7. Stack grows negatively

Segment	Base	Size	Grows Positive?
Code	32K	2K	1
Heap	34K	2K	1
Stack	28K	2K	0

Table 16.2: **Segment Registers (With Negative-Growth Support)**

With the hardware understanding that segments can grow in the negative direction, the hardware must now translate such virtual addresses slightly differently. Let's take an example stack virtual address and translate it to understand the process.

In this example, assume we wish to access virtual address 15KB, which should map to physical address 27KB. Our virtual address, in binary form, thus looks like this: 11 1100 0000 0000 (hex 0x3C00). The hardware uses the top two bits (11) to designate the segment, but then we are left with an offset of 3KB. To obtain the correct negative offset, we must subtract the maximum segment size from 3KB: in this example, a segment can be 4KB, and thus the correct negative offset is 3KB minus 4KB which equals -1KB. We simply add the negative offset (-1KB) to the base (28KB) to arrive at the correct physical address: 27KB. The bounds check can be calculated by ensuring the absolute value of the negative offset is less than the segment's size.

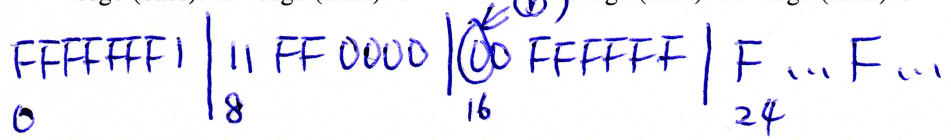
Assume virtual memory hardware that uses segmentation, and divides the address space in two by using the top bit of the virtual address. Each segment is thus relocated independently. What we'll be drawing in the question is what physical memory looks given some different parameters. We'll also label where a particular memory reference ends up.

For all questions, assume a virtual address space of size 16 bytes (yes tiny!) and a physical memory of size of 64 bytes. Thus, if we had a virtual address space placed in physical memory, it might look like this (with spaces between every 8 physical bytes):

```
0000FFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFF1111
```

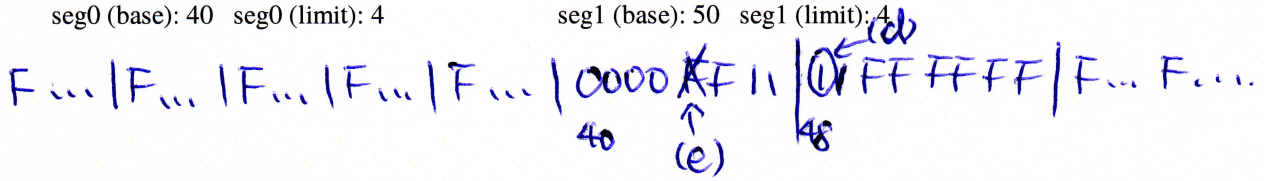
In this example, the segment 0 base register is 0, segment 1 base is 64 (it grows backwards), and both length registers are 4. 0's are used to record where segment 0 is in memory; 1's are for segment 1; F means free.

(a) What would physical memory look like if we had the following values instead? (draw a picture below)
seg0 (base): 12 seg0 (limit): 6 seg1 (base): 10 seg1 (limit): 3



(b) In your picture above, CIRCLE which byte of memory is accessed when the process generates a byte load of virtual address 4 (or DRAW AN X on the physical-memory address if the access is illegal)

(c) What would physical memory look like if we had the following values instead? (draw a picture below)
seg0 (base): 40 seg0 (limit): 4 seg1 (base): 50 seg1 (limit): 4



(d) In your picture above, CIRCLE which byte of memory is accessed when the process generates a byte load of virtual address 14 (or DRAW AN X on the physical-memory address if the access is illegal)

(e) In your picture above, CIRCLE which byte of memory is accessed when the process generates a byte load of virtual address 4 (or DRAW AN X on the physical-memory address if the access is illegal)