1 Activity 1 - Poisoning a Cache

Let’s think about how to absolutely ruin cache performance. Imagine that you have a “friend” who’s written a function, and you want it to \textit{always} have terrible performance. Here’s the function:

```c
int x;
int foo()
{
    return x;
}
```

Pretty simple, right? Let’s think about how to make this function very slow by making sure that every time the function is called, it hits a data-cache miss. Assume that:

- The cache lines are 256 bytes in size.
- There are exactly $4096 = 2^{12}$ cache lines in memory (only 1 line per set).
- You happen to know that the address of $x$ is $0x1234\_{\text{5678}}$.
- Your code (which messes things up) gets called every time, just before $\text{foo()}$ is called.

Write a bit of C or MIPS which will put the cache in a state such that $\text{foo()}$ will be slow. (NOTE: You don’t care if \textit{your} function is slow as well - so long as you reach your goal.)
2 Activity 2 - Snooping

Your friend is getting sneaky! Sometimes, they call \texttt{foo()}, but other times they don’t. Now, devise a way to check \textbf{whether or not} they have recently called \texttt{foo()}. (You don’t have to write code for it, just work out a plan for how you’ll do it.)

For this, you will need a high-resolution clock. Assume that you can call a function \texttt{get\_time\_ns()}, which gets the current clock time (in nanoseconds), as a 32-bit number. It wraps around a lot (once every 4 seconds), but it’s very cheap to access. (Many CPUs have a special-purpose register for this sort of thing; you can read this value in a single instruction.)

2.1 Activity 2(b) - Snooping, with More Limits

I’ll bet that your solution accessed \texttt{x}, right? Now modify your solution so that it doesn’t access that (directly) anymore.
3 Activity 3 - Reading a Secret

The war is escalating. Now your friend has created some secret variables. They’ve set things up so that it’s impossible for you to read it directly. However, they’ve set up a function that you can call at any time:

```c
// you can’t access either of these two variables, but you know
// their *ADDRESSES*
byte secret;
int dataArray[256];

// but you can call this function
int getEntry()
{
    return dataArray[secret];
}
```

Using what you’ve learned about cache timings, see what you can do to figure out the secret. How much detail can you extract (HINT: you can’t figure out the value exactly, just get close.)

3.1 Activity 3(b)

Your friend doesn’t realize that you’ve figured out how to hack into their code! So they’ve agreed to give you an updated function - after all, what can it hurt? Does this one give you more information about the secret?

(HINT: Yes, it does.)

```c
// you can’t access either of these two variables
byte secret;
int dataArrayB[256*256];

// but you can call this function
int getEntryB()
{
    return dataArrayB[secret * 64];
}
```
3.2 Activity 3(c)

One last version. What can you accomplish with this one?

```java
byte dummyArray[...];
int dataArrayB[256*256];

// but you can call this function
int getEntryC(int x)
{
    return dataArrayB[dummyArray[x] * 64];
}
```