7.10 Simply extend the comparison to include the valid bit as the high-order bit of the cache tag and extend the address tag by adding a high-order “1” bit. Now the values are equal only if the tags match and the valid bit is a 1.

7.11 The miss penalty is the time to transfer one block from main memory to the cache. Assume that it takes one clock cycle to send the address to the main memory.

a. Configuration (a) requires 16 main memory accesses to retrieve a cache block and words of the block are transferred 1 at a time.
   Miss penalty = $1 + 16 \times 10 + 16 \times 1 = 177$ clock cycles.

b. Configuration (b) requires 4 main memory accesses to retrieve a cache block and words of the block are transferred 4 at a time.
   Miss penalty = $1 + 4 \times 10 + 4 \times 1 = 45$ clock cycles.

c. Configuration (c) requires 4 main memory accesses to retrieve a cache block and words of the block are transferred 1 at a time.
   Miss penalty = $1 + 4 \times 10 + 16 \times 1 = 57$ clock cycles.

7.12 Effective CPI = Base CPI + Miss rate per instruction × Miss penalty.

For memory configurations (a), (b), and (c):

a. Effective CPI = $1.2 + 0.005 \times 177 = 2.085$ clocks/instruction.

b. Effective CPI = $1.2 + 0.005 \times 45 = 1.425$ clocks/instruction.

c. Effective CPI = $1.2 + 0.005 \times 57 = 1.485$ clocks/instruction.

Now

Speedup of A over B = Execution time B / Execution time A,

and

Execution time = Number of instructions × CPI × Clock cycle time.

We have the same CPU running the same software on the various memory configurations, so the number of instructions and the clock cycle time are fixed. Thus, speed can be compared by comparing CPI.

Speedup of configuration (b) over configuration (a) = $2.085/1.425 = 1.46$.

Speedup of configuration (b) over configuration (c) = $1.485/1.425 = 1.04$.

7.13 The shortest reference string will have 4 misses for C1 and 3 misses for C2; this leads to 32 versus 33 miss cycles. The following reference string will do: 0, 4, 8, 11.
7.27 Here are the cycles spent for each cache:

<table>
<thead>
<tr>
<th>Cache</th>
<th>Miss penalty</th>
<th>Instruction miss cycles per instruction</th>
<th>Data miss cycles per data reference</th>
<th>Total miss cycles per instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>6 + 1 = 7</td>
<td>4% × 7 = 0.28</td>
<td>8% × 7 = 0.56</td>
<td>0.56</td>
</tr>
<tr>
<td>C2</td>
<td>6 + 4 = 10</td>
<td>2% × 10 = 0.20</td>
<td>5% × 10 = 0.50</td>
<td>0.45</td>
</tr>
<tr>
<td>C3</td>
<td>6 + 4 = 10</td>
<td>2% × 10 = 0.20</td>
<td>4% × 10 = 0.40</td>
<td>0.40</td>
</tr>
</tbody>
</table>

So, C3 spends the least time on misses and C1 spends the most.

7.28 Execution time = CPI × Clock cycle × Instruction count

Execution time_{C1} = 0.56 × 2 ns × IC = 1.12 × IC × 10^{-9}

Execution time_{C2} = 0.45 × 2 ns × IC = 0.9 × IC × 10^{-9}

Execution time_{C3} = 0.40 × 2.4 ns × IC = 0.96 × IC × 10^{-9}

So C2 is the fastest and C1 is the slowest.

7.29 If direct-mapped and stride = 132, then we can assume without loss of generality that array[0] is in slot 0, along with array[1], [2], [3]. Then, slot 1 has [4], [5], [6], [7], slot 2 has [8], [9], [10], [11], and so on until slot 15 has [60], [61], [62], [63] (there are 16 slots × 16 bytes = 256 bytes in the cache). Then wrapping around, we find also that slot 0 has [64], [65], [66], [67] and also [128], [129], [130], [131], etc. Thus if we look at array[0], array[132], we are looking at two different slots. After filling these entries, there will be no misses. Thus the expected miss rate is about 0. If the stride equals 131, we are looking at slot 0 and slot 0 again. Each reference will bounce out the other, and we will have 100% misses. If the cache is two-way set-associative, even if two accesses are in the same cache line they can coexist, so the miss rate will be 0. Alternative explanation: 132 in binary is 10000100, and clearly when this is added to the word address of array[0], bit 2 will change and thus the slots are different. 131 in binary is 10000011, and in this case the last two bits are used for the block offset within the cache. The addition will not change the index, so the two entries will map to the same location.

7.30 We can build a 384-KB cache. The address breakdown is 15 bits for tag, 15 bits for set index, 0 bits for block offset, and 2 bits for byte offset. A total of 18 chips will be required; 6 for overhead.

7.31 No solution provided.

7.32 The total size is equal to the number of entries times the size of each entry. The number of entries is equal to the number of pages in the virtual address, which is

\[
\frac{2^{40} \text{ bytes}}{16 \text{ KB}} = \frac{2^{40} \text{ bytes}}{2^4 \cdot 2^{10} \text{ bytes}} = 2^{26}
\]

The width of each entry is 4 + 3 bits = 40 bits = 8 bytes. Thus the page table contains \(2^{29}\) bytes or 512 MB!

7.33 No solution provided.

7.34 No solution provided.
8 Solutions

8.1 Each transaction requires 50,000 instructions and 5 I/O operations. The CPU can handle transactions at a maximum rate of 50M/50K or 1000 transactions per second. The I/O limit for system A is 200 TPS and for system B it’s 150 TPS. These are the limiting rates.

8.2 For system A, 20 ms/IO × 5 IO/transaction × n transactions/10 = 1000 ms, n = 100 TPS.

Assume system B operates at greater than 100 TPS (and less than 500 I/O operations per second). Let n be the number over 100:

\[(18 \times 5 \times 100 + n \times 5 \times 25)/10 = 1000 \text{ ms}. \quad n = 8 \text{ or } 108 \text{ TPS.}\]

This solution ignores the fact that the first set of transactions that can be performed simultaneously numbers only 9. After the first 9 transactions, subsequent transactions can be processed in batches of 10, comprising one transaction dependent on a preceding transaction and a new set of 9 independent transactions.

8.3 After reading sector 7, a seek is necessary to get to the track with sector 8 on it. This will take some time (on the order of a millisecond, typically), during which the disk will continue to revolve under the head assembly. Thus, in the version where sector 8 is in the same angular position as sector 0, sector 8 will have already revolved past the head by the time the seek is completed and some large fraction of an additional revolution time will be needed to wait for it to come back again. By skewing the sectors so that sector 8 starts later on the second track, the seek will have time to complete, and then the sector will soon thereafter appear under the head without the additional revolution.

8.4 For ATM to be twice as fast, we would need

\[15 + 6 + 200 + 241 + \text{Transmission time}_{\text{Ethernet}} = 2 \times (50 + 6 + 207 + 360 + \text{Transmission time}_{\text{ATM}})\]

\[462 + X/1.125 = 2(623 + X/10), \text{ so } X = 1138 \text{ bytes.}\]

8.5 \[100 / (3 \times 10^8 \times .5) = .67 \text{ microsec per byte and } 5000 \times 10^3 / (3 \times 10^8 \times .5) = .034 \text{ seconds.}\]

8.6 \[.67 \times 10^{-6} \times 5 \times 10^6 = 3.35 \text{ bytes for the 100-meter network, and } .034 \times 5 \times 10^6 = .17 \text{ MB for the 5000-km network.}\]

8.7 4 KHz = 4 × 10^3 samples/sec × 2 bytes/sample = 8 × 10^3 bytes/sec × 100 conversations = 8 × 10^5 bytes/sec. Transmission time per packet is 1 KB/1 MB/sec = 1 millisecond plus a 350-microsecond latency for a total of 0.00135 seconds. The time to transmit all 800 packets collected from one second of monitoring is 800 × 0.00135 = 1.08 seconds. Thus, the chosen network with its given latency does not have sufficient effective bandwidth for the task.

8.8 We determine an average disk access time of 8 ms + 4.2 ms + .2 ms + 2 ms = 14.4 ms. Since each block processed involves two accesses (read and write), the disk component of the time is 28.8 ms per block processed. The (non-overlapped) computation takes 20 million cycles at 400 MHz, or another 50 ms. Thus, the total time to process one block is 78.8 ms, and the number of blocks processed per second is simply \[1/0.0788 = 12.7.\]
8.1 Each transaction requires 50,000 instructions and 5 I/O operations. The CPU can handle transactions at a maximum rate of 50M/50K or 1000 transactions per second. The I/O limit for system A is 200 TPS and for system B it’s 150 TPS. These are the limiting rates.

8.2 For system A, 20 ms/IO × 5 IO/transaction × n transactions/10 = 1000 ms, n = 100 TPS.

Assume system B operates at greater than 100 TPS (and less than 500 I/O operations per second). Let n be the number over 100:

\[(18 \times 5 \times 100 + n \times 5 \times 25)/10 = 1000\text{ ms.} n = 8\text{ or } 108\text{ TPS.}\]

This solution ignores the fact that the first set of transactions that can be performed simultaneously numbers only 9. After the first 9 transactions, subsequent transactions can be processed in batches of 10, comprising one transaction dependent on a preceding transaction and a new set of 9 independent transactions.

8.3 After reading sector 7, a seek is necessary to get to the track with sector 8 on it. This will take some time (on the order of a millisecond, typically), during which the disk will continue to revolve under the head assembly. Thus, in the version where sector 8 is in the same angular position as sector 0, sector 8 will have already revolved past the head by the time the seek is completed and some large fraction of an additional revolution time will be needed to wait for it to come back again. By skewing the sectors so that sector 8 starts later on the second track, the seek will have time to complete, and then the sector will soon thereafter appear under the head without the additional revolution.

8.4 For ATM to be twice as fast, we would need

\[15 + 6 + 200 + 241 + \text{Transmission time}_{\text{Ethernet}} = 2 \times (50 + 6 + 207 + 360 + \text{Transmission time}_{\text{ATM}})\]

\[462 + X/1.125 = 2(623 + X/10),\text{ so } X = 1138\text{ bytes.}\]

8.5 \[100 / (3 \times 10^8 \times .5) = .67\text{ microseconds and } 5000 \times 10^3 / (3 \times 10^8 \times .5) = .034\text{ seconds.}\]

8.6 \[.67 \times 10^{-6} \times 5 \times 10^6 = 3.35\text{ bytes for the 100-meter network, and } .034 \times 5 \times 10^6 = .17\text{ MB for the 5000-km network.}\]

8.7 \[4 \text{ KHz} = 4 \times 10^3 \text{ samples/sec} \times 2 \text{ bytes/sample} = 8 \times 10^3 \text{ bytes/sec} \times 100 \text{ conversations} = 8 \times 10^5 \text{ bytes/sec}.\text{ Transmission time per packet is } 1\text{ KB/1 MB/sec = 1 millisecond plus a 350-microsecond latency for a total of } 0.00135\text{ seconds. The time to transmit all 800 packets collected from one second of monitoring is } 800 \times 0.00135 = 1.08\text{ seconds.}\text{ Thus, the chosen network with its given latency does not have sufficient effective bandwidth for the task.}\]

8.8 We determine an average disk access time of 8 ms + 4.2 ms + .2 ms + 2 ms = 14.4 ms. Since each block processed involves two accesses (read and write), the disk component of the time is 28.8 ms per block processed. The (non-overlapped) computation takes 20 million cycles at 400 MHz, or another 50 ms. Thus, the total time to process one block is 78.8 ms, and the number of blocks processed per second is simply \(1/0.0788 = 12.7\).
NT-1. Consider a 4-way set associative cache that stores 2048 sets of 4-word blocks. This works with a byte-addressable DRAM with 32-bit addresses and 32-bit words. Specify exactly what kind of information is in the cache at each addressable location, taking into account whether the write policy is write back or write thru.

Each location (set) contains 4 blocks. Each block consists of a valid bit, a tag, 4 32-bit data words, and, if the write policy is write-back, a dirty bit. Since the cache has 2048 (= 2^11) sets or addressable locations, 11 bits of the address word are used to access the cache. These are bits 14-4 (bits 3, 2 locate a word in the block, 1 and 0 are byte locations). Bits 31-15 of the address make up the tag, i.e., there is a 17-bit tag in each block. So, for a write-thru cache, each block in the set contains 1+1+17+4x32 = 147 bits, and each set contains 4x147 = 588 bits for a write-back cache, or 584 bits for a write-thru cache.

NT-2. If the word at the following DRAM address is in the cache specified in the previous question, where would it be and how would it be found? 1010 1111 0000 0011 1100 0011 1111 1100

The cache is addressed by bits 14-4, which are 100 0011 1111. After reading out the set at that location, we match the tag: 1010 1111 0000 0011 1 against the tags for the 4 blocks in the set, also matching the valid bits against 1. If the word is in the cache, there will be a match and a valid bit equal to 1. We then, within the block, use the MUX to select the word from the block at the internal location given by the word location bits, 01 (bits 3 and 2) of the DRAM address.

NT-3. Key parameters of a magnetic disk drive are the seek time, rotational latency, controller time, transfer rate, and disk capacity. (a) Suppose a technology improvement allowed us to double the number of bits on a track, without changing the disk diameter or the number of tracks. How, if at all, would that affect each of the above listed parameters?

(b) Suppose, instead, the number of tracks could be doubled, without changing the disk diameter or the number of bits per track. How would this affect each of the key parameters?

(a) The transfer rate and the disk capacity would both be doubled, since we obviously are packing twice as many bits into the disk and, in a given unit of time, twice as many bits pass under the head for reading or writing. The other parameters are not affected.

(b) Only the disk capacity would be changed; it would be doubled. (The assumption here is that the number of tracks per inch of radius is doubled, without changing the region on the disk that is occupied by the tracks. If the tracks extended over a greater part of the disk radius, then the seek time would be increased.)