5.16 1)

Address	Virtual Page	TLB H/M	Valid	Tag	Physical Page	Time
4669	1	TLB Miss	1	b	12	5
0x123d		PT Hit	1	7	4	2
		Page Fault	1	3	6	4
			1	1	13	0
2227	0	TLB Miss	1	0	5	0
0x08b		PT Hit	1	7	4	3
			1	3	6	5
			1	1	13	1
13916	3	TLB Miss	1	0	5	1
0x365c		PT Hit	1	7	4	4
			1	3	6	0
			1	1	13	2
34587	8	TLB Miss	1	0	5	2
0x871b		PT Hit	1	8	14	0
		Page Fault	1	3	6	1
			1	1	13	3
48870	b	TLB Miss	1	0	5	3
0xbee6		PT Hit	1	8	14	1
			1	3	6	2
			1	b	12	0
12608	3	TLB Miss	1	*0*	5	0
0x3140		PT Hit	1	8	14	2
			1	3	6	3
			1	b	12	1
49225	c	TLB Miss	1	*c*	15	
0xc049		PT Hit	1	8	14	
		Page Fault	1	3	6	
			1	b	12	

5.16 2)

Address	Virtual Page (mod 3)	TLB H/M	Valid	Tag	Physical Page	Time
4669	1	TLB Miss	1	11	12	
0x123d		PT Hit	1	7	4	
			1	3	6	
			1	0	5	
2227	0	TLB Hit	1	11	12	
0x08b		PT Hit	1	7	4	
			1	3	6	
			1	0	5	
13916	0	TLB Hit	1	11	12	
0x365c		PT Hit	1	7	4	
			1	3	6	
			1	0	5	
34587	2	TLB Miss	1	2	13	
0x871b		PT Hit	1	7	4	
		Page Fault	1	3	6	
			1	0	5	
48870	2	TLB Hit	1	2	13	
0xbee6		PT Hit	1	7	4	
			1	3	6	
			1	0	5	
12608	0	TLB Hit	1	2	13	
0x3140		PT Hit	1	7	4	
			1	3	6	
			1	0	5	
49225	3	TLB Miss	1	2	13	
0xc049		PT Hit	1	7	4	
			1	3	6	
			1	0	5	

A larger page size can reduce the TLB miss rate, but there will be more fragmentation as a result. This means that there will be lower utilization of physical memory as well.

5.16 3)

Address	Virtual Page	Tag	Index	TLB H/M	Valid	Tag	Physical Page	Index
4669	1	0	1	TLB Miss	1	b	12	0
0x123d				PT Hit	1	7	4	1
				Page Fault	1	3	6	0
					1	0	13	1
2227	0	0	0	TLB Miss	1	0	5	0
0x08b				PT Hit	1	7	4	1
					1	3	6	0
					1	0	13	1
13916	3	1	1	TLB Miss	1	0	5	0
0x365c				PT Hit	1	1	6	1
					1	3	6	0
					1	1	13	1
34587	8	4	0	TLB Miss	1	0	5	0
0x871b				PT Hit	1	1	6	1
				Page Fault	1	4	14	0
					1	1	13	1
48870	b	5	1	TLB Miss	1	0	5	0
0xbee6				PT Hit	1	1	6	1
					1	4	14	0
					1	5	12	1
12608	3	1	1	TLB Hit	1	0	5	0
0x3140				PT Hit	1	1	16	1
					1	4	14	0
					1	5	12	1
49225	c	6	0	TLB Miss	1	6	15	0
0xc049				PT Miss	1	1	6	1
				Page Fault	1	4	14	0
					1	5	12	1

5.16 4)

Address	Virtual Page	Tag	Index	TLB H/M	Valid	Tag	Physical Page	Index
4669	1	0	1	TLB Miss	1	b	12	0
0x123d				PT Hit	1	0	13	1
				Page Fault	1	3	6	2
					1	4	9	3
2227	0	0	0	TLB Miss	1	0	5	0
0x08b				PT Hit	1	0	13	1
					1	3	6	2
					1	4	9	3
13916	3	0	1	TLB Miss	1	0	5	0
0x365c				PT Hit	1	0	13	1
					1	3	6	2
					1	0	6	3
34587	8	2	0	TLB Miss	1	2	14	0
0x871b				PT Hit	1	0	13	1
				Page Fault	1	3	6	2
					1	0	6	3
48870	b	2	1	TLB Miss	1	2	14	0
0xbee6				PT Hit	1	0	13	1
					1	3	6	2
					1	2	12	3
12608	3	0	1	TLB Hit	1	2	14	0
0x3140				PT Hit	1	0	13	1
					1	3	6	2
					1	0	6	3
49225	c	3	0	TLB Miss	1	3	15	0
0xc049				PT Miss	1	0	13	1
				Page Fault	1	3	6	2
					1	0	6	3

5.16 5)

A TLB is needed for high performance because it greatly reduces memory accesses. If there was no TLB, every single memory access would need to access RAM twice — one access to the page table, and one access to the requested data. This can get quite inefficient. Especially when there are many memory accesses in a program. This is why we use the TLB.

5.20
- 4 one-word blocks
- 20 address sequence
- 0, 1, 2, 3, 4, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0

1) 2-way set associative cache, LRU

0	1	2	3	4	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0
M	M	M	M	M	H	H	H	M	M	M	M	M	M	M	M	M	M	M	M

2) 2-way set associative cache, MRU

0	1	2	3	4	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0
M	M	M	M	M	M	H	M	M	M	M	H	H	M	M	M	M	M	M	H

3) 2-way set associative cache, Random replacement

0	1	2	3	4	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0
M	M	M	M	M	M	H	H	M	M	H	H	H	M	M	H	M	H	M	H

4)
Ideally, the best block to evict in a replacement would be the block that causes the least misses in the future. If this method was possible to implement then we would have 0 misses. However, this clearly is impossible because we can't see the future. Working with what is possible, MRU seems to work pretty well. There is slightly better performance with MRU than LRU, and random replacement is, of course, random. There may be times where random replacement is better than MRU and times when it is worse than MRU. Thus, MRU seems to be the best option. The hit/miss sequence can be seen in above.

5)
It's difficult to implement a cache replacement policy that is optimal for all address sequences because there is no way to see the future. In order to be optimal, we would need to know what addresses are coming up, so we could form our policy around that. There is no one-policy-fits-all solution because if we optimize one address sequence, there is going to be some inverse or opposite address sequence that has terrible

performance using that policy. There are simply too many possible combinations of address sequences for all of them to be optimal under a single replacement policy.

6)
This is an interesting idea, but impractical. An instance where this could be beneficial is when there is little temporal locality. It might be better to not even cache any of the values as there would be constant thrashing. However, in an instance where we know that an address will be accessed repeatedly (say, in a loop) then choosing to cache this value could be helpful. Overall though, this strategy is not practical because its success relies on something we can never achieve - the ability to read the future.