INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

COMPUTER SYSTEMS LABORATORY

Fall 2011

Overture

External software

- VMware Player *http://www.vmware.com/download/player/*
- LinuxVM-0910-1 *http://www.deetc.isel.ipl.pt/programacao/psc/Ferramentas/LinuxVM-0910-1.zip*
  Note: this is a temporary working environment, while a new one is being prepared

Setting up the basic environment

The following notes indicate how to setup the recommended environment. You may use a different Linux system if you want to, but the more you deviate from the recommended environment, the less we'll be able to help you with configuration issues. In any case, recent Ubuntu variants should cause less trouble than other options.

If you don't have *LinuxVM-0910-1* already installed:

- Install VMware Player
- Unzip *LinuxVM-0910-1.zip* into some convenient location (e.g.: `C:\LinuxVM\`)
- Run *LinuxVM.vmx* and select the option "*I copied it*" when the question pops up
- VMware Player will probably complain about VMware Tools not being up to date, but we'll try to live with that.

Your first boot disk

In this exercise, a tiny program will be placed in the very first sector of an otherwise empty 1.41MiB floppy disk. This sector will be loaded at physical address 0x07C00 by the BIOS. The program starts by setting a few processor registers with known values, then prints "Starting LSC…" on the screen, and terminates.

The following is the skeleton of the program's source file, `lscboot.s` :

```
    .equ START_ADDR, 0x7C00  # .equ defines a textual substitution

    .text                    # code section starts here
    .code16                  # this is real mode (16 bit) code

    cli                      # no interrupts while initializing
    # ... init ...           # initialization code...
    sti                      # interrupts enabled after initializing

    # ... prog ...           # main program body...

    # ... term ...           # end of execution...
```

```
      .section .rodata         # program constants (no real protection)
msgb: .asciz "Starting LSC..."
msge:

      .data                    # program variables (probably not needed)
lsc:  .long 2010

      .end
```

To use `cs` with a value of 0x0000 and have the boot code start at offset 0x7C00 of the code segment, the program should enforce these addresses in its early steps. A simple way to do it is by issuing a *long jump* instruction (a *far jump*, in Intel's parlance), which simultaneously sets `cs` and `ip` with the specified values:

```
          ...
          ljmp $0, $norm_cs
    norm_cs:
          ...
```

Next, all other segment registers are set to zero. As segment registers can only appear in register-register instructions, they have to be set to zero with code like:

```
          ...
          xorw %ax, %ax
          movw %ax, %ds
          ...
```

It is suggested that the stack pointer be set to 0x7C00, as it can then grow downwards from there.

To write the message on the screen, you should use the BIOS service INT 0x10 / AH=0x0E, which prints a single character in teletype mode. In this mode, the BIOS service will take care of line breaks and scrolling for you. The code to write a single char is:

```
          ...
          movb  $'@', %al    # The character: '@'
          movb  $7  , %bl    # Light gray on black
          xorb  %bh , %bh    # Using page 0

          movb $0x0E, %ah    # Identifying the service
          int  $0x10         # Invoking the BIOS service

          ...
```

After presenting the message, there's nothing else to do, which means that the processor can be stopped. To stop it until the next interrupt, use the halt instruction.

```
          ...
    stop:
          hlt
          jmp stop
          ...
```

Building the boot floppy

Generate object code from the assembly source in `lscboot.s`:

```
as -o lscboot.o lscboot.s
```

Create the file `bootrec.ld` with the following content:

```
OUTPUT_FORMAT("binary")
SECTIONS
{
    .bootrec 0x7C00 : {
       *(.text)
       *(.rodata)
       *(.data)
       . = 510;
       SHORT(0xAA55)
    }
}
```

The boot sector can now be generated with:

```
ld -T bootrec.ld -o lscboot.bin lscboot.o
```

Then use dd to produce the empty floppy space and to copy the boot sector:

dd if=/dev/zero of=lsc.fd bs=512 count=2880

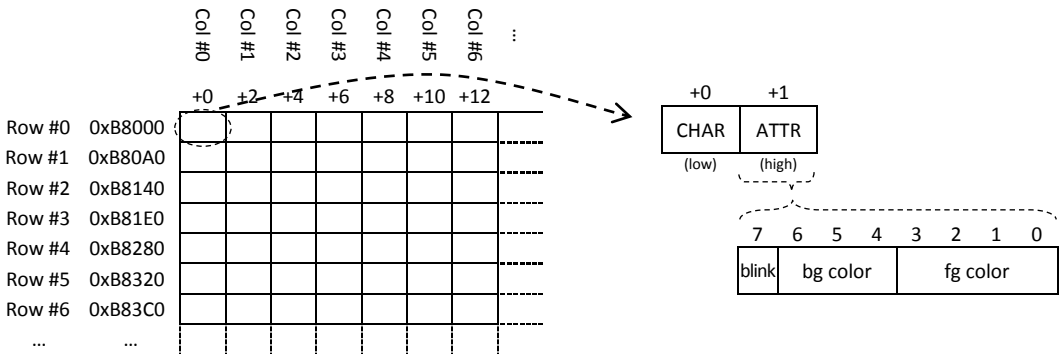
dd if=lscboot.bin of=lsc.fd bs=512 count=1 conv=notrunc

Finally setup a VMware virtual machine to boot from `lsc.fd`.

Direct access to screen in text-mode

To control the text mode screen we may use direct access to video memory at linear address 0xB8000, assuming the common text resolution of 80x25 colored screen cells.

Each screen cell is accessed via two consecutive bytes in video memory. The low order byte holds the numeric code of the character to be displayed, while the high order byte defines foreground and background colors, and the blink state of the cell.



Colors:

0	BLACK		8	DARK GRAY
1	BLUE		9	LIGHT BLUE
2	GREEN		10	LIGHT GREEN
3	CYAN		11	LIGHT CYAN
4	RED		12	LIGHT RED
5	MAGENTA		13	LIGHT MAGENTA
6	BROWN		14	YELLOW
7	LIGHT GRAY		15	WHITE

The position of the text cursor is defined as a 16-bit value stored in two 8-bit registers of the video controller. For some (X,Y) screen position, the 16-bit value is calculated as Y*COLS+X, where COLS is the number of text columns of the screen (80 in our case). The low-order 8-bits of this value are stored in register 0x0F (Cursor Location Low Register) of the video controller, while the high-order 8-bits are stored in register 0x0E (Cursor Location High Register).

To access the video controller registers, write the register number (0x0F or 0x0E, in our case) into I/O port 0x3D4 (CRTC Controller Address Register), and then read or write I/O port 0x3D5 (CRTC Controller Data Register).

More detailed information can be found at: http://www.osdever.net/FreeVGA/home.htm

---