INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

COMPUTER SYSTEMS LABORATORY

Fall 2011

Stage 1

Synopsis

During this stage you will develop a program that runs on a bare PC (i.e. without an operating system) using:

- Screen in graphics mode
- Timer

Additionally, the processor will be set to run in long mode (64-bit mode in AMD64/Intel64 processors) and a significant part of the code will be developed in C.

At the end of this stage, you should know:

- what are the basic characteristics of the instruction set of AMD64/Intel64 processors;
- the calling convention used by C compilers for 64-bit Linux, BSD, and Mac systems;
- how to access common devices, such as the screen (graphics mode) and the timer;
- how to program in C (and assembly) in a freestanding environment.

The base project

Attached to this document, you'll find a *.tar.gz* archive containing a small project for building a bootable floppy disk that changes the processor to long mode (64-bit), writes some message on the screen, and finally halts the processor.

Given the base project structure, you'll be able to proceed into stage 1 even if stage 0 is not yet concluded. It should be a trivial task to integrate the results of both stages, when needed.

The base project has two parts:

- ***bootrec*:** a boot record that loads a single file, of up to 8KiB, into address 0x10000;
- ***lsc.sys*:** the code to be loaded at 0x10000, which changes the processor to long mode, presents some message on the screen, and then halts the processor.
  NOTE: placing this file in the root directory of your MINIX file system, to be loaded by your VBR, should have a similar result to booting from this floppy.

The floppy has no file system. It contains *bootrec* in its first logical sector, and *lsc.sys* starts at the second logical sector and occupies up to 16 sectors. With this organization, only the first floppy track has to be read during boot. On the other hand, *lsc.sys* cannot exceed 8KiB.

The project has the following structure:

- *bootrec*
  - **bootrec.s** : boot record source code, which loads *lsc.sys* into 0x10000
  - **bootrec.ld** : boot record linker script, placing all sections (*.text*, *.rodata*, and *.data*) into 0x7C00, and carving the 0xAA55 signature
- *lsc.sys*
  - **start16.s** : real mode code, which enables A20, changes the processor to long mode, and finally jumps to 64-bit startup code present in start64.s
  - **start64.s** : long mode code, which sets up a minimal environment to run C code; in particular, it zeroes BSS memory and initializes the stack pointer; finally, *lsc_main* is called
  - **prog.c** : sample freestanding C program that prints a colored message on the screen
  - **lsc64.ld** : linker script for *lsc.sys*, which locates the contents of *start16.s* at the beginning of the target address range (0x10000), prepared for segmented addressing with the segment value of 0x1000; the contents of the remaining files follow, prepared for direct 64-bit addressing
- General
  - **Makefile** : builds *bootrec*, then *lsc.sys*, and produces the floppy disk
  - **lsc64.fd** : the name of the file that will contain the virtual floppy disk
  - **.bochrc** : bochs configuration file, which boots a virtual PC from *lsc64.fd*

At the base project directory, type *make*, and then run *bochs*.

VESA BIOS Extensions

The international standards body VESA defined an interface for accessing video boards, which presents itself as a set of BIOS extensions. Version 2 of this standard is widely supported, in particular with a real mode interface. As such, we will set a fixed graphics mode while still in real mode, and then access it from long mode via a linear frame buffer (LFB).

To get information about supported VESA modes, use INT 0x10 with AX set to 0x4F00 and ES:DI pointing to a 768 byte buffer. The first bytes of this space are organized as specified by struct vesa_general_info in http://lxr.linux.no/linux+v2.6.32.34/arch/x86/boot/vesa.h . The field signature should be set to “VBE2” before issuing INT 0x10.

After calling the service, AL should have the value 0x4F and AH will hold the error code, with 0x00 meaning success. Field video_mode_ptr will point to an array filled with all valid video mode ids, terminated by 0xFFFF.

To get information about a specific video mode, use INT 0x10 with AX set to 0x4F01, CX set to the specific video mode id, and ES:DI pointing to a 256 byte buffer, organized as specified by struct vesa_mode_info in http://lxr.linux.no/linux+v2.6.32.34/arch/x86/boot/vesa.h .

Exercise 1

Modify prog.c to show information about the following video modes, if available:

```
8bpp   640x480  800x600  1024x768
16bpp  640x480  800x600  1024x768
```

The first line of the screen should present the vendor string. The next 6 lines should present the details about the video modes, in the form:

```
ID  HRES x VRES  BPP  LFB  LFB_ADDR
```

where:

ID	video mode id
HRES	horizontal resolution
VRES	vertical resolution
BPP	bits per pixel
LFB	linear frame buffer supported (mode_attr & 0x80)
LFB_PTR	linear frame buffer address

Timer

The standard PC timer chip is the 8254 (originally 8253), known as Programmable Interval Timer (PIT). Three internal independent 16-bit counters are used to divide a fixed[1] input frequency of 1.193182 MHz. Loading a counter with the maximum value of 65536 and letting it count down to zero, at the fixed input frequency, will generate the minimum frequency, of about 18.2 Hz.

The three counters are accessible via three 8-bit I/O ports. A fourth I/O port is used for control:

Port	Direction	Name
0x40	Read / Write	Counter 0
0x41	Read / Write	Counter 1
0x42	Read / Write	Counter 2
0x43	Write Only	Control

Each counter may be used in one out of six modes. These modes determine whether the counter is one-shot or periodic, hardware or software triggered, and the form of the output signal:

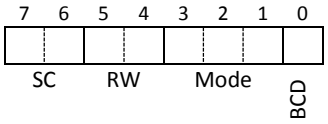
Mode	Name	Period	Trigger	Output
0	Interrupt on Terminal Count	One-shot	Software	Low
1	Programmable Monoflop	One-shot	Hardware	Low
2	Rate Generator	Periodic	Software or Hardware	Pulse
3	Square-Wave Generator	Periodic	Software or Hardware	Square
4	Software-Triggered Pulse	One-shot	Software	Pulse
5	Hardware-Triggered Pulse	One-shot	Hardware	Pulse

[1] The 8253/8254 has three individual clock input pins, but in the PC they are all connected to the same clock source.

The counters can be individually programmed to be accessed with 8 or 16 bit values, with the latter requiring two read or write accesses to get or set a counter value, respectively.

RW	Access Mode	Used for
01_2	Low counter byte only	Small divisor
10_2	High counter byte only	Course-grained divisor
11_2	Low byte then high byte	Full 16-bit divisor

To configure one of the counters, write a command byte to the control register:



The SC field selects the counter (0, 1 or 2) to be configured. BCD, if set to 1, configures the counter to work in BCD mode, producing values between 0000_{BCD} and 9999_{BCD}.

An additional RW value, 00_2, indicates a *counter latch command*. This will atomically read both bytes of the selected counter into a 16-bit latch. The next read operation will read the latched value, instead of the live counter. This is particularly important when the counter has been configured for 16-bit accesses, as individual reads of the low and high order bytes may produce erroneous 16-bit values (e.g.: reading a counter when its value changes from 256 to 255 may result in a low order byte of 0xFF and a high order byte of 0x01, for a final value of 511).

In the PC, the output of the counter 0 is connected to IRQ0 in the interrupt controller and it's the only counter that may generate interrupts. Counter 1 was used in the past to set the pace for DRAM refreshment, a task currently performed by dedicated logic, rendering this counter useless in practice. The output of counter 2 may be connected to the PC Speaker and is usually reserved for tone generation.

For more information about the timer, see:
The Indispensable PC Hardware Book, chapter 24 (2nd ed.), 27 (3rd ed.) or 19.1 (4th ed.)
http://download.intel.com/design/archives/periphrl/docs/23124406.pdf

Exercise 2

Develop a set of auxiliary functions to program the PC timer chip. Then develop the function *delay*, which returns after a specified time lapse, with a resolution of tens of milliseconds.

Write a test program that exhibits a 3 digit counter at the bottom right corner of the text-mode screen.

(to be continued)