On my honor, I have not given, nor received, nor witnessed any unauthorized assistance on this work.

Print name and sign: ______________________________

Question:	1	2	3	4	Total
Points:	7	7	6	10	30
Score:					

1. (7 points) Dr. Summet has been asked to give an expert mini-lecture on base and bounds addressing. Here's what she says:

   "Base/bounds-based virtual memory is really easy. Imagine you have a base register and a bounds register in each CPU. The base points to the physical memory location where an address space is relocated; the bounds tells us how big such an address space can be. Let's do an example to understand this better.

   Assume we have the following base/bounds pair:

   ```
   Base : 0x1000
   Bounds : 0x10
   ```

   Now assume we have the following virtual memory references by a process (in this order):

   ```
   0x0, 0x4, 0x8, 0xc
   ```

   The corresponding physical addresses that will be referenced are (in this order):

   ```
   0x1000, 0x1004, 0x1008, FAULT (because this one is out of bounds)
   ```

   Make sense?"

   Not to put too fine a point on it, but Prof. Summet isn't correct and has several errors in her explanation. Point out her errors and correct them.

   Solution: No, because 0xc is a legal virtual address (clearly less than the bounds of 0x10). Should have translated to 0x100c instead of faulting

2. Assume a system is using base and bounds with the following system characteristics:

- a 1KB (1024 bytes) virtual address space
- a base register set to 10000
- a bounds register set to 100

For each of the following *physical memory* locations, give the corresponding virtual address translation or state that the physical memory location could not be legally accessed by the running program.

(a) (1 point) 0 Not Possible

(b) (1 point) 1000 Not Possible

(c) (1 point) 10000 0

(d) (2 points) 10050 50

(e) (2 points) 10100 Not Possible

Solution: 1KB virtual address space means 1024 bytes can be accessed by program, from 0 ... 1023. Base and bounds places this address space into physical memory at physical address 10,000 (as above).

However, only the first 100 virtual addresses are legal because the bounds register contains the value 100. So only virtual addresses 0 ... 99 will not generate a fault, which translates to physical addresses 10,000 ... 10,099.

3. (6 points) Segmentation is a generalization of base-and-bounds. Give one advantage of segmentation over base and bounds. Then give one disadvantage.

Solution:

Advantages:

- Less physical memory "waste" (here, "waste" means unitilized space) because space between a stack and heap need not be allocated.
- Better sharing of code in memory because read-only code segments can be shared. For example, a parent and child process can share the same code segment and would only need separate data (stack and heap) segments.

Disadvantages:

- More hardware support is needed.

- More OS issues to handle, such as compaction when memory becomes too fragmented

It's worth noting that speed isn't a positive or a negative. Translation with segmentation is not faster nor slower because both are hardware based.

4. (10 points) Assume the following in a simple segmentation system that supports two segments: one (positive growing) for code and a heap (Segment 0), and one (negative growing) for a stack (Segment 1):

   - Virtual address space size 128 bytes
   - Physical memory size 512

   Segment register information:

   - Segment 0 base (grows positive) : 0
   - Segment 0 limit : 20 (decimal)
   - Segment 1 base (grows negative) : 0x200 (decimal 512)
   - Segment 1 limit : 20 (decimal)

   Circle all of the following which are valid *virtual memory* accesses

   A. `0x1d` (decimal: 29)
   B. `0x7b` (decimal: 123)
   C. `0x10` (decimal: 16)
   D. `0x5a` (decimal: 90)
   E. `0x0a` (decimal: 10)

Solution: There is extra information in this problem that isn't needed. Interestingly, we can correctly answer this problem without doing two things (which were probably your first instincts): 1) Calculating the bit representation of the addresses, figuring out which segment they're in, etc. 2) Mapping the virtual addresses to physical addresses.

Instead, look just as the size of the V.A. space (128 bytes) and the two bounds/limit registers (20 each). This means that only virtual addresses 0 ... 19 are going to be valid for segment 0 (since it grows towards positive addresses). Likewise, only virtual addresses 127 ... 108 will be valid because segment 1 is growing towards smaller addresses.

Pictorally, the space for valid virtual address space will be:

```
-----------------------------
|      Segment 0            |  0
|   Loosely, code & heap    |  1
|                           | ...
|   Grows to higher addr    | 18
|                           | 19
----------------------------- <-----  20 bytes long
                                      (segment 0 limit reg)


----------------------------- <----- 20 bytes long
|       Segment 1           | 108      (128-20 bytes (segment 1 limit reg)
|     Loosely, stack        | ...
|   Grows to smaller addr   | 126
|                           | 127
-----------------------------  <--- Virt. Addr. 128
```

Understanding this, we can look at our list of valid addresses and say that only B, C, and E are valid since they are the only ones that fall into either of the valid ranges.

You could also have converted each address to its bit representation, figured out what segment it was in, calculate the range of correct addresses and then determined the correct/valid addresses. But that's extra work which isn't needed as long as you carefully read the question and understand what it's asking.

Scoring: Each selection is worth 2 points. If you left an item uncircled and it was supposed to be uncircled, give yourself 2 points. If you correctly circled an item, give yourself two points. You would not score points if you misidentified an address.