Question 2 (20 points):

1. (**5 points**) How many bits are used for tag, index, and offset in a 128-KB cache that is 8-way set associative with 64-byte block used in a processor with 32-bit address?

$$
\begin{aligned}
\#\text{ of bytes per set} &= 8 \times 64 = 512 \\
\#\text{ of sets} &= \frac{128 \times 1024}{512} = 256 \\
\#\text{ of offset bits} &= \log_2 64 = 6 \text{ bits} \\
\#\text{ of index bits} &= \log_2 256 = 8 \text{ bits} \\
\#\text{ of tag bits} &= 32 - 6 - 8 = 18 \text{ bits}
\end{aligned}
$$

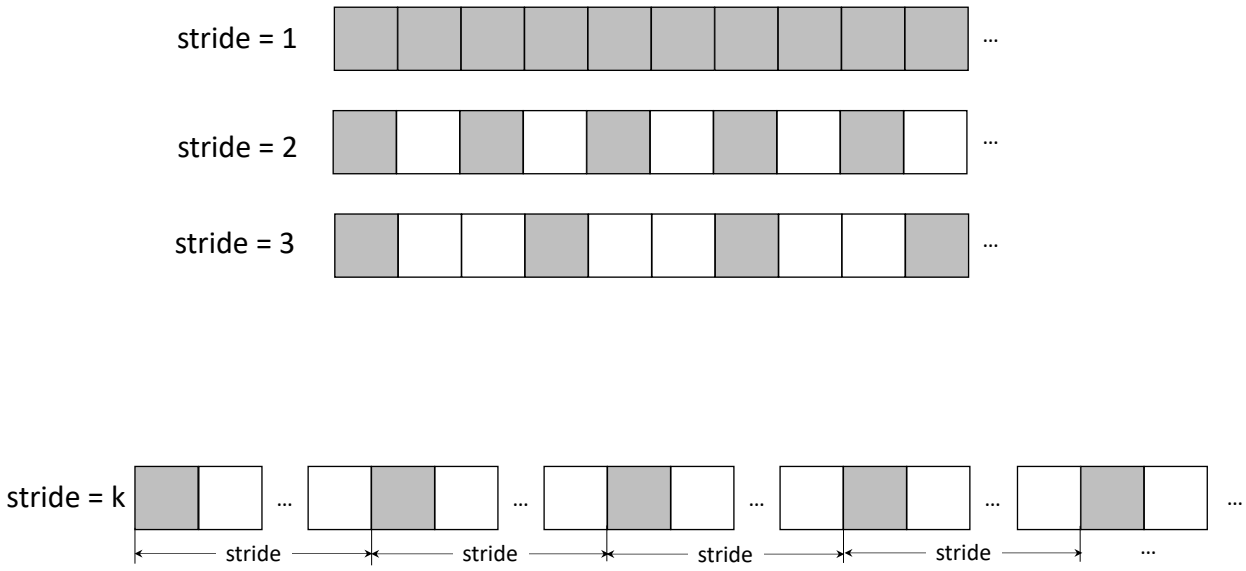

Figure 1: Illustration of strided access into a vector.

2. (**5 points**) A strided access into a vector consists in a program regularly skipping over elements of the vector. In the illustration shown in Figure 1, each small square represents a 32-bit integer value. A dark square represents a vector element that is accessed by a program while a white box is a vector element that is not accessed. Figure 1 illustrates accesses into a vector with different strides. Figure 2 shows the C code and the RISCV code for the function `StridedDotProduct`. We are interested in the accesses to the data cache that occur inside the loop body. The loop index `i` and the variable `Sum` are kept in registers. Assume that in a given

```
StridedDotProduct:
    bgtz    a3, doSum   # if N>0 goto doSum
    mv      a0, zero    # return Sum=0
    jalr    zero,ra,0
doSum:
    mv      t0, zero     # Sum <- 0
    mv      t1, zero     # i <- 0
next:
    sll     t2, t1, 2    # t2 <- 4*i
    add     t3, a0, t2   # t3 <- Addr(A[i])
    lw      t4, 0(t3)    # t4 <- A[i]
    add     t5, a1, t2   # t5 <- Addr(B[i])
    lw      t6, 0(t5)    # t6 <- B[i]
    mul     t7, t5, t6   # t7 <- A[i]*B[i]
    add     t1, t1, a2   # i <- i + stride
    blt     t1, a3, next # if i<N goto next
    jalr    zero,ra,0
```

(a) C code

```
StridedDotProduct:
    bgtz    a3, doSum   # if N>0 goto doSum
    mv      a0, zero    # return Sum=0
    jalr    zero,ra,0
doSum:
    mv      t0, zero     # Sum <- 0
    mv      t1, zero     # i <- 0
next:
    sll     t2, t1, 4    # t2 <- 4*i
    add     t3, a0, t2   # t3 <- Addr(A[i])
    lw      t4, 0(t3)    # t4 <- A[i]
    add     t5, a1, t2   # t5 <- Addr(B[i])
    lw      t6, 0(t5)    # t6 <- B[i]
    mul     t7, t5, t6   # t7 <- A[i]*B[i]
    add     t1, t1, a2   # i <- i + stride
    blt     t1, a3, next # if i<N goto next
    jalr    zero,ra,0
```

(b) RISCV code

Figure 2: Code for `StridedDotProduct`

execution of this function, `A = 0x40000000` and `B = 0x60000000`, and `N` is very large. We will study the memory hierarchy for executions of `StridedDotProduct` with different values of `stride`. In the table below write the memory addresses, expressed in hexadecimal, for the first eight data accesses in `StridedDotProduct` for each execution of the function with the value of `stride` specified at the top of the column.

Data Access	stride=1	stride=2	stride=4
1	0x40000000	0x40000000	0x40000000
2	0x60000000	0x60000000	0x60000000
3	0x40000004	0x40000008	0x40000010
4	0x60000004	0x60000008	0x60000010
5	0x40000008	0x40000010	0x40000020
6	0x60000008	0x60000010	0x60000020
7	0x4000000C	0x40000018	0x40000030
8	0x6000000C	0x60000018	0x60000030

3. (**5 points**) Assume that the machine where `StridedDotProduct` is executed has a 32KB data cache with 16-byte blocks. In the table below write the hit ratio for the data accesses while executing `StridedDotProduct` for the specified cache organization.

Cache Organization	stride=1	stride=2	stride=4
Direct Mapped	0%	0%	0%
Two-Way Set Associative	75%	50%	0%

4. (**5 points**) How would the answers to item (c) above change if the size of the cache was changed to 64KB for both possible cache organizations. Explain your answer in terms of spatial and temporal locality.

   The answers would not change because the hits in the L1 data cache are only due to spatial locality, and doubling the size of the cache does not affect the hits because of spatial locality.