CS314 LAB 6

- Atharva Swami (190010008)

Q1. Address Translation

(1.1)Seed=1

`python relocation.py -s 1 -c`

```
D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python relocation.py -s 1 -c

ARG seed 1
ARG address space size 1k
ARG phys mem size 16k

Base-and-Bounds register information:

  Base   : 0x0000363c (decimal 13884)
  Limit  : 290

Virtual Address Trace
  VA  0: 0x0000030e (decimal:  782) --> SEGMENTATION VIOLATION
  VA  1: 0x00000105 (decimal:  261) --> VALID: 0x00003741 (decimal: 14145)
  VA  2: 0x000001fb (decimal:  507) --> SEGMENTATION VIOLATION
  VA  3: 0x000001cc (decimal:  460) --> SEGMENTATION VIOLATION
  VA  4: 0x0000029b (decimal:  667) --> SEGMENTATION VIOLATION
```

Seed=2

`python relocation.py -s 2 -c`

```
D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python relocation.py -s 2 -c

ARG seed 2
ARG address space size 1k
ARG phys mem size 16k

Base-and-Bounds register information:

  Base   : 0x00003ca9 (decimal 15529)
  Limit  : 500

Virtual Address Trace
  VA  0: 0x00000039 (decimal:   57) --> VALID: 0x00003ce2 (decimal: 15586)
  VA  1: 0x00000056 (decimal:   86) --> VALID: 0x00003cff (decimal: 15615)
  VA  2: 0x00000357 (decimal:  855) --> SEGMENTATION VIOLATION
  VA  3: 0x000002f1 (decimal:  753) --> SEGMENTATION VIOLATION
  VA  4: 0x000002ad (decimal:  685) --> SEGMENTATION VIOLATION
```

Seed=3

python relocation.py -s 3 -c

```
D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python relocation.py -s 3 -c

ARG seed 3
ARG address space size 1k
ARG phys mem size 16k

Base-and-Bounds register information:

  Base   : 0x000022d4 (decimal 8916)
  Limit  : 316

Virtual Address Trace
  VA  0: 0x0000017a (decimal:  378) --> SEGMENTATION VIOLATION
  VA  1: 0x0000026a (decimal:  618) --> SEGMENTATION VIOLATION
  VA  2: 0x00000280 (decimal:  640) --> SEGMENTATION VIOLATION
  VA  3: 0x00000043 (decimal:   67) --> VALID: 0x00002317 (decimal: 8983)
  VA  4: 0x0000000d (decimal:   13) --> VALID: 0x000022e1 (decimal: 8929)
```

(1.2)

We have to set -l max_value + 1 which is 929 + 1 = 930. So, we have to add -l 930 to ensure all the generate virtual address are within bounds.

```
D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python relocation.py -s 0 -n 10 -c

ARG seed 0
ARG address space size 1k
ARG phys mem size 16k

Base-and-Bounds register information:

  Base   : 0x00003082 (decimal 12418)
  Limit  : 472

Virtual Address Trace
  VA  0: 0x000001ae (decimal:  430) --> VALID: 0x00003230 (decimal: 12848)
  VA  1: 0x00000109 (decimal:  265) --> VALID: 0x0000318b (decimal: 12683)
  VA  2: 0x0000020b (decimal:  523) --> SEGMENTATION VIOLATION
  VA  3: 0x0000019e (decimal:  414) --> VALID: 0x00003220 (decimal: 12832)
  VA  4: 0x00000322 (decimal:  802) --> SEGMENTATION VIOLATION
  VA  5: 0x00000136 (decimal:  310) --> VALID: 0x000031b8 (decimal: 12728)
  VA  6: 0x000001e8 (decimal:  488) --> SEGMENTATION VIOLATION
  VA  7: 0x00000255 (decimal:  597) --> SEGMENTATION VIOLATION
  VA  8: 0x000003a1 (decimal:  929) --> SEGMENTATION VIOLATION
  VA  9: 0x00000204 (decimal:  516) --> SEGMENTATION VIOLATION


D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python relocation.py -s 0 -n 10 -l 930 -c

ARG seed 0
ARG address space size 1k
ARG phys mem size 16k

Base-and-Bounds register information:

  Base   : 0x0000360b (decimal 13835)
  Limit  : 930

Virtual Address Trace
  VA  0: 0x00000308 (decimal:  776) --> VALID: 0x00003913 (decimal: 14611)
  VA  1: 0x000001ae (decimal:  430) --> VALID: 0x000037b9 (decimal: 14265)
  VA  2: 0x00000109 (decimal:  265) --> VALID: 0x00003714 (decimal: 14100)
  VA  3: 0x0000020b (decimal:  523) --> VALID: 0x00003816 (decimal: 14358)
  VA  4: 0x0000019e (decimal:  414) --> VALID: 0x000037a9 (decimal: 14249)
  VA  5: 0x00000322 (decimal:  802) --> VALID: 0x0000392d (decimal: 14637)
  VA  6: 0x00000136 (decimal:  310) --> VALID: 0x00003741 (decimal: 14145)
  VA  7: 0x000001e8 (decimal:  488) --> VALID: 0x000037f3 (decimal: 14323)
  VA  8: 0x00000255 (decimal:  597) --> VALID: 0x00003860 (decimal: 14432)
  VA  9: 0x000003a1 (decimal:  929) --> VALID: 0x000039ac (decimal: 14764)
```

(1.3)

The maximum value that base can be set is Max Base = Psize - limit = 16 * 1024 - 100 = 16384 - 100 = 16284

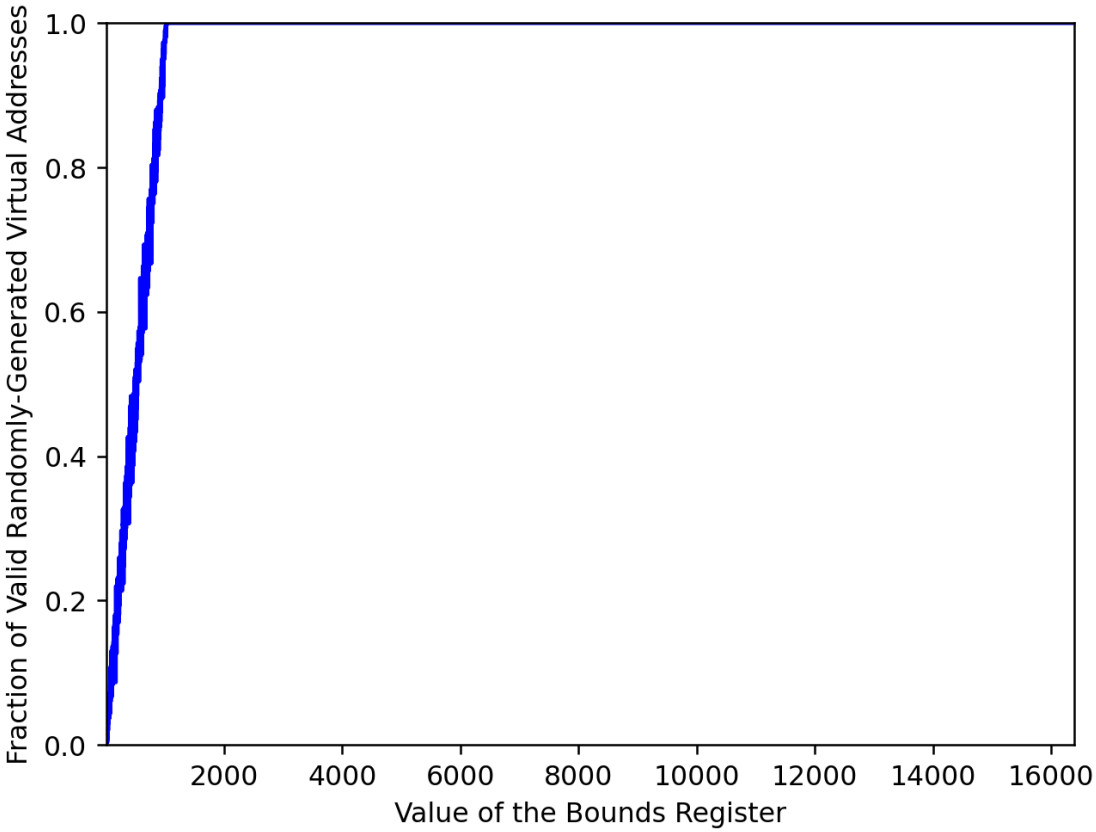
```
D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python relocation.py -s 1 -n 10 -l 100 -c

ARG seed 1
ARG address space size 1k
ARG phys mem size 16k

Base-and-Bounds register information:

  Base   : 0x00000899 (decimal 2201)
  Limit  : 100

Virtual Address Trace
  VA  0: 0x00000363 (decimal:  867) --> SEGMENTATION VIOLATION
  VA  1: 0x0000030e (decimal:  782) --> SEGMENTATION VIOLATION
  VA  2: 0x00000105 (decimal:  261) --> SEGMENTATION VIOLATION
  VA  3: 0x000001fb (decimal:  507) --> SEGMENTATION VIOLATION
  VA  4: 0x000001cc (decimal:  460) --> SEGMENTATION VIOLATION
  VA  5: 0x0000029b (decimal:  667) --> SEGMENTATION VIOLATION
  VA  6: 0x00000327 (decimal:  807) --> SEGMENTATION VIOLATION
  VA  7: 0x00000060 (decimal:   96) --> VALID: 0x000008f9 (decimal: 2297)
  VA  8: 0x0000001d (decimal:   29) --> VALID: 0x000008b6 (decimal: 2230)
  VA  9: 0x00000357 (decimal:  855) --> SEGMENTATION VIOLATION


D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python relocation.py -s 1 -n 10 -l 100 -b 16284 -c

ARG seed 1
ARG address space size 1k
ARG phys mem size 16k

Base-and-Bounds register information:

  Base   : 0x00003f9c (decimal 16284)
  Limit  : 100

Virtual Address Trace
  VA  0: 0x00000089 (decimal:  137) --> SEGMENTATION VIOLATION
  VA  1: 0x00000363 (decimal:  867) --> SEGMENTATION VIOLATION
  VA  2: 0x0000030e (decimal:  782) --> SEGMENTATION VIOLATION
  VA  3: 0x00000105 (decimal:  261) --> SEGMENTATION VIOLATION
  VA  4: 0x000001fb (decimal:  507) --> SEGMENTATION VIOLATION
  VA  5: 0x000001cc (decimal:  460) --> SEGMENTATION VIOLATION
  VA  6: 0x0000029b (decimal:  667) --> SEGMENTATION VIOLATION
  VA  7: 0x00000327 (decimal:  807) --> SEGMENTATION VIOLATION
  VA  8: 0x00000060 (decimal:   96) --> VALID: 0x00003ffc (decimal: 16380)
  VA  9: 0x0000001d (decimal:   29) --> VALID: 0x00003fb9 (decimal: 16313)
```

(1.4)

1g = 1024 * 1024 * 1024 - 100 = 1073741724

```
D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python relocation.py -s 1 -n 10 -l 100 -b 1g -a 512m -p 1g -c

ARG seed 1
ARG address space size 512m
ARG phys mem size 1g

Base-and-Bounds register information:

  Base   : 0x40000000 (decimal 1073741824)
  Limit  : 100

Error: address space does not fit into physical memory with those base/bounds values.
Base + Limit: 1073741924   Psize: 1073741824

D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python relocation.py -s 1 -n 10 -l 100 -b 1073741724 -a 1073741823 -p 1073741824 -c

ARG seed 1
ARG address space size 1073741823
ARG phys mem size 1073741824

Base-and-Bounds register information:

  Base   : 0x3fffff9c (decimal 1073741724)
  Limit  : 100

Virtual Address Trace
  VA  0: 0x08996c7c (decimal: 144272508) --> SEGMENTATION VIOLATION
  VA  1: 0x363c5ab5 (decimal: 909925045) --> SEGMENTATION VIOLATION
  VA  2: 0x30e1aeef (decimal: 820096751) --> SEGMENTATION VIOLATION
  VA  3: 0x10530d08 (decimal: 273878280) --> SEGMENTATION VIOLATION
  VA  4: 0x1fb5355d (decimal: 531969373) --> SEGMENTATION VIOLATION
  VA  5: 0x1cc4762b (decimal: 482637355) --> SEGMENTATION VIOLATION
  VA  6: 0x29b3b302 (decimal: 699642626) --> SEGMENTATION VIOLATION
  VA  7: 0x327a7180 (decimal: 846885248) --> SEGMENTATION VIOLATION
  VA  8: 0x0601cba3 (decimal: 100780963) --> SEGMENTATION VIOLATION
  VA  9: 0x01d071ef (decimal: 30437871) --> SEGMENTATION VIOLATION
```

(1.5)


Fraction of Valid Randomly-Generated Virtual Addresses
1.0
0.8
0.6
0.4
0.2
0.0
2000
4000
6000
8000
10000
12000
14000
16000
Value of the Bounds Register

Q2. Segmentation

(2.1)

```
D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 0 -c
ARG seed 0
ARG address space size 128
ARG phys mem size 512

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 20

  Segment 1 base  (grows negative) : 0x00000200 (decimal 512)
  Segment 1 limit                  : 20

Virtual Address Trace
  VA  0: 0x0000006c (decimal:  108) --> VALID in SEG1: 0x000001ec (decimal:  492)
  VA  1: 0x00000061 (decimal:   97) --> SEGMENTATION VIOLATION (SEG1)
  VA  2: 0x00000035 (decimal:   53) --> SEGMENTATION VIOLATION (SEG0)
  VA  3: 0x00000021 (decimal:   33) --> SEGMENTATION VIOLATION (SEG0)
  VA  4: 0x00000041 (decimal:   65) --> SEGMENTATION VIOLATION (SEG1)
```

VA 0: 0x0000006c (decimal: 108) --> PA or segmentation violation?

1. Convert the VA into decimal (108) base 10 = (1101100) base 2
2. Determine the Segment
    - (log2(address size) - 1)th bit is the segment bit.
    - In our case 1101100 it is 1 (The bit on the most significat position is segment bit).
    - It is telling that this virtual address is referring to some physical address in segment 1.
3. As our desired memory location is in segment 1 whose growth is negative. So, we have to use slightly different formula to get correct offset.
   Offset = Decimal Representation of (All Bits except Segment Bit (101100)) - Max. Segment
    - We can calculate max segment value by 2^[log2(address size) - number of segment bits]
    - Max Segment = 2^(log2(128) - 1) = 2^(7 - 1) = 2^(6) = 64

   So, our Offset calculation is as follow
    - Offset = Decimal Representation of 0b101100 - 64
        ➔ Offset = 44 - 64 = -20
    - Absolute of Offset should be less or equal than Bounds(Limit)
        ➔ |-20| <= 20 True (Valid)
4. Physical Address = Base + Offset = 512 - 20 = 492

Answer = VA 0: 0x0000006c (decimal: 108) --> Valid in Segment 1; 0x1ec or in decimal 492

VA 1: 0x00000061 (decimal: 97) --> PA or segmentation violation?

VA = 0b1100001
Seg 1
Offset = 33-64 = -31
|-31| > 20
So, Answer = Segmentation Fault

VA 2: 0x00000035 (decimal: 53) --> PA or segmentation violation?
VA = 0b0110101
Seg 0
Note that the formula for segment 0 is different because its growth is positive.
Offset = Decimal Representation of (All Bits except Segment Bit)
Offset = 53
|53| > 20
So, Answer = Segmentation Fault

VA 3: 0x00000021 (decimal: 33) --> PA or segmentation violation?
VA = 0b0100001
Seg 0
Offset = 33
33 > 20
So, Answer = Segmentation Fault

VA 4: 0x00000041 (decimal: 65) --> PA or segmentation violation?
VA = 0b1000001
Seg 1
Offset = 1 - 64 = -63
|-63| > 20
So, Answer = Segmentation Fault

Similary for the other 2 commands with seed 1 and 2.

```
D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 1 -c
ARG seed 1
ARG address space size 128
ARG phys mem size 512

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 20

  Segment 1 base  (grows negative) : 0x00000200 (decimal 512)
  Segment 1 limit                  : 20

Virtual Address Trace
  VA  0: 0x00000011 (decimal:   17) --> VALID in SEG0: 0x00000011 (decimal:   17)
  VA  1: 0x0000006c (decimal:  108) --> VALID in SEG1: 0x000001ec (decimal:  492)
  VA  2: 0x00000061 (decimal:   97) --> SEGMENTATION VIOLATION (SEG1)
  VA  3: 0x00000020 (decimal:   32) --> SEGMENTATION VIOLATION (SEG0)
  VA  4: 0x0000003f (decimal:   63) --> SEGMENTATION VIOLATION (SEG0)
```

```
D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 2 -c
ARG seed 2
ARG address space size 128
ARG phys mem size 512

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 20

  Segment 1 base  (grows negative) : 0x00000200 (decimal 512)
  Segment 1 limit                  : 20

Virtual Address Trace
  VA  0: 0x0000007a (decimal:  122) --> VALID in SEG1: 0x000001fa (decimal:  506)
  VA  1: 0x00000079 (decimal:  121) --> VALID in SEG1: 0x000001f9 (decimal:  505)
  VA  2: 0x00000007 (decimal:    7) --> VALID in SEG0: 0x00000007 (decimal:    7)
  VA  3: 0x0000000a (decimal:   10) --> VALID in SEG0: 0x0000000a (decimal:   10)
  VA  4: 0x0000006a (decimal:  106) --> SEGMENTATION VIOLATION (SEG1)
```

(2.2)

Highest Legal VA in Seg 0 = 19
Lowest Legal VA in Seg 1 = 108
Lowest Illegal VA in Whole Addr. Space = 20
Highest Illegal VA in Whole Addr. Space = 107

```
python segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 0 -c -A 20

python segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 0 -c -A 107
```

```
D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 0 -c -A 20
ARG seed 0
ARG address space size 128
ARG phys mem size 512

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 20

  Segment 1 base  (grows negative) : 0x00000200 (decimal 512)
  Segment 1 limit                  : 20

Virtual Address Trace
  VA  0: 0x00000014 (decimal:   20) --> SEGMENTATION VIOLATION (SEG0)


D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 0 -c -A 107
ARG seed 0
ARG address space size 128
ARG phys mem size 512

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 20

  Segment 1 base  (grows negative) : 0x00000200 (decimal 512)
  Segment 1 limit                  : 20

Virtual Address Trace
  VA  0: 0x0000006b (decimal:  107) --> SEGMENTATION VIOLATION (SEG1)
```

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

Seg 0 base = 0
Seg 0 bound = 2
Seg 1 base = 128
Seg 1 bound = 2

```
D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python segmentation.py -a 16 -p 128 -A 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 --b0 0 --l0 2 --b1 128 --l1 2 -c
ARG seed 0
ARG address space size 16
ARG phys mem size 128

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 2

  Segment 1 base  (grows negative) : 0x00000080 (decimal 128)
  Segment 1 limit                  : 2

Virtual Address Trace
  VA  0: 0x00000000 (decimal:    0) --> VALID in SEG0: 0x00000000 (decimal:    0)
  VA  1: 0x00000001 (decimal:    1) --> VALID in SEG0: 0x00000001 (decimal:    1)
  VA  2: 0x00000002 (decimal:    2) --> SEGMENTATION VIOLATION (SEG0)
  VA  3: 0x00000003 (decimal:    3) --> SEGMENTATION VIOLATION (SEG0)
  VA  4: 0x00000004 (decimal:    4) --> SEGMENTATION VIOLATION (SEG0)
  VA  5: 0x00000005 (decimal:    5) --> SEGMENTATION VIOLATION (SEG0)
  VA  6: 0x00000006 (decimal:    6) --> SEGMENTATION VIOLATION (SEG0)
  VA  7: 0x00000007 (decimal:    7) --> SEGMENTATION VIOLATION (SEG0)
  VA  8: 0x00000008 (decimal:    8) --> SEGMENTATION VIOLATION (SEG1)
  VA  9: 0x00000009 (decimal:    9) --> SEGMENTATION VIOLATION (SEG1)
  VA 10: 0x0000000a (decimal:   10) --> SEGMENTATION VIOLATION (SEG1)
  VA 11: 0x0000000b (decimal:   11) --> SEGMENTATION VIOLATION (SEG1)
  VA 12: 0x0000000c (decimal:   12) --> SEGMENTATION VIOLATION (SEG1)
  VA 13: 0x0000000d (decimal:   13) --> SEGMENTATION VIOLATION (SEG1)
  VA 14: 0x0000000e (decimal:   14) --> VALID in SEG1: 0x0000007e (decimal:  126)
  VA 15: 0x0000000f (decimal:   15) --> VALID in SEG1: 0x0000007f (decimal:  127)
```

(2.4)
We just need to set the bounds to be about 90% of the address space. For instance:
As 256 x (90/100) ~= 115.

```
python segmentation.py -a 256 -p 1024 --b0 0 --l0 115 --b1 1024 --l1 115 -c -n 100
```

```
D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python segmentation.py -a 256 -p 1024 --b0 0 --l0 115 --b1 1024 --l1 115 -c -n 100
ARG seed 0
ARG address space size 256
ARG phys mem size 1024

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 115

  Segment 1 base  (grows negative) : 0x00000400 (decimal 1024)
  Segment 1 limit                  : 115

Virtual Address Trace
  VA  0: 0x000000d8 (decimal:  216) --> VALID in SEG1: 0x000003d8 (decimal:  984)
  VA  1: 0x000000c2 (decimal:  194) --> VALID in SEG1: 0x000003c2 (decimal:  962)
  VA  2: 0x0000006b (decimal:  107) --> VALID in SEG0: 0x0000006b (decimal:  107)
  VA  3: 0x00000042 (decimal:   66) --> VALID in SEG0: 0x00000042 (decimal:   66)
  VA  4: 0x00000082 (decimal:  130) --> SEGMENTATION VIOLATION (SEG1)
  VA  5: 0x00000067 (decimal:  103) --> VALID in SEG0: 0x00000067 (decimal:  103)
  VA  6: 0x000000c8 (decimal:  200) --> VALID in SEG1: 0x000003c8 (decimal:  968)
  VA  7: 0x0000004d (decimal:   77) --> VALID in SEG0: 0x0000004d (decimal:   77)
  VA  8: 0x0000007a (decimal:  122) --> SEGMENTATION VIOLATION (SEG0)
  VA  9: 0x00000095 (decimal:  149) --> VALID in SEG1: 0x00000395 (decimal:  917)
  VA 10: 0x000000e8 (decimal:  232) --> VALID in SEG1: 0x000003e8 (decimal: 1000)
  VA 11: 0x00000081 (decimal:  129) --> SEGMENTATION VIOLATION (SEG1)
  VA 12: 0x00000048 (decimal:   72) --> VALID in SEG0: 0x00000048 (decimal:   72)
  VA 13: 0x000000c1 (decimal:  193) --> VALID in SEG1: 0x000003c1 (decimal:  961)
  VA 14: 0x0000009e (decimal:  158) --> VALID in SEG1: 0x0000039e (decimal:  926)
  VA 15: 0x00000040 (decimal:   64) --> VALID in SEG0: 0x00000040 (decimal:   64)
  VA 16: 0x000000e8 (decimal:  232) --> VALID in SEG1: 0x000003e8 (decimal: 1000)
  VA 17: 0x000000fb (decimal:  251) --> VALID in SEG1: 0x000003fb (decimal: 1019)
  VA 18: 0x000000cf (decimal:  207) --> VALID in SEG1: 0x000003cf (decimal:  975)
  VA 19: 0x000000e6 (decimal:  230) --> VALID in SEG1: 0x000003e6 (decimal:  998)
  VA 20: 0x0000004f (decimal:   79) --> VALID in SEG0: 0x0000004f (decimal:   79)
  VA 21: 0x000000ba (decimal:  186) --> VALID in SEG1: 0x000003ba (decimal:  954)
  VA 22: 0x000000e6 (decimal:  230) --> VALID in SEG1: 0x000003e6 (decimal:  998)
  VA 23: 0x000000af (decimal:  175) --> VALID in SEG1: 0x000003af (decimal:  943)
  VA 24: 0x00000078 (decimal:  120) --> SEGMENTATION VIOLATION (SEG0)
  VA 25: 0x00000019 (decimal:   25) --> VALID in SEG0: 0x00000019 (decimal:   25)
  VA 26: 0x0000006f (decimal:  111) --> VALID in SEG0: 0x0000006f (decimal:  111)
  VA 27: 0x0000009c (decimal:  156) --> VALID in SEG1: 0x0000039c (decimal:  924)
  VA 28: 0x000000e9 (decimal:  233) --> VALID in SEG1: 0x000003e9 (decimal: 1001)
  VA 29: 0x000000f7 (decimal:  247) --> VALID in SEG1: 0x000003f7 (decimal: 1015)
  VA 30: 0x0000007a (decimal:  122) --> SEGMENTATION VIOLATION (SEG0)
  VA 31: 0x000000dd (decimal:  221) --> VALID in SEG1: 0x000003dd (decimal:  989)
  VA 32: 0x00000042 (decimal:   66) --> VALID in SEG0: 0x00000042 (decimal:   66)
  VA 33: 0x000000ce (decimal:  206) --> VALID in SEG1: 0x000003ce (decimal:  974)
  VA 34: 0x0000008c (decimal:  140) --> SEGMENTATION VIOLATION (SEG1)
  VA 35: 0x00000003 (decimal:    3) --> VALID in SEG0: 0x00000003 (decimal:    3)
```

(2.5)

Yes, we can run the simulator such that no virtual addresses are valid by setting the value of

- segment 0 limit register(-l) = 0
- segment 1 limit register(-L) = 0
- segment 0 base register(-b) = 0
- segment 1 base register(-B) = 0

`python segmentation.py -b 0 -l 0 -B 0 -L 0 -c`

```
D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python segmentation.py -b 0 -l 0 -B 0 -L 0 -c
ARG seed 0
ARG address space size 1k
ARG phys mem size 16k

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 0

  Segment 1 base  (grows negative) : 0x00000000 (decimal 0)
  Segment 1 limit                  : 0

Virtual Address Trace
  VA  0: 0x00000360 (decimal:  864) --> SEGMENTATION VIOLATION (SEG1)
  VA  1: 0x00000308 (decimal:  776) --> SEGMENTATION VIOLATION (SEG1)
  VA  2: 0x000001ae (decimal:  430) --> SEGMENTATION VIOLATION (SEG0)
  VA  3: 0x00000109 (decimal:  265) --> SEGMENTATION VIOLATION (SEG0)
  VA  4: 0x0000020b (decimal:  523) --> SEGMENTATION VIOLATION (SEG1)
```

Q3. Paging: Linear Size

```
D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python paging-linear-size.py -c
ARG bits in virtual address 32
ARG page size 4k
ARG pte size 4

Recall that an address has two components:
[ Virtual Page Number (VPN) | Offset ]

The number of bits in the virtual address: 32
The page size: 4096 bytes
Thus, the number of bits needed in the offset: 12
Which leaves this many bits for the VPN: 20
Thus, a virtual address looks like this:

V V V V V V V V V V V V V V V V V V V V | 0 0 0 0 0 0 0 0 0 0 0 0

where V is for a VPN bit and O is for an offset bit
To compute the size of the linear page table, we need to know:
- The # of entries in the table, which is 2^(num of VPN bits): 1048576.0
- The size of each page table entry, which is: 4
And then multiply them together. The final result:
  4194304 bytes
  in KB: 4096.0
  in MB: 4.0
```

In the default case, there are 32 bits in the virtual address space, the page size is 4096 bytes (which requires 12 digits in a binary representation), thus the number of bits in the offset = 12, and the page table entry size = 4 bytes. Which leaves 32 - 12 = 20 bits for the VPN.

The number of entries in the tale = $2^{(number\ of\ VPN\ bits)}$ = $2^{(20)}$ = 1048576.
Therefore, the size of the linear page table
= (number of entries on the table) x (the size of each page table entry)
= 1048576 x 4
= 4194304 bytes
= **4 MB**

```
D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python paging-linear-size.py -v 64 -p 64g -e 64 -c
ARG bits in virtual address 64
ARG page size 64g
ARG pte size 64

Recall that an address has two components:
[ Virtual Page Number (VPN) | Offset ]

The number of bits in the virtual address: 64
The page size: 68719476736 bytes
Thus, the number of bits needed in the offset: 36
Which leaves this many bits for the VPN: 28
Thus, a virtual address looks like this:

V V V V V V V V V V V V V V V V V V V V V V V V V V V V | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

where V is for a VPN bit and O is for an offset bit
To compute the size of the linear page table, we need to know:
- The # of entries in the table, which is 2^(num of VPN bits): 268435456.0
- The size of each page table entry, which is: 64
And then multiply them together. The final result:
  17179869184 bytes
  in KB: 16777216.0
  in MB: 16384.0
```

In this case, there are 64 bits in the virtual address space, the page size is 64 GB (which requires 36 digits in a binary representation), thus the number of bits in the offset = 36, and the page table entry size = 64 bytes. Which leaves 64 - 36 = 28 bits for the VPN.

The number of entries in the tale = $2^{(number\ of\ VPN\ bits)}$ = $2^{(28)}$ = 268435456.

Therefore, the size of the linear page table

= (number of entries on the table) x (the size of each page table entry)

= 268435456 x 64

= 17179869184 bytes

= **16384 MB**

```
D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python paging-linear-size.py -v 4 -p 4 -e 4 -c
ARG bits in virtual address 4
ARG page size 4
ARG pte size 4

Recall that an address has two components:
[ Virtual Page Number (VPN) | Offset ]

The number of bits in the virtual address: 4
The page size: 4 bytes
Thus, the number of bits needed in the offset: 2
Which leaves this many bits for the VPN: 2
Thus, a virtual address looks like this:

V V | O O

where V is for a VPN bit and O is for an offset bit
To compute the size of the linear page table, we need to know:
- The # of entries in the table, which is 2^(num of VPN bits): 4.0
- The size of each page table entry, which is: 4
And then multiply them together. The final result:
  16 bytes
  in KB: 0.015625
  in MB: 1.52587890625e-05
```

In this case, there are 4 bits in the virtual address space, the page size is 4 bytes (which requires 2 digits in a binary representation), thus the number of bits in the offset = 2, and the page table entry size = 4 bytes. Which leaves 4 - 2 = 2 bits for the VPN.

The number of entries in the tale = $2^{(number\ of\ VPN\ bits)}$ = $2^{(2)}$ = 4.

Therefore, the size of the linear page table

= (number of entries on the table) x (the size of each page table entry)

= 4 x 4

= **16 bytes**

Q4. Paging: Linear Translate

(4.1) Formula to calculate Linear Page Table Size:

$$PTS = \frac{(VAS \times PTES)}{PS}$$

Where,
PTS ⇒Page Table Size
VAS ⇒ Virtual Address Space
PTES ⇒ Page Table Entry Size
PS ⇒ Page Size

Variation of Page Table Size with Address Space:

Command	Page Table Size
python paging-linear-translate.py -P 1k -a **1m** -p 512m -v -n 0	1024
python paging-linear-translate.py -P 1k -a **2m** -p 512m -v -n 0	2048
python paging-linear-translate.py -P 1k -a **4m** -p 512m -v -n 0	4096

From the formula, we can see that the Linear Page Table Size is directly proportional to Virtual Address Space Size. The above table also emphasizes the same point that the Linear Page Table Size increases with increasing Virtual Address Space Size. So the Page Table Size increases as the Address Space Size grows because we need more pages to cover the whole address space.

Variation of Page Table Size with Page Size:

Command	Page Table Size
python paging-linear-translate.py -P **1k** -a 1m -p 512m -v -n 0	1024
python paging-linear-translate.py -P **2k** -a 1m -p 512m -v -n 0	512
python paging-linear-translate.py -P **4k** -a 1m -p 512m -v -n 0	256

From the formula, we can see that the Linear Page Table Size is inversely proportional to Page Size. The above table also emphasizes the same point that the Linear Page Table Size decreases with increasing Page Size. So when the Page Size increases the Page Table Size decreases because we need fewer pages (because they are bigger in size) to cover the whole address space.

Thus we shouldn't use big pages in general because they cause Internal Fragmentation in the pages. It would be a waste of memory. Because most processes use very little memory.

(4.2) The -u flag changes the fraction of mappings that are valid, from 0% (-u 0) up to 100% (-u 100). The default is 50, which means that roughly 1/2 of the pages in the virtual address space will be valid. All the entries are invalid in VA to PA translation when 0% of address space is used. That's why every memory access failed to succeed. Thus it is illegal to allocate 0% of pages to the address space. From the output of the given 5 commands, we see that as the percentage of pages that are allocated or usage of address space is increased, more and more memory access operations become valid, however, free space decreases. Finally, all the entries become valid when 100% of the address space is used.

python paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 0 -c

```
Page Table (from entry 0 down to the max size)
  [       0]
0x00000000
  [       1]
0x00000000
  [       2]
0x00000000
  [       3]
0x00000000
  [       4]
0x00000000
  [       5]
0x00000000
  [       6]
0x00000000
  [       7]
0x00000000
  [       8]
0x00000000
  [       9]
0x00000000
  [      10]
0x00000000
  [      11]
0x00000000
  [      12]
0x00000000
  [      13]
0x00000000
  [      14]
0x00000000
  [      15]
0x00000000

Virtual Address Trace
  VA 0x00003a39 (decimal:    14905) -->  Invalid (VPN 14 not valid)
  VA 0x00003ee5 (decimal:    16101) -->  Invalid (VPN 15 not valid)
  VA 0x000033da (decimal:    13274) -->  Invalid (VPN 12 not valid)
  VA 0x000039bd (decimal:    14781) -->  Invalid (VPN 14 not valid)
  VA 0x000013d9 (decimal:     5081) -->  Invalid (VPN 4 not valid)
```

python paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 25 -c

```
Page Table (from entry 0 down to the max size)
  [       0]
0x80000018
  [       1]
0x00000000
  [       2]
0x00000000
  [       3]
0x00000000
  [       4]
0x00000000
  [       5]
0x80000009
  [       6]
0x00000000
  [       7]
0x00000000
  [       8]
0x80000010
  [       9]
0x00000000
  [      10]
0x80000013
  [      11]
0x00000000
  [      12]
0x8000001f
  [      13]
0x8000001c
  [      14]
0x00000000
  [      15]
0x00000000

Virtual Address Trace
  VA 0x00003986 (decimal:    14726) -->  Invalid (VPN 14 not valid)
  VA 0x00002bc6 (decimal:    11206) --> 00004fc6 (decimal    20422) [VPN 10]
  VA 0x00001e37 (decimal:     7735) -->  Invalid (VPN 7 not valid)
  VA 0x00000671 (decimal:     1649) -->  Invalid (VPN 1 not valid)
  VA 0x00001bc9 (decimal:     7113) -->  Invalid (VPN 6 not valid)
```

python paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 50 -c

```
Page Table (from entry 0 down to the max size)
  [       0]
0x80000018
  [       1]
0x00000000
  [       2]
0x00000000
  [       3]
0x8000000c
  [       4]
0x80000009
  [       5]
0x00000000
  [       6]
0x8000001d
  [       7]
0x80000013
  [       8]
0x00000000
  [       9]
0x8000001f
  [      10]
0x8000001c
  [      11]
0x00000000
  [      12]
0x8000000f
  [      13]
0x00000000
  [      14]
0x00000000
  [      15]
0x80000008

Virtual Address Trace
  VA 0x00003385 (decimal:    13189) --> 00003f85 (decimal    16261) [VPN 12]
  VA 0x0000231d (decimal:     8989) -->  Invalid (VPN 8 not valid)
  VA 0x000000e6 (decimal:      230) --> 000060e6 (decimal    24806) [VPN 0]
  VA 0x00002e0f (decimal:    11791) -->  Invalid (VPN 11 not valid)
  VA 0x00001986 (decimal:     6534) --> 00007586 (decimal    30086) [VPN 6]
```

python paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 75 -c

```
Page Table (from entry 0 down to the max size)
  [       0]
0x80000018
  [       1]
0x80000008
  [       2]
0x8000000c
  [       3]
0x80000009
  [       4]
0x80000012
  [       5]
0x80000010
  [       6]
0x8000001f
  [       7]
0x8000001c
  [       8]
0x80000017
  [       9]
0x80000015
  [      10]
0x80000003
  [      11]
0x80000013
  [      12]
0x8000001e
  [      13]
0x8000001b
  [      14]
0x80000019
  [      15]
0x80000000

Virtual Address Trace
  VA 0x00002e0f (decimal:    11791) --> 00004e0f (decimal    19983) [VPN 11]
  VA 0x00001986 (decimal:     6534) --> 00007d86 (decimal    32134) [VPN 6]
  VA 0x000034ca (decimal:    13514) --> 00006cca (decimal    27850) [VPN 13]
  VA 0x00002ac3 (decimal:    10947) --> 00000ec3 (decimal     3779) [VPN 10]
  VA 0x00000012 (decimal:       18) --> 00006012 (decimal    24594) [VPN 0]
```

python paging-linear-translate.py -P 1k -a 16k -p 32k -v -u 100 -c

```
Page Table (from entry 0 down to the max size)
  [       0]
0x80000018
  [       1]
0x80000008
  [       2]
0x8000000c
  [       3]
0x80000009
  [       4]
0x80000012
  [       5]
0x80000010
  [       6]
0x8000001f
  [       7]
0x8000001c
  [       8]
0x80000017
  [       9]
0x80000015
  [      10]
0x80000003
  [      11]
0x80000013
  [      12]
0x8000001e
  [      13]
0x8000001b
  [      14]
0x80000019
  [      15]
0x80000000

Virtual Address Trace
  VA 0x00002e0f (decimal:    11791) --> 00004e0f (decimal    19983) [VPN 11]
  VA 0x00001986 (decimal:     6534) --> 00007d86 (decimal    32134) [VPN 6]
  VA 0x000034ca (decimal:    13514) --> 00006cca (decimal    27850) [VPN 13]
  VA 0x00002ac3 (decimal:    10947) --> 00000ec3 (decimal     3779) [VPN 10]
  VA 0x00000012 (decimal:       18) --> 00006012 (decimal    24594) [VPN 0]
```

(4.3) The 1st and the 3rd are unrealistic in nature because the size of the 3rd-page table is too big whereas the size of the 1st-page table is small. This can be said by using the formula given below. The page size must also be considered. In this case, it is very small as compared to the actual physical address size of 1024 units and thus isn't appropriate to implement.

$$PTS = \frac{(VAS \times PTES)}{PS}$$

python paging-linear-translate.py -P 8 -a 32 -p 1024 -v -s 1 -c

```
D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python paging-linear-translate.py -P 8 -a 32 -p 1024 -v -s 1 -c
ARG seed 1
ARG address space size 32
ARG phys mem size 1024
ARG page size 8
ARG verbose True
ARG addresses -1


The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
  If the bit is 1, the rest of the entry is the PFN.
  If the bit is 0, the page is not valid.
Use verbose mode (-v) if you want to print the VPN # by
each entry of the page table.

Page Table (from entry 0 down to the max size)
  [       0]
0x00000000
  [       1]
0x80000061
  [       2]
0x00000000
  [       3]
0x00000000

Virtual Address Trace
  VA 0x0000000e (decimal:       14) --> 0000030e (decimal      782) [VPN 1]
  VA 0x00000014 (decimal:       20) -->  Invalid (VPN 2 not valid)
  VA 0x00000019 (decimal:       25) -->  Invalid (VPN 3 not valid)
  VA 0x00000003 (decimal:        3) -->  Invalid (VPN 0 not valid)
  VA 0x00000000 (decimal:        0) -->  Invalid (VPN 0 not valid)
```

python paging-linear-translate.py -P 8k -a 32k -p 1m -v -s 2 -c

```
D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python paging-linear-translate.py -P 8k -a 32k -p 1m -v -s 2 -c
ARG seed 2
ARG address space size 32k
ARG phys mem size 1m
ARG page size 8k
ARG verbose True
ARG addresses -1


The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
  If the bit is 1, the rest of the entry is the PFN.
  If the bit is 0, the page is not valid.
Use verbose mode (-v) if you want to print the VPN # by
each entry of the page table.

Page Table (from entry 0 down to the max size)
  [       0]
0x80000079
  [       1]
0x00000000
  [       2]
0x00000000
  [       3]
0x8000005e

Virtual Address Trace
  VA 0x000055b9 (decimal:    21945) -->  Invalid (VPN 2 not valid)
  VA 0x00002771 (decimal:    10097) -->  Invalid (VPN 1 not valid)
  VA 0x00004d8f (decimal:    19855) -->  Invalid (VPN 2 not valid)
  VA 0x00004dab (decimal:    19883) -->  Invalid (VPN 2 not valid)
  VA 0x00004a64 (decimal:    19044) -->  Invalid (VPN 2 not valid)
```

python paging-linear-translate.py -P 1m -a 256m -p 512m -v -s 3 -c

```
D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python paging-linear-translate.py -P 1m -a 256m -p 512m -v -s 3 -c
ARG seed 3
ARG address space size 256m
ARG phys mem size 512m
ARG page size 1m
ARG verbose True
ARG addresses -1


The format of the page table is simple:
The high-order (left-most) bit is the VALID bit.
  If the bit is 1, the rest of the entry is the PFN.
  If the bit is 0, the page is not valid.
Use verbose mode (-v) if you want to print the VPN # by
each entry of the page table.

Page Table (from entry 0 down to the max size)
  [       0]
0x00000000
  [       1]
0x800000bd
  [       2]
0x80000140
  [       3]
0x00000000
  [       4]
0x00000000
  [       5]
0x80000084
  [       6]
0x00000000
  [       7]
0x800000f0
  [       8]
0x800000f3
  [       9]
0x8000004d
  [      10]
0x800001bc
  [      11]
0x8000017b
  [      12]
0x80000020
  [      13]
0x8000012e
  [      14]
0x00000000
```

```
0x00000000
  [     252]
0x00000000
  [     253]
0x00000000
  [     254]
0x80000159
  [     255]
0x00000000

Virtual Address Trace
  VA 0x0308b24d (decimal: 50901581) --> 1f68b24d (decimal 526955085) [VPN 48]
  VA 0x042351e6 (decimal: 69423590) -->  Invalid (VPN 66 not valid)
  VA 0x02feb67b (decimal: 50247291) --> 0a9eb67b (decimal 178173563) [VPN 47]
  VA 0x0b46977d (decimal: 189175677) -->  Invalid (VPN 180 not valid)
  VA 0x0dbcceb4 (decimal: 230477492) --> 1f2cceb4 (decimal 523030196) [VPN 219]
```

(4.4) The program doesn't work when:

- Page Size is zero:
  `python paging-linear-translate.py -P 0 -v -c`

```
D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python paging-linear-translate.py -P 0 -v -c
ARG seed 0
ARG address space size 16k
ARG phys mem size 64k
ARG page size 0
ARG verbose True
ARG addresses -1

Traceback (most recent call last):
  File "D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6\paging-linear-translate.py", line 85, in <module>
    mustbemultipleof(asize, pagesize, 'address space must be a multiple of the pagesize')
  File "D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6\paging-linear-translate.py", line 14, in mustbemultipleof
    if (int(float(bignum)/float(num)) != (int(bignum) / int(num))):
ZeroDivisionError: float division by zero
```

- Address Space Size is zero:
  `python paging-linear-translate.py -a 0 -v -c`

```
D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python paging-linear-translate.py -a 0 -v -c
ARG seed 0
ARG address space size 0
ARG phys mem size 64k
ARG page size 4k
ARG verbose True
ARG addresses -1

Error: must specify a non-zero address-space size.
```

- Physical Memory Size is less or equal to the Address Space Size.
  `python paging-linear-translate.py -a 65k -v -c`

```
D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python paging-linear-translate.py -a 65k -v -c
ARG seed 0
ARG address space size 65k
ARG phys mem size 64k
ARG page size 4k
ARG verbose True
ARG addresses -1

Error: physical memory size must be GREATER than address space size (for this simulation)
```

- Address Space must be a multiple of the pagesize:
  python paging-linear-translate.py -P 5k -v -c

```
D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python paging-linear-translate.py -P 5k -v -c
ARG seed 0
ARG address space size 16k
ARG phys mem size 64k
ARG page size 5k
ARG verbose True
ARG addresses -1

Error in argument: address space must be a multiple of the pagesize
```

- Physical Memory must be a multiple of the pagesize:
  python paging-linear-translate.py -P 3 -a 3 -p 7 -v -c

```
D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python paging-linear-translate.py -P 3 -a 3 -p 7 -v -c
ARG seed 0
ARG address space size 3
ARG phys mem size 7
ARG page size 3
ARG verbose True
ARG addresses -1

Error in argument: physical memory must be a multiple of the pagesize
```

- Physical Memory Size is zero:
  python paging-linear-translate.py -p 0 -v -c

```
D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python paging-linear-translate.py -p 0 -v -c
ARG seed 0
ARG address space size 16k
ARG phys mem size 0
ARG page size 4k
ARG verbose True
ARG addresses -1

Error: must specify a non-zero physical memory size.
```

- Page size or address space size is not the power of 2. This is because it will result in discontinuous addresses.
  python paging-linear-translate.py -P 3 -a 3 -p 6 -v -c

```
D:\IIT-Dharwad\6th-Sem\OS-Lab\lab6\cs314oslaboratory6>python paging-linear-translate.py -P 3 -a 3 -p 6 -v -c
ARG seed 0
ARG address space size 3
ARG phys mem size 6
ARG page size 3
ARG verbose True
ARG addresses -1

Error in argument: address space must be a power of 2
```