Assignment 4

This assignment is focusing on the interrupt capabilities of the system.
In difference to the software part in previous assignments, your first instruction should be a jump instruction to your main program. Also, at the address derived from your chosen interrupt line, place a jump instruction to the interrupt handler. Your main routine should set initialize the stack pointer, the memory locations in data memory for `axialmeter` and `prevaxialstate`, and lastly set the runtime priority to zero.
Note that the stack pointer grows downwards.

Question 4.1

Read the `irq_controller.vhd` file in the `soc/core/` subdirectory to understand how the interrupt controller handles incoming interrupt requests.
Extend your button and switch controller to be able to produce maskable interrupt requests. Maskable in this context mean that the generation of interrupts should be possible to be enabled or disabled by software. Realize that by making the upper 16 bit in the components register a writable mask.
Any change in state of the switches should generate an interrupt. The interrupt signal should of course be present in the components port interface.
Test it by building a test bench for it. Do not forget to add, commit and push your changed and created work.

Question 4.2

Connect the interrupt signal to the interrupt controller of the system.
Upon receiving and acknowledging an interrupt request, the processor will perform a few stack operations, saving status register and link register, and will then jump to an address derived from the vector number supplied by the interrupt controller, shifted by one. Determine that address.

Question 4.3

The function the joystick is to achieve is to change a value depending on which direction it is pressed, and only once per press. The value to be changed is an 8-bit variable to be called `axialmeter`.

a) Since you not only need a variable to store the `axialmeter` value, but also a notion of what the previous state of the joystick was, you need another variable `prevaxialstate`.
The logical function applied on the bitvectors of the axial button's state to determine which state changed is easy, see Equation 1. But this not only indicated a pressing of the button, but also its release, and it is needed to derive only presses, see Equation 2.

$$\text{change} = \text{current state} \oplus \text{previous state} \tag{1}$$

$$\text{presses} = (\text{current state} \oplus \text{previous state}) \wedge \text{current state} \tag{2}$$

Determine how you can implement the equation in assembler.

b) Set up a new assembler program file, `assignment4.prog`. Write an interrupt service routine for the joystick.
It should load both pointers for `axialmeter` and `prevaxialstate` from the instruction memory. Note: The pointers should point to a memory location in the supplied DRAM component, not instruction memory.
The change to be applied to `axialmeter` depends on the joystick movement. Moving the joystick up increases the value by one. Moving it down decreases it by one. Left or right pressing the joystick should shift the value in the respective direction. Shifting takes precedence before in- and decrementing.
Also add the preamble and initialization as described in the beginning of the assignment. For testing, add an endless loop to the end of your main routine.

Add and commit the program file.

c) Create a testbench for the integrated system. Make it simulate joystick inputs and verify that your setup triggers the right interrupts and the value in memory gets changed as expected.
Add and commit the testbench.

Question 4.4

Complete your main routine. After initialization, it should periodically check the value in `axialmeter`, and display it, using the 8 leds as an 8 bit unsigned representation.
Synthesize and demonstrate your solution to complete the assignment. Do not forget to add, commit and push your changed and created work.

Question 4.5 - Challenge

This assignment's *challenge* is to diplay the value of `axialmeter` on the LCD instead of the LEDs. The displayed value should either be encoded as decimal or hexadecimal.
Hint: Check the internet for implementations of the modulo operator without using multiplication of division.