Question 4 (10 points): Consider the following segment of C code:

```
...
V[V[i]] = V[V[i+1]];
...
```

The variable `i` is declared as an integer, and variable `V` is declared as an array of integers.

a. (**5 points**) Immediately prior to the execution of this statement, `V` and `i` are in RISC-V registers. How many load instructions and how many store instructions must be used in the RISC-V assembly code that executes this C statement?

3 loads and one store

```
# assume that  V is in s0 and i is in s1
slli t0, s1, 2  # t0 <- 4*i
add  t1, s0, t0 # t1 <- Address(V[i])
lw   t2, 4(t1)  # t2 <- V[i+1]
slli t3, t2, 2  # t3 <- 4*V[i+1]
add  t4, s0, t3 # t4 <- Address(V[V[i+1]])
lw   t5, 0(t4)  # t5 <- V[V[i+1]]
lw   t6, 0(s0)  # t6 <- V[i]
slli t7, t6, 2  # t7 <- 4*V[i]
add  t8, t7, s0 # t8 <- Address(V[V[i]])
sw   t4, 0(t8)  # V[V[i]] <- V[V[i+1]]
```

b. (**5 points**) Immediately prior to the execution of this statement, `V` and `i` are in memory locations in the stack frame of the function. How many load instructions and how many store instructions must be used in the RISC-V assembly code used to generate the code for this assignment?

5 loads and one store

```
# assume that  V is in Mem[sp] and i is in Mem[sp+4]
lw  s0, 0(sp) # s0 <- V
lw  s1, 4(sp) # s1 <- i
slli t0, s1, 2  # t0 <- 4*i
add  t1, s0, t0 # t1 <- Address(V[i])
lw   t2, 4(t1)  # t2 <- V[i+1]
slli t3, t2, 2  # t3 <- 4*V[i+1]
add  t4, s0, t3 # t4 <- Address(V[V[i+1]])
lw   t5, 0(t4)  # t5 <- V[V[i+1]]
lw   t6, 0(s0)  # t6 <- V[i]
slli t7, t6, 2  # t7 <- 4*V[i]
add  t8, t7, s0 # t8 <- Address(V[V[i]])
sw   t4, 0(t8)  # V[V[i]] <- V[V[i+1]]
```