COSC 211 Lab 9 – Interrupts and I/O

0.Introduction

In this lab, you will have the opportunity to explore using interrupts to read and write data from the keyboard. This will build on the materials covered in lab 7. There are a total of 20 marks available for this assignment. Appendix A-7 and A-8 are available on Canvas for review.

The assignment has one coding activity. The code is to be submitted through Canvas by the **end of the day on Thursday, December 7th, 2023**. Submissions will be accepted earlier. Late submissions will not be accepted. Even if you do not complete all functionality, please try them all and submit what you have for partial marks.

1. Programming Assignment

1. (/20) In this lab, you will add more functionality to your exception handler from lab 7 so that it can handle interrupts generated by input from the keyboard. When a keyboard interrupt occurs, your handler should

    - read the input byte from the Receiver Data Register,
    - save it to the register $s0, and
    - echo the typed character to the output console.

    SPIM supports one I/O device: a memory-mapped character-based terminal console, which will read characters from and output to the terminal. There are four memory-mapped device registers that control its operation. To output the character, use the polling IO strategy (Topic 18 – starting at slide #19). Also, please review appendix A.8 for further details. Your handler should not proceed until the character has been transmitted. The handler will need to both print out the character and copy it into $s0. **Do not use a SYSCALL to print the character but use the transmitter control register transmitter data register to output the character.** In order to deal with IO in this way, memory-mapped IO needs to be enabled. This is set by Simulator->Settings from the main menu bar (Figure 1). Select the MIPS tab from the settings window and make sure Enabled Mapped IO is checked (Figure 2). Ensure that your simulator is also setup to handle exceptions and your handler is loaded correctly as per lab 7.

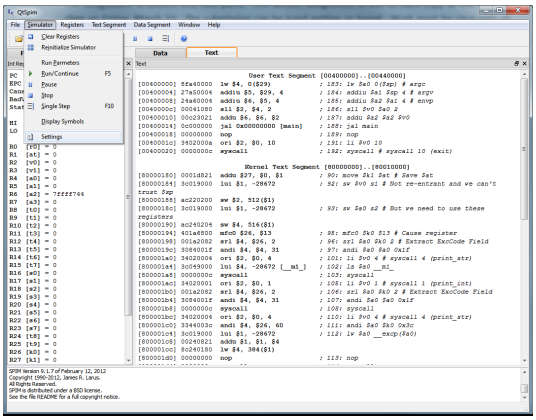

Figure 1

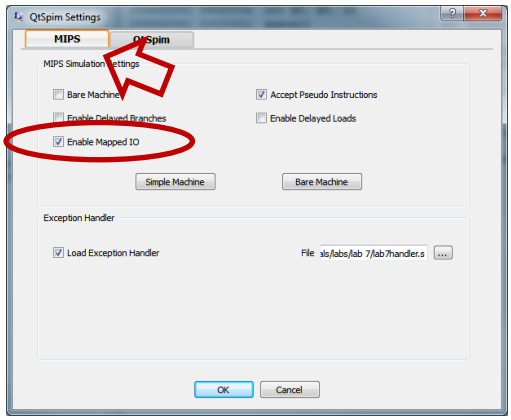

Figure 1

Important:

- **The goal of your code is to modify your exception handler to be able to handle an interrupt (not an exception as this is an asynchronous operation) from the keyboard and print the character out to the console.**
- **In your handler, do not use the SYSCALL to print out the character; the character that you have received via the interrupt will need to be copied into the transmit data register.**

Use the program io.s to test your program. It just loops until the register $s0 contains the capital letter Z. To make the memory-mapped work, we have to

1. set the interrupt-enable bit of the **Receiver Control Register**
2. set the interrupt-enable bit of the **Status Register**

in the startup routine (`__start`) of your exception handler. Update the `__start` section of your handler to include the following code:

```
#configure the status register to enable the keyboard interrupt (hardware interrupt enabled)
mfc0 $K0, $12
ori $k0, $k0, 0x1
mtc0 $k0, $12

#configure receiver control register
lw $k1, 0xFFFF0000              #l get the contents of the Rx control register into register (p. A-39)
ori $k1, $k1, 0x2               # set the second bit (bit position 2) to enable the interrupt
sw $k1, 0xFFFF0000              # copy the results back to the Rx control register

jal main
li $v0, 10
syscall
```

Make sure to add this as per the class notes. Once this is in place, it will cause the processor to generate an interrupt when a character is received asynchronously from your main code; it will cause the user space code to halt and change control to the kernel space (your exception handler).

The second thing that you will need to add to your handler is a section of code to check if the exception handler was generated by an interrupt. To do this, you will need to check the cause register and specifically look at the bits for exception codes. If no exception bits have been set, the you can assume that the handler has been triggered by an interrupt. Here is an example of how this could be accomplished:

```
mfc0 $k0, $13                   # k0 holds Cause
mfc0 $k1, $14                   # k1 holds EPC

srl $k0, $k0, 2
andi $k0, $k0, 0xF              # isolate the the exceptions code bits with a mask

beq $k0, $zero, interrupt       # if no exceptions are set, then branch to interrupt handler
                                # your interrupt handler should then be at the interrupt: label
```

Once your code is able to determine that an interrupt has occurred and branched to your interrupt handler, you will need to process the received character. When a character is received with an interrupt, it will be put into the `Receiver Data` register (address `0xFFFF0004`). You will need retrieve the character, and setup the transmit register to send the same character back. The steps you will need to use are:

1. load into a general register the word from `0xFFFF0004` (`Receiver Data` register). Hint: use `lw` with the register address as the base address.
2. Load the contents of the `Transmitter Control` register into a general register (address of the `Transmitter Control` register is `0xFFFF0008`).
3. Check the Ready bit (bit 0) using a mask; if bit 0 is a zero, then the transmitter is busy, and you will need to loop and reload the contents of the `Transmitter Control` register and keep checking until bit 0 is a one (which indicates that it is free). When this occurs, it indicates that the transmitter is ready to transmit a character.
4. Copy the **byte** that was received from the receiver into the `Transmitter Data` register. You can copy the entire word from the `Receiver Data` register into the `Transmitter Data` register (address `0xFFFF000C`). Recall that only bytes are send. Thus, only the lowest 8-bits of the word will be actually transmitted.
5. Exit the handler.

Test your code to make sure it works.

Include a program comment header that has your name, student number and program name. **Name your file lab9handler.s.**

Code Submission

Submit your code via Canvas before the indicated due date.