Serial Programming

I/O Commands

- I/O devices are managed by I/O controller hardware
  - Transfers data to/from device
  - Synchronizes operations with software
- Command registers
  - Cause device to do something
- Status registers
  - Indicate what the device is doing and occurrence of errors
- Data registers
  - Write: transfer data to a device
  - Read: transfer data from a device
I/O Register Mapping

- Memory mapped I/O
  - Registers are addressed in same space as memory
  - Address decoder distinguishes between them
  - OS uses address translation mechanism to make them only accessible to kernel

- I/O instructions
  - Separate instructions to access I/O registers
  - Can only be executed in kernel mode
  - Example: x86

An Example of A Memory-Mapped Device

0x0100 - 0x0103: store zero turns display off
  nonzero turns display on
  fetch returns zero if display off
  returns one if display on

0x0104 - 0x0107: video parameter register used for info such as RGB type, RGB data
  fetch returns brightness

Exercise: write an assembly code segment to turn off display
Our Programming Environment

/mipcon:
automatically connects a frontend machine to a backend router,
sends the commands to boot XINU, and runs your program.

Backend Routers

- File /etc/xinu-consoled.conf has the names of the backend routers. Router names include:
  - voc
  - xoanon
  - cass
  - toos
  - dask

- After you issue /mipcon, some info is printed. Which router is running my program? Look for
  - *** command status = 0
  - CFE> boot -elf 192.168.6.10:xoanon.boot

- For detailed info, refer to
  http://xinu.mscs.mu.edu/HOWTO:Build_Backend_Pool
Last Homework

• Read from and write to serial ports
  – Serial ports used to serve as console

• UART is between the MIPS processor and serial ports. We will program to interact with UART

• UART has 12 registers, which are memory mapped in processor’s view
  – The base address on our system is 0xB8000400

  | Transmitter Holding Buffer | base address + 0 |
  | Receiver Buffer             | base address + 0 |
  | Line Status Register (LSR)  | base address + 5 |

Line Status Register

The bits in LSR indicate status. You can only read from LSR, do not try to write to it

<table>
<thead>
<tr>
<th>Bit</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Data Ready, set simply means that there is data available for your software to extract from the UART</td>
</tr>
<tr>
<td>1</td>
<td>Overrun Error</td>
</tr>
<tr>
<td>2</td>
<td>Parity Error</td>
</tr>
<tr>
<td>3</td>
<td>Framing Error</td>
</tr>
<tr>
<td>4</td>
<td>Break Interrupt</td>
</tr>
<tr>
<td>5</td>
<td>Empty Transmitter Holding Register. Set means the UART is capable of receiving more characters</td>
</tr>
<tr>
<td>6</td>
<td>Empty Data Holding Registers</td>
</tr>
<tr>
<td>7</td>
<td>Error in Received FIFO</td>
</tr>
</tbody>
</table>
You interact with your program through console 1, the interactions include keyboard input, print, etc.

If your program writes data to the serial port, the info will be shown on console 2.

Example: assume you want to write a character ‘A’ to the serial port, what should the algorithm and assembly code segment look like?

Here is a demo

**I/O Example**

- **Input: Read from 2nd console into v0**

  ```assembly
  li t0, 0xB8000400 #fff0000
  Waitloop:
  lb t1, 5(t0) #control
  andi t1, t1, 0x1
  beq t1, zero, Waitloop
  lb v0, 0(t0) #data
  ```

- **Output: Write to 2nd console from a0**

  ```assembly
  li t0, 0xB8000400 #fff0000
  Waitloop:
  lb t1, 5(t0) #control
  andi t1, t1, 0b00100000
  beq t1, zero, Waitloop
  sb a0, 0(t0) #data
  ```

- Processor waiting for I/O called “Polling”
- “Ready” bit from processor’s point of view!
References