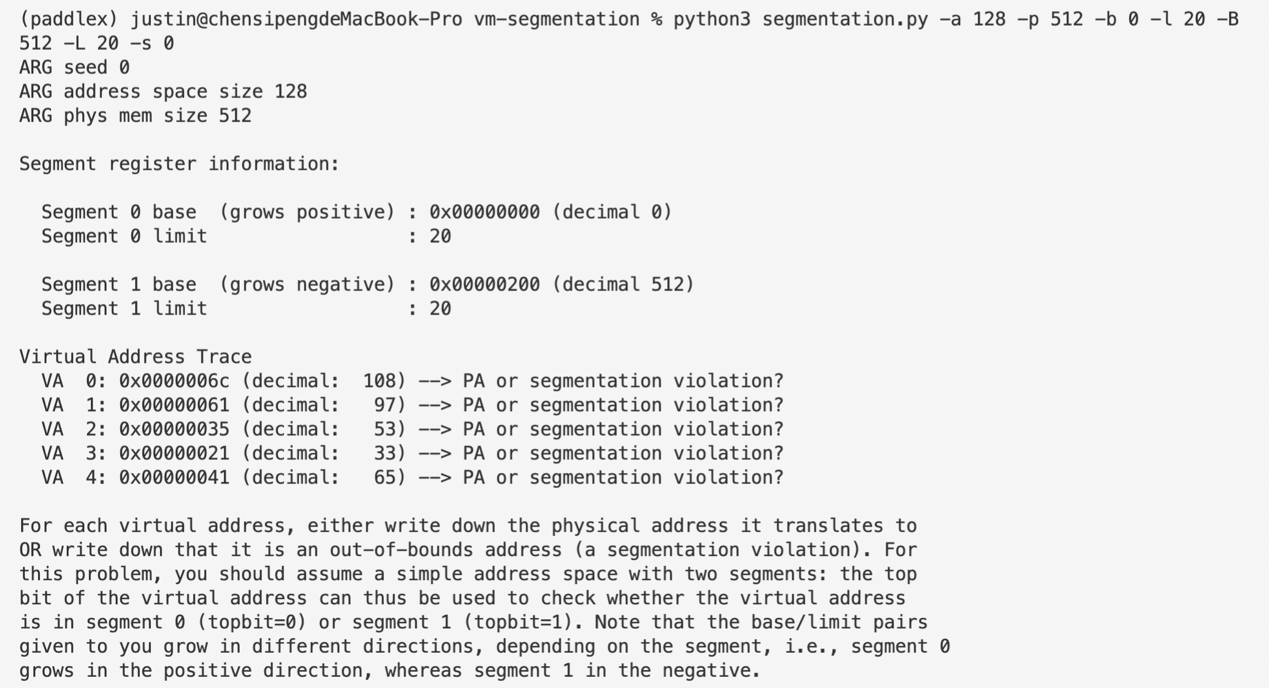
**Cpt 16**

**Homework (Simulation)**

1. **First let’s use a tiny address space to translate some addresses. Here’s a simple set of parameters with a few different random seeds; can you translate the addresses?**

Run the python file with given sets of parameters. In the first example, the address space has a size of 128, which equals 27, thus any virtual address space should comprise seven binary digits. Also, since here we only have 2 segments, the index for memory segment should consist of only one binary digit. With this in mind, we can translate the virtual addresses by first converting them to binaries, then cut out the first digit of all seven, and finally do the translation with the first digit as the segment index and the last six as offset.



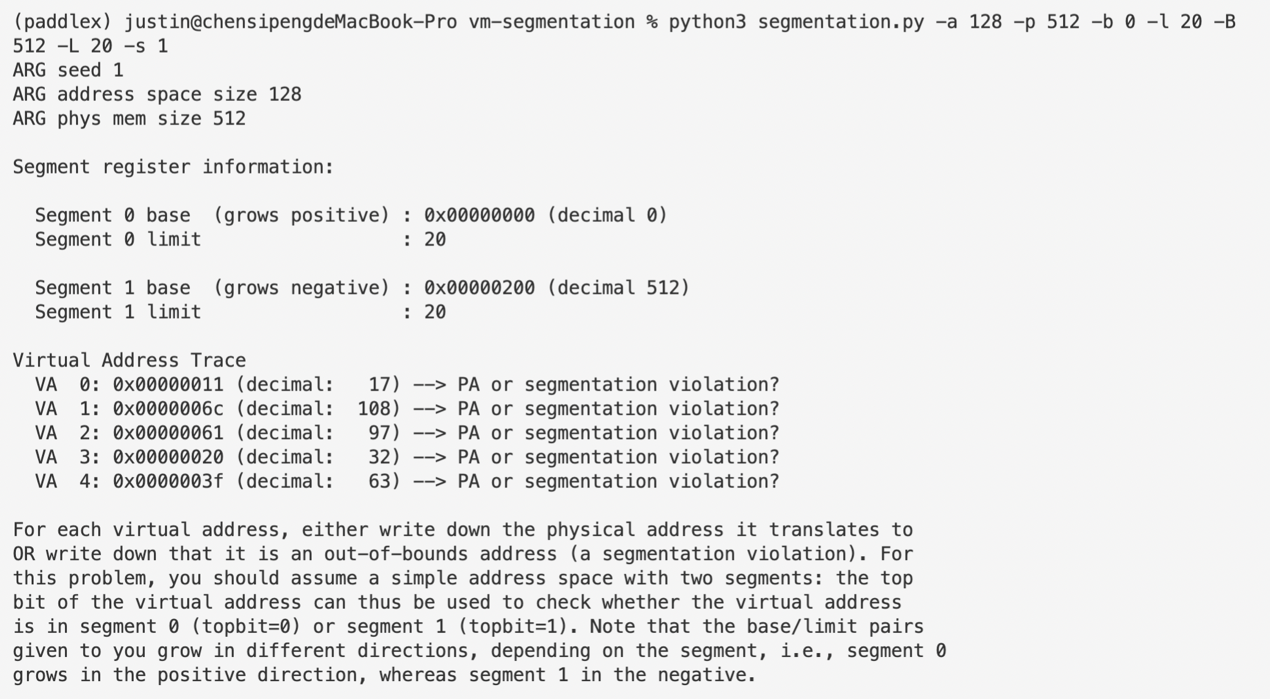
VA 0 -> 0x000001ec (offset = (2c)16 – (128 / 2)10 = -2010)

VA 1 -> segmentation violation (offset = 2116 – (128 / 2)10 = -2110, out of range)

VA 2 -> segmentation violation (offset = 35, out of range)

VA 3 -> segmentation violation (offset = 21, out of range)

VA 4 -> segmentation violation (offset = 0116 – (128 / 2)10 = -6310, out of range)



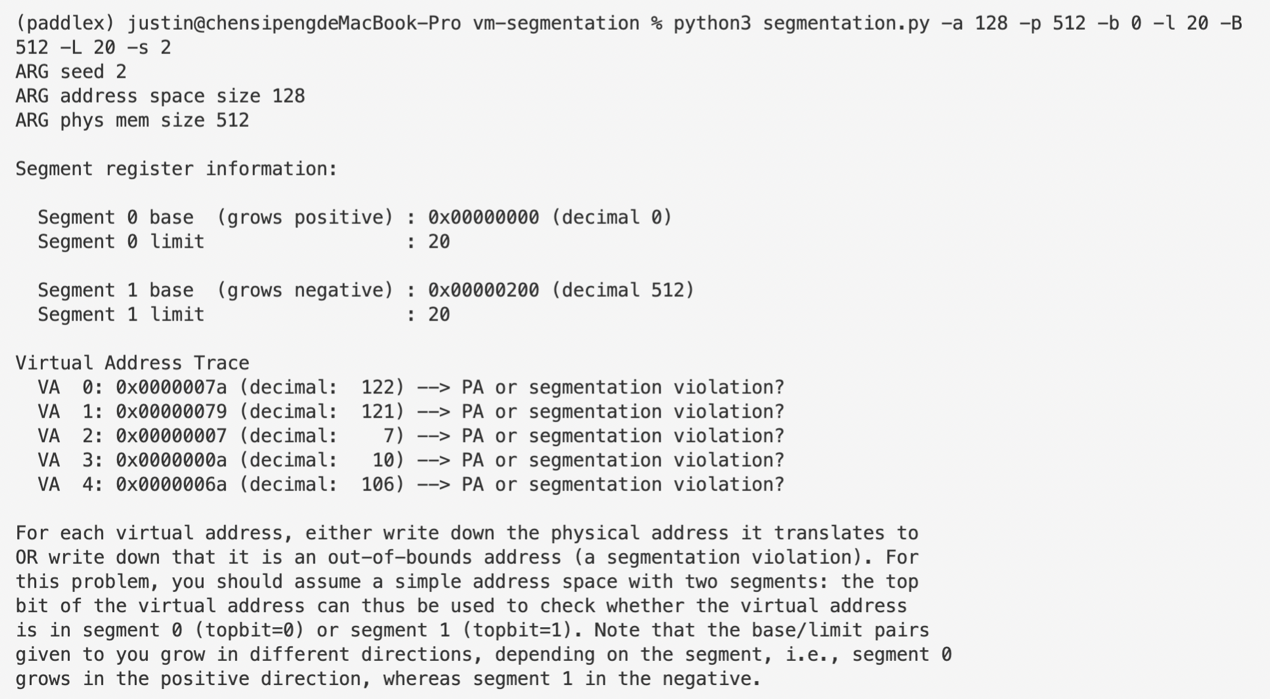
VA 0 -> 0x00000011 (offset = 11)

VA 1 -> 0x000001ec (offset = (2c)16 – (128 / 2)10 = -2010)

VA 2 -> segmentation violation (offset = 2116 – (128 / 2)10 = -2110, out of range)

VA 3 -> segmentation violation (offset = 2016 = 3210, out of range)

VA 4 -> segmentation violation (offset = 6310, out of range)



VA 0 -> 0x000001fa (offset = 3a16 – (128 / 2)10 = -610)

VA 1 -> 0x000001f9 (offset = -710)

VA 2 -> 0x00000007 (offset = 7)

VA 3 -> 0x0000000a (offset = a)

VA 4 -> segmentation violation (offset = 2a16 – 6410 = -2210)

1. **Now, let’s see if we understand this tiny address space we’ve constructed (using the parameters from the question above). What is the highest legal virtual address in segment 0? What about the lowest legal virtual address in segment 1? What are the lowest and highest illegal addresses in this entire address space? Finally, how would you run segmentation.py with the -A flag to test if you are right?**

Highest legal VA in seg 0: 0x00000000 + 2010 = 0x00000014

Lowest legal VA in seg 1: 0x00000200 – 2010 = 0x000001ec

Lowest illegal VA: 0x00000015

Highest illegal VA: 0x000001eb

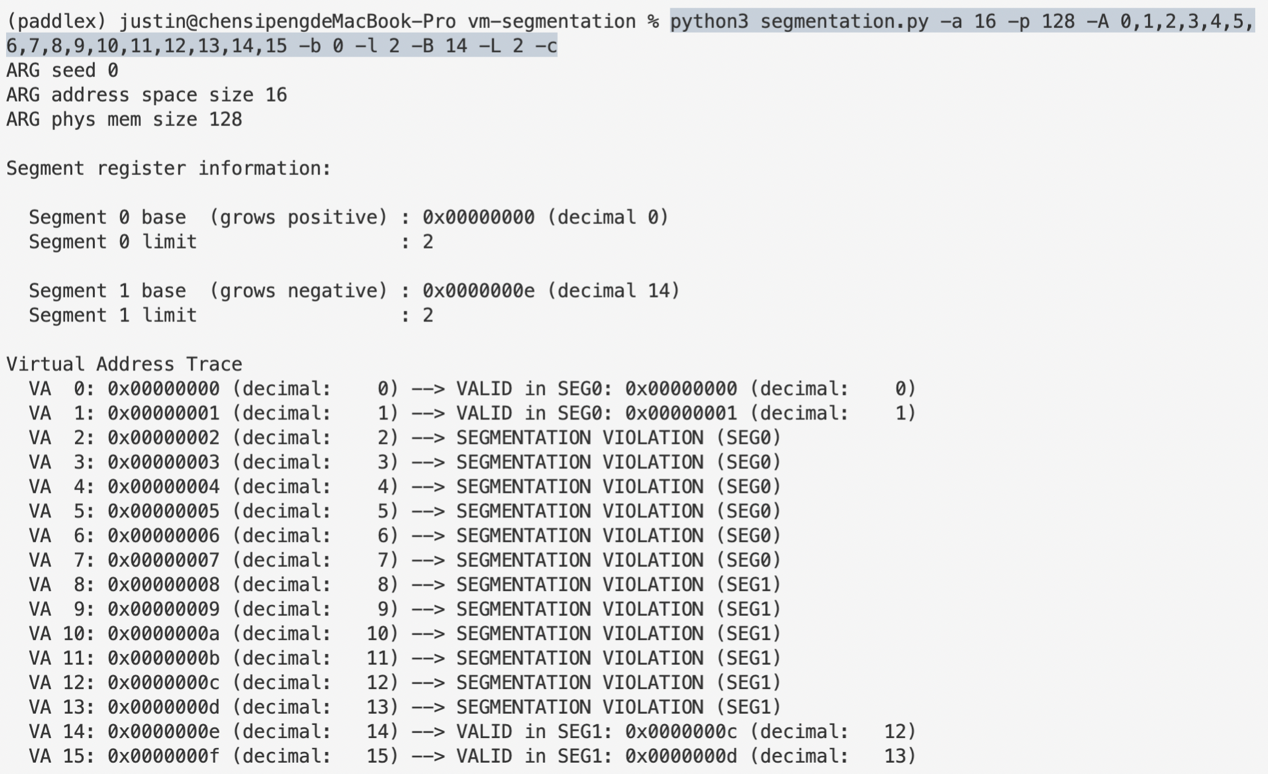
1. **Let’s say we have a tiny 16-byte address space in a 128-byte physical memory. What base and bounds would you set up so as to get the simulator to generate the following translation results for the specified address stream: valid, valid, violation, ..., violation, valid, valid? Assume the following parameters:**

segmentation.py -a 16 -p 128 -A 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 -b0 ? -l0 ? -b1 ? -l1 ?

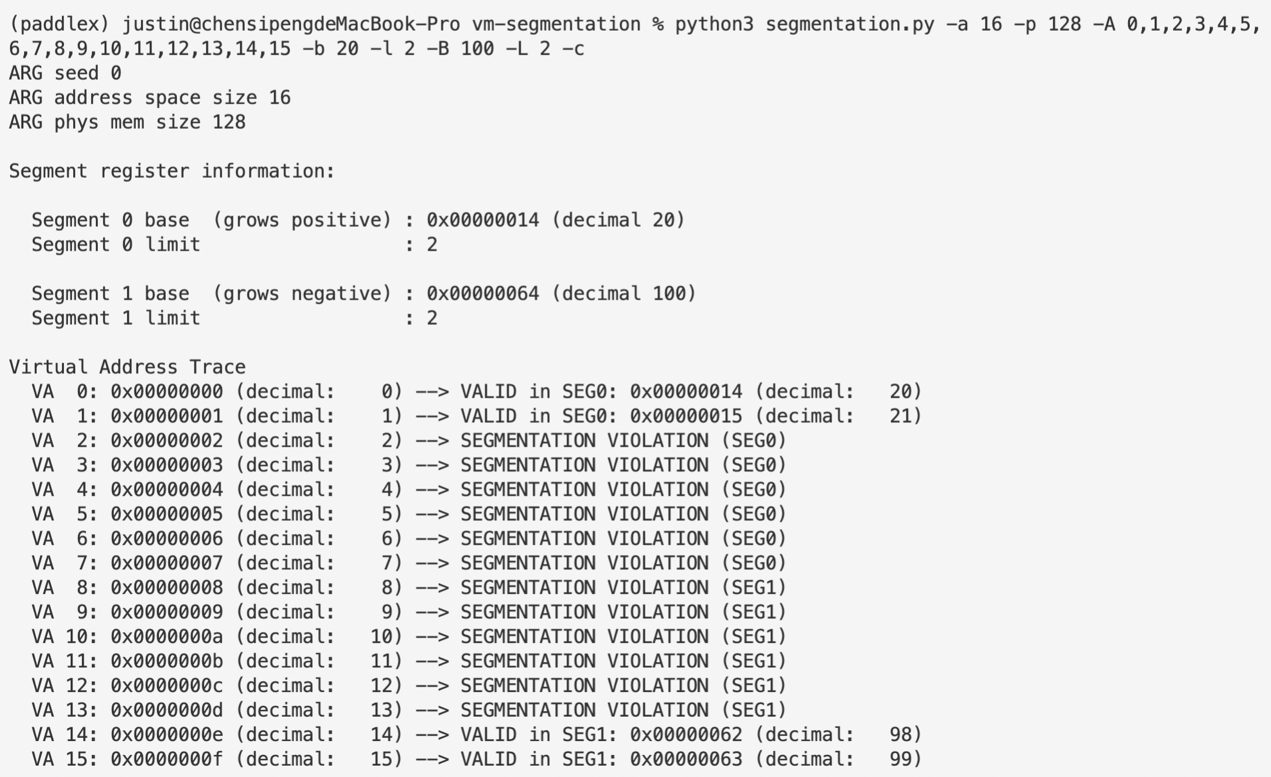
To do so, we simply need to set the limits for both segments 2, which is because 0 and 1 in VA will always be projected to the first 2 addresses above the baseline of the first segment and similarly, e and f will always be projected to the first 2 addresses below the baseline of the second segment. That is to say, what we will have to do is to simply set -l and -L 2, and the value of -b and -B can be totally arbitrary.

Try the examples below:

python3 segmentation.py -a 16 -p 128 -A 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 -b 0 -l 2 -B 14 -L 2 -c



python3 segmentation.py -a 16 -p 128 -A 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 -b 20 -l 2 -B 100 -L 2 -c



1. **Assume we want to generate a problem where roughly 90% of the randomly generated virtual addresses are valid (not segmentation violations). How should you configure the simulator to do so? Which parameters are important to getting this outcome?**

Since what we have here is a one-on-one projection from virtual address space to physical address space, to make 90% of physical addresses valid we simply need to make sure that the address space has a size which is 90% that of our physical space, which is the case of the example below:

python3 segmentation.py -a 90 -p 100 -b 0 -l 45 -B 100 -L 45 -c

1. **Can you run the simulator such that no virtual addresses are valid? How?**

We can achieve this by simply setting the limits for both segments 0, for instance:

python3 segmentation.py -l 0 -L 0