Question 1: Briefly describe the data structure(s) that your kernel uses to manage the allocation of physical memory. What information is recorded in this data structure? When your VM system is initialized, how is the information in this data structure initialized?

Our kernel uses a coremap(coremap.h). The coremap is indexed by integers from 0 to coremapsize-1. The information contained within the coremap is the paddr of the frame, and the vaddr that its map to. Other information such as len of block, used bit, flags, pid and time stamp information. The vm system primarily uses the flag,used, and len block to manage allocation of memory(explained in detail in Q2). When our VM system is initialize, call ram_getsize to get available memory after, all the locks and variable have been declared and created. We then find, how many pages the actual coremap needs, so we take the address of the first free remaining memory (freeaddr) we initialize [firstaddr,freeaddr) coremap entry with used =1 and flags = 0x1 for fixed, and the free pages [freeaddr,lastaddr]] with used = 0 and flags =0. We go through each coremap entry and initialize len,pid, timestaps, as -1. We then give a paddr for each coremap entry via (firstaddr+ i*PAGE_SIZE), it is noteworthy that the paddr are properly aligned. We dont initialize vaddr, since no mapppings have been set yet for each frame. After the coremap is ready we initialize pt_initialize =1 (the global variable that indicates our vm system is ready)

Question 2: When a single physical frame needs to be allocated, how does your kernel use the above data structure to choose a frame to allocate? When a physical frame is freed, how does your kernel update the above data structure to support this?

To choose a frame to allocate, our kernel finds the first available frame that isn't fixed or used (flags =0x0, used =0). To allocate it, we set the first available frame's used bit to 1 and len = 1 and return the paddr of the frame (getppages). To free the frame, you do the opposite, you set the used bit back of the coremap entry to 0, and len to -1 (default value to indicate this block isnt freed)

Question 3: Does your physical-memory system have to handle requests to allocate/free multiple (physically) contiguous frames? Under what circumstances? How does your physical-memory manager support this?
Yes, the circumstances when allocation npages is when, a single frame isn't enough to hold the data. If we need to allocate n pages, we search through the coremap to find n contiguous pages that have used = 0(free) and fixed = 0(to ensure we never touch fixed memory). We set the first frame's len= n, and all the n frames used bit to 1. We keep the flags the same since we did not implemented page replacement; however, if we did, then we would set the flags to 0x2(dirty) that is pages that are allocated but no copy in the swapfile. To free a page, we find the physical frame that the vaddr maps to, the vaddr maybe a kernel vaddr so we also use paddr_to_kvaddr to check each paddr in coremap. Once we found, the index of the frame, we just set the used bit back to 0 and len = -1 on all frames that it needs to free.

Question 4: Are there any synchronization issues that arise when the above data structures are

used? Why or why not?
Yes, there are synchronization issues. The reason being is that , two processes can run concurrently, to ensure we never allocate the same frame to them, we protect the coremap with a lock. That is, it ensures that the coremap can never give two different processes the same frame(s) that another uses.

Question 5: Briefly describe the data structure(s) that your kernel uses to describe the virtual address space of each process. What information is recorded about each address space?

Question 6: When your kernel handles a TLB miss, how does it determine whether the required page is already loaded into memory?

Question 7: If, on a TLB miss, your kernel determines that the required page is not in memory, how does it determine where to find the page?

Question 8: How does your kernel ensure that read-only pages are not modified?

Question 9: Briefly describe the data structure(s) that your kernel uses to manage the swap file. What information is recorded and why? Are there any synchronization issues that need to be handled? If so what are they and how were they handled?

Question 10: What page replacement algorithm did you implement? Why did you choose this algorithm?Is this a good choice, why or why not? What were some of the issues you encountered when trying to design and implement this algorithm