1

a. Which caller-saved registers are preserved in this function, if any?
a0, a1, t0, t1, ra

b. Which callee-saved registers are preserved in this function, if any?
There are no callee-saved registers preserved in this function

c. Are there caller-saved registers that are used but not preserved? If so, why is this okay?
Yes: t3, t4, t5, t6. This is okay because the user must know that the swap_s function does not alter those registers.

d. How many bytes of the stack are actually used by this function?
64

e. Does this function use pointer-based array access or indexed-based array access?
Index-based array access

2.
This C function recursively counts the number of times a character appears in a string. If the first character in the string is the null terminator, the function returns 0. If the first character of the string is equal to c, the variable addval is set to 1. The function then returns addval plus a recursive call to the method where a substring of the original string is passed in.

```
.global count_rec_s

# a0 - char *str
# a1 - char c

# t0 - addval
# t1 - temp reg


count_rec_s:
     li t0, 0                    # addval = 0

     lb t2, (a0)                 # load first char into t2

     beq t2, zero, str_done
```

```
      beq t2, a1, char_found

      j char_found

char_found:
      li t0, 1                    # addval = 1

recur:
      addi a0, a0, 1
      call count_rec_s
      add a0, a0, t0              # add addval to res
      ret

str_done:
      li a0, 0
      ret
```

3.

```
uint32_t extract = (iw >> 20) & ((1 << 12) - 1);

int shifted = (extract >> 11) & 1;
int temp = shifted == 1;

int32_t signed;

if (temp) {
      signed = ~((1 << 12) - 1);
} else {
      signed = 0;
}

int32_t imm = signed | extract;
```