```
   enable();

/* ... Process the interrupt, e.g. get the character in
   the COM1: port... */

/* Clear the if flag in the CPU flags register. */

   disable();

/* Re-enable COM (asynch) interrupts by ANDing a zero
   into bit 4 of the 8259 port 0x21 value. */

   outportb(0x21, inportb(0x21) & 0xEF);
   }
```

For your information only, in DOS, you can install NewCOMHandler (insert its address at 0xC shown on slide 7 in chapter2a.ps) using the following statements in a program:

```
main()
   {
   void interrupt (*oldcomhand)(void);

/* Save the address of the old handlers so that we can
   restore it later. */

   oldcomhand = getvect(0xC);

/* Set the address of the routine NewCOMHandler into
interrupt vector 0xC. */

   setvect(0xC, NewCOMHandler);

/* Busy wait loop */

   while (1)
      { /* Do something such as check variables that
           NewCOMHandler may change*/ };

/* Restore the DOS handler. */

   setvect(0xC, oldcomhand);
}
```

Homework problem #6 asks you why it is necessary to provide a hardware mechanim for disabling interrupts on the 8259A and the CPU separately. The following code fragment will help shed some light on this for you.

NewCOMHandler is the skeleton of an interrupt service routine (ISR) that is called whenever an event occurs on the COM1: port, such as the arrival of a character from a serial line or modem. The COM1: interrupt line to the 8259A is connected to IRQ4 (see slide 8 of chapter2a.ps) which means it has lower priority than the Timer, the Keyboard and COM2:. However, if the COM1: interrupt is delivered to the 8259A just before, for example, a Timer interrupt, the sequence of events carried out by the hardware (described on slide 9 of chapter2a.ps) will not permit these other higher priority interrupts from getting handled. This is true because the **if** in the CPU is cleared (0) during the execution of NewCOMHandler. It is worth noteing that the 8259A will send interrupts of higher priority to the CPU but the CPU will not acknowledge them.

The code in NewCOMHandler() can change this situation simply enough by re-enabling CPU interrupts (by setting the **if** flag to 1). This gives the desired result since the 8259A will NOT raise the INTR line to the CPU for additional COM1: interrupts, or interrupts from lower priority devices, until a special value, End Of Interrupt, is written to an 8259A register at port 0x20. Therefore, by setting **if** flag to 1 as the first instruction in NewCOMHandler(), we allow interrupts of higher priority to interrupt the processing of a COM1: interrupt but not interrupts of equal or lower priority.

I mentioned in class that the 8259A has a 8-bit mask register (at port address 0x21) that controls whether or not the 8259A *recognizes* interrupts individually from each of the devices connected to it. Why do we need this individual control mask since it appears we can get away with using the **if** flag to do everything we need ? Right ? No. One additional capability of having the mask register on the 8259A in addition to the **if** flag control on the CPU is that we can ignore interrupts from individual devices. However, another possibility (the answer that I am looking for) is given in the code fragment below. What does this code allow ?

```
void interrupt NewCOMHandler(void)
   {

/* Disable other COM (asynchronous) interrupts in
   8259A.*/

   outportb(0x21, inportb(0x21) | 0x10);

/* Send EOI(End of Interrupt) to 8259A port 20.*/

   outportb(0x20, 0x20);

/* Set the if flag in the CPU flags register.*/
```