Lab 4

CMPUT 229

University of Alberta

Outline

1 **Lab 4 Assignment**
- Memory-Mapped I/O
- Polling
- Interrupts
- Interrupt Handling
- Questions

The Assignment: A Countdown Timer

- You will implement a simple countdown timer in MIPS.
- This will consist of a main program, as well as an interrupt handler.
- The timer will be driven by timer interrupts and displayed using memory-mapped output.
    - The program will prompt the user for a number of seconds.
    - The remaining time will be displayed as *mm:ss* starting at the corresponding time entered by user.
    - You do not need to handle anything beyond 99:59, i.e. 5999 seconds.
- The timer will be controlled by the keyboard, using memory-mapped input.
    - After entering a number, [ENTER] starts the timer.
    - *q* exits.

Memory-Mapped I/O

- Control and data registers for the keyboard and display live at memory addresses outside of the real memory range.
    - Keyboard control at `0xffff 0000`.
    - Keyboard data at `0xffff 0004`.
    - Display control at `0xffff 0008`.
    - Display data at `0xffff 000C`.
- Run SPIM or XSPIM with the `-mapped_io` flag to enable them.
- Use `lw` and `sw` to access and modify them, just like you would with normal memory locations

Output Using Polling

- To write data to the display, we have to wait for it to become ready, then print a character. We poll the control register, then write data to the register.
- To write a string, we have to do this for every character.

Output Using Polling

```
.data
str:
        .asciiz   ''Hi."

.text
main:   la        $t0 str
loop:   lb        $t1 0($t0)
        beqz      $t1 done
poll:   lw        $t2 0xffff0008
        andi      $t2 $t2 0x01
        beqz      $t2 poll
        sw        $t1 0xffff000C
        addi      $t0 $t0 1
        j         loop
done:
        jr        $ra
```

Keyboard Input with Interrupts

- You must enable the keyboard using its control register.
- When there is a character read from the keyboard, you will get an *interrupt* (we'll discuss interrupts later).
- Then you can read the character from its data register.

Keyboard Input with Interrupts

```
main:          lw    $t0 0xffff0000    # Enable Keyboard interrupt
               ori   $t0 $t0 0x02
               sw    $t0 0xffff0000


                                       # In the exception handler
               lw    $a0 0xffff0000    # Check the keyboard status
               andi  $a0 $a0 0x01
               bnez  $a0 nkeyboard
               ...   ...
nkeyboard:     lw    $a0 0xffff0004
```

Interrupts

- *Interrupts* (also called exceptions) are events that invoke the interrupt handler code.
- Interrupts can be generated in response to external events (e.g. keypresses on the keyboard), by errors in code (e.g. arithmetic overfow or misaligned loads), or by software itself.
- The interrupt handler in MIPS resides in the kernel text section of memory, and must start at address 0x8000 0180.
- MIPS uses co-processor 0 to generate and handle interrupts. Co-processor 0 has a number of registers that you can access using the mfc0 and mtc0 instructions.

A Basic Interrupt Handler (1)

- We define two locations in the kernel data section to store $v0 and $a0. In kernel mode, you have free use of $k0 and $k1, but you must leave other registers as you found them.
- $13 on CP0 is the cause register. $12 on CP0 is the status register. We get them with `mfc0`.
- The exception code is in bits 6-2 of the cause register.
- We clear the cause register before returning.
- We reload $a0 and $v0 so that the user code doesn't know the exception has happened.
- We make sure interrupts are enabled by setting bit 0 of the status register.
- The `eret` instruction returns control to the user code at the point where the exception was thrown.

A Basic Interrupt Handler (2)

```
.kdata:              s1:      .word 0
                     s2:      .word 0
.ktext 0x80000180    sw       $v0 s1        # Reload a0 and v0
                     sw       $a0 s2
                     mfc0     $k0 $13       # Get the cause re
                     srl      $a0 $k0 0x02
                     andi     $a0 $a0 0x1f  # Extract the exce
                     li       $v0 4         # Print it
                     syscall
                     mtc0     $0 $13        # Clear the cause
                     mfc0     $k0 $12       # Enable interrupt
                     ori      $k0 $k0 0x01
                     mtc0     $k0 12
                     lw       $v0 s1        # Reload a0 and v0
                     lw       $a0 s2
                     eret                   # Return control t
```

Interrupt Handler Notes

- SPIM provides a default exception handler, which is in */usr/local/lib/spim/exceptions.s* or in */usr/lib/spim/exceptions.s*. It might be useful for you to look at, and you can use it as a starting point for your assignment if you wish.
- You will need to run SPIM without this default exception handler to provide your own. Use the `-notrap` flag.
- This default exception handler contains, among other things, the code that runs your main method when SPIM starts. Without it, SPIM runs code starting at the global symbol `_start`.
- In main you usually returned using jr `$ra`. You can't do this in start because `$ra` wasn't set by a jal instruction.

The Timer

- SPIM provides a timer mechanism that triggers an interrupt periodically.
- Specifically, this involves two registers on co-processor 0: $9 and $11.
- $9 increments by 1 automatically every 10 milliseconds.
- An interrupt is raised when $9 == $11.
- This means if you want an interrupt in 1000 milliseconds, you should set $11 to 100 and $9 to 0

Things to Remember

- You may use the default exception handler (`/usr/local/lib/spim/exceptions.s`) to start your assignment. It is **not** plagiarism to hand in code from file.
- The marksheet has been posted. Look at the items that will be evaluated carefully.
- You can write your solution any way you like, but following the method outlined in class is probably a good way to go.
- The usual: your code should be formatted accordingly, no late submissions, and make sure it runs on the lab machines.

Lab 4 Questions?