Question 2 (10 points):

a. (**8 points**) Assume the following C function declaration:

```
void foo(...) {
int i, j, x;
int v[ArrayLength];
int *p;
...
}
```

Thus, `x`, `v`, `i`, `j`, `p` are variables with an allocation in the stack frame of `foo`. In the table below, write how many load instructions and how many store instructions are needed to implement each one of the C statements listed. Assume that independent code needs to be generated for each statement — do not assume that the result of instructions executed for one of the statements could be used for the subsequent statement.

C code statement	# of loads	# of stores
`x = v[i]`	2	1
`v[j] = v[i]`	3	1
`v[v[j]] = *p`	4	1
`j = v[*p]`	3	1

`x = v[i]`: load `i`, load `v[i]`, store `i`
`v[j] = v[i]`: load `i`, load `v[i]`, load `j`, store `v[j]`
`v[v[j]] = *p`: load `p`, load `*p`, load `j`, load `v[j]`, store `v[v[j]]` `*(p+3) = v[*p]`: load `p`, load `*p`, load `v[*p]`, store `j`

b. (**2 points**) Assume the following declaration for a C function:

```
int bar(int v[], int k){
...
}
```

You are tasked with trying to determine what the function `bar` does based on the RISC-V code for this function.

```
bar: sll  t0, a1, 2
     add  t1, a0, t0
     lw   t2, 0(t1)
     sll  t3, t2, 2
     add  t4, a0, t3
     lw   a0, 0(t4)
     jalr zero, ra, 0
```

Interpret this assembly code and write an equivalent C code for function `bar` that performs exactly the same computation as this assembly code.

```
int bar(int v[], int k){
return(v[v[k]]);
}
```