Lab 5

Introduction

This lab serves to introduce us to the topics of exceptions and interrupts, and how to handle interrupts. Exceptions are events that are scheduled out of order that disrupt the program (edit the control flow). Interrupts are specific exceptions that occur from an external source.

Assignment 1

Description: Observe the file "io.s" and record its output, making sure to note what goes on upon execution.

Register	Before	After
PC	0	4000bc
EPC	0	4000bc
Cause	0	4
BadVAddr	0	0
Status	3000ff10	3000ff11

The program infinite-loops when running via single-step. The cause register is a starts off at 400. The status register starts off at 0X3000ff10. This means that the last bit is a 0, which means that all interrupts are disabled. While entering the first three words, it still goes into an infinite loop. When entering the next three words, it doesn't go into that infinite loop.

Basically, when entering the words one by one, while entering the first 3 words, it uses polling. While entering the next 3 words, it uses interrupts.

The difference between polling and interrupts are that in polling, the CPU constantly needs to read the keyboard latch memory location to see if a key is pressed. This makes it very inefficient.

Interrupt driven I/O is when the keyboard alerts the CPU of a key-press. Tasks can be concurrently performed this way, making it more efficient.

Assignment 2

Description: We wish to modify assignment 1 such that an interrupt handler will handle the event of a user initiating a keystroke which should generate a random integer. An attempt to try this is given below in the code below (also seen in interrupt_handler.asm)

```
.data

prompt: .asciiz "SPIM IO Test.\n Please type 6 input lines:\n"
nl:     .asciiz "\n"

.text
.globl main

main:
li $v0, 4
la $a0, prompt
syscall

li $s0 3        # Loop counter

li $t0 0xffff0000    # recv ctrl
li $t1 0xffff0004    # recv buf
li $t2 0xffff0008    # trans ctrl
li $t3 0xffff000c    # trans buf

# Second, read and echo 1 lines of input by through interrupts:

li $t4, 0
li $t5, 100000
li $t6, 0        #counter

addi $t4, $t4, 5

l3:    j l3 # Loop waiting for interrupts

# Interrupt handler. Replaces the standard SPIM handler.

.ktext 0x80000180
mfc0 $t4,$13        # Get ExcCode field from Cause reg
srl $t5,$t4,2
and $t5,$t5,0x1f    # ExcCode field
bne $t5,0,exception # Not interrupt, it is exception

# An interrupt:
and $t5,$t4,0x800    # Check for IP3
beq $t5,0,check_trans

# Receiver interrupt:
lw $t5,0($t0)        # Check receiver ready
and $t5,$t5,1
beq $t5,0,no_recv_ready    # Error if receiver is not ready

lw $t6,0($t1)        # Read character
li $t7,1            # Set a flag

beq $t6 0xa decr2    # New line (nl)
bne $t6 0xd next    # Carriage return (cr)

decr2:
add $s0 $s0 -1        # Decrement line counter

next:
mfc0 $t4 $13        # Get Cause register
li $t5,0xfffff7ff
```

```
and $t4,$t4,$t5        # Clear IP3 bit in Cause register
mtc0 $t4,$13

check_trans:
beq $t7 0 ret_handler    # No char to write yet

and $t5 $t4 0x400    # Check for IP2
beq $t5 0 check_loop

# Transmitter interrupt:
lw $t5 0($t2)        # Check transmitter ready
and $t5 $t5 1
beq $t5 0 no_trans_ready

sw $t6 0($t3)        # Write character
li $t7 0

mfc0 $t4 $13        # Get Cause register
and $t4 0xfffffbff    # Clear IP2 bit
mtc0 $t4 $13

check_loop:
bne $s0 0 ret_handler    # If line counter not zero, get another line

# Done echoing, so terminate program.
li $v0 10
syscall            # syscall 10 (exit)

# Return from handler.
ret_handler:

mfc0 $t4 $12        # Enable interrupts and mask in Status register
ori $t4 $t4 0xff01
mtc0 $t4 $12
```

```
eret                # return to interrupted instruction

exception:
li $v0 4            # Non-interrupt exception
la $a0 other_str    # Print message and ignore
syscall
j ret_handler

no_recv_ready:
li $v0 4            # Receiver was not ready after interrupt
la $a0 no_recv_str    # Print message and ignore
syscall
j ret_handler

bad_int:
li $v0 4            # Interrupt was not from recv or trans

la $a0 bad_int_str    # Print message and ignore
syscall
j ret_handler

no_trans_ready:
li $v0 4            # Transmitter was not ready after interrupt
la $a0 no_trans_str    # Print message and ignore
syscall
j ret_handler

.data

other_str: .asciiz "Non-interrupt exception\n"
no_recv_str: .asciiz "Receiver not ready\n"
no_trans_str: .asciiz "Transmitter not ready\n"
bad_int_str: .asciiz "Unknown interrupt\n"
```

Conclusion

In this lab, I learned the strict difference between interrupts and exceptions, although they are very much related, as well as how to create my own interrupt handler, even if that portion is incomplete.