5

Solutions

5.1

5.1.1 2.

5.1.2 I, J, and B[I][0].

5.1.3 A[I][J].

5.1.4 I, J, and B[I][0].

5.1.5 A(J, I) and B[I][0].

5.1.6 32,004 with Matlab. 32,008 with C.

The code references 8*8000 = 64,000 integers from matrix A. At two integers per 16-byte block, we need 32,000 blocks.

The code also references the first element in each of eight rows of Matrix B. Matlab stores matrix data in column-major order; therefore, all eight integers are contiguous and fit in four blocks. C stores matrix data in row-major order; therefore, the first element of each row is in a different block.

5.2

5.2.1

Word Address	Binary Address	Tag	Index	Hit/Miss
0x03	0000 0011	0	3	M
0xb4	1011 0100	b	4	M
0x2b	0010 1011	2	b	M
0x02	0000 0010	0	2	M
0xbf	1011 1111	b	f	M
0x58	0101 1000	5	8	M
0xbe	1011 1110	b	e	M
0x0e	0000 1110	0	e	M
0xb5	1011 0101	b	5	M
0x2c	0010 1100	2	c	M
0xba	1011 1010	b	a	M
0xfd	1111 1101	f	d	M

5.2.2

Word Address	Binary Address	Tag	Index	Offset	Hit/Miss
0x03	0000 0011	0	1	1	M
0xb4	1011 0100	b	2	0	M
0x2b	0010 1011	2	5	1	M
0x02	0000 0010	0	1	0	H
0xbf	1011 1111	b	7	1	M
0x58	0101 1000	5	4	0	M
0xbe	1011 1110	b	6	0	H
0x0e	0000 1110	0	7	0	M
0xb5	1011 0101	b	2	1	H
0x2c	0010 1100	2	6	0	M
0xba	1011 1010	b	5	0	M
0xfd	1111 1101	f	6	1	M

5.2.3

Word Address	Binary Address	Tag	Cache 1		Cache 2		Cache 3	
			Index	Hit/miss	Index	Hit/miss	Index	Hit/miss
0x03	0000 0011	0x00	3	M	1	M	0	M
0xb4	1011 0100	0x16	4	M	2	M	1	M
0x2b	0010 1011	0x05	3	M	1	M	0	M
0x02	0000 0010	0x00	2	M	1	M	0	M
0xbf	1011 1111	0x17	7	M	3	M	1	M
0x58	0101 1000	0x0b	0	M	0	M	0	M
0xbe	1011 1110	0x17	6	M	3	H	1	H
0x0e	0000 1110	0x01	6	M	3	M	1	M
0xb5	1011 0101	0x16	5	M	2	H	1	M
0x2c	0010 1100	0x05	4	M	2	M	1	M
0xba	1011 1010	0x17	2	M	1	M	0	M
0xfd	1111 1101	0x1F	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.

5.3

5.3.1 Total size is 364,544 bits = 45,568 bytes

Each word is 8 bytes; each block contains two words; thus, each block contains 16 = 2^4 bytes.

The cache contains 32KiB = 2^15 bytes of data. Thus, it has 2^15/2^4 = 2^11 lines of data.

Each 64-bit address is divided into: (1) a 3-bit word offset, (2) a 1-bit block offset, (3) an 11-bit index (because there are 2^11 lines), and (4) a 49-bit tag (64 − 3 − 1 − 11 = 49).

The cache is composed of: 2^15 * 8 bits of data + 2^11*49 bits of tag + 2^11*1 valid bits = 364,544 bits.

5.3.2 549,376 bits = 68,672 bytes. This is a 51% increase.

Each word is 8 bytes; each block contains 16 words; thus, each block contains 128 = 2^7 bytes.

The cache contains 64KiB = 2^16 bytes of data. Thus, it has 2^16/2^7 = 2^9 lines of data.

Each 64-bit address is divided into: (1) a 3-bit word offset, (2) a 4-bit block offset, (3) a 9-bit index (because there are 2^9 lines), and (4) a 48-bit tag (64 − 3 − 4 − 9 = 48).

The cache is composed of: 2^16 * 8 bits of data + 2^9*48 bits of tag + 2^9*1 valid bits = 549,376 bits

5.3.3 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.

5.3.4 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 two-way set associate cache with LRU replacement, even one with a significantly smaller overall capacity, would hit on every access after the first two.

5.4 Yes it is possible. To implement a direct-mapped cache, we need only a function that will take an address as input and produce a 10-bit output. Although it is possible to implement a cache in this manner, it is not clear that such an implementation will be beneficial. (1) The cache would require a larger tag and (2) there would likely be more conflict misses.

5.5

5.5.1 Each cache block consists of four 8-byte words. The total offset is 5 bits. Three of those 5 bits is the word offset (the offset into an 8-byte word). The remaining two bits are the block offset. Two bits allows us to enumerate 2^2 = 4 words.

5.5.2 There are five index bits. This tells us there are 2^5 = 32 lines in the cache.

5.5.3 The ratio is 1.21. The cache stores a total of 32 lines * 4 words/block * 8 bytes/word = 1024 bytes = 8192 bits.

In addition to the data, each line contains 54 tag bits and 1 valid bit. Thus, the total bits required = 8192 + 54*32 + 1*32 = 9952 bits.

5.5.4

Byte Address	Binary Address	Tag	Index	Offset	Hit/Miss	Bytes Replaced
0x00	0000 0000 0000	0x0	0x00	0x00	M	
0x04	0000 0000 0100	0x0	0x00	0x04	H	
0x10	0000 0001 0000	0x0	0x00	0x10	H	
0x84	0000 1000 0100	0x0	0x04	0x04	M	
0xe8	0000 1110 1000	0x0	0x07	0x08	M	
0xa0	0000 1010 0000	0x0	0x05	0x00	M	
0x400	0100 0000 0000	0x1	0x00	0x00	M	0x00-0x1F
0x1e	0000 0001 1110	0x0	0x00	0x1e	M	0x400-0x41F
0x8c	0000 1000 1100	0x0	0x04	0x0c	H	
0xc1c	1100 0001 1100	0x3	0x00	0x1c	M	0x00-0x1F
0xb4	0000 1011 0100	0x0	0x05	0x14	H	
0x884	1000 1000 0100	0x2	0x04	0x04	M	0x80-0x9f

5.5.5 4/12 = 33%.

5.5.6
```
<index, tag, data>
<0, 3, Mem[0xC00]-Mem[0xC1F]>
<4, 2, Mem[0x880]-Mem[0x89f]>
<5, 0, Mem[0x0A0]-Mem[0x0Bf]>
<7, 0, Mem[0x0e0]-Mem[0x0ff]>
```

5.6

5.6.1 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.

5.6.2 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.

5.6.3 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.

5.7

5.7.1 When the CPI is 2, there are, on average, 0.5 instruction accesses per cycle. 0.3% of these instruction accesses cause a cache miss (and subsequent memory request). Assuming each miss requests one block, instruction accesses generate an average of 0.5*.003*64 = 0.096 bytes/cycle of read traffic.

25% of instructions generate a read request. 2% of these generate a cache miss; thus, read misses generate an average of 0.5*0.25*0.02*64 = 0.16 bytes/cycle of read traffic.

10% of instructions generate a write request. 2% of these generate a cache miss. Because the cache is a write-through cache, only one word (8 bytes) must be written back to memory; but, every write is written through to memory (not just the cache misses). Thus, write misses generate an average of 0.5*0.1*8 = 0.4 bytes/cycle of write traffic. Because the cache is a write-allocate cache, a write miss also makes a read request to RAM. Thus, write misses require an average of 0.5*0.1*0.02*64 = 0.064 bytes/cycle of read traffic.

Hence: The total read bandwidth = 0.096 + 0.16 + 0.064 = 0.32 bytes/cycle, and the total write bandwidth is 0.4 bytes/cycle.

5.7.2 The instruction and data read bandwidth requirement is the same as in 5.4.4.

With a write-back cache, data are only written to memory on a cache miss. But, it is written on every cache miss (both read and write), because any line could have dirty data when evicted, even if the eviction is caused by a read request. Thus, the data write bandwidth requirement becomes 0.5*(0.25 + 0.1)*0.02*0.3*64 = 0.0672 bytes/cycle.

5.8

5.8.1 The addresses are given as word addresses; each 32-bit block contains four words. Thus, every fourth access will be a miss (i.e., a miss rate of 1/4). 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.

5.8.2 The miss rates are 1/2, 1/8, and 1/16, respectively. The workload is exploiting spatial locality.

5.8.3 In this case the miss rate is 0: The pre-fetch buffer always has the next request ready.

5.9

5.9.1 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.

5.9.2 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

5.9.3 B = 128 is optimal: Minimizing the miss rate minimizes the total miss latency.

5.10

5.10.1

P1	1.515 GHz
P2	1.11 GHz

5.10.2

P1	6.31 ns	9.56 cycles
P2	5.11 ns	5.68 cycles

For P1 all memory accesses require at least one cycle (to access L1). 8% of memory accesses additionally require a 70 ns access to main memory. This is 70/0.66 = 106.06 cycles. However, we can't divide cycles; therefore, we must round up to 107 cycles. Thus, the Average Memory Access time is 1 + 0.08*107 = 9.56 cycles, or 6.31 ps.

For P2, a main memory access takes 70 ns. This is 70/0.66 = 77.78 cycles. Because we can't divide cycles, we must round up to 78 cycles. Thus the Average Memory Access time is 1 + 0.06*78 = 5.68 cycles, or 6.11 ps.

5.10.3

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

For P1, every instruction requires at least one cycle. In addition, 8% of all instructions miss in the instruction cache and incur a 107-cycle delay. Furthermore, 36% of the instructions are data accesses. 8% of these 36% are cache misses, which adds an additional 107 cycles.

1 + .08*107 + .36*.08*107 = 12.64

With a clock cycle of 0.66 ps, each instruction requires 8.34 ns.

Using the same logic, we can see that P2 has a CPI of 7.36 and an average of only 6.63 ns/instruction.

5.10.4

AMAT = 9.85 cycles	Worse

An L2 access requires nine cycles (5.62/0.66 rounded up to the next integer).

All memory accesses require at least one cycle. 8% of memory accesses miss in the L1 cache and make an L2 access, which takes nine cycles. 95% of all L2 access are misses and require a 107 cycle memory lookup.

1 + .08[9 + 0.95*107] = 9.85

5.10.5

13.04

Notice that we can compute the answer to 5.6.3 as follows: AMAT + %memory * (AMAT-1).

Using this formula, we see that the CPI for P1 with an L2 cache is 9.85 * 0.36*8.85 = 13.04

5.10.6 Because the clock cycle time and percentage of memory instructions is the same for both versions of P1, it is sufficient to focus on AMAT. We want

AMAT with L2 < AMAT with L1 only

1 + 0.08[9 + m*107] < 9.56

This happens when m< .916.

5.10.7 We want P1's average time per instruction to be less than 6.63 ns. This means that we want

(CPI_P1 * 0.66) < 6.63. Thus, we need CPI_P1 < 10.05

CPI_P1 = AMAT_P1 + 0.36(AMAT_P1 – 1)

Thus, we want

AMAT_P1+ 0.36(AMAT_P1-1) < 10.05

This happens when AMAT_P1< 7.65.

Finally, we solve for

1+ 0.08[9+ m*107] < 7.65

and find that

m< 0.693

This miss rate can be at most 69.3%.

5.11

5.11.1 Each line in the cache will have a total of six blocks (two in each of three ways). There will be a total of 48/6 = 8 lines.

5.11.2 T(x) is the tag at index x.

Word Address	Binary Address	Tag	Index	Offset	Hit/Miss	Way 0	Way 1	Way 2
0x03	0000 0011	0x0	1	1	M	T(1)=0		
0xb4	1011 0100	0xb	2	0	M	T(1)=0 T(2)=b		
0x2b	0010 1011	0x2	5	1	M	T(1)=0 T(2)=b T(5)=2		
0x02	0000 0010	0x0	1	0	H	T(1)=0 T(2)=b T(5)=2		
0xbe	1011 1110	0xb	7	0	M	T(1)=0 T(2)=b T(5)=2 T(7)=b		
0x58	0101 1000	0x5	4	0	M	T(1)=0 T(2)=b T(5)=2 T(7)=b T(4)=5		
0xbf	1011 1111	0xb	7	1	H	T(1)=0 T(2)=b T(5)=2 T(7)=b T(4)=5		
0x0e	0000 1110	0x0	7	0	M	T(1)=0 T(2)=b T(5)=2 T(7)=b T(4)=5	T(7)=0	
0x1f	0001 1111	0x1	7	1	M	T(1)=0 T(2)=b T(5)=2 T(7)=b T(4)=5	T(7)=0	T(7)=1
0xb5	1011 0101	0xb	2	1	H	T(1)=0 T(2)=b T(5)=2 T(7)=b T(4)=5	T(7)=0	T(7)=1
0xbf	1011 1111	0xb	7	1	H	T(1)=0 T(2)=b T(5)=2 T(7)=b T(4)=5	T(7)=0	T(7)=1
0xba	1011 1010	0xb	5	0	M	T(1)=0 T(2)=b T(5)=2 T(7)=b T(4)=5	T(7)=2 T(5)=b	T(7)=1
0x2e	0010 1110	0x2	7	0	M	T(1)=0 T(2)=b T(5)=2 T(7)=b T(4)=5	T(7)=2 T(5)=b	T(7)=1
0xce	1100 1110	0xc	7	0	M	T(1)=0 T(2)=b T(5)=2 T(7)=b T(4)=5	T(7)=2 T(5)=b	T(7)=c

5.11.3 No solution given.

5.11.4 Because this cache is fully associative and has one-word blocks, there is no index and no offset. Consequently, the word address is equivalent to the tag.

Word Address	Binary Address	Tag	Hit/Miss	Contents
0x03	0000 0011	0x03	M	3
0xb4	1011 0100	0xb4	M	3, b4
0x2b	0010 1011	0x2b	M	3, b4, 2b
0x02	0000 0010	0x02	M	3, b4, 2b, 2
0xbe	1011 1110	0xbe	M	3, b4, 2b, 2, be
0x58	0101 1000	0x58	M	3, b4, 2b, 2, be, 58
0xbf	1011 1111	0xbf	M	3, b4, 2b, 2, be, 58, bf
0x0e	0000 1110	0x0e	M	3, b4, 2b, 2, be, 58, bf, e
0x1f	0001 1111	0x1f	M	b4, 2b, 2, be, 58, bf, e, 1f
0xb5	1011 0101	0xb5	M	2b, 2, be, 58, bf, e, 1f, b5
0xbf	1011 1111	0xbf	H	2b, 2, be, 58, e, 1f, b5, bf
0xba	1011 1010	0xba	M	2, be, 58, e, 1f, b5, bf, ba
0x2e	0010 1110	0x2e	M	be, 58, e, 1f, b5, bf, ba, 2e
0xce	1100 1110	0xce	M	58, e, 1f, b5, bf, ba, 2e, ce

5.11.5 No solution given.

5.11.6 Because this cache is fully associative, there is no index. (Contents shown in the order the data were accessed. Order does not imply physical location.)

Word Address	Binary Address	Tag	Offset	Hit/Miss	Contents
0x03	0000 0011	0x01	1	M	[2,3]
0xb4	1011 0100	0x5a	0	M	[2,3], [b4,b5]
0x2b	0010 1011	0x15	1	M	[2,3], [b4,b5], [2a,2b]
0x02	0000 0010	0x01	0	H	[b4,b5], [2a,2b], [2,3]
0xbe	1011 1110	0x5f	0	M	[b4,b5], [2a,2b], [2,3], [be, bf]
0x58	0101 1000	0x2c	0	M	[2a,2b], [2,3], [be, bf], [58, 59]
0xbf	1011 1111	0x5f	1	H	[2a,2b], [2,3], [58, 59], [be, bf]
0x0e	0000 1110	0x07	0	M	[2,3], [58, 59], [be, bf], [e,f]
0x1f	0001 1111	0x0f	1	M	[58, 59], [be, bf], [e,f], [1e,1f]
0xb5	1011 0101	0x5a	1	M	[be, bf], [e,f], [1e,1f], [b4, b5]
0xbf	1011 1111	0x5f	1	H	[e,f], [1e,1f], [b4, b5], [be, bf]
0xba	1011 1010	0x5d	0	M	[1e,1f], [b4, b5], [be, bf], [ba, bb]
0x2e	0010 1110	0x17	0	M	[b4, b5], [be, bf], [ba, bb], [2e, 2f]
0xce	1100 1110	0x67	0	M	[be, bf], [ba, bb], [2e, 2f], [ce,cf]

5.11.7 (Contents shown in the order the data were accessed. Order does not imply physical location.)

Word Address	Binary Address	Tag	Offset	Hit/Miss	Contents
0x03	0000 0011	0x01	1	M	[2,3]
0xb4	1011 0100	0x5a	0	M	[2,3], [b4,b5]
0x2b	0010 1011	0x15	1	M	[2,3], [b4,b5], [2a,2b]
0x02	0000 0010	0x01	0	H	[b4,b5], [2a,2b], [2,3]
0xbe	1011 1110	0x5f	0	M	[b4,b5], [2a,2b], [2,3], [be, bf]
0x58	0101 1000	0x2c	0	M	[b4,b5], [2a,2b], [2,3], [58, 59]
0xbf	1011 1111	0x5f	1	M	[b4,b5], [2a,2b], [2,3], [be, bf]
0x0e	0000 1110	0x07	0	M	[b4,b5], [2a,2b], [2,3], [e, f]
0x1f	0001 1111	0x0f	1	M	[b4,b5], [2a,2b], [2,3], [1e, 1f]
0xb5	1011 0101	0x5a	1	H	[2a,2b], [2,3], [1e, 1f], [b4,b5]
0xbf	1011 1111	0x5f	1	M	[2a,2b], [2,3], [1e, 1f], [be, bf]
0xba	1011 1010	0x5d	0	M	[2a,2b], [2,3], [1e, 1f], [ba, bb]
0x2e	0010 1110	0x17	0	M	[2a,2b], [2,3], [1e, 1f], [2e, 2f]
0xce	1100 1110	0x67	0	M	[2a,2b], [2,3], [1e, 1f],, [ce, cf]

5.11.8 Because this cache is fully associative, there is no index.

Word Address	Binary Address	Tag	Offset	Hit/Miss	Contents
0x03	0000 0011	0x01	1	M	[2,3]
0xb4	1011 0100	0x5a	0	M	[2,3], [b4,b5]
0x2b	0010 1011	0x15	1	M	[2,3], [b4,b5], [2a,2b]
0x02	0000 0010	0x01	0	H	[2,3], [b4,b5], [2a,2b]
0xbe	1011 1110	0x5f	0	M	[2,3], [b4,b5], [2a,2b], [be, bf]
0x58	0101 1000	0x2c	0	M	[58,59], [b4,b5], [2a,2b], [be, bf]
0xbf	1011 1111	0x5f	1	H	[58,59], [b4,b5], [2a,2b], [be, bf]
0x0e	0000 1110	0x07	0	M	[e,f], [b4,b5], [2a,2b], [be, bf]
0x1f	0001 1111	0x0f	1	M	[1e,1f], [b4,b5], [2a,2b], [be, bf]
0xb5	1011 0101	0x5a	1	H	[1e,1f], [b4,b5], [2a,2b], [be, bf]
0xbf	1011 1111	0x5f	1	H	[1e,1f], [b4,b5], [2a,2b], [be, bf]
0xba	1011 1010	0x5d	0	M	[1e,1f], [b4,b5], [ba,bb], [be, bf]
0x2e	0010 1110	0x17	0	M	[1e,1f], [b4,b5], [2e,2f], [be, bf]
0xce	1100 1110	0x67	0	M	[1e,1f], [b4,b5], [ce,cf], [be, bf]

5.12

5.12.1 Standard memory time: Each cycle on a 2-Ghz machine takes 0.5 ps. Thus, a main memory access requires 100/0.5 = 200 cycles.

- L1 only: 1.5 + 0.07*200 = 15.5
- Direct mapped L2: 1.5 + .07 × (12 + 0.035 × 200) = 2.83
- 8-way set associated L2: 1.5 + .07 × (28 + 0.015 × 200) = 3.67.

Doubled memory access time (thus, a main memory access requires 400 cycles)

- L1 only: 1.5 + 0.07*400 = 29.5 (90% increase)
- Direct mapped L2: 1.5 + .07 × (12 + 0.035 × 400) = 3.32 (17% increase)
- 8-way set associated L2: 1.5 + .07 × (28 + 0.015 × 400) = 3.88 (5% increase).

5.12.2 $1.5 = 0.07 \times (12 + 0.035 \times (50 + 0.013 \times 100)) = 2.47$

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

5.12.3 No size will achieve the performance goal.

We want the CPI of the CPU with an external L2 cache to be at most 2.83. Let x be the necessary miss rate.

1.5 + 0.07*(50 + x*200) < 2.83

Solving for x gives that $x < -0.155$. This means that even if the miss rate of the L2 cache was 0, a 50-ns access time gives a CPI of 1.5 + 0.07*(50 + 0*200) = 5, which is greater than the 2.83 given by the on-chip L2 caches. As such, no size will achieve the performance goal.

5.13

5.13.1

3 years and 1 day	1096 days	26304 hours

5.13.2

1095/1096 = 99.90875912%

5.13.3 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.

5.13.4 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.

5.14

5.14.1 9. For SEC, we need to find minimum p such that $2^p >= p + d + 1$ and then add one. That gives us p = 8. We then need to add one more bit for SEC/DED.

5.14.2 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 better cost/performance ratio.

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

5.15 Instructors can change the disk latency, transfer rate and optimal page size for more variants. Refer to Jim Gray's paper on the 5-minute rule 10 years later.

5.15.1 32 KB.

To solve this problem, I used the following gnuplot command and looked for the maximum.

```
plot [16:128] log((x*1024/128) *0.7)/(log(2)*(10 + 0.1*x))
```

5.15.2 Still 32 KB. (Modify the gnuplot command above by changing 0.7 to 0.5.)

5.15.3 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.

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

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

5.15.4 (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.

5.16

5.16.1

Address	Virtual Page	TLB H/M	TLB		
			Valid	Tag	Physical Page
4669 0x123d	1	TLB miss PT hit PF	1	b	12
			1	7	4
			1	3	6
			1 (last access 0)	1	13
2227 0x08b3	0	TLB miss PT hit	1 (last access 1)	0	5
			1	7	4
			1	3	6
			1 (last access 0)	1	13
13916 0x365c	3	TLB miss PT hit	1 (last access 1)	0	5
			1	7	4
			1 (last access 2)	3	6
			1 (last access 0)	1	13
34587 0x871b	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 0xbee6	b	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)	b	12
12608 0x3140	3	TLB miss PT hit	1 (last access 1)	0	5
			1 (last access 3)	8	14
			1 (last access 5)	3	6
			1 (last access 4)	b	12
49225 0xc040	c	TLB miss PT hit PF	1 (last access 6)	c	15
			1 (last access 3)	8	14
			1 (last access 5)	3	6
			1 (last access 4)	b	12

5.16.2

Address	Virtual Page	TLB H/M	TLB		
			Valid	Tag	Physical Page
4669 0x123d	1	TLB miss PT hit	1	11	12
			1	7	4
			1	3	6
			1 (last access 0)	0	5
2227 0x08b3	0	TLB hit	1	11	12
			1	7	4
			1	3	6
			1 (last access 1)	0	5
13916 0x365c	0	TLB hit PT hit	1	11	12
			1	7	4
			1	3	6
			1 (last access 2)	0	5
34587 0x871b	2	TLB miss PT hit PF	1 (last access 3)	2	13
			1	7	4
			1	3	6
			2	0	5
48870 0xbee6	2	TLB hit PT hit	1 (last access 4)	2	13
			1	7	4
			1	3	6
			1 (last access 2)	0	5
12608 0x3140	0	TLB hit PT hit	1 (last access 4)	2	13
			1	7	4
			1	3	6
			5	0	5
49225 0xc040	3	TLB hit PT hit	1 (last access 4)	2	13
			1	7	4
			1 (last access 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.

5.16.3 Two-way set associative

Address	Virtual Page	Tag	Index	TLB H/M	TLB			
					Valid	Tag	Physical Page	Index
4669 0x123d	1	0	1	TLB miss PT hit PF	1	b	12	0
					1	7	4	1
					1	3	6	0
					1 (last access 0)	0	13	1
2227 0x08b3	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 0x365c	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 0x871b	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 0xbee6	b	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 0x3140	3	1	1	TLB hit PT 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 0xc049	c	6	0	TLB miss PT miss PF	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

5.16.4 Direct mapped

Address	Virtual Page	Tag	Index	TLB H/M	TLB			
					Valid	Tag	Physical Page	Index
4669 0x123d	1	0	1	TLB miss PT hit PF	1	b	12	0
					1	0	13	1
					1	3	6	2
					0	4	9	3
2227 0x08b3	0	0	0	TLB miss PT hit	1	0	5	0
					1	0	13	1
					1	3	6	2
					0	4	9	3
13916 0x365c	3	0	3	TLB miss PT hit	1	0	5	0
					1	0	13	1
					1	3	6	2
					1	0	6	3
34587 0x871b	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 0xbee6	b	2	3	TLB miss PT hit	1	2	14	0
					1	0	13	1
					1	3	6	2
					1	2	12	3
12608 0x3140	3	0	3	TLB hit PT hit	1	2	14	0
					1	0	13	1
					1	3	6	2
					1	0	6	3
49225 0xc049	c	3	0	TLB miss PT miss PF	1	3	15	0
					1	0	13	1
					1	3	6	2
					1	0	6	3

5.16.5 Without a TLB, almost every memory access would require two accesses to RAM: An access to the page table, followed by an access to the requested data.

5.17

5.17.1 The tag size is 32–log2(8192) = 32–13 = 19 bits. All five page tables would require 5 × (2^19 × 4) bytes = 10 MB.

5.17.2 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 contains 2^(19 – 8) = 2048 entries, requiring 2048 × 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 1st-level 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.

5.17.3 The page index is 13 bits (address bits 12 down to 0).

A 16 KB direct-mapped cache with two 64-bit words per block would have 16-byte blocks and thus 16 KB/16 bytes = 1024 blocks. Thus, it would have 10 index bits and 4 offset bits and the index would extend outside of the page index.

The designer could increase the cache's associativity. This would reduce the number of index bits so that the cache's index fits completely inside the page index.

5.18

5.18.1 Worst case is 2^(43 − 12) = 2^31 entries, requiring 2^(31) × 4 bytes = 2^33 = 8 GB.

5.18.2 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.

5.18.3 Yes, if segment table entries are assumed to be the physical page numbers of segment pages, and one bit is reserved as the valid bit, then each one has an effective reach of (2^{31}) * 4KiB = 8TiB, which is more than enough to cover the physical address space of the machine (16 GiB).

5.18.4 Each page table level contains 4KiB/4B = 1024 entries, and so translates log2(1024) = 10 bits of virtual address. Using 43-bit virtual addresses and 4KiB pages, we need ceil((43 − 12)/10) = 4 levels of translation.

5.18.5 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.

5.19

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

5.19.2 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.

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

5.20

5.20.1 There are no hits.

5.20.2 Direct mapped

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

5.20.3 Answers will vary.

5.20.4 MRU is an optimal policy.

5.20.5 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.

5.20.6 If you knew that an address had limited temporal locality and would conflict with another block in the cache, choosing not to cache it could improve the miss rate. On the other hand, you could worsen the miss rate by choosing poorly which addresses to cache.

5.21

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

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

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

The CPI of a machine running on native hardware is 1.5 + 120/10000*15 = 1.68. To keep the performance degradation to 10%, we need

1.5 + 120/10000*(15 + x) < 1.1*1.68

Solving for x shows that a trap to the VMM can take at most 14 cycles.

5.21.2 Non-virtualized CPI = 1.5 + 120/10000 × 15 + 30/10000 × 1100 = 4.98

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

Non-virtualized CPI with half I/O = 1.5 + 120/10000 × 15 + 15/10000 × 1100 = 3.33

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

5.22 Virtual memory aims to provide each application with the illusion of the entire address space of the machine. Virtual machines aim to provide each operating system with the illusion of having the entire machine at 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.

5.23 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 degradation 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 recent Intel processors that do micro-op fusion, allowing several instructions to be handled by fewer instructions.

5.24

5.24.1 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.

5.24.2 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.

5.24.3 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.

5.25

5.25.1 There are six 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).

5.25.2 Direct mapped

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 write X block in cache		X block is invalided
X[1] = 3;	write X block in cache		

5.25.3 Best case:

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

Worst case:

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

5.25.4

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)

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

Result: (2,0).

5.25.6 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.

5.26

5.26.1 Benchmark A

AMATprivate = 1 + 0.03*[5 + 0.1*180] = 1.69

AMATshared = 1 + 0.03*[20 + 0.04*180] = 1.82

Benchmark B

AMATprivate = 1+ 0.03*[5 + 0.02*180] = 1.26

AMATshared = 1+ 0.03*[20 + 0.01*180] = 1.65

Private cache is superior for both benchmarks.

5.26.2 In a private cache system, the first link of the chip is the link from the private L2 caches to memory. Thus, the memory latency doubles to 360. In a shared cache system, the first link off the chip is the link to the L2 cache. Thus, in this case, the shared cache latency doubles to 40.

Benchmark A

$AMAT_{private} = 1 + .03*[5 + .1*360] = 2.23$

$AMAT_{shared} = 1 + .03*[40 + .04*180] = 2.416$

Benchmark B

$AMAT_{private} = 1 + .03*[5 + .02*360] = 1.37$

$AMAT_{shared} = 1 + .03*[40 + .01*180] = 2.25$

Private cache is superior for both benchmarks.

5.26.3

	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. No disadvantage.	Threads often have private working sets, and using a private L2 prevents cache contamination and conflict misses between threads.
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.

5.26.4 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.

5.26.5 Four times.

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

5.27

5.27.1 `srcIP` and `refTime` fields. Two misses per entry.

5.27.2 Group the `srcIP` and `refTime` fields into a separate array. (I.e., create two parallel arrays. One with `srcIP` and `refTime`, and the other with the remaining fields.)

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

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

5.28

5.28.1 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.

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

5.28.3 Answers will vary.

5.29

5.29.1 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.

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

Native: L; NPT: $L \times (L + 2)$.

5.29.3 Shadow page table: page fault rate.

NPT: TLB miss rate.

5.29.4 Shadow page table: 1.03

NPT: 1.04.

5.29.5 Combining multiple page table updates.

5.29.6 NPT caching (similar to TLB caching).