Meltdown and Spectre Attacks

H.1. When we read from a memory address multiple times, the second access is usually faster than the first access, what is the reason?

H.2. Regarding the Flush+Reload technique used in the book, why do we use an array of size 256*4096, why not use an array of size 256?

H.3. How do we use CPU cache as a side channel to send out a number 89?

H.4. We have defined an array of size 1024 (bytes). If the CPU cache size is 128 bytes (i.e., when memory address x is accessed, the memory from x to x+127 will be cached by the CPU). Please describe how many distinct values we can send out using this array and the Flush+Reload technique.

H.5. A secret number is stored at the kernel address `0xfb102000`. The following user-level program tries to access and print out this number. What is going to happen?

```
#include <stdio.h>
int main()
{
  char *kernel_data_addr = (char*)0xfb102000;  ①
  char kernel_data = *kernel_data_addr;        ②
  printf("I have reached here.\n");            ③
  return 0;
}
```

H.6. A secret number is stored at the kernel address `0xfb102000`. You are trying to print it out. Please use plain English to describe how you can do this.

H.7. To improve the success rate of the Meltdown attack, it is better that the targeted kernel memory has already been cached by CPU. Why?

H.8. If CPU does not do out-of-order execution, can we still launch the Meltdown attack? What is the downside?

H.9. KAISER can be used to defeat the Meltdown attack. Its main idea is to not map kernel memory in the user space. Explain why this idea works.

H.10. In the Spectre attack, why do we need to train the CPU?

H.11. If CPU does not do out-of-order execution, can we still launch the Spectre attack?

H.12. In the Spectre attack scenario, the protected memory and the attacker program are in the same process, why can't attackers directly access the protected memory?

H.13. When we run the `SpectreAttack.c` program listed in the book, why do we always see `array[0*4096 + DELTA]` in the cache?

H.14. Your program runs in a sandbox, and it is allowed to access the first 100 elements of a protected buffer; the access has to go through the following sandbox API. The address of the buffer is at `0xbfff0200`. There is a secret number stored at `0xbfff0100`, but your program cannot access it due to the sandbox protection. Can you get this secret number? If so, please describe how.

```
int low  = 0;
int high = 100;
int restrictedAccess(int x)
{
  if (x > low && x < high) {
     return buffer[x];
  } else {
     return 0;
  }
}
```