1

- The loop runs 65 times.
- It is `x144A` stored in `R3`.

2

- `RET` pops the return address from the system stack, which is typically used for subroutine calls and not for interrupts. Interrupts push the return address to the supervisor stack. So, if `RET` is used instead of `RTI`, the program will return to an incorrect address, leading to undefined behavior.
- When an interrupt occurs, the LC-3 switches from user mode to supervisor mode. The `RTI` instruction is designed to restore the previous privilege mode (user mode) after servicing the interrupt. If `RET` is used instead, the system remains in supervisor mode even after returning from the interrupt service routine. This could lead to serious security issues as subsequent user mode code would be executed in supervisor mode.

3

- The student wants to print `I`.
- No he can't. Because when the program execute the second `JSR` instruction, the address stored in `R7` will be changed, and he didn't store the former address so the `RET` instruction right after the label `A` can not work correctly.

4

- Signal `X` is wehther Access Violation happened.
- It will be `1` if it is user mode but trying to access the privilege aera.

5

- In LC-3, we can check if there is a new character input from the keyboard by checking the highest bit of the Keyboard Status Register (KBSR). If the highest bit is 1, then there is a new character input.
- Once a new character input is detected, we can read this character from the keyboard by reading the lower 8 bits of the Keyboard Data Register (KBDR).
- To display the read character on the screen, you can use the `TRAP x21` instruction. This instruction will output the character stored in the R0 register to the console.
- Here is a simple LC-3 assembly code snippet that demonstrates this process:

```
       .ORIG x3000
       LEA R0, KBDR
       LD R1, KBSR
WAIT LDR R2, R1, #0
       BRz WAIT
       LDR R0, R0, #0
       TRAP x21
       HALT
KBSR .FILL xFE00
KBDR .FILL xFE02
       .END
```

6

- Here's the completed program:

```
    HEX_INPUT
    ST R1, SAVE_R1 ; R1 = Constant 1
    ST R2, SAVE_R2 ; R2 = Constant 2
    ST R3, SAVE_R3 ; R3 = Chars left (counter)
    ST R4, SAVE_R4 ; R4 = **DELETED**
    LD R1, C1
    LD R2, C2
    AND R3, R3, #0
    ADD R3, R3, #4
    AND R0, R0, #0 ; R0 stores our result
GETCHAR
    ; **DELETED**
    ADD R0, R0, R0
    ADD R0, R0, R0
    ADD R0, R0, R0
    ADD R0, R0, R0
WAIT
    LDI R4, KBSR ; Check keyboard status
    BRzp WAIT ; **DELETED**
    LDI R4, KBDR ; Get KBDR
    ADD R4, R4, R1 ; Check if it is a letter
    BRzp LETTER ; Got a capital letter
    ADD R4, R4, R2 ; Not a letter -> digit
    BR CONTINUE
LETTER
    ADD R4, R4, #10 ; **DELETED**
CONTINUE
    ADD R0, R0, R4 ; Add to result
    ADD R3, R3, #-1 ; Decr counter
    BRp GETCHAR ; Wait for another char
    ; Restore regs
    LD R1, SAVE_R1
    LD R2, SAVE_R2
    LD R3, SAVE_R3
    LD R4, SAVE_R4
    RET

    ; Data
    C1 .FILL #-65 ; **DELETED**
    C2 .FILL #-48 ; **DELETED**
    KBSR .FILL xFE00
    KBDR .FILL xFE02
    SAVE_R1 .BLKW 1
    SAVE_R2 .BLKW 1
    SAVE_R3 .BLKW 1
    SAVE_R4 .BLKW 1
```

- The four consecutive `ADD R0, R0, R0` instructions are used to shift the contents of R0 left by 4 bits. This is equivalent to multiplying the value in R0 by 16 (or 2^4). This is done because each hex digit represents 4 bits, so when a new hex digit is read, the previous digits need to be shifted left by 4 bits to make room for the new digit.

- This instruction is necessary because we don't know what value R0 holds before the subroutine is called. Since `R0` is used to store the result, we need to ensure that it starts from 0. If we don't clear R0, the final result could be incorrect because it would include whatever value was previously in `R0`.

7

- The output of the program will be the string `H3ll0_W0r1d` followed by the character `!`, which is the ASCII representation of the decimal number 33.
- The LC-3 uses 16-bit words, so each instruction or piece of data takes up 2 bytes. The program has 5 instructions (`LEA`, `PUTS`, `LD`, `OUT`, `HALT`), 1 label with a string of length 11 (including the null terminator), and 1 label with a single value. So, the total memory occupied by the program is:

$$5 \times 2 + 11 \times 2 + 1 \times 2 = 34\,\text{Bytes}$$

8

- If a program does not check the Keyboard Status Register (KBSR) before reading from the Keyboard Data Register (KBDR), it might read stale or invalid data.
- If the keyboard does not check the Keyboard Status Register (KBSR) before writing to the Keyboard Data Register (KBDR), it might overwrite the existing data in the KBDR before it has been read by the program.
- The problem of a program not checking the Keyboard Status Register (KBSR) before reading from the Keyboard Data Register (KBDR) is more likely to occur. This is because the responsibility of checking the KBSR falls on the programmer who writes the code. If the programmer is not aware of the need to check the KBSR before reading the KBDR, or if they forget to do so, the program could read stale or invalid data.

  On the other hand, the problem of the keyboard not checking the KBSR before writing to the KBDR is less likely to occur. This is because the process of writing to the KBDR is typically managed by the hardware or the operating system, not by the programmer. The hardware or operating system would be designed to handle this correctly, so it's less likely that this problem would occur.

9

The output of this program will be `F !`.

10