1. Virtual memory address layout example
    a. 12-bit virtual addresses
    b. Page size of 256 bytes
        i. Offset into page is just like cache offset
        ii. Offset = $\log_2 256 = 8$ bits
        iii. Frame size will be the same as page size
    c. Physical memory size = 2 KB = 2^{11} B
        i. Therefore, physical address size is $\log_2 2^{11} = 11$ bits
    d. Layout of address below

Virtual Page Number (VPN)	**Offset into Virtual Page**	=	**Virtual Address Size**
Whatever's left = 12 – 8 = 4 bits	$\log_2$ (page size) = 8 bits		Given, 12 bits

Physical Frame Number (PFN)	**Offset into Physical Frame**	=	**Physical Address Size**
Whatever's left = 11 – 8 = 3 bits	Same as page offset, 8 bits		$\log_2$ (memory size) = 11 bits

    e. Offset is always same size for both addresses
        i. Same offset that we're accessing in the page is the same offset into the frame
    f. Translation is from 4-bit VPN to 3-bit PPN
        i. Page table tells us exactly what that translation should be

2. Page table example *(not covered in lecture)*
    a. Using same parameters as last problem, let's look at a memory and a page table
    b. Memory is 2 KB, and frame size = page size = 256 B
    c. Thus, divide memory into 8 frames of size 256 B each below
        i. PFN for each frame is equal to first three bits of each frame's address range

Memory

Physical Frame Number	**Size**	**Address Range**	**Status**
000	256 B	0x000 – 0x0FF	Busy
001	256 B	0x100 – 0x1FF	Busy
010	256 B	0x200 – 0x2FF	Empty
011	256 B	0x300 – 0x3FF	Busy
100	256 B	0x400 – 0x4FF	Empty
101	256 B	0x500 – 0x5FF	Process A
110	256 B	0x600 – 0x6FF	Busy
111	256 B	0x700 – 0x7FF	Empty

    d. Busy frames are already assigned to another process, empty frames are unallocated
        i. One other page from our current process A in memory right now

3. Page table access example – page fault *(not covered in lecture)*
    a. Processor generates the following virtual address for process A
        i. 0x716 = 0111 0001 0110
    b. First, need to translate this virtual address into the physical address
        i. Look at page table below to do so
        ii. Page table is essentially a hash table
            1. Key is the VPN of the address, data is the PFN we want
            2. Page table contains $2^{\text{VPN bits}}$ number of entries = 2^4 = 16 entries here
            3. Each entry is (PFN bits) bits wide = 3 bits here

Page Table

Virtual Page Number	Status
0000	On Disk
0001	Frame 101
0010	On Disk
0011	On Disk
0100	On Disk
0101	On Disk
0110	On Disk
0111	On Disk
1000	On Disk
1001	On Disk
1010	On Disk
1011	On Disk
1100	On Disk
1101	On Disk
1110	On Disk
1111	On Disk

    c. Use the first four bits of the address 0x716 above
        i. For entry 0111, the page is on the disk
        ii. We have a page fault, the page is not in RAM
    d. On a page fault, OS gets called in
        i. Must go out and get the page from memory
        ii. Then needs to decide where to place it
        iii. Selects empty frame 010 since that was the first one it found that was free
        iv. Copy the data from the disk into that frame in RAM
        v. Update page table and list of frames

Memory

Physical Frame Number	Size	Address Range	Status
000	256 B	0x000 – 0x0FF	Busy
001	256 B	0x100 – 0x1FF	Busy
010	256 B	0x200 – 0x2FF	Process A
011	256 B	0x300 – 0x3FF	Busy
100	256 B	0x400 – 0x4FF	Empty
101	256 B	0x500 – 0x5FF	Process A
110	256 B	0x600 – 0x6FF	Busy
111	256 B	0x700 – 0x7FF	Empty

Page Table

Virtual Page Number	Status
0000	On Disk
0001	Frame 101
0010	On Disk
0011	On Disk
0100	On Disk
0101	On Disk
0110	On Disk
0111	Frame 010
1000	On Disk
1001	On Disk
1010	On Disk
1011	On Disk
1100	On Disk
1101	On Disk
1110	On Disk
1111	On Disk

e. Now our memory has the page we requested in frame 010
    i. Our page table has been updated to match
f. Use page table to convert address
    i. Take 4-bit VPN, replace with 3-bit PFN
    ii. Virtual address 0x716 = 0111 0001 0110
        1. Replace first 4 bits with 3-bit sequence 010
        2. Offset stays the same
    iii. Thus, corresponding physical address = 0x216 = 010 0001 0110
g. Now that we have the physical address, go out to cache/RAM to retrieve data at that location
    i. Completely possible to have a page fault, then a cache miss
    ii. This could even occur for the page table, since the page table needs to be stored in RAM too!

4. Page table access example – hit *(not covered in lecture)*
    a. Use same address format and modified memory contents from previous problem
    b. Now assume we're given the next virtual address
        i. 0x15F = 0001 0101 1111
    c. Look in the page table for the VPN 0001
        i. Page table tells us the frame is in memory, and the PFN is 101
    d. Translate address using the PFN
        i. Virtual address 0x15F = 0001 0101 1111
            1. Replace first 4 bits with 3-bit sequence 101
            2. Offset stays the same
        ii. Thus, physical address = 0x55F = 101 0101 1111
    e. Now that we have the physical address, go out to cache/RAM to retrieve data at that location

5. Translation lookaside buffer (TLB)
    a. Need to make these translations extremely often
    b. Instead of constantly going to RAM for page tables, cache recent translations from VPN to PFN
    c. Cache needs to have extremely high hit rate, so use FA cache
        i. Various other techniques can increase the hit rate, too
    d. Hit in TLB? Great, immediately have translation ready
    e. Miss in TLB? Must go to page table to obtain translation (or find out there's a page fault)

6. Virtual memory summary
    a. Virtual memory is a mapping from the process' virtual space to the real/physical RAM space
    b. Need to translate virtual addresses from the currently running process to physical addresses in RAM
    c. Page table tells us exactly how to make that translation, and if page is currently in RAM or not
    d. Can have more memory for one process than the amount of RAM we have
    e. Currently unused frames are stored on disk, and brought out when needed
    f. TLB is used to cache recent virtual to physical translations