Solution 5.2

5.2.1 4

5.2.2

a.	I, J, B[J][0]
b.	I, J

5.2.3

a.	A[I][J]
b.	A[J][I]

5.2.4

a.	$3186 = 8 \times 800/4 \times 2 - 8 \times 8/4 + 8/4$
b.	$3596 = 8 \times 800/4 \times 2 - 8 \times 8/4 + 8000/4$

5.2.5

a.	I, J, B(J, 0)
b.	I, J

5.2.6

a.	A(I, J), A(J, I), B(J, 0)
b.	A(I, J)

Solution 5.3

5.3.1

a.	Binary address: 1_2, 10000110_2, 11010100_2, 1_2, 10000111_2, 11010101_2, 10100010_2, 10100001_2, 10_2, 101100_2, 101001_2, 11011101_2 Tag: Binary address >> 4 bits Index: Binary address mod 16 Hit/Miss: M, M, M, H, M, M, M, M, M, M, M, M
b.	Binary address: 00000110_2, 11010110_2, 10101111_2, 11010110_2, 00000110_2, 01010100_2, 01000001_2, 10101110_2, 01000000_2, 01101001_2, 01010101_2, 11010111_2 Tag: Binary address >> 4 bits Index: Binary address modulus 16 Hit/Miss: M, M, M, H, M, M, M, M, M, M, M, M

5.3.2

a.	Binary address: 1_2, 10000110_2, 11010100_2, 1_2, 10000111_2, 11010101_2, 10100010_2, 10100001_2, 10_2, 101100_2, 101001_2, 11011101_2 Tag: Binary address >> 3 bits Index: (Binary address >> 1 bit) mod 8 Hit/Miss: M, M, M, H, H, H, M, M, M, M, M, M
b.	Binary address: 00000110_2, 11010110_2, 10101111_2, 11010110_2, 00000110_2, 01010100_2, 01000001_2, 10110000_2, 01000000_2, 01101001_2, 01010101_2, 11010111_2 Tag: Binary address shift right 3 bits Index: (Binary address shift right 1 bit) modulus 8 Hit/Miss: M, M, M, H, M, M, M, H, H, M, H, M

5.3.3

a.	C1: 1 hit, C2: 3 hits, C4: 2 hits. C1: Stall time = 25 × 11 + 2 × 12 = 299, C2: Stall time = 25 × 9 + 3 × 12 = 261, C3: Stall time = 25 × 10 + 4 × 12 = 298
b.	C1: 1 hit, stall time = 25 × 11 + 2 × 12 = 299 cycles C2: 4 hits, stall time = 25 × 8 + 3 × 12 = 236 cycles C3: 4 hits, stall time = 25 × 8 + 5 × 12 = 260 cycles

5.3.4

a.	Using equation on page 351, n = 14 bits, m = 0 (1 word per block) $2^{14} \times (2^0 \times 32 + (32 - 14 - 0 - 2) + 1) = 802$ Kbits Calculating for 16 word blocks, m = 4, if n = 10 then the cache is 541 Kbits, and if n = 11 then the cache is 1 Mbit. Thus the cache has 128 KB of data. The larger cache may have a longer access time, leading to lower performance.
b.	Using equation total cache size = $2^n \times (2^m \times 32 + (32 - n - m - 2) + 1)$, n = 13 bits, m = 1 (2 words per block) $2^{13} \times (2^1 \times 32 + (32 - 13 - 1 - 2) + 1) = 2^{13} \times (64 + 17) = 663$ Kbits total cache size For m = 4 (16 word blocks), if n = 10 then the cache is 541 Kbits and if n = 11 then cache is 1 Mbits. Thus the cache has 64 KB of data. The larger cache may have a longer access time, leading to lower performance.

5.3.5 For a larger direct-mapped cache to have a lower or equal miss rate than a smaller 2-way set associative cache, it would need to have at least double the cache block size. The advantage of such a solution is less misses for near by addresses (spatial locality), but with the disadvantage of suffering longer access times.

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

Solution 5.5

5.5.1

a.	L1 => Write-back buffer => L2 => Write buffer
b.	L1 => Write-back buffer => L2 => Write buffer

5.5.2

a.	1. Allocate cache block for the missing data, select a replacement victim; 2. If victim dirty, put it into the write-back buffer, which will be further forwarded into L2 write buffer; 3. Issue write miss request to the L2 cache; 4. If hit in L2, source data into L1 cache; if miss, send write request to memory; 5. Data arrives and is installed in L1 cache; 6. Processor resumes execution and hits in L1 cache, set the dirty bit.
b.	1. If L1 miss, allocate cache block for the missing data, select a replacement victim; 2. If victim dirty, put it into the write-back buffer, which will be further forwarded into L2 write buffer; 3. Issue write miss request to the L2 cache; 4. If hit in L2, source data into L1 cache, goto (8); 5. If miss, send write request to memory; 6. Data arrives and is installed in L2 cache; 7. Data arrives and is installed in L1 cache; 8. Processor resumes execution and hits in L1 cache, set the dirty bit.

5.5.3

a.	Similar to 5.5.2, except that (2) If victim clean, put it into a victim buffer between the L1 and L2 caches; If victim dirty, put it into the write-back buffer, which will be further forwarded into L2 write buffer; (4) If hit in L2, source data into L1 cache, invalidate the L2 copy;
b.	Similar to 5.5.2, except that – if L1 victim clean, put it into a victim buffer between the L1 and L2 caches; – if L1 victim dirty, put it into the write-back buffer, which will be further forwarded into L2 write buffer; – if hit in L2, source data into L1 cache, invalidate copy in L2;

5.5.4

a.	0.166 reads and 0.160 writes per instruction (0.5 cycles). Minimal read/write bandwidths are 0.664 and 0.640 byte-per-cycle.
b.	0.152 reads and 0.120 writes per instruction (0.5 cycles). Minimal read/write bandwidths are 0.608 and 0.480 byte-per-cycle.

5.5.5

a.	0.092 reads and 0.0216 writes per instruction (0.5 cycles). Minimal read/write bandwidths are 0.368 and 0.0864 byte-per-cycle.
b.	0.084 reads and 0.0162 writes per instruction (0.5 cycles). Minimal read/write bandwidths are 0.336 and 0.0648 byte-per-cycle.

5.5.6

a.	Write-back, write-allocate cache saves bandwidth. Minimal read/write bandwidths are 0.4907 and 0.1152 byte-per-cycle.
b.	Write-back, write-allocate cache saves bandwidth. Minimal read/write bandwidths are 0.4478 and 0.0863 byte-per-cycle

Solution 5.6

5.6.1

12.5% miss rate. The miss rate doesn't change with cache size or working set. These are cold misses.

5.6.2

25%, 6.25% and 3.125% miss rates for 16-byte, 64-byte and 128-byte blocks. Spatial locality.

5.6.3 With next-line prefetching, miss rate will be near 0%.

5.6.4

a.	16-byte.
b.	8-byte.

5.6.5

a.	32-byte.
b.	8-byte.

5.6.6

a.	64-byte.
b.	64-byte.

Solution 5.7

5.7.1

a.	P1	1.61 GHz
	P2	1.52 GHz
b.	P1	1.04 GHz
	P2	926 MHz

5.7.2

a.	P1	8.60 ns	13.87 cycles
	P2	6.26 ns	9.48 cycles
b.	P1	3.97 ns	4.14 cycles
	P2	3.46 ns	3.20 cycles

5.7.3

a.	P1	5.63	P2
	P2	4.05	
b.	P1	2.13	P2
	P2	1.79	

5.7.4

a.	8.81 ns	14.21 cycles	Worse
b.	3.65 ns	3.80 cycles	Better

5.7.5

a.	5.76
b.	2.01

5.7.6

a.	P1 with L2 cache: CPI = 5.76. P2: CPI = 4.05. P2 is still faster than P1 even with an L1 cache
b.	P1 with L2 cache: CPI = 2.01. P2: CPI = 1.79. P2 is still faster than P1 even with an L1 cache