5.10.1

a.

Virtual page number: Address >> 12 bits
H: Hit in TLB, M: Miss in TLB hit in page table, PF: Page Fault

0, 7, 3, 3, 1, 1, 2 (M, H, M, H, PF, H, PF)

TLB

Valid	Tag	Physical Page Number
1	3	6
1	7	4
1	1	13
1	2	14

Page table

Valid	Physical page or in disk
1	5
1	13
1	14
1	6
1	9
1	11
0	Disk
1	4
0	Disk
0	Disk
1	12

b.

Binary address: (all hexadecimal), {bits 15–12 virtual page, 11–0 page offset}
2 4EC, Page Fault, disk => physical page D, (=> TLB slot 3)
7 8F4, Hit in TLB
4 AC0, Miss in TLB, (=> TLB slot 0)
B 5A6, Miss in TLB, (=> TLB slot 2)
9 4DE, Page Fault, disk => physical page E, (=> TLB slot 3)
4 10D, Hit in TLB
B D60 Hit in TLB

TLB

Valid	Tag	Physical Page
1	4	9
1	7	4
1	B	C
1	9	E

Page table

Valid	Physical page
1	5
0	disk
1	D
1	6
1	9
1	B
0	disk
1	4
0	disk
1	E
1	3
1	C

5.10.2

a.

Virtual page number: Address >> 14 bits
H: Hit in TLB, M: Miss in TLB hit in page table, PF: Page Fault

0, 3, 1, 1, 0, 0, 1 (M, H, PF, H, H, H, H)

TLB

Valid	Tag	Physical Page Number
1	1	13
1	7	4
1	3	6
1	0	5

Page table

Valid	Physical page or in disk
1	5
1	13
0	Disk
1	6
1	9
1	11
0	Disk
1	4
0	Disk
0	Disk
1	3
1	12

Larger page sizes allow for more addresses to be stored in a single page, potentially decreasing the amount of pages that must be brought in from disk and increasing the coverage of the TLB. However, if a program uses addresses in a sparse fashion (for example randomly accessing a large matrix), then there will be an extra penalty from transferring larger pages compared to smaller pages.

b.

Binary address: (all hexadecimal), {bits 15–14 virtual page, 13–0 page offset}
0 24EC, Miss in TLB, (=> TLB slot 3)
1 38F4, Page Fault, disk => physical page D, (=> TLB slot 1)
1 0AC0, Hit in TLB
2 35A6, Page Fault, disk => physical page E, (=> TLB slot 2)
2 14DE, Hit in TLB
1 010D, Hit in TLB
2 3D60, Hit in TLB

TLB

Valid	Tag	Physical Page
1	B	C
1	1	D
1	2	E
1	0	5

Page table

Valid	Physical page
1	5
1	D
1	E
1	6
1	9
1	B
0	disk
1	4
0	disk
0	disk
1	3
1	C

Larger page sizes allow for more addresses to be stored in a single page, potentially decreasing the amount of pages that must be brought in from disk and increasing the coverage of the TLB. However, if a program uses addresses in a sparse fashion (for example randomly accessing a large matrix), then there will be an extra penalty from transferring larger pages compared to smaller pages.

5.10.3

a.

Virtual page number: Address >> 12 bits

0, 7, 3, 3, 1, 1, 2 => 0, 111, 011, 011, 001, 001, 010

2 way set associative:
Tag: VPN >> 1 bit

TLB

Valid	Tag	PPN	Valid	Tag	PPN
1	0	5	1	01	14
1	0	13	1	01	6

Direct-mapped:
Tag: VPN >> 2 bits

TLB

Valid	Tag	Physical Page Number
1	0	5
1	0	13
1	0	14
1	0	6

The TLB is important to avoiding paying high access times to memory in order to translate virtual addresses to physical addresses. If memory accesses are frequent, then the TLB will become even more important. Without a TLB, the page table would have to be referenced upon every access using a virtual addresses, causing a significant slowdown.

b.

Binary address: (all hexadecimal), {bits 15–12 virtual page, 11–0 page offset}

2 4EC, Page Fault, disk => physical page D, (=> TLB set 0, slot 1)
7 8F4, Miss in TLB, (=> TLB set 1, slot 1)
4 AC0, Miss in TLB, (=> TLB set 0, slot 0)
B 5A6, Miss in TLB, (=> TLB set 1, slot 0)
9 4DE, Page Fault, disk => physical page E, (=> TLB set 1, slot 1)
4 10D, Hit in TLB
B D60 Hit in TLB

2-way set associative TLB {bits 15–12 virtual page => bits 15–13 tag, bits 12 set}
(note: time stamps according to at start physical page numbers)

Valid	Tag(/Time)	Physical Page
1	4 /4	9
1	2 /2	D
1	B /5	C
1	9 /4	E

Binary address: (all hexadecimal), {bits 15–12 virtual page, 11–0 page offset}
2 4EC, Page Fault, disk => physical page D, (=> TLB slot 2)
7 8F4, Hit in TLB
4 AC0, Miss in TLB, (=> TLB slot 0)
B 5A6, Miss in TLB, (=> TLB slot 3)
9 4DE, Page Fault, disk => physical page F, (=> TLB slot 1)
4 10D, Hit in TLB
B D60 Hit in TLB

direct-mapped TLB {bits 15–12 virtual page => bits 13–12 TLB slot}

Valid	Tag	Physical Page
1	4	9
1	9	F
1	2	D
1	B	C

The TLB is important to avoiding paying high access times to memory in order to translate virtual addresses to physical addresses. If memory accesses are frequent, then the TLB will become even more important. Without a TLB, the page table would have to be referenced upon every access using a virtual addresses, causing a significant slowdown.

5.10.4

a.	4 KB page = 12 offset bits, 20 page number bits 2^{20} = 1 M page table entries 1 M entries × 4 bytes/entry = ~4 MB (2^{22} bytes) page table per application 2^{22} bytes × 5 apps = 20.97 MB total
b.	virtual address size of 64 bits 16 KB (2^{14}) page size, 8 (2^{3}) bytes per page table entry 64 – 14 = 40 bits or 2^{40} page table entries with 8 bytes per entry, yields total of 2^{43} bytes for each page table Total for 5 applications = 5 × 2^{43} bytes

5.11.1

a.	virtual address 32 bits, physical memory 4 GB page size 8 KB or 13 bits, page table entry 4 bytes or 2 bits #PTE = 32 – 13 = 19 bits or 512K entries PT physical memory = 512K × 4 bytes = 2 MB
b.	virtual address 64 bits, physical memory 16 GB page size 4 KB or 12 bits, page table entry 8 bytes or 3 bits #PTE = 64 – 12 = 52 bits or 2^{52} entries PT physical memory = $2^{52} \times 2^{3} = 2^{55}$ bytes

5.11.2

a.	virtual address 32 bits, physical memory 4 GB page size 8 KB or 13 bits, page table entry 4 bytes or 2 bits #PTE = 32 – 13 = 19 bits or 512K entries 8 KB page/4 byte PTE = 2^{11} pages indexed per page Hence with 2^{19} PTEs will need 2-level page table setup. Each address translation will require at least 2 physical memory accesses.
b.	virtual address 64 bits, physical memory 16 GB page size 4 KB or 12 bits, page table entry 8 bytes or 3 bits #PTE = 64 – 12 = 52 bits or 2^{52} entries 4 KB page/8 byte PTE = 2^{9} pages indexed per page Hence with 2^{52} PTEs will need 6-level page table setup. Each address translation will require at least 6 physical memory accesses.

5.12.1

a.	0 hits
b.	2 hits

5.12.2

a.	3 hits
b.	3 hits

5.12.3

a.	3 hits or fewer
b.	3 hits or fewer