Homework

• Reading
  – Review previous material on “interrupts”

• Machine Projects
  – MP4 Due today
  – Starting on MP5 (Due at start of Class 28)

• Labs
  – Continue in labs with your assigned section
Discussion of MP4

• What did you learn?
• Did anyone do the optional software UART?
• Let’s look at the code for it as an exercise
Introduction to MP5

• Adding new code to provided tutor “cmds.c”
• Writing a COM1 port driver for Tutor to use
  – Started and stopped by the application (Tutor)
• Tutor cycles driver through this sequence:
  – Receives and buffers user entered data
    (with full duplex echo back to COM1 port)
  – Returns to callback function with receive data buffer
  – Transmits buffer of application data (prompt)
  – Returns to callback function when done
SAPC as Host to a User on COM1

User on COM1
Sees prompts and enters data as if on a host connection

SYSADMIN
Controls SAPC with Tutor and verifies data in from the user on COM1 port
What Code is Needed?

- **In** `cmds.c`:  
  - The `spi` command function has been written for you  
  - Write two call back functions  
    - one for processing last interrupt in transmission and re-starting receiver interrupts  
    - one for processing last interrupt in receiving and re-starting transmitter interrupts

- **In** `comintspack`:
  - Write init and shutdown for COM1 interrupts  
  - Write an interrupt handler for IRQ4 (must handle either a transmit or a receive interrupt each call)
What’s in cmds.c

• New PC-tutor command

  `spi <on|off>`

• Descriptions

  `spi on calls init_comints to enable COM1 in transmit mode with transmit call back function (to print prompt first)`

  `spi off calls shutdown_comints to disable both transmit and receive interrupts`
What’s in cmds.c

- **Receive callback function** *(process_input)*
  - Process input completion (print buffer on COM2)
  - Disable input receiving via shutdown_comints()
  - Enable output transmission via init_comints()

- **Transmit callback function** *(process_output)*
  - Disable output transmission via shutdown_comints()
  - Enable input receiving via init_comints()

- These cause alternate COM1 transmit and receive
What’s in comintspack.h?

• API symbolic constants
  
  /* mode values */
  #define TRANSMIT 0
  #define RECEIVE 1

• API function prototypes
  
  void init_comints (int mode,
    void (*callback)(char *),
    char *buffer,
    int size);
  
  void shutdown_comints (void);

• You do NOT modify this file. Use it as-is!
What’s in comintspack.c?

• Initialize COM1 port (*init_comints*)
  – Save callback function, buffer, and size in static memory
  – Clear out any characters already received
  – Set the interrupt gate
  – Enable the PIC for the IRQ4
  – For RX mode, enable RX interrupts in the UART’s IER
  – For TX mode, enable TX interrupts in the UART’s IER

• This function is called with interrupts disabled
What’s in comintspack.c?

• Shut down COM1 port (shutdown_comints)
  – Disable the PIC for the COM IRQ
  – Disable both interrupts in the UART’s IER
• This function is called with interrupts disabled
What’s In comintspack.c?

• **Interrupt Handler** *(irq4inthandc)*
  – Acknowledge the PIC interrupt
  – For Receive
    • Input the character from COM1
    • Echo the character to COM1
    • Add to accumulated data in the application buffer
    • On end of line, call callback function passing buffer
  – For Transmit
    • Get the next outgoing character from application buffer
    • If not end of string (‘\0’), output the character
    • Otherwise output CR and call callback function
Comintspack Ladder Diagram

Sysadmin
COM2

spi on

User
COM1

Confirm

TUTOR

API

COMINTSPACK

UART

Write

static
storage

* * *

Prompt Character to user

Last Prompt Character to user

Transmit callback function

shutdown_comints ()

init_comints (tx mode)

irq4inthehandc

Int

Read

Int

Read

Write

init_comints (rx mode)
UART Interrupts

• The UART is a real interrupt driven I/O device
• At system reset, all interrupts are disabled
• The UART has four conditions for interrupting
  • We’ll use two alternately - the receiver “data ready” and transmitter “THR empty” interrupts
• We program the UART to enable them via the COM1 Interrupt Enable Register (IER = 0x3f9)
UART Interrupts

- The UART interrupts each time it receives a char or the THR goes empty (depending on the interrupt enabled)
- COM1 is connected to pin IR4 on the PIC, its IRQ is 4.
- The nn code generated by the PIC for COM1 is 0x24, so its interrupt gate descriptor is IDT[0x24]
- ISR must send an EOI command to the PIC
- The ISR must read the received char or write the THR to cause the UART to remove its interrupt
- The UART hardware detects the inb or outb for the character and completes its interrupt-in-progress
UART Interrupts

• Two Parts of the Interrupt Handler
  • irq4inthand – the outer assembly language interrupt handler
    – Save registers
    – Call C function irq4inthandc
    – Restore registers
    – iret
  • irq4inthandc - the C interrupt handler
    – Does the work described earlier
Demonstration of Both Windows

<table>
<thead>
<tr>
<th>COM1</th>
<th>COM2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prompt:</td>
<td>Prompt:</td>
</tr>
<tr>
<td>see me type data</td>
<td>comints for COM1 on</td>
</tr>
<tr>
<td>Prompt:</td>
<td>prompt:</td>
</tr>
<tr>
<td>timeon 5</td>
<td>I can still enter a PC-tutor cmd</td>
</tr>
<tr>
<td>Timer is operating independently</td>
<td>of the COM1 port with interrupts</td>
</tr>
<tr>
<td>more data1</td>
<td>more data1^M^M</td>
</tr>
<tr>
<td>Prompt:</td>
<td>(2)</td>
</tr>
<tr>
<td>Prompt:</td>
<td>(3)</td>
</tr>
<tr>
<td>more data2</td>
<td>more data2^M^M</td>
</tr>
<tr>
<td>Prompt:</td>
<td>timer off</td>
</tr>
<tr>
<td>Another PC-tutor command</td>
<td></td>
</tr>
<tr>
<td>Exception 3 at EIP=00100110: Breakpoint</td>
<td></td>
</tr>
<tr>
<td>~q</td>
<td>Quit handler:</td>
</tr>
<tr>
<td>~q</td>
<td>killing process 12521 Leaving board #7</td>
</tr>
<tr>
<td>Quit handler:</td>
<td>killing process 12932 Leaving board #-1</td>
</tr>
</tbody>
</table>