Tutorial 5 Sample Answer

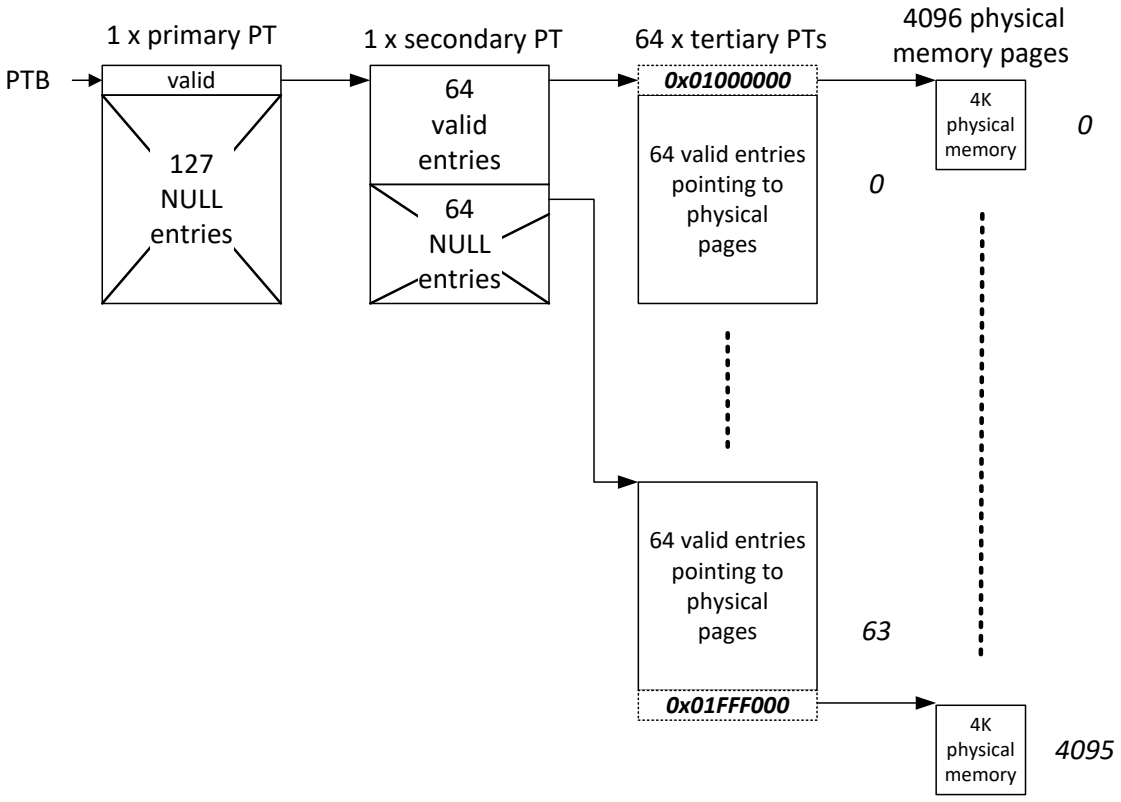
(i) What page sizes are supported by the MMU?

7-7-6-12 - 2^{12} = 4K (4 Kilo bytes)
7-7-5-13 - 2^{13} = 8K

(ii) What are the sizes of the various page tables in the hierarchy (both options)?

7-7-6-12 –	primary	2^7 = 128x4 = 512 bytes (4 assume each table entry is 4 bytes)
	secondary	2^7 = 128x4 = 512 bytes
	tertiary	2^6 = 64x4 = 256 bytes
7-7-5-13 –	primary	2^7 = 128x4 = 512 bytes (4 assume each table entry is 4 bytes)
	secondary	2^7 = 128x4 = 512 bytes
	tertiary	2^5 = 32x4 = 128 bytes

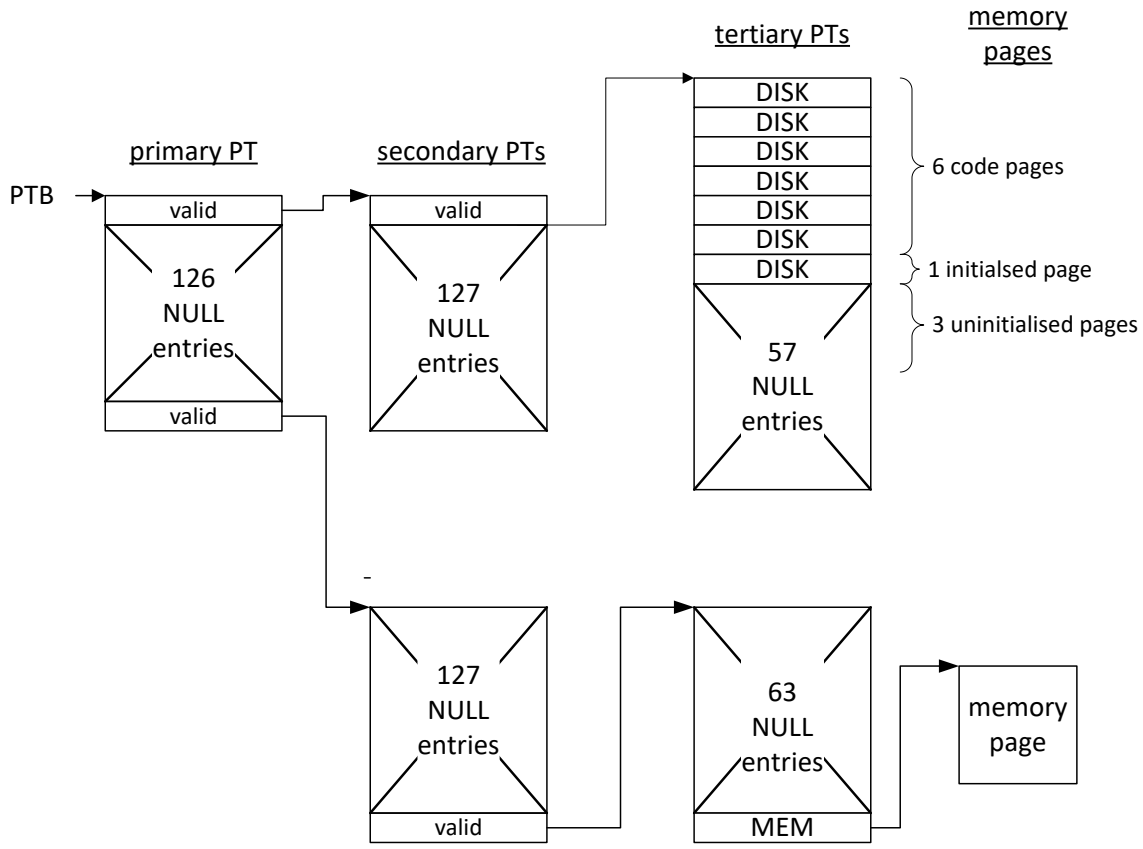
(iii) Outline the structure and size of the kernel page table assuming that the kernel's virtual address space [starting at address 0x00000000] is mapped directly onto 16MB of physical memory starting at address physical 0x01000000 [assume a 4Kbyte page size].



Easier to think about this question backwards! As 16MB of physical memory (4096 x 4K pages) needs to be mapped, 64 tertiary page tables are required since each tertiary page table can map 64 pages. "Half" a secondary page table is then needed to point to 64 tertiary page tables. Finally, the first entry in the primary page table is needed to point to the secondary page table. Note that the tertiary page table entries (PTEs) contain the physical addresses of the memory pages they point to.

(iv) A user process is started which has a code/text size of 0x5fe0 bytes, an initialised data size of 0x6ac bytes, an uninitialised data [bss] size of 0x2888 bytes and 642 bytes of stack data copied from its parent. Outline the structure and size of its initial user page table [assume a 4Kbyte page size].

Each code/data section is rounded up to a multiple of the page size. This allows each section to have its own protection level (text – read/execute, data – read/write). 6 x 4K pages are needed for code, 1 page for initialised data, 3 pages for uninitialised data and 1 page for the stack.



The initial memory footprint of a process is small. Initially, the physical memory allocated is that needed for the initial page table structure and the initial stack. The PTEs for the code and initialised data pages are set to type DISK. A DISK PTE contains the disk block where the corresponding code/data is stored on disk. This will be faulted in on demand as the process executes.

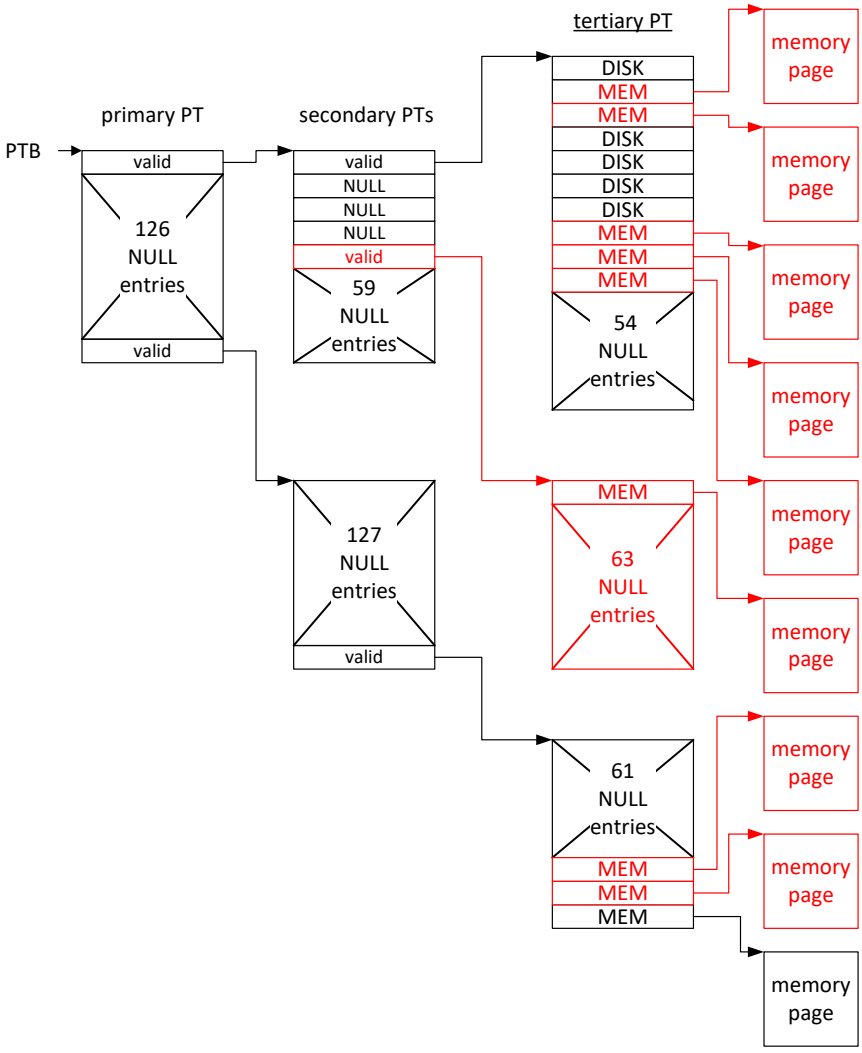
Initial memory footprint:

512 + 2 x 512 + 2 x 256 – 2 K for page table structure
4K – for stack page

total : 6K.

(v) Given that the TCD2019 processor accesses the following virtual memory addresses when executing the user process [U = user address, S = supervisor address], indicate the memory area being accessed and the changes that would be made to the initial page table structure when handling any page faults.

```
U  0x00001000 - code page fault, allocate page, get code from disk
U  0x00001004 - code access
U  0x00001008 - code access
U  0x00002000 - code page fault, allocate page, get code from disk
U  0x00008000 - uninitialised data fault, allocate zeroed page
S  0x00001000 - kernel access
U  0x00007000 - uninitialised data fault, allocate zeroed page
S  0x00002000 - kernel access
U  0xFFFFE000 - stack page fault, allocate zeroed page
U  0xFFFFD000 - stack page fault, allocate zeroed page
S  0x00FFE000 - kernel stack access
S  0x00FFC000 - kernel stack access
U  0x00002000 - code access
U  0x00100000 - data access (heap) fault, allocate zeroed page
U  0x00002000 - code access
U  0x00009000 - uninitialised data fault, allocate zeroed page
```



Changes to the process page table structure as a result of the above sequence of memory accesses are coloured red. Note that the OS dynamically allocates a tertiary page table when virtual address 0x00100000 is accessed.

(vi) The TCD2019's MMU has an integral 8 entry fully associative TLB. Outline the organisation of the TLB and explain how kernel & user virtual to physical mappings can share the TLB. Given the above sequence of memory accesses, calculate the TLB hit rate and the resulting contents of the TLB [assume LRU replacement].

```
U  0x00001000 – miss [use cache line 0]
U  0x00001004 – hit
U  0x00001008 – hit
U  0x00002000 – miss [use cache line 1]
U  0x00008000 – miss [use cache line 2]
S  0x00001000 – miss [use cache line 3]
U  0x00007000 – miss [use cache line 4]
S  0x00002000 – miss [use cache line 5]
U  0xFFFFE000 – miss [use cache line 6]
U  0xFFFFD000 – miss [use cache line 7]
S  0x00FFE000 – miss [reuse cache line 0]
S  0x00FFC000 – miss [reuse cache line 1]
U  0x00002000 – miss [reuse cache line 2]
U  0x00100000 – miss [reuse cache line 3]
U  0x00002000 – hit
U  0x00009000 - miss [reuse cache line 4]
```

hit rate 3/16 = 18.75%

NB: TLB uses the U/S bit to distinguish between kernel and user virtual addresses.