Homework 12

I. Virtual Memory

1. Which type of address is used in each of following scenarios, virtual or physical address?
    1) The address of variables in C program V
    2) The address stored in a C pointer V
    3) The address of a C pointer V
    4) The address used in looking up L1 cache P
    5) The value in CR3 P
    6) The address in L2 PTE P
    7) The address in L4 PTE P
    8) The value in PC register V
2. The lookup of L1 cache usually consists of three steps: locating the set; comparing the tag of each cache line in the set; returning bytes from cache or loading value from next level memory system. As the lookup uses physical address, the hardware can only start the cache lookup after address translation is completed. How to make cache lookup and address translation parallelized? Show the requirements to the parameters of your paging and cache system.

    Make sure the set bits of cache are in the page offset bits, so the hardware could locate the set and do address translation simultaneously. This requires s+b <= len(po)

II. Page Table

1. Why could multiple level page table save memory?
    1) If we only use 0x400000-0x800000, compare the memory usage of page table between single level and 4 level.

        single: (2^48B/4KB)*8B = 2^39B = 512GB

        4 level: (2 L4 page table+3 L1/L2/L3 page table) * 4KB = 20KB
    2) If we have enough physical memory and use the whole $2^{48}\mathrm{B}$ virtual space, compare the memory usage again.

        single: (2^48B/4KB)*8B = 2^39B = 512GB

        4 level: (1+512+512^2+512^3) * 4KB $\approx$ 513GB
2. Given the following page table and cr3=0x1000, please translate the virtual addresses to physical addresses.

    (Assume the machine is x86_64, which uses 4 level page table)

Physical Address of PTE	Address Part in PTE
0x1000	0x4000
0x1008	0x4000
0x4000	0x5000
0x5000	0x7000
0x5008	0x8000
0x5010	0x9000
0x5018	0x3000
0x5020	0x5000
0x7008	0x8000

0x7010	0x10000

Physical Address	Virtual Address
0x1234	0x8234
0x2234	0x10234
0x8000802135	0x9135

III. Demand paging & TLB

Assume on a x86_64 machine without cache, we have an `int a[70]` at virtual address 0x8000344f00, a program access a[0], a[1], ..., a[69] one by one. Before the first access, we have only an empty L1 page table.

1. How many PP is used after the program access a[0], a[64] and a[69]? (Remember the page table is stored in PP too)
   after access a[0]: 4 page table pages + 1 content pages
   after access a[64]: 4 + 2
   after access a[69]: 4 + 2
2. How many memory accesses and page fault happened in the program if we have no TLB?
   (4+1)*70+1+4=355
   4+1 for each successful array access
   first access to a[0] need one access to L1 PTE to trigger page fault
   first access to a[64] need four access to L1, L2, L3 and L4 PTE to trigger page fault

   2 page faults
3. How many memory accesses and page fault happened in the program if we have a full-associative infinite TLB?
   70+4*2+1+4=83
   1 for each successful array access
   successful accesses to a[0], a[64] will trigger two page table walk and fill TLB
   first access to a[0] need one access to L1 PTE to trigger page fault
   first access to a[64] need four access to L1, L2, L3 and L4 PTE to trigger page fault

   2 page faults