Question 1(a)

Number of cache blocks $= \frac{16 \text{ Kilobytes}}{64 \text{ Bytes}} = 256$

X is an array of 4096 elements, each consisting of 4 Bytes, so X is 4 Bytes $*$ 4096 = 16 Kilobytes in total. X and Y are consecutive in memory, so the start of Y is 16 kilobytes after X, exactly the size of the cache. Because the cache is direct-mapped, this means for all i X[i] and Y[i] will map to the same cache block.

In each iteration, the memory operations are a load to X[i], a load to Y[i], then a store to X[i].

We assume that the start of X maps to the start of a cache block, i.e. the offset for X[0] is 0.

Compulsory Misses

Each cache block can store $\frac{64 \text{ Bytes}}{4 \text{ Bytes}} = 16$ elements of either array, so only the first time we access an element per 16 element subset of each array is it a compulsory miss.

Each array occupies $\frac{4096}{16} = 256$ cache lines, so there are 256 compulsory misses per array for a total of 512 compulsory misses.

Conflict Misses

Every store to X is a conflict miss since it is displaced by the preceding load to Y, and the remaining 15 out of every 16 loads to Y will miss since it occupies the same cache block as the preceding load to X. This gives a total of 4096 + (15/16) * 4096 = 4096 + 3840 = 7936 conflict misses.

Capacity Misses

There are no capacity misses in a direct-mapped cache because each memory address maps to a unique cache line, meaning the cache can only experience conflict or compulsory misses but not misses due to insufficient space.

Total Misses and Miss Rate

The total number of misses is 7936 conflict misses plus the 512 compulsory misses. Therefore the final total is $7936 + 512 = 8448$ misses out of $3 * 4096$ memory accesses, since each iteration has three memory accesses and there are 4096 iterations. The miss rate is then $\frac{8448}{12288} = 68.75\%$.

Question 1(b)

An example of a software solution that significantly reduces the number of data cache misses is adding a buffer between the end of X and the start of Y so that X[i] and Y[i] do not map to the same set. If we put a buffer between them equal to the size of a single cache block, then X[0] and Y[0] will map to blocks n and $n+1$, avoiding the multiple conflict misses per iteration since they can be accessed without evicting one another.

The compulsory misses on X, excluding X[0], are also conflict misses since X[i] maps to the same block as Y[$i-16$], but this is not relevant for counting total misses since they are already counted as compulsory misses. Similarly, the last compulsory miss on Y maps to the same set as X[0] and can also be labeled a conflict miss. It is acceptable to list conflict misses as 256 or 0.

The total number of misses is then only the 512 compulsory misses, leading to a miss rate of $\frac{512}{12288} \approx 4.17\%$.

Question 1(c)

An example of a hardware solution that significantly reduces the number of data cache misses is making the cache two-way set associative. Doing so allows X[i] and Y[i] to be stored in the same set instead of evicting one another.

Number of sets = $\frac{256}{2} = 128$

The number of compulsory misses is still 512, since the arrays still occupy 256 cache lines each.

Conflict misses are reduced to 256, since the first 2048 iterations have zero conflict misses and the second 2048 iterations each have 2 conflict misses, one when loading X and one when loading Y. This is the case because there are only 128 sets and X occupies 256 cache lines, so X[2048] now maps to the same set as X[0]. The same is true for Y. It is also acceptable to list the number of conflict misses as zero, because each of these conflict misses is also already a compulsory miss.

The number of capacity misses is also 256, because each conflict miss is also a capacity miss due to the cache being full and thus the set is full. Again, these misses are also counted as compulsory misses already, and thus listing capacity misses as zero is also acceptable.

The total number of misses is 512 compulsory misses, leading to a miss rate of $\frac{512}{12288} = 4.17\%$.

Question 2(1)

Number of cache blocks $= \frac{16\times 1024}{64} = 256$.

Since the cache is 2-way set associative, number of sets $= \frac{256}{2} = 128$.

Number of bits for set index $= \log_2(128) = 7$.

Number of bits for block offset $= \log_2(64) = 6$.

Number of bits for tag $= 32 - 7 - 6 = 19$.

Therefore, the cache uses 7 bits for index and 19 bits for tag.

Question 2(2)

Number of bits used for page offset $= \log_2(4096) = 12$.

Number of bits used for virtual page number $= 32 - 12 - 20$.

For a virtual address, bits 31 to 12 form the virtual page number and bits 12 to 6 form the set index. If the cache is implemented with a VI-PT organization, bits 31 to 12 are used to access the TLB to obtain the physical page number and bits 12 to 6 are used to access the L1 cache to obtain the (physical) tag number. The tag number is compared with the tag bits in the physical page number to determine whether an access hits in the cache. This leads to two potential problems:

Homonyms. Multiple physical addresses can be mapped to the same virtual address in different processes. Suppose that two distinct physical addresses have the same bits in every position except at bit 12 and are mapped to the same virtual address. Since the virtual addresses are the same, both physical addresses will have the same virtual index and be mapped to the same cache set. Since the physical addresses have the same bits 31 to 13, they have the same physical tag. This makes it impossible to distinguish between the two physical addresses when comparing the physical tags after accessing the cache with the virtual index, causing false cache hits and accesses to physical addresses that are supposed to be private to other processes.

For example, suppose that the physical address 0x0000 is mapped to virtual address 0x1000 in process 0 and physical address 0x1000 is also mapped to virtual address 0x1000 in process 1. First, process 0 accessed virtual address 0x1000 so the data at physical address 0x0000 is cached in set 64. Next, process 1 accesses virtual address 0x1000. Translating virtual page number 1 for process 1 returns physical page number 1, which has tag 0. In parallel, looking up set 64 in the cache gives a physical tag 0 that is valid. This is a cache hit. If the access to 0x1000 by process 1 is a load, then process 1 would have read data at physical address 0x0000 instead of physical address 0x0000. If the access to 0x1000 by process 1 is a store, then process 1 would have written data to the cached copy which can be read by process 0.

Synonyms. Multiple virtual addresses can point to a single physical address. Since bit 12 is part of the virtual page number, different virtual addresses can have different values for bit 12. However, bit 12 is also used in the set index. This allows a single physical address to be cached in two different sets and updates to one of the cached copy would not be reflected in the other cached copy. As a result, reading the data from the cache could return a stale value.

Consider this example:

Process 0 and process 1 share data at physical address 0x0000 and the data is initially 0. Process 0 maps physical address 0x0000 to virtual address 0x0000 while process 1 maps physical address 0x0000 to virtual address 0x1000. When process 0 writes value 1 to the shared variable, the block is allocated in set 0 and the value is written to the cached block. Suppose that after some synchronization event, process 1 reads from address 0x1000. Translating virtual page number 1 gives physical page number 0. In parallel, looking up set 64 in the cache gives no valid block. So the block containing physical address 0x0000 is fetched from memory, allocated in set 64 and the value of 0 is returned to process 1. As a result, process 1 reads the stale value of the shared data.

Question 2(3)

The key is to ensure that all of the bits used for the set index are the same in both the physical address and the virtual address. Solutions include:

Increasing associativity to 4-way while keeping cache and block size constant. Since the cache still has 256 blocks but there are 4 blocks in each set, there are $\frac{256}{4} = 64$ sets. So 6 bits are used for the set index. The total number of bits used for set index and block offset $= 6 + 6 = 12 =$ number of bits used for the page offset.

Page coloring. When the OS is creating the mapping from virtual to physical addresses, ensure that bit 12 of the virtual address retains the same value as in the physical address.

Increasing page size from 4 KB to 8 KB. This increases the number of bits used for page offset to 13 which is the same as the total number of bits used for the set index and block offset.