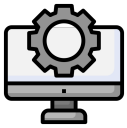
Advanced Computer Architecture



HW1

Prof :

Andreas Moshovos

Student :

Hooman Keshvari

Student Number :

1011293869

Question 1

1. The key challenge is that previous cache side-channel attacks relied on L1 cache which is practically small to probe and also infeaseble due to each core having its own distinguished L1 cache in an environment like the cloud but the last level cache is shared across all cores but it is slower in terms of access time which make previous approaches ineffective.
   The goal is to actually develop an effective side-channel attack across virtual machines on the cloud which relies on the shared last level cache between these virtual machines regardless of special conditions like shared memory or VMM level vulnerabilities.

2. Acording to the **CEASER** paper:

   > the group of lines that map to the same set of the cache and can cause an eviction is called an Eviction Set

   This basically means that eviction set is the collection of memory segments that when brought to the cache, would replace a certain line and throw out whatever is in that line (flush that line of cache)

Question 2

1. First lets see the wanted MIPS code :

```
.data
    x:  .word 12
    array1_size: .word 16
    array1: .space 160
    array2: .space 256 * 512
    y:      .byte 0
    ourx:   .space 200
    i:      .word 0

.text
.globl main

main:
    la $t1, ourx
    lw $t2, i
    add $t1, $t1, $t2
    lw $t2, ($t1)
    sw $t2, x
victim:
    lw $a0, x
    lw $a1, array1_size
    ble $a1, $a0, reset
    ...
reset:
    lw $t0, i
    addi $t0, $t0, 1
    sw $t0, i
    beq $a0, $a0, main
EXIT:
    jr $ra
```

2. We know that bits 11-2 would determin what index in our predictor would be used to see that last time this branch was taken or not. So $bit[(PC\&0xFFC) >> 2]$ actually determins what instructions would be fetched next when a branch instruction is seen. So if we have our first x as something less than 16 (like 2) then the branch would be taken(executes the victim code) and since we have a **last-outcome predictor** and the previous branch was taken, predictor would assume we will also take it this time and if we choose our second x as something out of bound(256), then $array1[256]$ could potentially be fetched and cached. So $ourx = [2, 256]$

3. So with a biomodal predictor, we actually use 2 bits to determin wether a branch has been taken or not. If we assume initially all entries are initialized to weakly taken, then 10 would be the initial value of the branch predictor for that branch. Now, if our next x is 256, then it would predict taken because it has the initial state weakly taken (although it is not taken) and would fetch array1[256]. So $ourx = [256]$

4. The global history register has 4 entries which means we could have 16 states for the GHT (NNNN - NNNT - ... - TTTT) which look at the previous 4 branches. Now we assume that we are in a state of NNNN at first and the predictor for every state of the GHT of every branch is weakly not taken(01). To make this analysis easier, we name the **while loop** branch A and **if statement** branch B Now first we enter A and take the branch which transitions our state into NNNT and A[NNNN] = 10(weakly taken). Then we give x=2 as an input to B which would cause a transition in GHT to NNTT and B[NNNT] = 10(weakly taken). Afterwards we go back to A and take it which transitions GHT to NTTT and A[NNTT]=10(weakly taken) and after that we give x again as 2 and this would cause to take the branch and GHT becomes TTTT and B[NTTT]=10(weakly taken) and we loop back to A but since this time we see TTTT, we just change A[TTTT]=10(weakly taken) and then we input 2 as x again so that B[TTTT]=10(weakly taken). Now A would be taken again and A[TTTT]=11(strongly taken) and now we know that B[TTTT]=10(weakly taken) so the branch would be predicted as taken and we could input 256 as x and access the out-of-bound data. So $ourx = [2, 2, 2, 256]$
   ****Note**:** Of course the first state of the GHT and A[GHT] and B[GHT] would determin ourx but the worst case is that we start in NNNN and every entry for B[GHT] and A[GHT] is 00(strongly not taken), in this case if we use $ourx = [2, 2, 2, 2, 256]$ then in the first 2 itterations GHT would turn from NNNN to TTTT (A and B would be taken 2 times each) and then in the third itteration, B[TTTT]=01(weakly not taken) and in the 4th itterations B[TTTT]=10(weakly taken) and in the fifth itteration it would be taken so we input 256.