Question 4 (18 points):

When generating initial code, a compiler generates naive code where the instructions for the execution of each memory access or for the computation of each expression is done independently of other such accesses or computation in the code. Consider the following function written in C to compute the Hadamard product of two vectors:

```
void Hadamard(int *A, int *B, int N){
    for(i=0; i<N; i++){
        A[i] = A[i] * B[i];
    }
}
```

```
HadamardNaive:
        add     t0, zero, zero  # i <- 0
loop:   bge     t0, a2, done    # if i >= N goto done
        slli    t1, t0, 2       # t1 <- 4*i
        add     t2, a0, t1      # t2 <- Address(A[i])
        lw      t3, 0(t2)       # t3 <- A[i]
        slli    t4, t0, 2       # t4 <- 4*i
        add     t5, a1, t4      # t5 <- Address(B[i])
        lw      t6, 0(t5)       # t6 <- B[i]
        mul     t7, t3*t6       # t7 <- A[i]*B[i]
        slli    t8, t0, 2       # t8 <- 4*i
        add     t9, a0, t8      # t9 <- Address(B[i])
        sw      t7, 0(t9)       # A[i] <- A[i]*B[i]
        addi    t0, t0, 1       # i <- i+1
        jal     zero, loop
done:   jalr zero, ra, 0
```

(a) Naive Version

```
HadamardOpt:
        slli    t0, a2, 2       # t0 <- 4*N
        add     t1, a0, t0      # t1 <- Address(A[N])
        bge     zero, a2, done  # if(0<=N) done
loop:   lw      t2, 0(a0)       # t2 <- A[i]
        lw      t3, 0(a1)       # t3 <- B[i]
        mul     t4, t2, t3      # t4 <- A[i]*B[i]
        sw      t4, 0(a0)       # A[i] <- A[i]*B[i]
        addi    a0, a0, 4       # pA++
        addi    a1, a1, 4       # pB++
        blt     a0, t1, loop    # if pA < Address(A[N])
done:   jalr zero, ra, 0
```

(b) Optimized Version

Figure 1: Two versions for a Dot Product function.

Figure **??** shows the RISC-V assembly code for a naive version and for an optimized version of the this function. In this question you will study the performance of these two versions. A performance study was conducted to determine the average number of clock cycles for different types of instructions. Based on this study, the instructions used in Figure **??** are classified into the following classes of instructions:

ALU Instructions (`add`, `slli`, `addi`): 1 cycle

Jumps and Branches (`bge`, `blt`, `jal`, `jalr`): 3 cycles

Memory instructions (`lw`, `sw`): 5 cycles

Multiplication Instructions (`mul`): 10 cycles

a. (**3 points**) Assuming that `N` is very large, what is the CPI of the naive version and what is the CPI of the optimized version?

b. (**4 points**) Again, assuming that N is very large, which version is faster and by how much?

c. (**5 points**) A given invocation of the optimized version of `Hadamard` executes in 15 seconds in a baseline machine with a clock cycle of 2 GHz (1 GHz = 10^9 Hz). What was the value of N for this invocation of the `Hadamard` function?

d. (**6 points**) A new version of the same processor has been designed that implements the following changes:

- The clock frequency is 3 GHz
- The average number of cycles required to execute memory instructions is reduced to 3 cycles
- The number of cycles required to execute a multiplication instruction is also reduced

To discover what is the average number of cycles required to execute a multiplication operation in this new version of the machine you run an experiment where you execute the optimized version of `Hadamard` with $\mathrm{N} = 2 \times 10^6$. You find out that the execution of such an invocation of `Hadamard` takes 12.7 ms (1 ms = $10^{-3} seconds$). What is the average number of cycles required for multiplication operations in this new version of the machine?