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	5	Total
Points:	2	1	12	6	9	30
Score:						

Consider a system with the following specs:

- 128 byte pages
- 1K (1024 byte) virtual address space
- 512 bytes physical memory

1. (2 points) How many bits do we need to represent a virtual address and a physical address?

   Solution: virtual: $2^{10} == 1024$, so we need 10 bits for a virtual address.

   physical: 512 is half of 1024, so we need 1 fewer bit to represent physcial addresses: 9 bits.

2. (1 point) Why do we need 3 bits to represent the virtual page number (VPN) and 2 bits to represent the physical frame number (PFN)?

   Solution: We need to address 8 pages (1024/128) in virtual address space and 4 pages (512/128) in physical memory. Thus, of our 10 bit virtual address (Q1), the upper 3 bits will be the VPN. Of our 9 bit virtual address (Q1), the upper 2 bits will be the PFN.

3. Our system has a linear page table which stores an 8-bit PTE in the form:
   `| 1 present bit | 3 RWX bits | 1 valid bit | 1 dirty bit | 2 bit PFN |`
   The page table currently contains:
   `0xCB`
   `0x7E`
   `0x43`
   `0x7D`
   `0xC9`
   `0xEA`
   `0x6C`
   `0xD8`

Translate the following virtual addresses to physical addresses. Give your answer as either hex or binary.

(a) (4 points) `0x280`

Solution: Breaking the VA into the VP and the offset: `101 0000000` Virtual page is 5. Page 5 PTE is `0xEA`. Convert to binary: 1110 1010. Means the present bit is 1 and the valid bit is 1. The PFN is 10 (2), so the physical address is PFN + offset == 10 0000000 == `0x100`

(b) (4 points) `0x05E`

Solution: Breaking the VA into the VP and the offset: `000 1010110`. Virtual page is 0. Page 0 PTE is `0xCB`. Convert to binary: `1100 1011`. The present bit is 1 and the valid bit is 1. The PFN is 11 (3), so the physical address is `11 1010110` == `0x3D6`

(c) (4 points) `0x3C9`

Solution: Breaking the VA into the VP and the offset: `111 1001001` Virtual page is 7. Page 5 PTE is `0xD8`. Convert to binary: 1101 1000. Means the present bit is 1 but the valid bit is 0. Thus, this page isn't valid and we can't translate the virtual address.

4. For each of the following virtual addresses, state whether the page is currently in physical memory or in disk swap space. If the page is in physical memory, give its PFN.

(a) (3 points) `0x1EC`

Solution: 1EC == 01 1110 1100 == VPN of 011
Looking up page 3 in the page table, we find the PTE: `0x7d`.
7d == 0111 1101. The present bit (leftmost bit) is 0 which means the page is in swap space on disk.

(b) (3 points) `0x235`

Solution: 235 == 10 0011 0101 == VPN of 100
Looking up page 4 in the page table, we find the PTE: `0xC9`
C9 == 1100 1001. The present bit (leftmost bit) is 1 which means the page is in physical memory at PFN 1.

5. The TLB has the following contents in each entry: a 3-bit VPN, a 2-bit PFN, and an 3-bit PID field (in that order). This TLB only has four entries, and they look like this:

`0x68`
`0x71`
`0xE0`
`0x9A`

(a) (3 points) If process 0 generates a virtual address with a VPN of 3, is this a TLB hit or miss?

hit, first entry of TLB

(b) (3 points) If process 2 generates a virtual address with a VPN of 5, is this a TLB hit or miss?

miss, not in TLB

(c) (2 points) Describe what happens when there is a TLB miss in a situation like this (where the TLB is full).

Solution: If there's a TLB miss, we must look the VPN up in the page table. We then have to update the TLB with that info. Depending on the replacement algorithm, a translation will need to be kicked out of the TLB. At that point, we'll have room for the most recently used translation.

(If the page isn't in physical memory – ie, it's a page that's on disk – it would need to be swapped in and the page table would need to be updated, but you don't have to talk about that in this question.)

(d) (1 point) Explain why paging-based systems need a TLB to store frequently used translations.

Solution: The page table resides in memory too. And going out to memory is slow. If we need to look up an address in the page table for each memory access, we double our memory accesses! So we store a few frequently used addresses in on-chip memory (fast!) so that we don't need the page table access slowing us down.