You want to transfer secret messages to your friend in a non-encrypted app. You are concerned with others seeing your messages. Thus you come up with a very simple encoding/decoding mechanism. This is a very weak encryption system, but you feel that it is enough to thwart the technically naive people that you are concerned about reading your messages.

Your encryption/decryption algorithm simply flips one bit in each character in the message. Flipping a bit consists in changing a 0 to a 1 or changing a 1 to a 0. The algorithm flips the bit 0 of the character 0, the bit 1 of the character 1, and so on. Once it gets to bit 7, it goes back to bit 0. Thus, it will flip the bit 0 of character 8, the bit 1 of character 9, and so on.

To implement this algorithm you will implement two functions.

Question 5 (20 points): Write RISC-V assembly for the function `flipBitInByte` that flips a single bit in a byte. The arguments for this function are:

- a0: address of character in memory
- a1: an integer between zero and seven that specifies a bit to be flipped

`flipBitInByte` replaces the specified character in memory with a character where the specified bit is flipped. `flipBitInByte` has no return value.

The assembly code that you write must follow all the register saving/restoring conventions for RISC-V.

```
35 # flipBitInByte:
36 # a0: address of byte in memory
37 # a1: an integer between zero and seven specifies
38 #     a bit to be flipped
39 # Pseudo code:
40 # masc <- a byte with a single bit equal one
41 # bit <- masc&byte
42 # if bit != 0
43 #    temp <- byte&~masc
44 # else
45 #    temp <- byte|masc
46 # Mem[a0] <- temp
47 flipBitInByte:
48 lbu     t0, 0(a0)       # t0 <- byte
49 li      t1, 1           # t1
50 sll     t2, t1, a1      # masc <- only specified bit is 1
51 xor     t3, t0, t1      # t3 <- byteXORmasc flip only one bit
52 sb      t3, 0(a0)       # store byte
53 jalr    zero, ra, 0
```

Figure 1: A solution for `flipBitInByte`.

```
55 flipBitInByte:
56     lbu     t0, 0(a0)       # t0 <- byte
57     li      t1, 1           # t1
58     sll     t2, t1, a1      # masc <- only specified bit is
59     and     t3, t0, t1      # t3 <- byte&masc
60     beq     t3, zero, bitIsZero # if t3 is zero the bit is
61     xori    t4, t2, 0x00FF  # t4 <- ~masc
62     and     t0, t0, t4      # t0 <- byte&~masc
63     j       skipOver
64 bitIsZero:
65     or      t0, t0, t2      # t0 <- byte|masc
66 skipOver:
67     sb      t0, 0(a0)       # store byte
68     jalr    zero, ra, 0
```

Figure 2: Another solution for `flipBitInByte`.

A simpler version of the solution appears in Figure **??**. A slightly more complicated solution — for programmers that did not realized that a XOR 0 = a and that a XOR 1 = /a — is shown in Figure Figure **??**