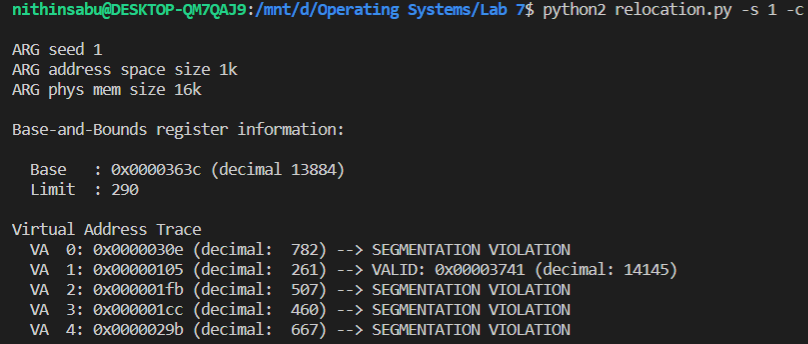
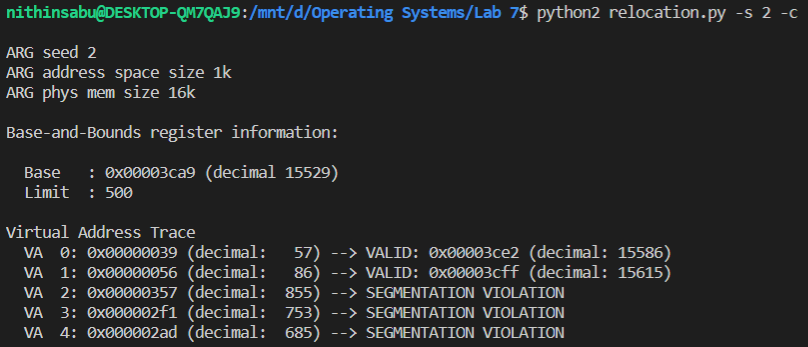
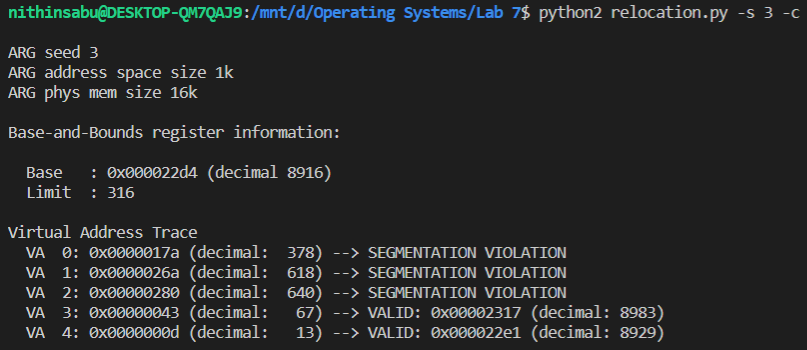
**Operating Systems Assignment 7**

-210010032, Nithin Sabu

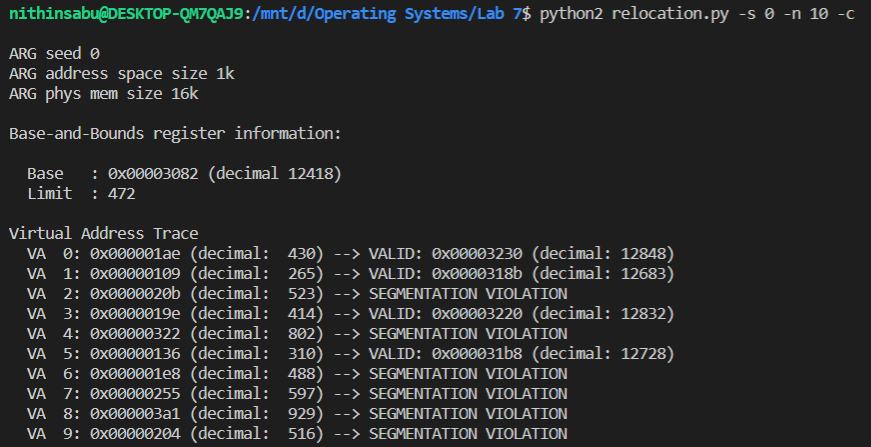
1.1 Run with seeds 1, 2 and 3:



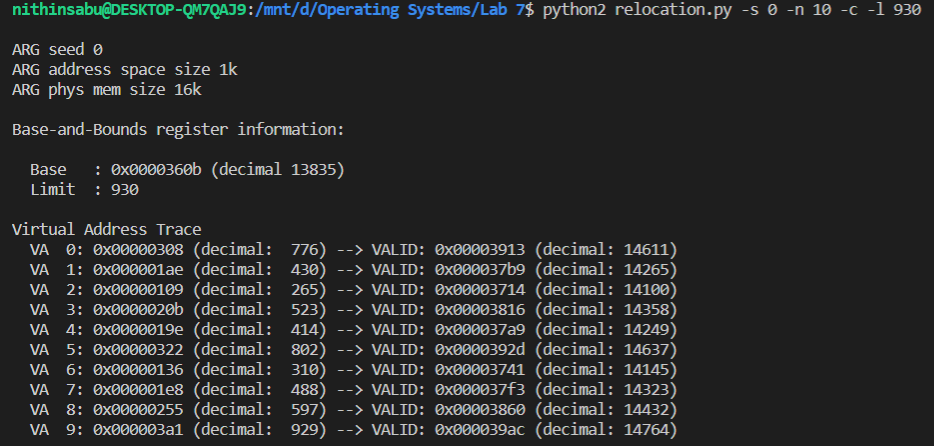


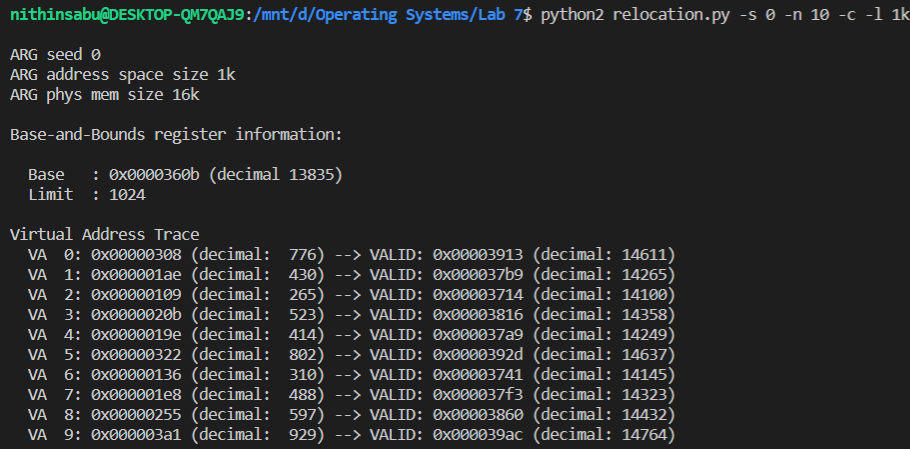


* 1. Run with flags -s 0 and -n 10

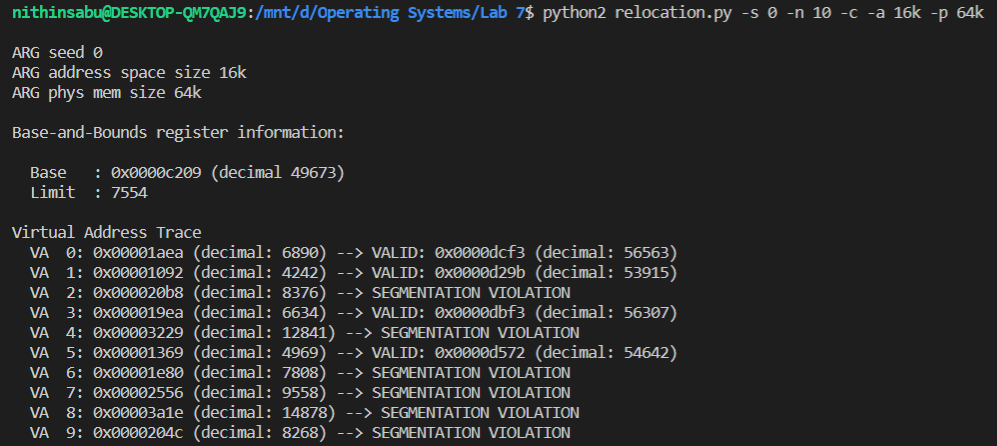


To get all virtual addresses within bounds, the ideal value of bounds register would be equal to the address space = 1k here. But since the seed generates the same values every time, we can say for **this specific case**, bound register must be greater than 929.

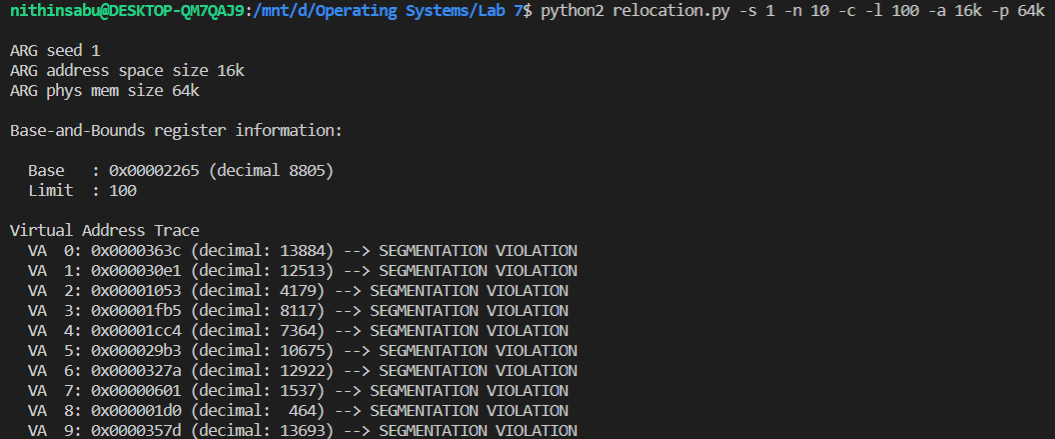




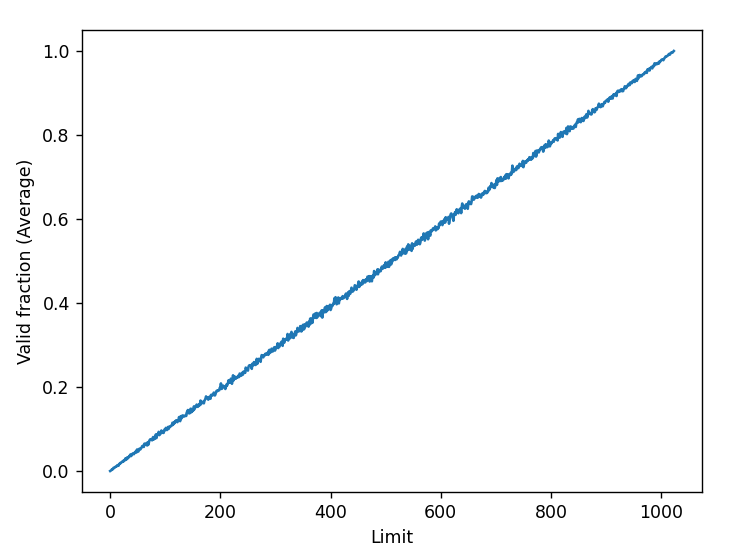
* 1. The maximum value base can be set to would be 16k-100 = 16284.
  2. Running Q2 with -a 16k -p 64k



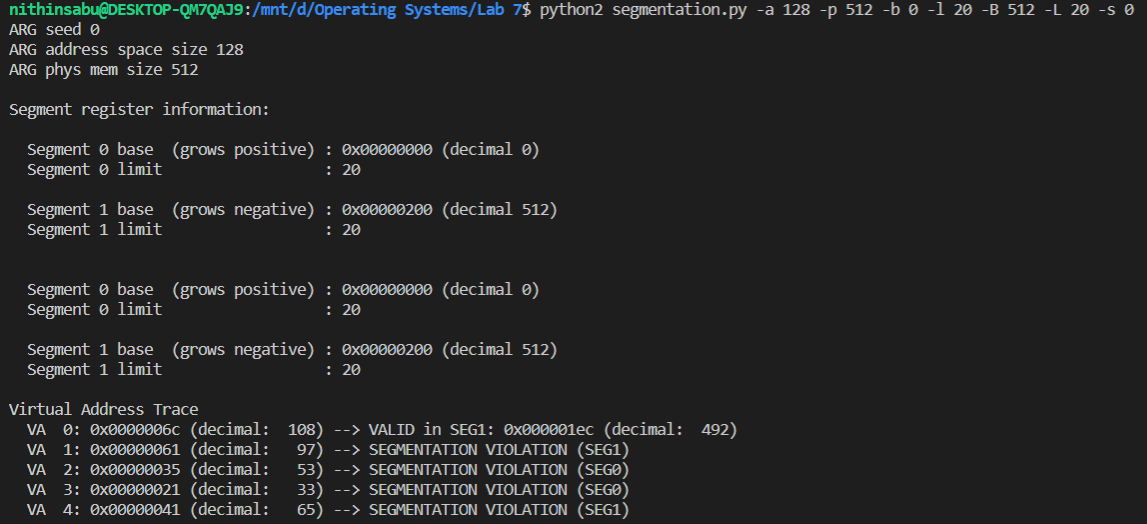
Running Q3 with -a 16k -p 64k

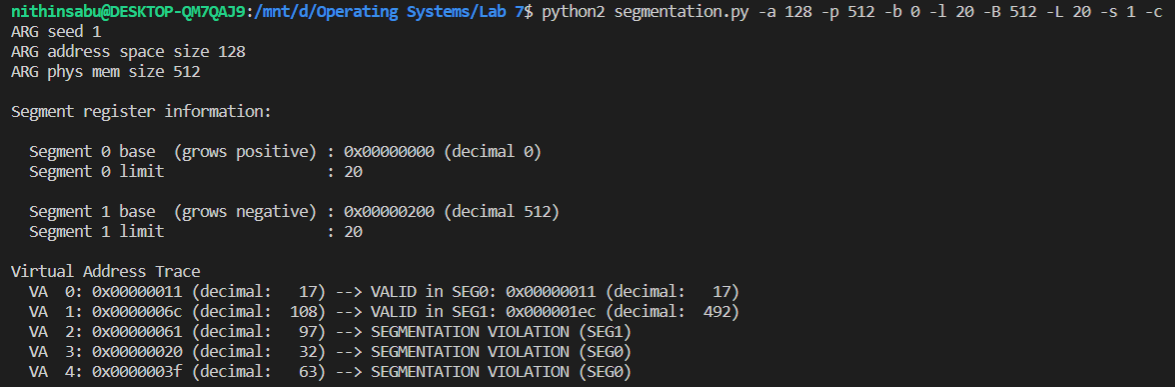


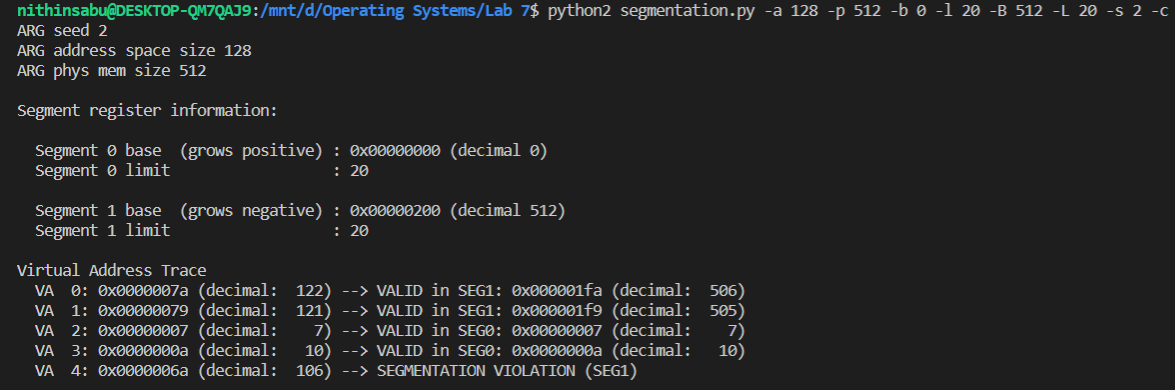
* 1. Ran with address space = 1k and physical memory size = 16k, for 10000 iterations.



2.1



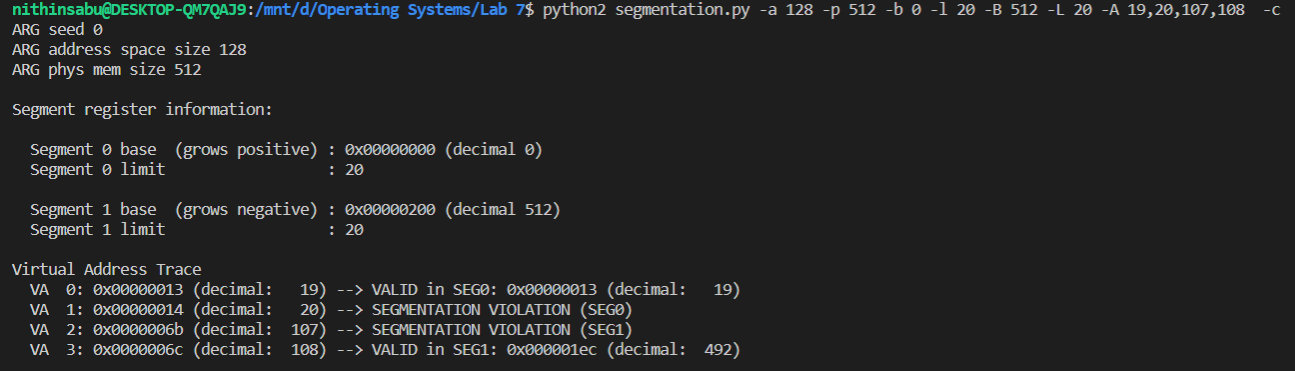




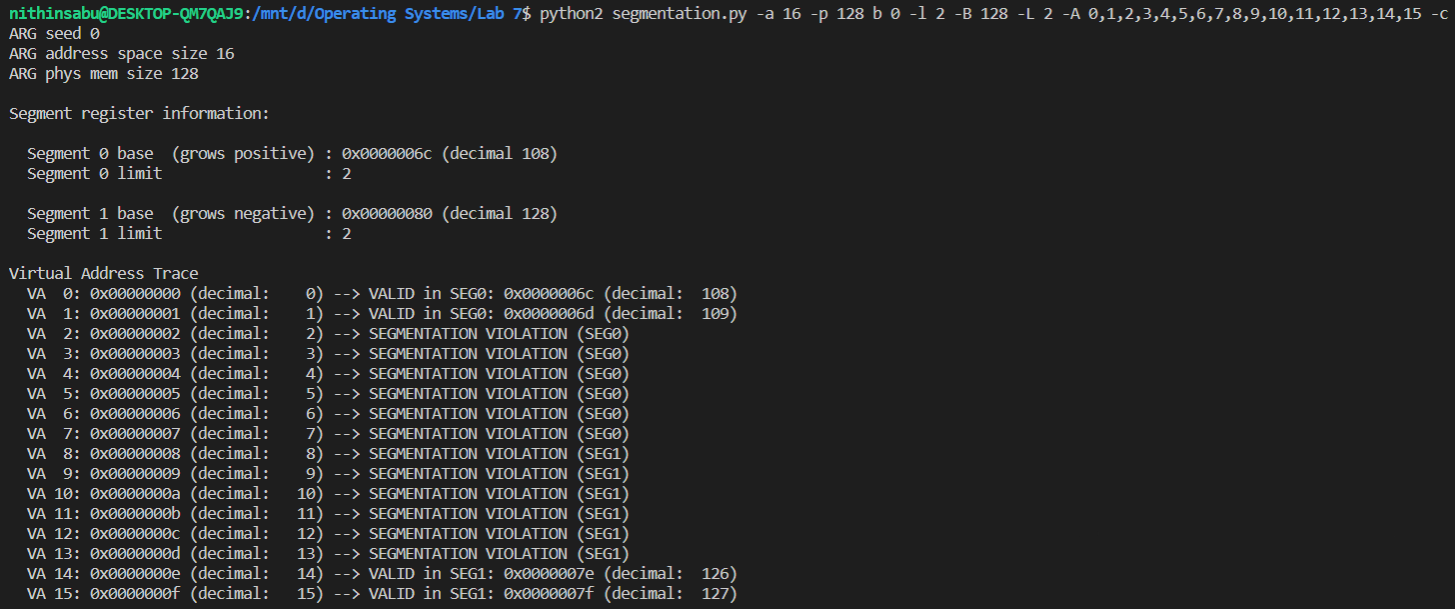
* 1. Highest legal virtual address of Segment 0 is 19 (0+19) and lowest legal virtual address of Segment 1 is

108 (127-19). Highest illegal virtual address is 107, lowest illegal virtual address is 20.

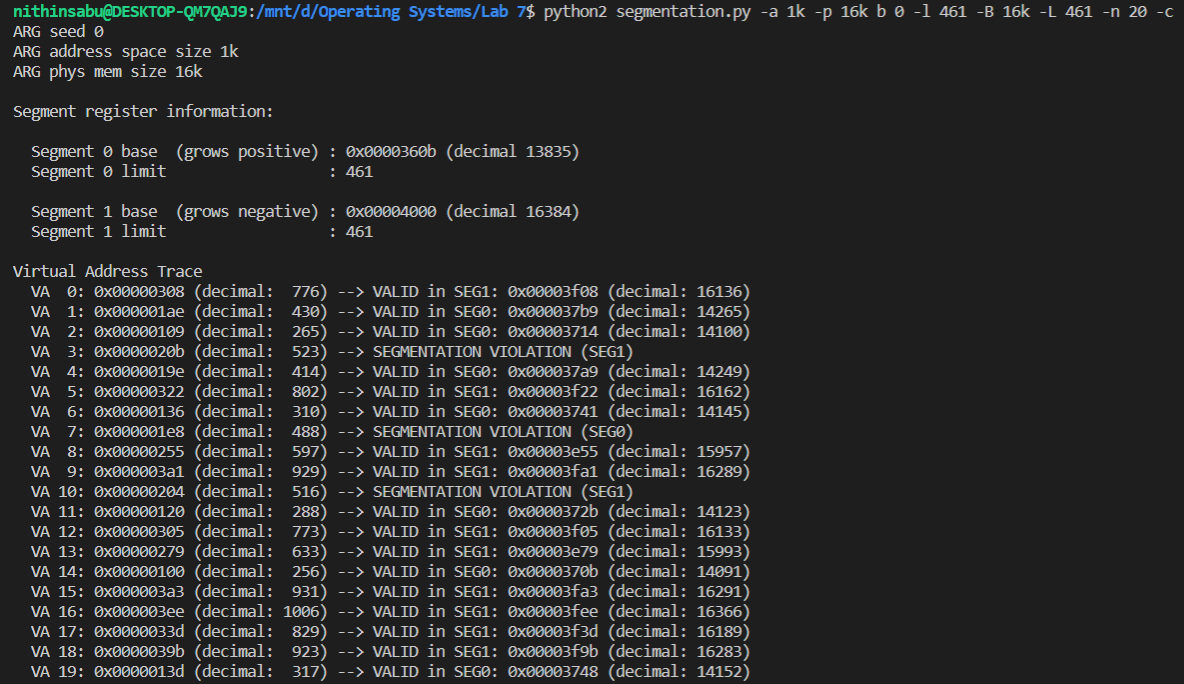
We pass flag as -A 19,20,107,108 to verify:



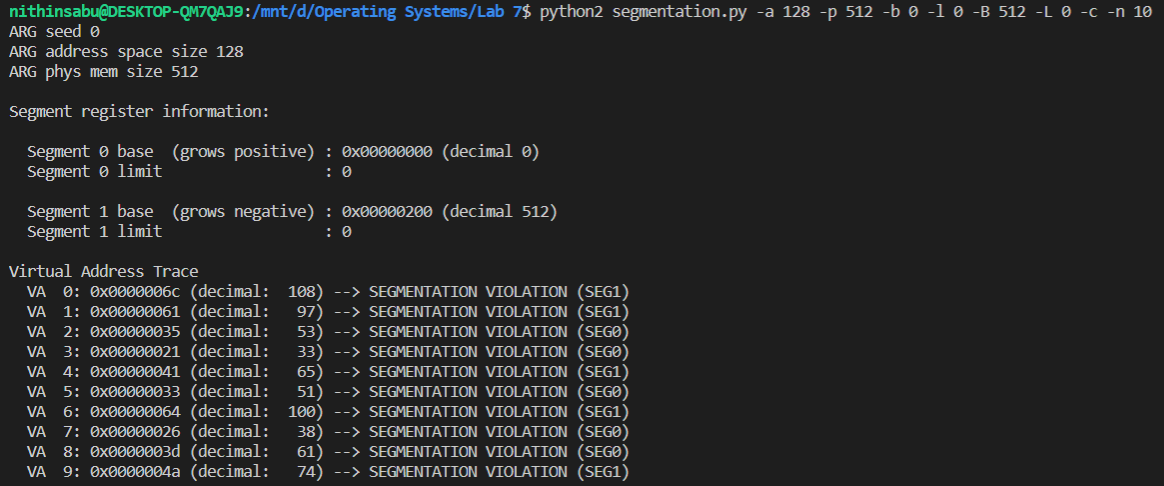
* 1. Values for b = 0, B =128, l = 2, L = 2 will output the required result. This is because virtual addresses between 2 and 14 (inclusive) are all illegal.



* 1. For roughly 90% of valid virtual addresses, we keep both limit registers as approximately virtual address space\*0.45. In the case of below image, 1024\*0.45 ≈ 461.



* 1. Set both limit registers to 0.



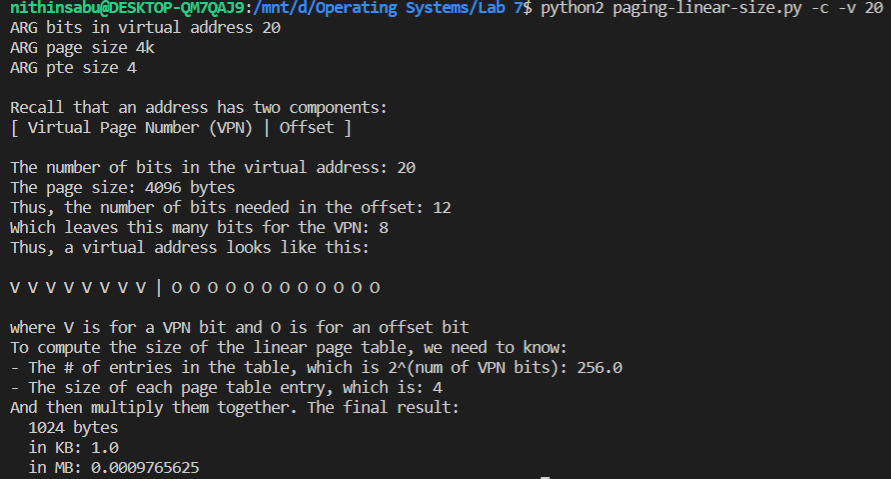
Q3.

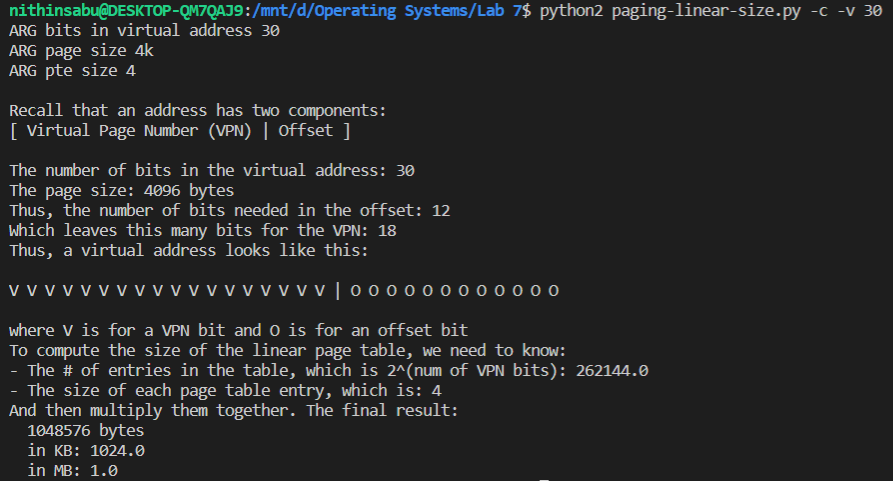
Note here that :

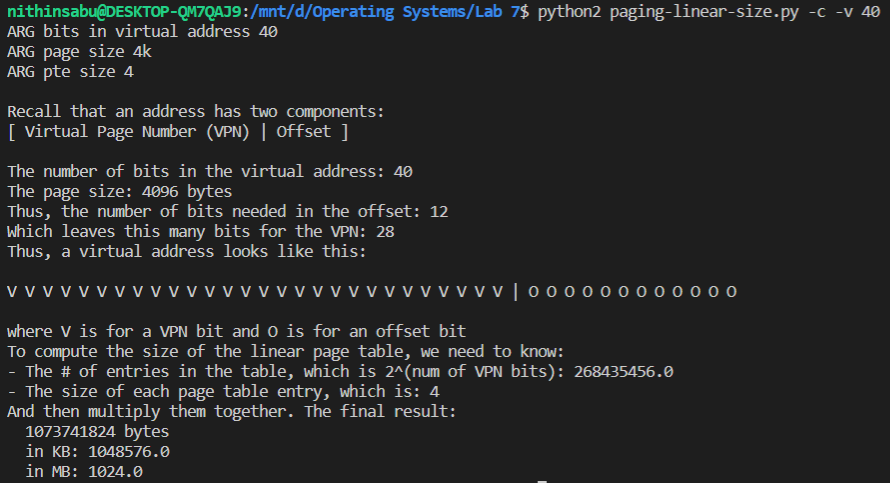
page table size = 2(VPN bits) × (Page table entry size)

1. Running by varying virtual address bytes:

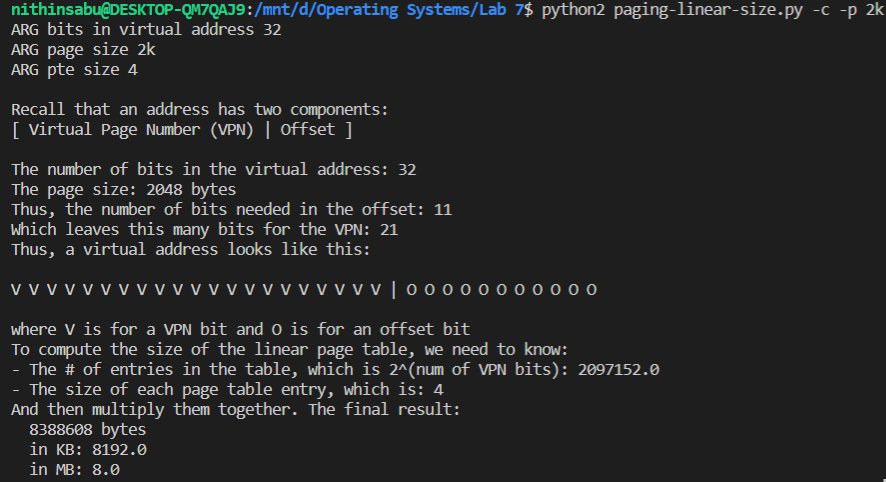
We observe that each bit increase in the virtual address size increases the page table size twice. In the below examples, increasing the address size by 10 bytes increased the page table size by a factor of 210.

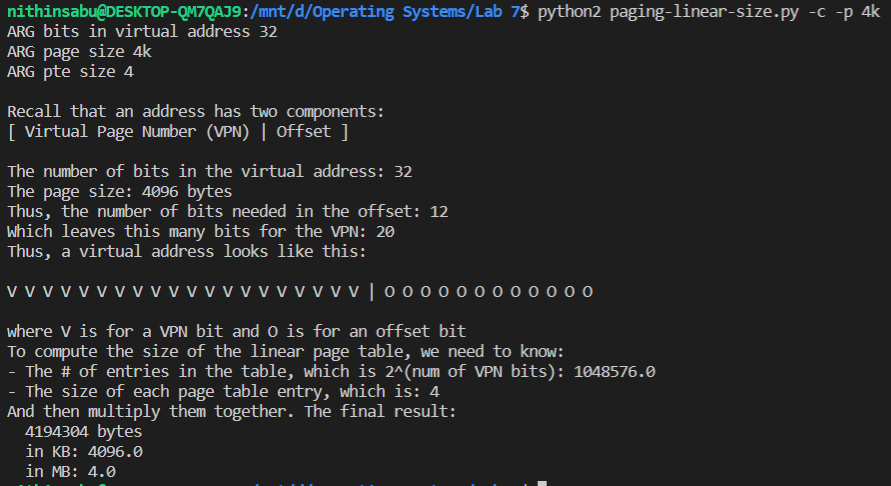


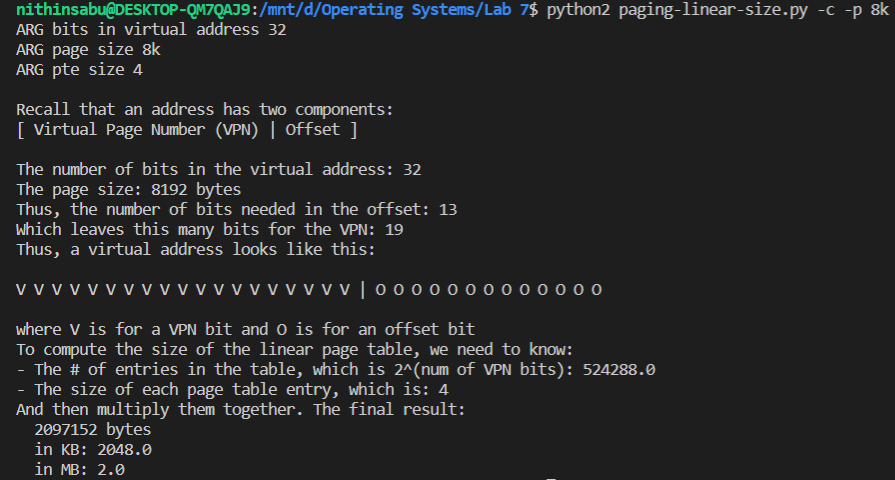




1. Running by varying the page size: doubling the page size halves the page table size. This is because there is more offset bits on increasing page size, hence lesser virtual page number bits. (Page table size = 2(VPN bits) )



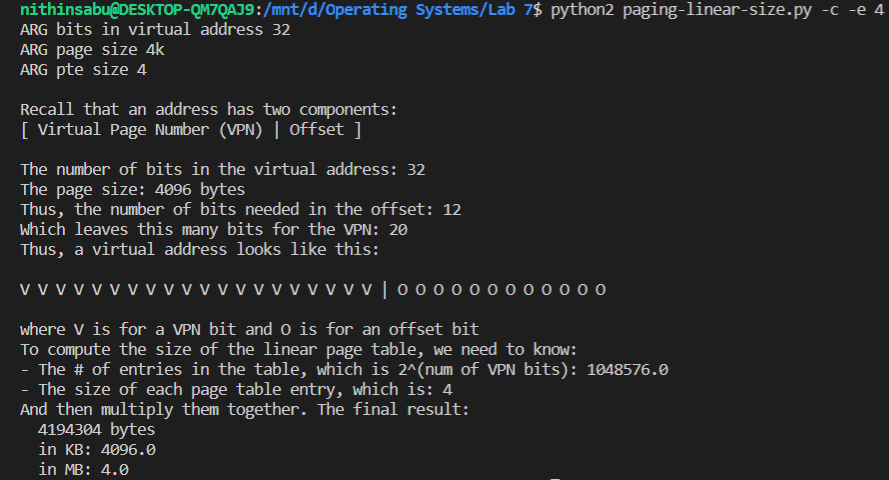


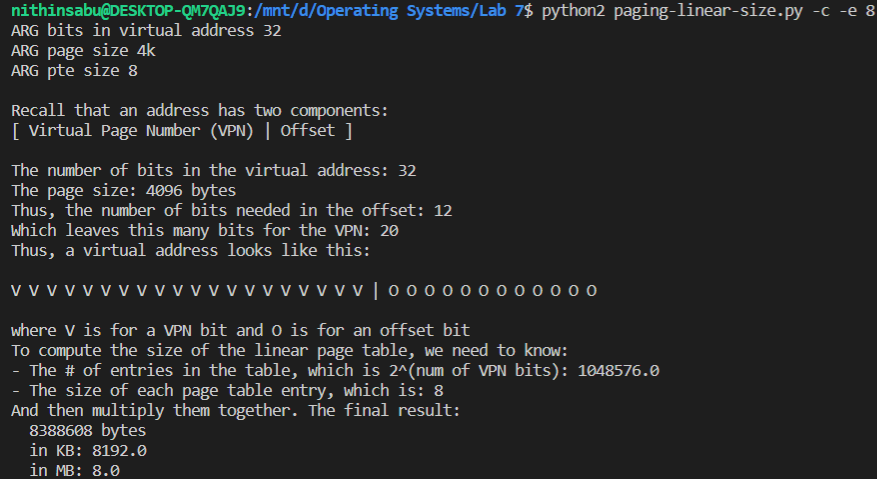


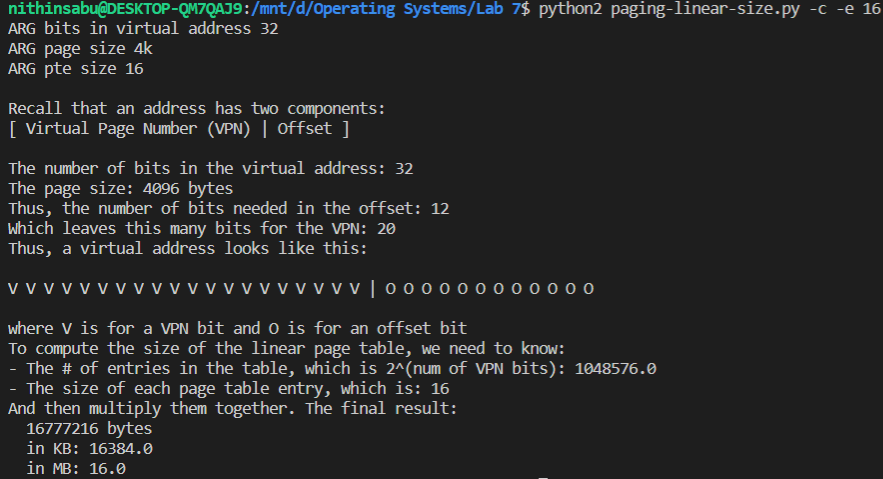
1. Varying page table entry size:

We observe page table size doubling on doubling page table entry size. This is because page table size

Is directly proportional to the page table entry size.

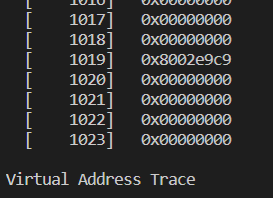
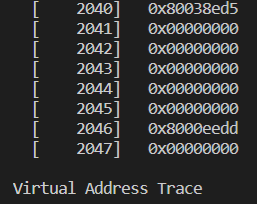




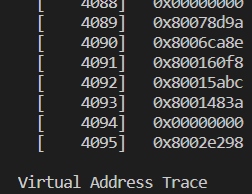


* 1. Increasing the address space increases the page table size proportionately.

-a 1m : -a 2m :

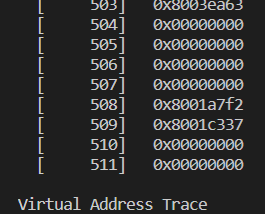
 

-a 4m :

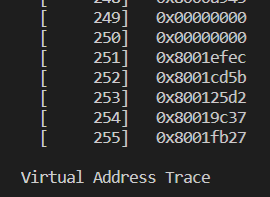


Increasing page size decreases the page table size proportionately:

-P 1k : -P 2k :

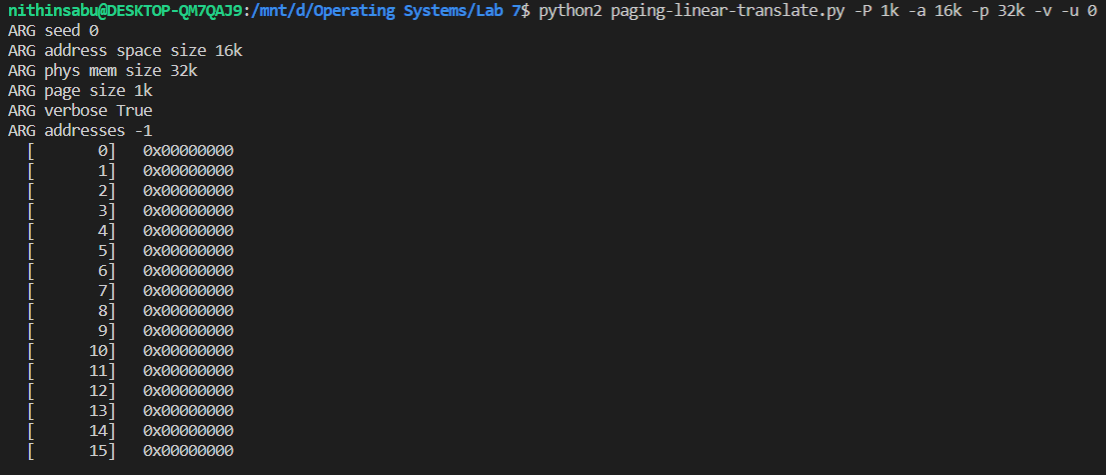
-P 4k :



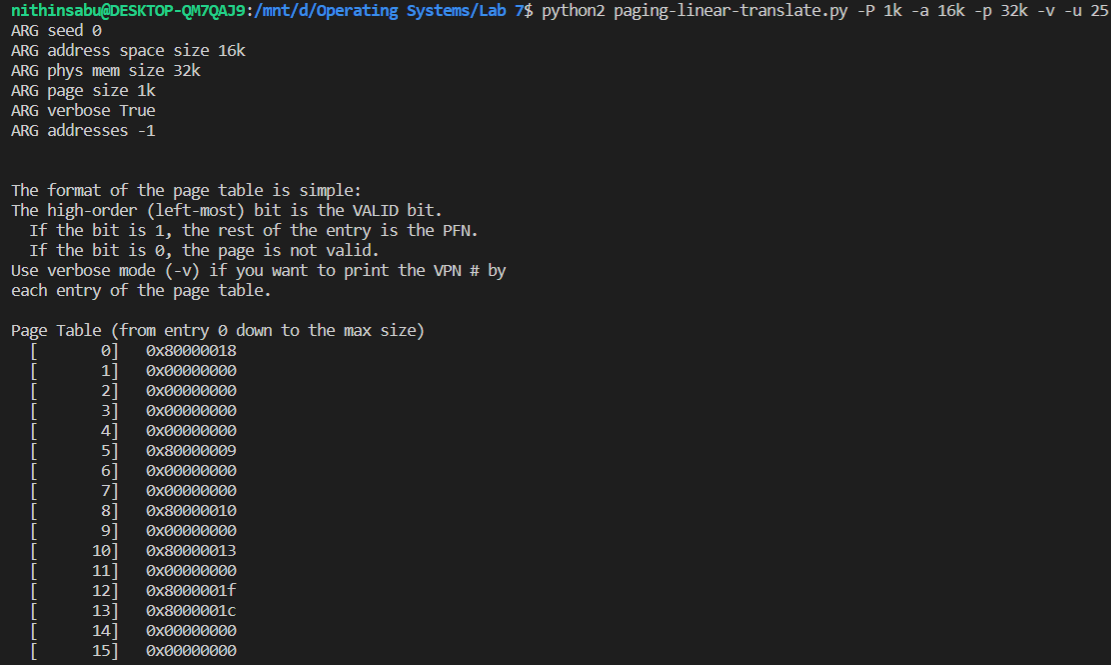
Larger page size decreases the page table size, but has a trade off, such as increases fragmentation, and more memory overhead.

4.2. As u is increased the number of allocated pages increases.

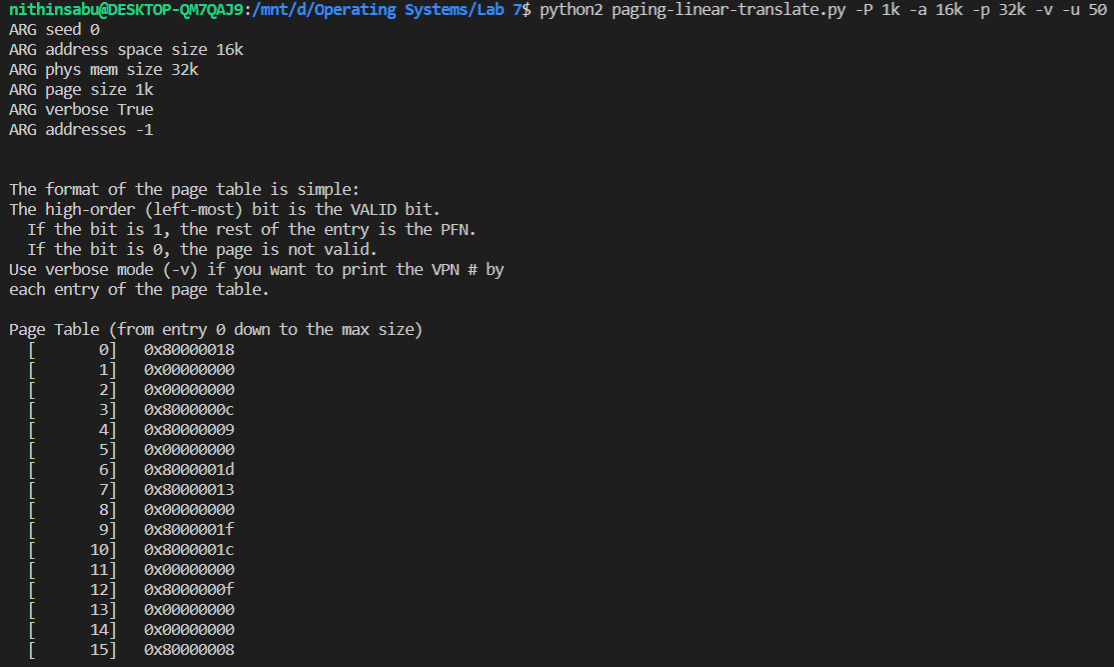
-u 0 : No entries are allocated.



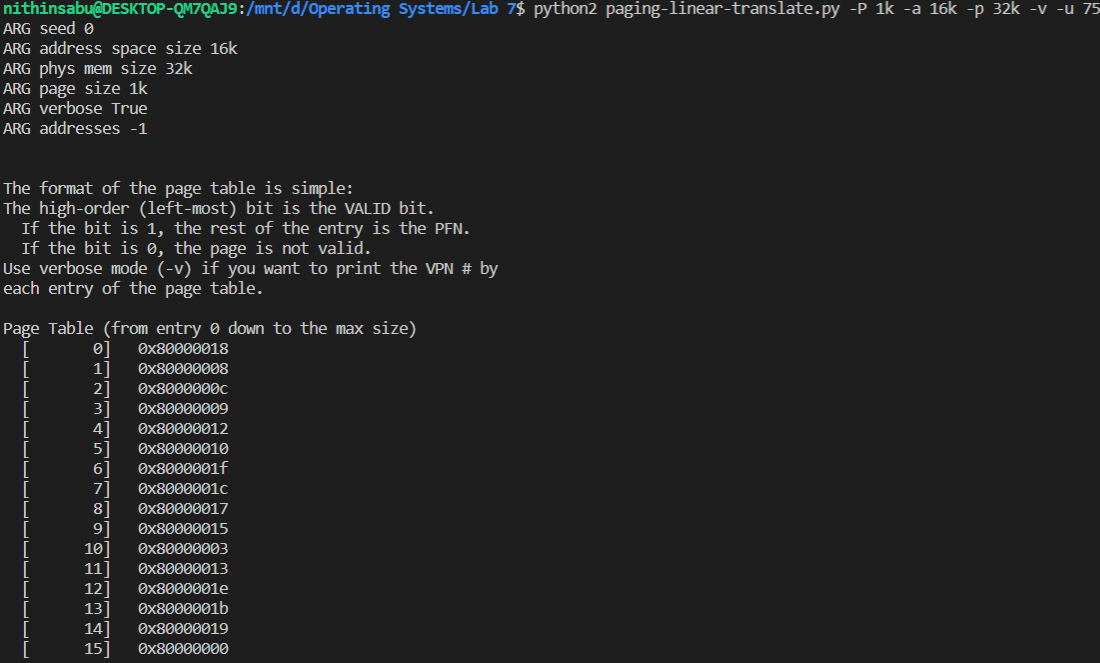
-u 25 : 6 entries are allocated.



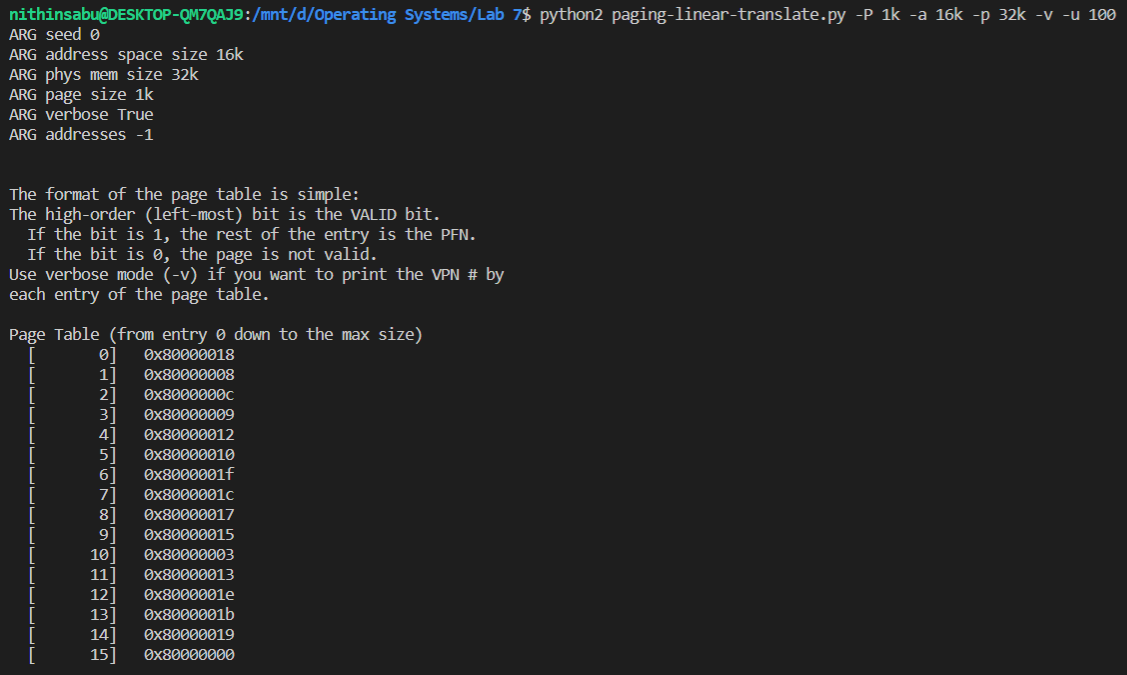
-u 50 : 9 entries are allocated.



-u 75 : all entries are allocated.



-u 100 : all entries are allocated.

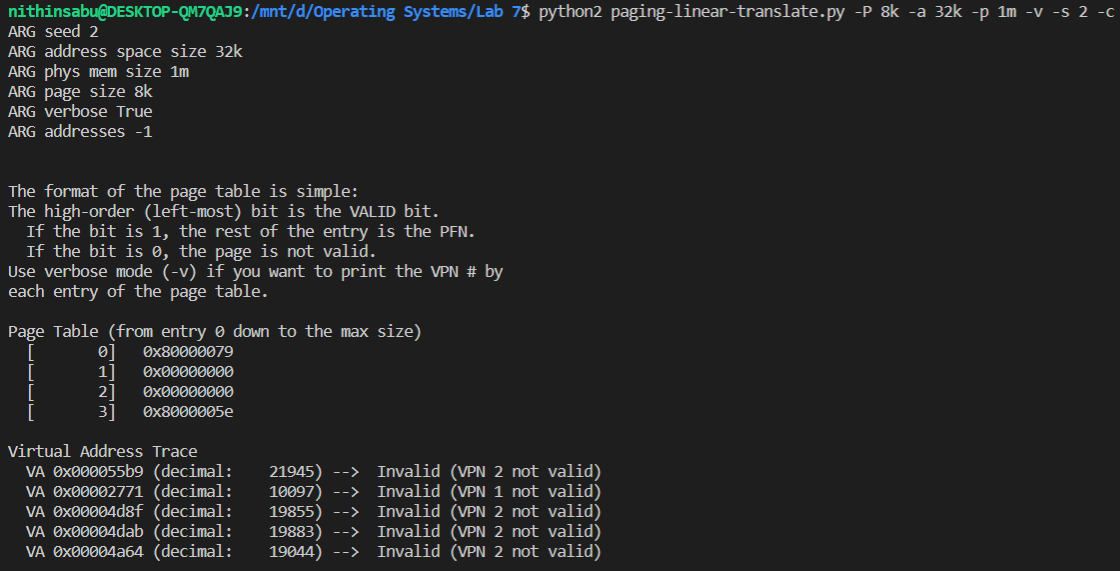


4.3 All three cases seem to be unrealistic:

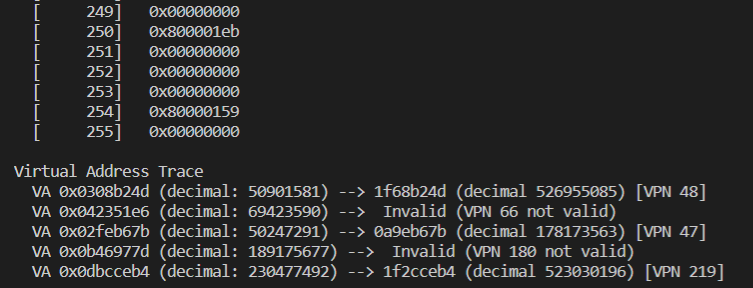
Case 1: because each page is allotted only 8B, and at most 4 pages can be present in the address space



Case 2: because at most 4 pages in the address space.



Case 3: Page size is 1MB which is wastage of memory, hence unrealistic.



4.4 The limitations of the code are:  
 1. address space, physical memory, page size etc. must all be positive.

2. physical memory and address space must be multiple of page size.

3. page size should be a multiple of 2.

4. page size should be smaller than the physical memory, but larger than address space size.