Lab 4

CMPUT 229

University of Alberta

Outline

The Assignment: A Countdown Timer

- You will implement 6 sychronized countdown timers in RISC-V.
- This will consist of a main program, as well as an interrupt handler.
- The timers will be driven by timer interrupts and displayed using memory-mapped output.
  - Any timer can be set with a specific amount of time, at any time from an input in the form of seconds@timer.
  - The remaining time will be displayed as *mm:ss* starting at the corresponding time entered by user, or 00:00 which is the default state of the timer.
  - You do not need to handle anything beyond 59:59, i.e. 3599 seconds.
- The timer will be controlled by the keyboard, using memory-mapped input.
  - After entering a number and desired timer, [ENTER] starts the timer.
  - *q* pressed at any time terminates the program.

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 RARS simulator with the `Keyboard and Display MMIO Simulator` tool and the `Timer Tool`, connect them 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:
        .asciz  "Hi."

.text
main:   la      t0, str
loop:   lb      t1, 0(t0)
        beqz    t1, done
poll:   lw      t2, 0xFFFF0008
        beqz    t2, poll
        li      t3, 0xFFFF000C
        sw      t1, 0(t3)
        addi    t0, t0, 1
        j       loop
done:
        jr      ra, 0
```

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

```
                # Enable keyboard interrupts
main:           lw      t0, 0xFFFF0000
                ori     t0  t0, 0x02
                sw      t0, 0xFFFF0000

                # In the exception handler check the keyboard
                # status
                csrrc   t0, 66, t1
                li      t1, 0x7FFFFFFF
                and     t0, t0, t1
                bnez    t0, 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 addresss of the interrupt handler in RISC-V resides in the utvec register, store the address of your handler to the utvec register at the start of your progam.
- RISC-V uses Control and Status Registers to generate and handle interrupts. There are a number of these registers that you can access using the csrr instructions.

A Basic Interrupt Handler (1)

- In the handler, you have to store the registers in the stack before using them (even the temporary registers).
- `ucause (66)` in CSR is the cause register. `ustatus (0)` in CSR is the status register. We get them with `cssr` instructions.
- The exception code is in the ucause CSR
- We reload registers used from stack so that the user code doesn't know the exception has happened.
- The `uret` instruction returns control to the user code at the point where the exception was thrown.

A Basic Interrupt Handler (2)

```
handler:  addi    sp, sp, -8     # Store registers
          sw      a0, 0(sp)
          sw      a7, 4(sp)

          csrrc   a0, 66, zero   # Get the exception code
          li      a7, 34
          ecall                  # Print it

          lw      a0, 0(sp)
          lw      a7, 4(sp)
          addi    sp, sp, 8      # Reload registers
          uret                   # Return control to user
```

The Timer

- RARS provides a timer mechanism that triggers an interrupt at a spceified time.
- Specifically, this involves two addresses used by the timer hart: Time at `0xFFFF0018` and TimeCmp at `0xFFFF0020`.
- value in `0xFFFF0018` increments by 1 automatically every 1 millisecond.
- An interrupt is raised when the values at the two addresses are equal (or when Time is greater than TimeCmp).
- This means if you want an interrupt in 1000 milliseconds (1 sec), you should set the value at address `0xFFFF0020` to the current value in `0xFFFF0018` plus 1000.

Things to Remember

- 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?