, and offset.

Quick Start:

1. Download the file cache-lab.tar
2. Type: `tar -xvf cache-lab.tar`
3. A new directory named cache-lab will be creaded.
4. Type `cd cache-lab`
5. Inside that directory, you will find 3 files:
    a. cache.c this is the main file that you need to fill-in the functions there. We have already written the main() function for you that reads the user inputs and sets some global variables. You need to write the other functions whose declarations appear at the top of the file. Read the rules at the top of that file before you start writing.
    b. cache-orig: This is an executable file that you need to compare your own file with. Type ./cache-orig and press enter to see a help message about how to run the program.
    c. genaddrs: This is another executable that you can use to generate a file with random addresses. This address file can be used as a test for the cache program.

How to compile cache.c ?

```
gcc -std=c99 -o cache cache.c -lm
```

This will generate an executable with the name cache.

What does the program do?

The programs ./cache and ./cache-orig take 3 arguments:

1. total cache size in bytes
2. the associativity
3. the block size in bytes
4. the name of a text file that contains a series of addresses

The output of the programs will be the specifications of the cache (such as size, number of sets, associativity, total number of blocks, the number of bits used for tag, set, and offset). The programs will then read the address file, which is text file that contains randomly generated 64-bit addresses, and for each address will generate the tag, set, and offset.

Try it yourself:

1. Example: type ./genaddrs 10 address.txt
2. This will generate a text file with the name address.txt that contains 10 randomly generated addresses.
3. Type ./cache-orig 65536 1 256 address.txt this command will make the program assume we have a cache with total size 65536 (which is 64KB) with associativity of 1 and a block size of 256, and will generate the specs that we mentioned earlier.
4. After you are done with your program cache.c and build it with the command line mentioned earlier, you will have an executable called cache. If we type the same command as in #3 above but instead of ./cache-orig we use ./cache we must get the same output.

What to submit?

Just cache.c

Enjoy!