5

Solutions

5.1

5.1.1 4

5.1.2 I, J

5.1.3 A[I][J]

5.1.4 3596 = 8 × 800/4 × 2−8×8/4 + 8000/4

5.1.5 I, J

5.1.6 A(J, I)

5.2

5.2.1

Word Address	Binary Address	Tag	Index	Hit/Miss
3	0000 0011	0	3	M
180	1011 0100	11	4	M
43	0010 1011	2	11	M
2	0000 0010	0	2	M
191	1011 1111	11	15	M
88	0101 1000	5	8	M
190	1011 1110	11	14	M
14	0000 1110	0	14	M
181	1011 0101	11	5	M
44	0010 1100	2	12	M
186	1011 1010	11	10	M
253	1111 1101	15	13	M

5.2.2

Word Address	Binary Address	Tag	Index	Hit/Miss
3	0000 0011	0	1	M
180	1011 0100	11	2	M
43	0010 1011	2	5	M
2	0000 0010	0	1	H
191	1011 1111	11	7	M
88	0101 1000	5	4	M
190	1011 1110	11	7	H
14	0000 1110	0	7	M
181	1011 0101	11	2	H
44	0010 1100	2	6	M
186	1011 1010	11	5	M
253	1111 1101	15	6	M

5.2.3

Word Address	Binary Address	Tag	Cache 1		Cache 2		Cache 3	
			index	hit/miss	index	hit/miss	index	hit/miss
3	0000 0011	0	3	M	1	M	0	M
180	1011 0100	22	4	M	2	M	1	M
43	0010 1011	5	3	M	1	M	0	M
2	0000 0010	0	2	M	1	M	0	M
191	1011 1111	23	7	M	3	M	1	M
88	0101 1000	11	0	M	0	M	0	M
190	1011 1110	23	6	M	3	H	1	H
14	0000 1110	1	6	M	3	M	1	M
181	1011 0101	22	5	M	2	H	1	M
44	0010 1100	5	4	M	2	M	1	M
186	1011 1010	23	2	M	1	M	0	M
253	1111 1101	31	5	M	2	M	1	M

Cache 1 miss rate = 100%

Cache 1 total cycles = $12 \times 25 + 12 \times 2 = 324$

Cache 2 miss rate = 10/12 = 83%

Cache 2 total cycles = $10 \times 25 + 12 \times 3 = 286$

Cache 3 miss rate = 11/12 = 92%

Cache 3 total cycles = $11 \times 25 + 12 \times 5 = 335$

Cache 2 provides the best performance.

5.2.4 First we must compute the number of cache blocks in the initial cache configuration. For this, we divide 32 KiB by 4 (for the number of bytes per word) and again by 2 (for the number of words per block). This gives us 4096 blocks and a resulting index field width of 12 bits. We also have a word offset size of 1 bit and a byte offset size of 2 bits. This gives us a tag field size of 32 − 15 = 17 bits. These tag bits, along with one valid bit per block, will require $18 \times 4096 = 73728$ bits or 9216 bytes. The total cache size is thus 9216 + 32768 = 41984 bytes.

The total cache size can be generalized to

totalsize = datasize + (validbitsize + tagsize) × blocks

totalsize = 41984

datasize = blocks × blocksize × wordsize

wordsize = 4

tagsize = 32 − log2(blocks) − log2(blocksize) − log2(wordsize)

validbitsize = 1

Increasing from 2-word blocks to 16-word blocks will reduce the tag size from 17 bits to 14 bits.

In order to determine the number of blocks, we solve the inequality:

41984 <= 64 × blocks + 15 × blocks

Solving this inequality gives us 531 blocks, and rounding to the next power of two gives us a 1024-block cache.

The larger block size may require an increased hit time and an increased miss penalty than the original cache. The fewer number of blocks may cause a higher conflict miss rate than the original cache.

5.2.5 Associative caches are designed to reduce the rate of conflict misses. As such, a sequence of read requests with the same 12-bit index field but a different tag field will generate many misses. For the cache described above, the sequence 0, 32768, 0, 32768, 0, 32768, …, would miss on every access, while a 2-way set associate cache with LRU replacement, even one with a significantly smaller overall capacity, would hit on every access after the first two.

5.2.6 Yes, it is possible to use this function to index the cache. However, information about the five bits is lost because the bits are XOR'd, so you must include more tag bits to identify the address in the cache.

5.3

5.3.1 8

5.3.2 32

5.3.3 1+ (22/8/32) = 1.086

5.3.4 3

5.3.5 0.25

5.3.6 <Index, tag, data>

<000001_2, 0001_2, mem[1024]>

<000001_2, 0011_2, mem[16]>

<001011_2, 0000_2, mem[176]>

<001000_2, 0010_2, mem[2176]>

<001110_2, 0000_2, mem[224]>

<001010_2, 0000_2, mem[160]>

5.4

5.4.1 The L1 cache has a low write miss penalty while the L2 cache has a high write miss penalty. A write buffer between the L1 and L2 cache would hide the write miss latency of the L2 cache. The L2 cache would benefit from write buffers when replacing a dirty block, since the new block would be read in before the dirty block is physically written to memory.

5.4.2 On an L1 write miss, the word is written directly to L2 without bringing its block into the L1 cache. If this results in an L2 miss, its block must be brought into the L2 cache, possibly replacing a dirty block which must first be written to memory.

5.4.3 After an L1 write miss, the block will reside in L2 but not in L1. A subsequent read miss on the same block will require that the block in L2 be written back to memory, transferred to L1, and invalidated in L2.

5.4.4 One in four instructions is a data read, one in ten instructions is a data write. For a CPI of 2, there are 0.5 instruction accesses per cycle, 12.5% of cycles will require a data read, and 5% of cycles will require a data write.

The instruction bandwidth is thus $(0.0030 \times 64) \times 0.5 = 0.096$ bytes/cycle. The data read bandwidth is thus $0.02 \times (0.13+0.050) \times 64 = 0.23$ bytes/cycle. The total read bandwidth requirement is 0.33 bytes/cycle. The data write bandwidth requirement is $0.05 \times 4 = 0.2$ bytes/cycle.

5.4.5 The instruction and data read bandwidth requirement is the same as in 5.4.4. The data write bandwidth requirement becomes $0.02 \times 0.30 \times (0.13+0.050) \times 64 = 0.069$ bytes/cycle.

5.4.6 For CPI=1.5 the instruction throughput becomes $1/1.5 = 0.67$ instructions per cycle. The data read frequency becomes $0.25 / 1.5 = 0.17$ and the write frequency becomes $0.10 / 1.5 = 0.067$.

The instruction bandwidth is $(0.0030 \times 64) \times 0.67 = 0.13$ bytes/cycle.

For the write-through cache, the data read bandwidth is $0.02 \times (0.17 +0.067) \times 64 = 0.22$ bytes/cycle. The total read bandwidth is 0.35 bytes/cycle. The data write bandwidth is $0.067 \times 4 = 0.27$ bytes/cycle.

For the write-back cache, the data write bandwidth becomes $0.02 \times 0.30 \times (0.17+0.067) \times 64 = 0.091$ bytes/cycle.

Address	0	4	16	132	232	160	1024	30	140	3100	180	2180
Line ID	0	0	1	8	14	10	0	1	9	1	11	8
Hit/miss	M	H	M	M	M	M	M	H	H	M	M	M
Replace	N	N	N	N	N	N	Y	N	N	Y	N	Y

5.5

5.5.1 Assuming the addresses given as byte addresses, each group of 16 accesses will map to the same 32-byte block so the cache will have a miss rate of 1/16. All misses are compulsory misses. The miss rate is not sensitive to the size of the cache or the size of the working set. It is, however, sensitive to the access pattern and block size.

5.5.2 The miss rates are 1/8, 1/32, and 1/64, respectively. The workload is exploiting temporal locality.

5.5.3 In this case the miss rate is 0.

5.5.4 AMAT for B = 8: $0.040 \times (20 \times 8) = 6.40$

AMAT for B = 16: $0.030 \times (20 \times 16) = 9.60$

AMAT for B = 32: $0.020 \times (20 \times 32) = 12.80$

AMAT for B = 64: $0.015 \times (20 \times 64) = 19.20$

AMAT for B = 128: $0.010 \times (20 \times 128) = 25.60$

B = 8 is optimal.

5.5.5 AMAT for B = 8: $0.040 \times (24 + 8) = 1.28$

AMAT for B = 16: $0.030 \times (24 + 16) = 1.20$

AMAT for B = 32: $0.020 \times (24 + 32) = 1.12$

AMAT for B = 64: $0.015 \times (24 + 64) = 1.32$

AMAT for B = 128: $0.010 \times (24 + 128) = 1.52$

B = 32 is optimal.

5.5.6 B=128

5.6

5.6.1

P1	1.52 GHz
P2	1.11 GHz

5.6.2

P1	6.31 ns	9.56 cycles
P2	5.11 ns	5.68 cycles

5.6.3

P1	12.64 CPI	8.34 ns per inst
P2	7.36 CPI	6.63 ns per inst

5.6.4

6.50 ns	9.85 cycles	Worse

5.6.5 13.04

5.6.6 P1 AMAT = 0.66 ns + 0.08 × 70 ns = 6.26 ns

P2 AMAT = 0.90 ns + 0.06 × (5.62 ns + 0.95 × 70 ns) = 5.23 ns

For P1 to match P2's performance:

5.23 = 0.66 ns + MR × 70 ns

MR = 6.5%

5.7

5.7.1 The cache would have 24 / 3 = 8 blocks per way and thus an index field of 3 bits.

Word Address	Binary Address	Tag	Index	Hit/Miss	Way 0	Way 1	Way 2
3	0000 0011	0	1	M	T(1)=0		
180	1011 0100	11	2	M	T(1)=0 T(2)=11		
43	0010 1011	2	5	M	T(1)=0 T(2)=11 T(5)=2		
2	0000 0010	0	1	M	T(1)=0 T(2)=11 T(5)=2	T(1)=0	
191	1011 1111	11	7	M	T(1)=0 T(2)=11 T(5)=2 T(7)=11	T(1)=0	
88	0101 1000	5	4	M	T(1)=0 T(2)=11 T(5)=2 T(7)=11 T(4)=5	T(1)=0	
190	1011 1110	11	7	H	T(1)=0 T(2)=11 T(5)=2 T(7)=11 T(4)=5	T(1)=0	
14	0000 1110	0	7	M	T(1)=0 T(2)=11 T(5)=2 T(7)=11 T(4)=5	T(1)=0 T(7)=0	
181	1011 0101	11	2	H	T(1)=0 T(2)=11 T(5)=2 T(7)=11 T(4)=5	T(1)=0 T(7)=0	

44	0010 1100	2	6	M	T(1)=0 T(2)=11 T(5)=2 T(7)=11 T(4)=5 T(6)=2	T(1)=0 T(7)=0	
186	1011 1010	11	5	M	T(1)=0 T(2)=11 T(5)=2 T(7)=11 T(4)=5 T(6)=2	T(1)=0 T(7)=0 T(5)=11	
253	1111 1101	15	6	M	T(1)=0 T(2)=11 T(5)=2 T(7)=11 T(4)=5 T(6)=2	T(1)=0 T(7)=0 T(5)=11 T(6)=15	

5.7.2 Since this cache is fully associative and has one-word blocks, the word address is equivalent to the tag. The only possible way for there to be a hit is a repeated reference to the same word, which doesn't occur for this sequence.

Tag	Hit/Miss	Contents
3	M	3
180	M	3, 180
43	M	3, 180, 43
2	M	3, 180, 43, 2
191	M	3, 180, 43, 2, 191
88	M	3, 180, 43, 2, 191, 88
190	M	3, 180, 43, 2, 191, 88, 190
14	M	3, 180, 43, 2, 191, 88, 190, 14
181	M	181, 180, 43, 2, 191, 88, 190, 14
44	M	181, 44, 43, 2, 191, 88, 190, 14
186	M	181, 44, 186, 2, 191, 88, 190, 14
253	M	181, 44, 186, 253, 191, 88, 190, 14

5.7.3

Address	Tag	Hit/ Miss	Contents
3	1	M	1
180	90	M	1, 90
43	21	M	1, 90, 21
2	1	H	1, 90, 21
191	95	M	1, 90, 21, 95
88	44	M	1, 90, 21, 95, 44
190	95	H	1, 90, 21, 95, 44
14	7	M	1, 90, 21, 95, 44, 7
181	90	H	1, 90, 21, 95, 44, 7
44	22	M	1, 90, 21, 95, 44, 7, 22
186	143	M	1, 90, 21, 95, 44, 7, 22, 143
253	126	M	1, 90, 126, 95, 44, 7, 22, 143

The final reference replaces tag 21 in the cache, since tags 1 and 90 had been reused at time=3 and time=8 while 21 hadn't been used since time=2.

Miss rate = 9/12 = 75%

This is the best possible miss rate, since there were no misses on any block that had been previously evicted from the cache. In fact, the only eviction was for tag 21, which is only referenced once.

5.7.4 L1 only:

$.07 \times 100 = 7$ ns

CPI = 7 ns / .5 ns = 14

Direct mapped L2:

$.07 \times (12 + 0.035 \times 100) = 1.1$ ns

CPI = ceiling(1.1 ns/.5 ns) = 3

8-way set associated L2:

$.07 \times (28 + 0.015 \times 100) = 2.1$ ns

CPI = ceiling(2.1 ns / .5 ns) = 5

Doubled memory access time, L1 only:

$.07 \times 200 = 14$ ns

CPI = 14 ns / .5 ns = 28

Doubled memory access time, direct mapped L2:

$.07 \times (12 + 0.035 \times 200) = 1.3$ ns

CPI = ceiling(1.3 ns/.5 ns) = 3

Doubled memory access time, 8-way set associated L2:

$.07 \times (28 + 0.015 \times 200) = 2.2$ ns

CPI = ceiling(2.2 ns / .5 ns) = 5

Halved memory access time, L1 only:

$.07 \times 50 = 3.5$ ns

CPI = 3.5 ns / .5 ns = 7

Halved memory access time, direct mapped L2:

$.07 \times (12 + 0.035 \times 50) = 1.0$ ns

CPI = ceiling(1.1 ns/.5 ns) = 2

Halved memory access time, 8-way set associated L2:

$.07 \times (28 + 0.015 \times 50) = 2.1\,\text{ns}$

CPI = ceiling(2.1 ns / .5 ns) = 5

5.7.5 $.07 \times (12 + 0.035 \times (50 + 0.013 \times 100)) = 1.0\,\text{ns}$

Adding the L3 cache does reduce the overall memory access time, which is the main advantage of having a L3 cache. The disadvantage is that the L3 cache takes real estate away from having other types of resources, such as functional units.

5.7.6 Even if the miss rate of the L2 cache was 0, a 50 ns access time gives

AMAT = $.07 \times 50 = 3.5$ ns, which is greater than the 1.1 ns and 2.1 ns given by the on-chip L2 caches. As such, no size will achieve the performance goal.

5.8

5.8.1

1096 days	26304 hours

5.8.2

0.9990875912%

5.8.3 Availability approaches 1.0. With the emergence of inexpensive drives, having a nearly 0 replacement time *for hardware* is quite feasible. However, replacing file systems and other data can take significant time. Although a drive manufacturer will not include this time in their statistics, it is certainly a part of replacing a disk.

5.8.4 MTTR becomes the dominant factor in determining availability. However, availability would be quite high if MTTF also grew measurably. If MTTF is 1000 times MTTR, it the specific value of MTTR is not significant.

5.9

5.9.1 Need to find minimum p such that $2^p >= p + d + 1$ and then add one. Thus 9 total bits are needed for SEC/DED.

5.9.2 The (72,64) code described in the chapter requires an overhead of 8/64=12.5% additional bits to tolerate the loss of any single bit within 72 bits, providing a protection rate of 1.4%. The (137,128) code from part a requires an overhead of 9/128=7.0% additional bits to tolerate the loss of any single bit within 137 bits, providing a protection rate of 0.73%. The cost/performance of both codes is as follows:

(72,64) code => 12.5/1.4 = 8.9

(136,128) code => 7.0/0.73 = 9.6

The (72,64) code has a better cost/performance ratio.

5.9.3 Using the bit numbering from section 5.5, bit 8 is in error so the value would be corrected to 0x365.

5.10 Instructors can change the disk latency, transfer rate and optimal page size for more variants. Refer to Jim Gray's paper on the five-minute rule ten years later.

5.10.1 32 KB

5.10.2 Still 32 KB

5.10.3 64 KB. Because the disk bandwidth grows much faster than seek latency, future paging cost will be more close to constant, thus favoring larger pages.

5.10.4 1987/1997/2007: 205/267/308 seconds. (or roughly five minutes)

5.10.5 1987/1997/2007: 51/533/4935 seconds. (or 10 times longer for every 10 years).

5.10.6 (1) DRAM cost/MB scaling trend dramatically slows down; or (2) disk $/access/sec dramatically increase. (2) is more likely to happen due to the emerging flash technology.

5.11

5.11.1

Address	Virtual Page	TLB H/M	TLB		
			Valid	Tag	Physical Page
4669	1	TLB miss PT hit PF	1	11	12
			1	7	4
			1	3	6
			1 (last access 0)	1	13
2227	0	TLB miss PT hit	1 (last access 1)	0	5
			1	7	4
			1	3	6
			1 (last access 0)	1	13
13916	3	TLB hit	1 (last access 1)	0	5
			1	7	4
			1 (last access 2)	3	6
			1 (last access 0)	1	13
34587	8	TLB miss PT hit PF	1 (last access 1)	0	5
			1 (last access 3)	8	14
			1 (last access 2)	3	6
			1 (last access 0)	1	13
48870	11	TLB miss PT hit	1 (last access 1)	0	5
			1 (last access 3)	8	14
			1 (last access 2)	3	6
			1 (last access 4)	11	12
12608	3	TLB hit	1 (last access 1)	0	5
			1 (last access 3)	8	14
			1 (last access 5)	3	6
			1 (last access 4)	11	12
49225	12	TLB miss PT miss	1 (last access 6)	12	15
			1 (last access 3)	8	14
			1 (last access 5)	3	6
			1 (last access 4)	11	12

5.11.2

Address	Virtual Page	TLB H/M	TLB		
			Valid	Tag	Physical Page
4669	0	TLB miss PT hit	1	11	12
			1	7	4
			1	3	6
			1 (last access 0)	0	5
2227	0	TLB hit	1	11	12
			1	7	4
			1	3	6
			1 (last access 1)	0	5
13916	0	TLB hit	1	11	12
			1	7	4
			1	3	6
			1 (last access 2)	0	5
34587	2	TLB miss PT hit PF	1 (last access 3)	2	13
			1	7	4
			1	3	6
			1 (last access 2)	0	5
48870	2	TLB hit	1 (last access 4)	2	13
			1	7	4
			1	3	6
			1 (last access 2)	0	5
12608	0	TLB hit	1 (last access 4)	2	13
			1	7	4
			1	3	6
			1 (last access 5)	0	5
49225	3	TLB hit	1 (last access 4)	2	13
			1	7	4
			1 (last axxess 6)	3	6
			1 (last access 5)	0	5

A larger page size reduces the TLB miss rate but can lead to higher fragmentation and lower utilization of the physical memory.

5.11.3 Two-way set associative

Address	Virtual Page	Tag	Index	TLB H/M	TLB			
					Valid	Tag	Physical Page	Index
4669	1	0	1	TLB miss PT hit PF	1	11	12	0
					1	7	4	1
					1	3	6	0
					1 (last access 0)	0	13	1
2227	0	0	0	TLB miss PT hit	1 (last access 1)	0	5	0
					1	7	4	1
					1	3	6	0
					1 (last access 0)	0	13	1
13916	3	1	1	TLB miss PT hit	1 (last access 1)	0	5	0
					1 (last access 2)	1	6	1
					1	3	6	0
					1 (last access 0)	1	13	1
34587	8	4	0	TLB miss PT hit PF	1 (last access 1)	0	5	0
					1 (last access 2)	1	6	1
					1 (last access 3)	4	14	0
					1 (last access 0)	1	13	1
48870	11	5	1	TLB miss PT hit	1 (last access 1)	0	5	0
					1 (last access 2)	1	6	1
					1 (last access 3)	4	14	0
					1 (last access 4)	5	12	1
12608	3	1	1	TLB hit	1 (last access 1)	0	5	0
					1 (last access 5)	1	6	1
					1 (last access 3)	4	14	0
					1 (last access 4)	5	12	1
49225	12	6	0	TLB miss PT miss	1 (last access 6)	6	15	0
					1 (last access 5)	1	6	1
					1 (last access 3)	4	14	0
					1 (last access 4)	5	12	1

Direct mapped

Address	Virtual Page	Tag	Index	TLB H/M	TLB			
					Valid	Tag	Physical Page	Index
4669	1	0	1	TLB miss PT hit PF	1	11	12	0
					1	0	13	1
					1	3	6	2
					0	4	9	3
2227	0	0	0	TLB miss PT hit	1	0	5	0
					1	0	13	1
					1	3	6	2
					0	4	9	3
13916	3	0	3	TLB miss PT hit	1	0	5	0
					1	0	13	1
					1	3	6	2
					1	0	6	3
34587	8	2	0	TLB miss PT hit PF	1	2	14	0
					1	0	13	1
					1	3	6	2
					1	0	6	3
48870	11	2	3	TLB miss PT hit	1	2	14	0
					1	0	13	1
					1	3	6	2
					1	2	12	3
12608	3	0	3	TLB miss PT hit	1	2	14	0
					1	0	13	1
					1	3	6	2
					1	0	6	3
49225	12	3	0	TLB miss PT miss	1	3	15	0
					1	0	13	1
					1	3	6	2
					1	0	6	3

All memory references must be cross referenced against the page table and the TLB allows this to be performed without accessing off-chip memory (in the common case). If there were no TLB, memory access time would increase significantly.

5.11.4 Assumption: "half the memory available" means half of the 32-bit virtual address space for each running application.

The tag size is $32 - \log_2(8192) = 32 - 13 = 19$ bits. All five page tables would require $5 \times (2\text{^}19/2 \times 4)$ bytes $= 5$ MB.

5.11.5 In the two-level approach, the 2^19 page table entries are divided into 256 segments that are allocated on demand. Each of the second-level tables contain $2\text{^}(19-8) = 2048$ entries, requiring $2048 \times 4 = 8$ KB each and covering 2048×8 KB $= 16$ MB (2^24) of the virtual address space.

If we assume that "half the memory" means 2^31 bytes, then the minimum amount of memory required for the second-level tables would be 5 × (2^31 / 2^24) * 8 KB = 5 MB. The first-level tables would require an additional 5 × 128 × 6 bytes = 3840 bytes.

The maximum amount would be if all segments were activated, requiring the use of all 256 segments in each application. This would require 5 × 256 × 8 KB = 10 MB for the second-level tables and 7680 bytes for the first-level tables.

5.11.6 The page index consists of address bits 12 down to 0 so the LSB of the tag is address bit 13.

A 16 KB direct-mapped cache with 2-words per block would have 8-byte blocks and thus 16 KB / 8 bytes = 2048 blocks, and its index field would span address bits 13 down to 3 (11 bits to index, 1 bit word offset, 2 bit byte offset). As such, the tag LSB of the cache tag is address bit 14.

The designer would instead need to make the cache 2-way associative to increase its size to 16 KB.

5.12

5.12.1 Worst case is 2^(43−12) entries, requiring 2^(43−12) × 4 bytes = 2 ^33 = 8 GB.

5.12.2 With only two levels, the designer can select the size of each page table segment. In a multi-level scheme, reading a PTE requires an access to each level of the table.

5.12.3 In an inverted page table, the number of PTEs can be reduced to the size of the hash table plus the cost of collisions. In this case, serving a TLB miss requires an extra reference to compare the tag or tags stored in the hash table.

5.12.4 It would be invalid if it was paged out to disk.

5.12.5 A write to page 30 would generate a TLB miss. Software-managed TLBs are faster in cases where the software can pre-fetch TLB entries.

5.12.6 When an instruction writes to VA page 200, and interrupt would be generated because the page is marked as read only.

5.13

5.13.1 0 hits

5.13.2 1 hit

5.13.3 1 hits or fewer

5.13.4 1 hit. Any address sequence is fine so long as the number of hits are correct.

5.13.5 The best block to evict is the one that will cause the fewest misses in the future. Unfortunately, a cache controller cannot know the future! Our best alternative is to make a good prediction.

5.13.6 If you knew that an address had limited temporal locality and would conflict with another block in the cache, it could improve miss rate. On the other hand, you could worsen the miss rate by choosing poorly which addresses to cache.

5.14

5.14.1 Shadow page table: (1) VM creates page table, hypervisor updates shadow table; (2) nothing; (3) hypervisor intercepts page fault, creates new mapping, and invalidates the old mapping in TLB; (4) VM notifies the hypervisor to invalidate the process's TLB entries. Nested page table: (1) VM creates new page table, hypervisor adds new mappings in PA to MA table. (2) Hardware walks both page tables to translate VA to MA; (3) VM and hypervisor update their page tables, hypervisor invalidates stale TLB entries; (4) same as shadow page table.

5.14.2 Native: 4; NPT: 24 (instructors can change the levels of page table)

Native: L; NPT: L×(L+2)

5.14.3 Shadow page table: page fault rate.

NPT: TLB miss rate.

5.14.4 Shadow page table: 1.03

NPT: 1.04

5.14.5 Combining multiple page table updates

5.14.6 NPT caching (similar to TLB caching)

5.15

5.15.1 CPI= 1.5 + 120/10000 × (15+175) = 3.78

If VMM performance impact doubles => CPI = 1.5 + 120/10000 × (15+350) =5.88

If VMM performance impact halves => CPI = 1.5 + 120/10000 × (15+87.5) =2.73

5.15.2 Non-virtualized CPI = 1.5 + 30/10000 × 1100 = 4.80

Virtualized CPI = 1.5 + 120/10000 × (15+175) + 30/10000 × (1100+175) = 7.60

Virtualized CPI with half I/O= 1.5 + 120/10000 × (15+175) + 15/10000 × (1100+175) = 5.69

I/O traps usually often require long periods of execution time that can be performed in the guest O/S, with only a small portion of that time needing to be spent in the VMM. As such, the impact of virtualization is less for I/O bound applications.

5.15.3 Virtual memory aims to provide each application with the illusion of the entire address space of the machine. Virtual machines aims to provide each operating system with the illusion of having the entire machine to its disposal. Thus they both serve very similar goals, and offer benefits such as increased security. Virtual memory can allow for many applications running in the same memory space to not have to manage keeping their memory separate.

5.15.4 Emulating a different ISA requires specific handling of that ISA's API. Each ISA has specific behaviors that will happen upon instruction execution, interrupts, trapping to kernel mode, etc. that therefore must be emulated. This can require many more instructions to be executed to emulate each instruction than was originally necessary in the target ISA. This can cause a large performance impact and make it difficult to properly communicate with external devices. An emulated system can potentially run faster than on its native ISA if the emulated code can be dynamically examined and optimized. For example, if the underlying machine's ISA has a single instruction that can handle the execution of several of the emulated system's instructions, then potentially the number of instructions executed can be reduced. This is similar to the case with the recent Intel processors that do micro-op fusion, allowing several instructions to be handled by fewer instructions.

5.16

5.16.1 The cache should be able to satisfy the request since it is otherwise idle when the write buffer is writing back to memory. If the cache is not able to satisfy hits while writing back from the write buffer, the cache will perform little or no better than the cache without the write buffer, since requests will still be serialized behind writebacks.

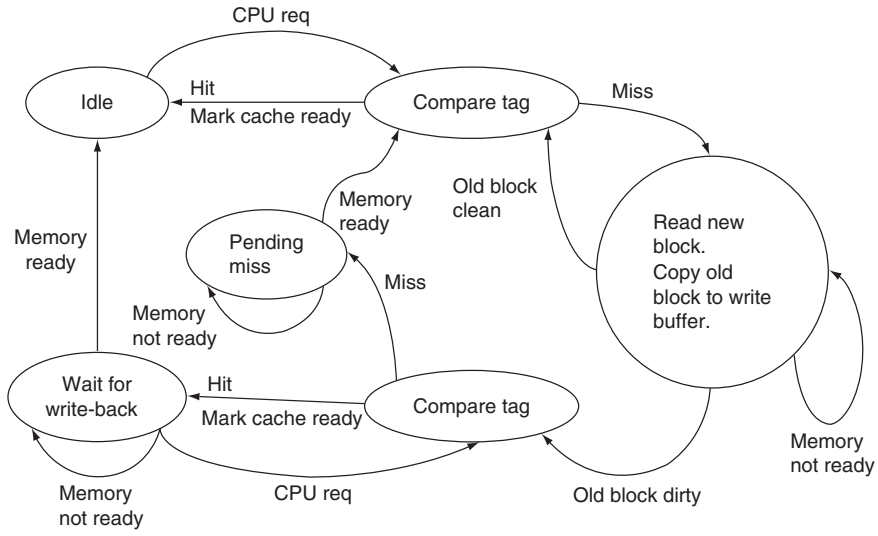
5.16.2 Unfortunately, the cache will have to wait until the writeback is complete since the memory channel is occupied. Once the memory channel is free, the cache is able to issue the read request to satisfy the miss.

5.16.3 Correct solutions should exhibit the following features:

1. The memory read should come before memory writes.

2. The cache should signal "Ready" to the processor before completing the write.

Example (simpler solutions exist; the state machine is somewhat underspecified in the chapter):



5.17

5.17.1 There are 6 possible orderings for these instructions.

Ordering 1:

P1	P2
X[0]++;	
X[1] = 3;	
	X[0]=5
	X[1] += 2;

Results: (5,5)

Ordering 2:

P1	P2
X[0]++;	
	X[0]=5
X[1] = 3;	
	X[1] += 2;

Results: (5,5)

Ordering 3:

P1	P2
	X[0]=5
X[0]++;	
	X[1] += 2;
X[1] = 3;	

Results: (6,3)

Ordering 4:

P1	P2
X[0]++;	
	X[0]=5
	X[1] += 2;
X[1] = 3;	

Results: (5,3)

Ordering 5:

P1	P2
	X[0]=5
X[0]++;	
X[1] = 3;	
	X[1] += 2;

Results: (6,5)

Ordering 6:

P1	P2
	X[0]=5
	X[1] += 2;
X[0]++;	
X[1] = 3;	

(6,3)

If coherency isn't ensured:

P2's operations take precedence over P1's: (5,2)

5.17.2

P1	P1 cache status/ action	P2	P2 cache status/action
		X[0]=5	invalidate X on other caches, read X in exclusive state, write X block in cache
		X[1] += 2;	read and write X block in cache
X[0]++;	read value of X into cache		X block enters shared state
	send invalidate message		X block is invalided
	write X block in cache		
X[1] = 3;	write X block in cache		

5.17.3 Best case:

Orderings 1 and 6 above, which require only two total misses.

Worst case:

Orderings 2 and 3 above, which require 4 total cache misses.

5.17.4 Ordering 1:

P1	P2
A = 1	
B = 2	
A += 2;	
B++;	
	C = B
	D = A

Result: (3,3)

Ordering 2:

P1	P2
A = 1	
B = 2	
A += 2;	
	C = B
B++;	
	D = A

Result: (2,3)

Ordering 3:

P1	P2
A = 1	
B = 2	
	C = B
A += 2;	
B++;	
	D = A

Result: (2,3)

Ordering 4:

P1	P2
A = 1	
	C = B
B = 2	
A += 2;	
B++;	
	D = A

Result: (0,3)

Ordering 5:

P1	P2
	C = B
A = 1	
B = 2	
A += 2;	
B++;	
	D = A

Result: (0,3)

Ordering 6:

P1	P2
A = 1	
B = 2	
A += 2;	
	C = B
	D = A
B++;	

Result: (2,3)

Ordering 7:

P1	P2
A = 1	
B = 2	
	C = B
A += 2;	
	D = A
B++;	

Result: (2,3)

Ordering 8:

P1	P2
A = 1	
	C = B
B = 2	
A += 2;	
	D = A
B++;	

Result: (0,3)

Ordering 9:

P1	P2
	C = B
A = 1	
B = 2	
A += 2;	
	D = A
B++;	

Result: (0,3)

Ordering 10:

P1	P2
A = 1	
B = 2	
	C = B
	D = A
A += 2;	
B++;	

Result: (2,1)

Ordering 11:

P1	P2
A = 1	
	C = B
B = 2	
	D = A
A += 2;	
B++;	

Result: (0,1)

Ordering 12:

P1	P2
	C = B
A = 1	
B = 2	
	D = A
A += 2;	
B++;	

Result: (0,1)

Ordering 13:

P1	P2
A = 1	
	C = B
	D = A
B = 2	
A += 2;	
B++;	

Result: (0,1)

Ordering 14:

P1	P2
	C = B
A = 1	
	D = A
B = 2	
A += 2;	
B++;	

Result: (0,1)

Ordering 15:

P1	P2
	C = B
	D = A
A = 1	
B = 2	
A += 2;	
B++;	

Result: (0,0)

5.17.5 Assume B=0 is seen by P2 but not preceding A=1

Result: (2,0)

5.17.6 Write back is simpler than write through, since it facilitates the use of exclusive access blocks and lowers the frequency of invalidates. It prevents the use of write-broadcasts, but this is a more complex protocol.

The allocation policy has little effect on the protocol.

5.18

5.18.1 Benchmark A

$$AMAT_{private} = (1/32) \times 5 + 0.0030 \times 180 = 0.70$$

$$AMAT_{shared} = (1/32) \times 20 + 0.0012 \times 180 = 0.84$$

Benchmark B

$$AMAT_{private} = (1/32) \times 5 + 0.0006 \times 180 = 0.26$$

$$AMAT_{shared} = (1/32) \times 20 + 0.0003 \times 180 = 0.68$$

Private cache is superior for both benchmarks.

5.18.2 Shared cache latency doubles for shared cache. Memory latency doubles for private cache.

Benchmark A

$$AMAT_{private} = (1/32) \times 5 + 0.0030 \times 360 = 1.24$$

$$AMAT_{shared} = (1/32) \times 40 + 0.0012 \times 180 = 1.47$$

Benchmark B

$$AMAT_{private} = (1/32) \times 5 + 0.0006 \times 360 = 0.37$$

$$AMAT_{shared} = (1/32) \times 40 + 0.0003 \times 180 = 1.30$$

Private is still superior for both benchmarks.

5.18.3

	Shared L2	Private L2
Single threaded	No advantage. No disadvantage.	No advantage. No disadvantage.
Multi-threaded	Shared caches can perform better for workloads where threads are tightly coupled and frequently share data.	Threads often have private working sets, and using a private L2 prevents cache contamination and conflict misses between threads.
	No disadvantage.	
Multiprogrammed	No advantage except in rare cases where processes communicate. The disadvantage is higher cache latency.	Caches are kept private, isolating data between processes. This works especially well if the OS attempts to assign the same CPU to each process.
Having private L2 caches with a shared L3 cache is an effective compromise for many workloads, and this is the scheme used by many modern processors.		

5.18.4 A non-blocking shared L2 cache would reduce the latency of the L2 cache by allowing hits for one CPU to be serviced while a miss is serviced for another CPU, or allow for misses from both CPUs to be serviced simultaneously. A non-blocking private L2 would reduce latency assuming that multiple memory instructions can be executed concurrently.

5.18.5 4 times.

5.18.6 Additional DRAM bandwidth, dynamic memory schedulers, multi-banked memory systems, higher cache associativity, and additional levels of cache.

f. Processor: out-of-order execution, larger load/store queue, multiple hardware threads;

Caches: more miss status handling registers (MSHR)

Memory: memory controller to support multiple outstanding memory requests

5.19

5.19.1 `srcIP` and `refTime` fields. 2 misses per entry.

5.19.2 Group the `srcIP` and `refTime` fields into a separate array.

5.19.3 `peak_hour (int status); // peak hours of a given status`

Group `srcIP`, `refTime` and status together.

5.19.4 Answers will vary depending on which data set is used.

Conflict misses do not occur in fully associative caches.

Compulsory (cold) misses are not affected by associativity.

Capacity miss rate is computed by subtracting the compulsory miss rate and the fully associative miss rate (compulsory + capacity misses) from the total miss rate. Conflict miss rate is computed by subtracting the cold and the newly computed capacity miss rate from the total miss rate.

The values reported are miss rate per instruction, as opposed to miss rate per memory instruction.

5.19.5 Answers will vary depending on which data set is used.

5.19.6 apsi/mesa/ammp/mcf all have such examples.

Example cache: 4-block caches, direct-mapped vs. 2-way LRU.

Reference stream (blocks): 1 2 2 6 1.