segmentation

./segmentation.py

Seg 0 base (grows positive) : 6890
Seg 0 limit : 472

Seg 1 base (grows negative) : 4692
Seg 1 limit : 450

X VA 0: 523 → binary 00....11
top bit = 1 ⇒ seg = 1
523 < 450 x
Seg violation

X VA 1: 414 → top bit = 0 ⇒ seg = 0
414 < 472 ✓
Valid 7304

X VA 2: 802 → top bit = 0 ⇒ seg = 0
802 < 472 x
Seg violation

X VA 3: 310 → top bit = 0 ⇒ seg = 0
310 < 472 ✓

X VA4 : 488 → valid 7200
top bit = 0 ⇒ seg = 0
488 < 472 x
Seg violation

Figuring out of top bit
vaddr >= asize/2 (this works since only logical segment it figures out, then base, limit check actual VA can be valid or not
523 >= 1024
523 >= 512 ✓ seg 1

paddr = nbase1 + (vaddr - asize)
= 4692 + 523 - 1024
= 4191

paddr < base = 4242
Seg violation

VA1 : 414 >= asize/2 = 512 x
seg 0 414 < 472
PA = 6890 + 414 = 7304

VA2 : 802 >= 512 ✓

Seg 1 = 802 - asize = 802 - 1024
= -222

abs(-222) < 450 ✓

PA = 4692 - 222 = 4470

PA > base 4470 > 4242

VA 3 : 310 >= 512 X

Seg 0 310 < 472

PA = 6890 + 310 = 7200

VA 4 : 488 >= 512 X

Seg 0 488 < 472 X

Seg violation

./segmentation.py -s 100 -a 16 -p 48

Seg 0 base = 36, limit = 4

Seg 1 base = 25 limit = 5

VA 0 : 12 >= asize/2 = 16/2 = 8 ✓

Seg 1 offset = 12 - 16 = -4, abs(-4) < limit

PA = 25 + (-4) = 21 valid

VA 1 : 8 >= 8 ✓

seg1 offset = 8 − 16 = −8 abs(-8) < limit ✗

Seg violation

VA2: 1 >= 8 ✗

seg0 1 >= 4 ✗ Valid

PA = base + 1 = 37

VA3: 7 >= 8 ✗

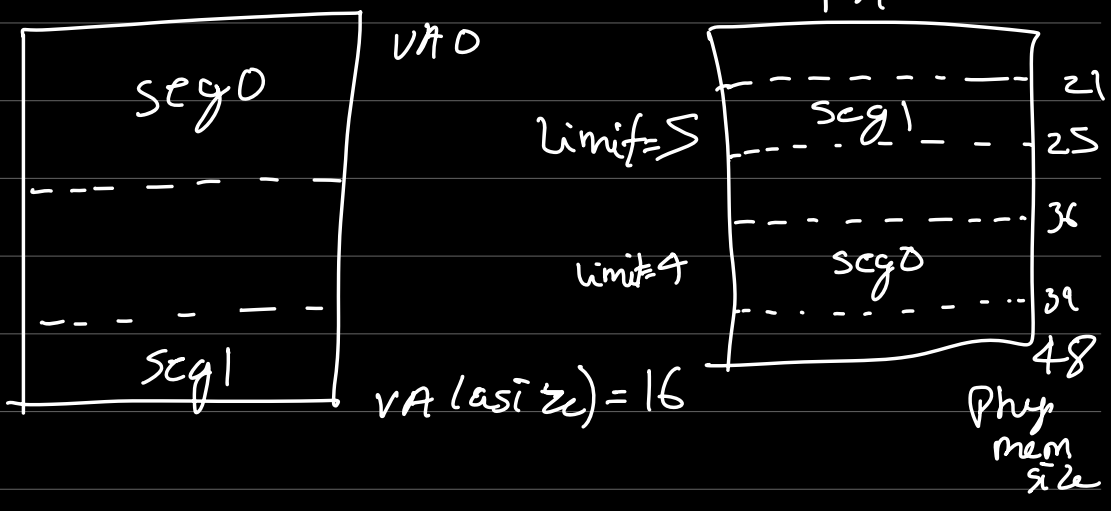
seg0 7 >= 4 ✓

seg violation

VA4: 0 >= 8 ✗

seg0 0 >= 4 ✗

PA1 = base + 0 = 36



-s 100 -a 16 -p 98 --b0 20 --l0 4
--b1 40 --l1 5
with given base/limit for segments

seg0: base = 20, limit = 4
seg1: base = 40, limit = 5

VA0: 2 >= asize/2 = 8 x
seg0 2 >= limit 4 x
PA = base + 2 = 22 Valid

VA1: 7 >= 8 x
seg0 7 >= 4 ✓
seg violation

VA2: 12 >= 8 ✓
seg1 offset = 12 - asize = -4
abs(-4) < 5 ✓ Valid
PA = base + offset = 36

VA3: 11 >= 8
seg1 off-set = 11 - 16 = -5

PA = 40 - 5 = 35 < 35 ×

Valid

① -a 128 -p512 -b 0 -l 20 -B 512 -L 20 -s 0

asize = 128, Phy mem size = 512

Seg 0: base = 0, limit = 20

Seg 1: base = 512, limit = 20

VA 0 : 108 >= asize/2 = 64 ✓

seg1, offset = 108 - 128 = |-20| <= limit 20

PA = 512 + (-20) = 492 ✓

VA 1 : 97 >= 64 ✓

seg1, offset = 97 - 128 = -31

abs(-31) <= 20 ×

Seg. violation

VA 2 : 53 >= 64 ×

seg0, 53 <= limit 20 ×

Seg violation

VA 3: 33 >= 64 x

seg0, 33 <= limit 20 x

seg violation

VA 4: 65 >= 64 ✓

seg1, offset = 65 - 128 = -63

abs(-63) <= 20 x

seg violation

same with seed = 1

VA 0: 17 >= 64 x

seg0, 17 <= limit 20 ✓

PA = base + 17 = 17
↘0

VA 1: 108 >= 64 ✓

seg1, offset 108 - 128 = -20

abs(-20) <= 20 ✓

PA = base + (-20) = 512 - 20 = 492

VA 2: 32 >= 64 x

seg0, 32 <= 20 x

Seg. violation

VA3: 637/64 x

seg0, 63 <= 20 x

SV

VA4: 97 // 64 ✓

seg1, offset = 97 - 128 = -31

abs(-31) <= 20 x

SV

With seed = 2

VA0: 122 // 64 ✓

seg1, offset = 122 - 128 = -6

abs(-6) <= 20 ✓

PA = 512 - 6 = 506

VA1: 121 // 64 ✓

seg1, offset = 121 - 128 = -7

PA = 505

VA2: 7 // 64 x

seg0, 7 < 20 ✓

PA = 0+7 = 7

VA3 : 107 > 64 ✗

seg0, 10 ≤ 20 ✓

PA = 10

VA4 : 106 > 64 ✓

seg1, offset = 106-128 = -22

abs(-22) <= 20 ✗

SV

(2) Seg 0, highest VA = 20 ✗ (19)

Seg 1, lowest VA = 108

Lowest illegal address = 21 ✗ (20)

highest illegal address = 107, (128 out of bounds)

Seg0, highest VA legal = 19

seg0, VA < limit Validity check

[0, 20)

seg1, [492, 512)

(3) asize = 16, physical mem size = 128

VA0 : 0 → valid
VA1 : 1 → valid
VA2 : 2 → SV
VA3 : 13 → SV
VA4 : 14 → valid
VA5 : 15 → valid

since "2" is SV ⇒ l0 = 2
"13" is SV
⇒ 13 ≥ 8 ✓
seg1 offset = 13 − 16 = −3
abs(−3) <= l1 ✗
l1 = 2 ✓
since two valid 14, 15

b0 = 0, b1 = 128 } other values would work too
just need two valid addresses window

```
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)
```

④ 90% valid randomly generated virtual addresses

available

the no. of VAs should be close to no. of available physical addresses in segements; 90%.

Eg:- asize = 16
Phy addr = 128
} Taking same values from previous question

Now let's update limits to fit more addresses

90% valid = 9 valid out of 10 addresses

15 valid addresses out of 16 = 0.93

base0 = 0, limit0 = 7

base1 = 128, limit1 = 8

```
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                  : 7

  Segment 1 base  (grows negative) : 0x00000080 (decimal 128)
  Segment 1 limit                  : 8

Virtual Address Trace
  VA  0: 0x0000000d (decimal:   13) --> VALID in SEG1: 0x0000007d (decimal:  125)
  VA  1: 0x0000000c (decimal:   12) --> VALID in SEG1: 0x0000007c (decimal:  124)
  VA  2: 0x00000006 (decimal:    6) --> VALID in SEG0: 0x00000006 (decimal:    6)
  VA  3: 0x00000004 (decimal:    4) --> VALID in SEG0: 0x00000004 (decimal:    4)
  VA  4: 0x00000008 (decimal:    8) --> VALID in SEG1: 0x00000078 (decimal:  120)
  VA  5: 0x00000006 (decimal:    6) --> VALID in SEG0: 0x00000006 (decimal:    6)
  VA  6: 0x0000000c (decimal:   12) --> VALID in SEG1: 0x0000007c (decimal:  124)
  VA  7: 0x00000004 (decimal:    4) --> VALID in SEG0: 0x00000004 (decimal:    4)
  VA  8: 0x00000007 (decimal:    7) --> SEGMENTATION VIOLATION (SEG0)
  VA  9: 0x00000009 (decimal:    9) --> VALID in SEG1: 0x00000079 (decimal:  121)
```

0.9

```
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                  : 8

  Segment 1 base  (grows negative) : 0x00000080 (decimal 128)
  Segment 1 limit                  : 7

Virtual Address Trace
  VA  0: 0x0000000d (decimal:   13) --> VALID in SEG1: 0x0000007d (decimal:  125)
  VA  1: 0x0000000c (decimal:   12) --> VALID in SEG1: 0x0000007c (decimal:  124)
  VA  2: 0x00000006 (decimal:    6) --> VALID in SEG0: 0x00000006 (decimal:    6)
  VA  3: 0x00000004 (decimal:    4) --> VALID in SEG0: 0x00000004 (decimal:    4)
  VA  4: 0x00000008 (decimal:    8) --> SEGMENTATION VIOLATION (SEG1)
  VA  5: 0x00000006 (decimal:    6) --> VALID in SEG0: 0x00000006 (decimal:    6)
  VA  6: 0x0000000c (decimal:   12) --> VALID in SEG1: 0x0000007c (decimal:  124)
  VA  7: 0x00000004 (decimal:    4) --> VALID in SEG0: 0x00000004 (decimal:    4)
  VA  8: 0x00000007 (decimal:    7) --> VALID in SEG0: 0x00000007 (decimal:    7)
  VA  9: 0x00000009 (decimal:    9) --> VALID in SEG1: 0x00000079 (decimal:  121)
```

0.9

By just reducing limit by 1

0.8

```
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                  : 7

  Segment 1 base  (grows negative) : 0x00000080 (decimal 128)
  Segment 1 limit                  : 7

Virtual Address Trace
  VA  0: 0x0000000d (decimal:   13) --> VALID in SEG1: 0x0000007d (decimal:  125)
  VA  1: 0x0000000c (decimal:   12) --> VALID in SEG1: 0x0000007c (decimal:  124)
  VA  2: 0x00000006 (decimal:    6) --> VALID in SEG0: 0x00000006 (decimal:    6)
  VA  3: 0x00000004 (decimal:    4) --> VALID in SEG0: 0x00000004 (decimal:    4)
  VA  4: 0x00000008 (decimal:    8) --> SEGMENTATION VIOLATION (SEG1)
  VA  5: 0x00000006 (decimal:    6) --> VALID in SEG0: 0x00000006 (decimal:    6)
  VA  6: 0x0000000c (decimal:   12) --> VALID in SEG1: 0x0000007c (decimal:  124)
  VA  7: 0x00000004 (decimal:    4) --> VALID in SEG0: 0x00000004 (decimal:    4)
  VA  8: 0x00000007 (decimal:    7) --> SEGMENTATION VIOLATION (SEG0)
  VA  9: 0x00000009 (decimal:    9) --> VALID in SEG1: 0x00000079 (decimal:  121)
```

larger address spaces

0.85

```
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                  : 60

  Segment 1 base  (grows negative) : 0x00000080 (decimal 128)
  Segment 1 limit                  : 55

Virtual Address Trace
  VA  0: 0x0000006c (decimal:  108) --> VALID in SEG1: 0x0000006c (decimal:  108)
  VA  1: 0x00000061 (decimal:   97) --> VALID in SEG1: 0x00000061 (decimal:   97)
  VA  2: 0x00000035 (decimal:   53) --> VALID in SEG0: 0x00000035 (decimal:   53)
  VA  3: 0x00000021 (decimal:   33) --> VALID in SEG0: 0x00000021 (decimal:   33)
  VA  4: 0x00000041 (decimal:   65) --> SEGMENTATION VIOLATION (SEG1)
  VA  5: 0x00000033 (decimal:   51) --> VALID in SEG0: 0x00000033 (decimal:   51)
  VA  6: 0x00000064 (decimal:  100) --> VALID in SEG1: 0x00000064 (decimal:  100)
  VA  7: 0x00000026 (decimal:   38) --> VALID in SEG0: 0x00000026 (decimal:   38)
  VA  8: 0x0000003d (decimal:   61) --> SEGMENTATION VIOLATION (SEG0)
  VA  9: 0x0000004a (decimal:   74) --> VALID in SEG1: 0x0000004a (decimal:   74)
  VA 10: 0x00000074 (decimal:  116) --> VALID in SEG1: 0x00000074 (decimal:  116)
  VA 11: 0x00000040 (decimal:   64) --> SEGMENTATION VIOLATION (SEG1)
  VA 12: 0x00000024 (decimal:   36) --> VALID in SEG0: 0x00000024 (decimal:   36)
  VA 13: 0x00000060 (decimal:   96) --> VALID in SEG1: 0x00000060 (decimal:   96)
  VA 14: 0x0000004f (decimal:   79) --> VALID in SEG1: 0x0000004f (decimal:   79)
  VA 15: 0x00000020 (decimal:   32) --> VALID in SEG0: 0x00000020 (decimal:   32)
  VA 16: 0x00000074 (decimal:  116) --> VALID in SEG1: 0x00000074 (decimal:  116)
  VA 17: 0x0000007d (decimal:  125) --> VALID in SEG1: 0x0000007d (decimal:  125)
  VA 18: 0x00000067 (decimal:  103) --> VALID in SEG1: 0x00000067 (decimal:  103)
  VA 19: 0x00000073 (decimal:  115) --> VALID in SEG1: 0x00000073 (decimal:  115)
```

(5) No virtual addresses are valid

simple set limits for segments equal to 0.
Or bases to be out of physical address space

```
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                  : 0

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

Virtual Address Trace
  VA  0: 0x0000000d (decimal:   13) --> SEGMENTATION VIOLATION (SEG1)
  VA  1: 0x0000000c (decimal:   12) --> SEGMENTATION VIOLATION (SEG1)
  VA  2: 0x00000006 (decimal:    6) --> SEGMENTATION VIOLATION (SEG0)
  VA  3: 0x00000004 (decimal:    4) --> SEGMENTATION VIOLATION (SEG0)
  VA  4: 0x00000008 (decimal:    8) --> SEGMENTATION VIOLATION (SEG1)
  VA  5: 0x00000006 (decimal:    6) --> SEGMENTATION VIOLATION (SEG0)
  VA  6: 0x0000000c (decimal:   12) --> SEGMENTATION VIOLATION (SEG1)
  VA  7: 0x00000004 (decimal:    4) --> SEGMENTATION VIOLATION (SEG0)
  VA  8: 0x00000007 (decimal:    7) --> SEGMENTATION VIOLATION (SEG0)
  VA  9: 0x00000009 (decimal:    9) --> SEGMENTATION VIOLATION (SEG1)
```

```
ARG seed 0
ARG address space size 128
ARG phys mem size 512

Error: seg0 is not in physical memory
```