1. Example use of I/O instructions: boot loader

Below is a WeensyOS boot loader (for WeensyOS Schedos, which is part for lab 4).

It may be helpful to understand the overall picture (given in the comments; see also our class notes l03, Section 2(b).)

The main point for today's class is that this code demonstrates I/O, specifically with the disk: the bootloader reads in the kernel from the disk.

See the functions waitdisk() and readsect(). Compare to Figures 36.4 and 36.5 in OSTEP.

```c
// Read 'filesz' bytes at 'offset' from kernel into virtual address 'va',
then clear the memory from 'va+filesz' up to 'va+memsz' (set it to 0).
void
readseg(uint32_t va, uint32_t filesz, uint32_t memsz, uint32_t sect)
{
    // Read 'filesz' bytes at 'offset' from kernel into virtual address 'va',
    // then clear the memory from 'va+filesz' up to 'va+memsz' (set it to 0).
    void
readsect(void *dst, uint32_t sect);
    readseg(uint32_t va, uint32_t filesz, uint32_t memsz, uint32_t sect);
    void
bootstrap(void)
{
    struct Proghdr *ph, *eph;
    // load each program segment (ignores ph flags)
    ph = (struct Proghdr*) ((uint8_t *) ELFHDR + ELFHDR->e_phoff);
    eph = ph + ELFHDR->e_phnum;
    for (; ph < eph; ph++)
    {
        // is this a valid ELF?
        if (ELFHDR->e_magic != ELF_MAGIC)
            return;
        // is this a valid ELF?
        if (ELFHDR->e_magic != ELF_MAGIC)
            return;
        // jump to the kernel, clearing eax
        __asm __volatile("movl %0, %%esp; ret" : : "r" (&ELFHDR->e_entry), "a" (0));
    }
    // jump to the kernel, clearing eax
    __asm __volatile("movl %0, %%esp; ret" : : "r" (&ELFHDR->e_entry), "a" (0));
}
```
2. Two more examples of I/O instructions

(a) Reading keyboard input

The code below is an excerpt from WeensyOS’s x86.c.

This reads a character typed at the keyboard (which shows up on the "keyboard data port" (KBDATAP)), and converts it to a digit. This code is not called in lab4; it was called in lab1. (Use grep to convince yourself of this!)

/* Excerpt from WeensyOS x86.c. Comments from kbd.h in x64 */
#define KBSTATP 0x64  // keyboard controller status port (I)
#define KBS_DIB 0x01  // keyboard data in buffer
#define KBDATAP 0x60  // keyboard data port (I)

int console_read_digit(void)
{
    uint8_t data;
    if ((inb(KBSTATP) & KBS_DIB) == 0)
        return −1;
    data = inb(KBDATAP);
    if (data >= 0x02 && data <= 0x0A)
        return data − 0x02 + 1;
    if (data == 0x0B)
        return 0;
    if (data >= 0x47 && data <= 0x4D)
        return data − 0x47 + 7;
    if (data == 0x53)
        return 0;
    return −1;
}

(b) Setting the cursor position

The code below is also excerpted from WeensyOS’s x86.c.

It clears the console (see next items on the handout) and then uses I/O instructions to set a blinking cursor in the upper left of the screen.

/* console_clear */
/* Clear the console by writing spaces to it, and move the cursor to the upper left (row 0, column 0). */

void console_clear(void)
{
    int i;
    /* what’s this?? (see next items on handout) */
    cursorpos = (uint16_t *) 0xB8000;
    for (i = 0; i < 80 * 25; i++)
        cursorpos[i] = ' ' | color;
    /* */
    /* prints a character to the console at the specified cursor position in the specified color. */
    /* Question: what is going on in the check */
    if (c == \n')
        ?
    /* Hint: \n’ is ”C” for ”newline” (the user pressed enter). */
    static uint16_t *
    console_putc(uint16_t *cursor, unsigned char c, int color)
    {
        if (cursor >= CONSOLE_END)
            cursor = CONSOLE_BEGIN;
        if (c == \n')
            int pos = (cursor - CONSOLE_BEGIN) % 80;
            /* what does this do */
            for (; pos != 80; pos++)
                *cursor++ = ' ' | color;
        } else
            *cursor++ = c | color;
        return cursor;
    }