1.5.5 Exercise 5:

Comment what each code block does. Each block runs in isolation. Assume that there is an array, int arr[6] = 3, 1, 4, 1, 5, 9, which starts at memory address 0xBFFFFF00, and a linked list struct (as defined below), struct ll* lst, whose first element is located at address 0xABCD0000. Let s0 contain arr's address 0xBFFFFF00, and let s1 contain lst's address 0xABCD0000. You may assume integers and pointers are 4 bytes and that structs are tightly packed. Assume that lst's last node's next is a NULL pointer to memory address 0x00000000.

```
    struct ll {
int val;
struct ll* next;
}
```

1.
   ```
   lw t0, 0(s0)
   lw t1, 8(s0)
   add t2, t0, t1
   sw t2, 4(s0)
   ```

2.
   ```
   loop: beq s1, x0, end
   lw t0, 0(s1)
   addi t0, t0, 1
   sw t0, 0(s1)
   lw s1, 4(s1)
   jal x0, loop
   end:
   ```

3.
   ```
   add t0, x0, x0
   loop: slti t1, t0, 6
   beq t1, x0, end
   slli t2, t0, 2
   add t3, s0, t2
   lw t4, 0(t3)
   sub t4, x0, t4
   sw t4, 0(t3)
   addi t0, t0, 1
   jal x0, loop
   end:
   ```

1.5.6 Exercise 6:

In a function called myfunc, we want to call two functions called GenerateRandom and reverse.
myfunc takes in 3 arguments: a0, a1, a2
generate random takes in no arguments and returns a random integer to a0.
reverse takes in 4 arguments: a0, a1, a2, a3 and doesn't return anything.

```
    myfunc:
# Prologue (omitted)

# assign registers to hold arguments to myfunc
addi t0 a0 0
addi s0 a1 0
```

```
addi a7 a2 0

# Save the registers in 4.2
jal GenerateRandom
# Load the registers stored from 4.2

# store and process return value
addi t1 a0 0
slli t5 t1 2

# setup arguments for reverse
add a0 t0 x0
add a1 s0 x0
add a2 t5 x0
addi a3 t1 0

# Save the registers in 4.3
jal reverse
# Load the registers stored from 4.2

# additional computations
add t0 s0 x0
add t1 t1 a7
add s9 s8 s7
add s3 x0 t5

# Epilogue (omitted)
ret
```

1. Which registers, if any, need to be saved on the stack in the prologue?
2. Which registers do we need to save on the stack before calling generate random?
3. Which registers do we need to save on the stack before calling reverse?
4. Which registers need to be recovered in the epilogue before returning?

1.5.7 Exercise 7:

Please answer true/false to the following questions, and include an explanation:

1. Let a0 point to the start of an array x. lw s0, 4(a0) will always load x[1] into s0.
2. Assuming no compiler or operating system protections, it is possible to have the code jump to data stored at 0(a0) (offset 0 from the value in register a0) and execute instructions from there.
3. jalr is a shorthand expression for a jal that jumps to the specified label and does not store a return address anywhere.
4. After calling a function and having that function return, the t registers may have been changed during the execution of the function, while a registers cannot.
5. In order to use the saved registers (s0-s11) in a function, we must store their values before using them and restore their values before returning.
6. The stack should only be manipulated at the beginning and end of functions, where the callee saved registers are temporarily saved.
7. Assume that s0 and s1 contain signed integers. Without any pseudoinstructions, how can we branch on the following conditions to jump to some LABEL?
   $s0 < s1$ $s0 \neq s1$ $s0 \leq s1$ $s0 > s1$

1.5.8 Exercise 8:

Translate between RISC-V and C.

1.
   ```
   // s0 -> a, s1 -> b
   int a = 5, b = 10;
   if(a + a == b)
   a = 0;
   else
   b = a - 1;
   ```

2.
   ```
   addi s0, x0, 0
   addi s1, x0, 1
   addi t0, x0, 30
   loop:
   beq s0, t0, exit
   add s1, s1, s1
   addi s0, s0, 1
   jal x0, loop
   exit:
   ```

3.
   ```
   // s0 -> n, s1 -> sum
   // assume n > 0 to start
   for(int sum = 0; n > 0; n–)
   sum += n;
   ```