```
CS5600 Week11.b

1. Two examples of I/O instructions

  (a) Reading keyboard input

    The code below is an excerpt from WeensyOS.
    (details in PS/2 controller: https://wiki.osdev.org/%228042%22_PS/2_Controller)
    This reads a character typed at the keyboard (which shows up on the
    "keyboard data port" (kEYBOARD_DATAREG)).

    /* Excerpt from WeensyOS x86-64.h */
    // Keyboard programmed I/O
    #define KEYBOARD_STATUSREG 0x64
    #define KEYBOARD_STATUS_READY 0x01
    #define KEYBOARD_DATAREG 0x60

    int keyboard_readc(void) {
      static uint8_t modifiers;
      static uint8_t last_escape;

      if ((inb(KEYBOARD_STATUSREG) & KEYBOARD_STATUS_READY) == 0) {
        return -1;
      }

      uint8_t data = inb(KEYBOARD_DATAREG);
      uint8_t escape = last_escape;
      last_escape = 0;

      if (data == 0xE0) { // mode shift
        last_escape = 0x80;
        return 0;
      } else if (data & 0x80) { // key release: matters only
                                //  for modifier keys
        int ch = keymap[(data & 0x7F) | escape];
        if (ch >= KEY_SHIFT && ch < KEY_CAPSLOCK) {
          modifiers &= ~(1 << (ch - KEY_SHIFT));
        }
        return 0;
      }

      int ch = (unsigned char) keymap[data | escape];

      if (ch >= 'a' && ch <= 'z') {
        if (modifiers & MOD_CONTROL) {
          ch -= 0x60;
        } else if (!(modifiers & MOD_SHIFT) != \
                   !(modifiers & MOD_CAPSLOCK)) {
          ch -= 0x20;
        }
      } else if (ch >= KEY_CAPSLOCK) {
        modifiers ^= 1 << (ch - KEY_SHIFT);
        ch = 0;
      } else if (ch >= KEY_SHIFT) {
        modifiers |= 1 << (ch - KEY_SHIFT);
        ch = 0;
      } else if (ch >= CKEY(0) && ch <= CKEY(21)) {
        ch = complex_keymap[ch - CKEY(0)].map[modifiers & 3];
      } else if (ch < 0x80 && (modifiers & MOD_CONTROL)) {
        ch = 0;
      }

      return ch;
    }

```



```

(b) Setting the cursor position

  The code below is also excerpted from WeensyOS. It uses I/O
  instructions to set a blinking cursor. To set the cursor to
  the upper left of the screen, run:  console_show_cursor(0)

  // console_show_cursor(cpos)
  //   Move the console cursor to position 'cpos',
  //   which should be between 0 and 80 * 25.

  void console_show_cursor(int cpos) {
    if (cpos < 0 || cpos > CONSOLE_ROWS * CONSOLE_COLUMNS)
      cpos = 0;

    outb(0x3D4, 14); // Command 14 = upper byte of position
    outb(0x3D5, cpos / 256); // upper byte (256 = 2^8)
    outb(0x3D4, 15); // Command 15 = lower byte of position
    outb(0x3D5, cpos % 256); // lower byte

  }

  // if interested, see details: https://wiki.osdev.org/Text_Mode_Cursor
```

```
89
90  2. Memory-mapped I/O
91
92    (a) Here is a 32-bit PC's physical memory map:
93
94        +------------------+ <- 0xFFFFFFFF (4GB)
95        |       32-bit     |
96        |   memory mapped  |
97        |      devices     |
98        |                  |
99        /\/\/\/\/\/\/\/\/\/\
100
101       /\/\/\/\/\/\/\/\/\/\
102       |                  |
103       |      Unused      |
104       |                  |
105       +------------------+ <- depends on amount of RAM
106       |                  |
107       |                  |
108       |  Extended Memory |
109       |                  |
110       |                  |
111       +------------------+ <- 0x00100000 (1MB)
112       |     BIOS ROM     |
113       +------------------+ <- 0x000F0000 (960KB)
114       |  16-bit devices, |
115       |  expansion ROMs  |
116       +------------------+ <- 0x000C0000 (768KB)
117       |   VGA Display    |
118       +------------------+ <- 0x000A0000 (640KB)
119       |                  |
120       |   Low Memory     |
121       |                  |
122       +------------------+ <- 0x00000000
123
124     [Credit to Frans Kaashoek, Robert Morris, and
125     Nickolai Zeldovich for this picture]
126
```



```
127
128   (b) Loads and stores to the device memory "go to hardware".
129
130     Here is an excerpt of the console printing code from WeensyOS.
131
132
133     /* Compare the address below to the map in panel 2(a). */
134     PROVIDE(console = 0xB8000);
135
136     This is an excerpt about printing; notice how it uses the address
137     "console":
138
139     /*
140     * prints a character to the console at the specified
141     * cursor position in the specified color.
142     * Question: what is going on in the check
143     * if (c == '\n')
144     * ?
145     * Hint: '\n' is "newline" (the user pressed enter).
146     */
147     static void console_putc(printer* p, unsigned char c, int color) {
148       console_printer* cp = (console_printer*) p;
149       if (cp->cursor >= console + CONSOLE_ROWS * CONSOLE_COLUMNS) {
150         cp->cursor = console;
151       }
152       if (c == '\n') {
153         int pos = (cp->cursor - console) % 80;
154         for (; pos != 80; pos++) {
155           *cp->cursor++ = ' ' | color;
156         }
157       } else {
158         *cp->cursor++ = c | color;
159       }
160     }
```