OS Lab03——实模式启动

PB15000102 王嵩超

背景知识与技能

BIOS

来自维基百科

BIOS, an acronym for **Basic Input/Output System** and also known as the **System BIOS**, **ROM BIOS** or **PC BIOS**) is a type of firmware used to perform hardware initialization during the booting process (power-on startup) on IBM PC compatible computers, and to provide runtime services for operating systems and programs.

启动协议

BIOS固件里包含的代码，会在机器启动按照可配置的引导顺序，将启动扇区的启动代码加载到RAM（0x7C00处）并跳至此处执行。

对于软盘启动，若软盘设备第一个扇区的最后两字节为0xAA55，则该扇区被认作启动扇区。

为什么要关中断

一方面可能的原因是，从实模式切换至保护模式过程短暂而关键，没有必要打断。

更重要的是：

> 详细的解释位于http://stackoverflow.com/questions/16536035/why-do-interrupts-need-to-be-disabled-before-switching-to-protected-mode-from-re

计算机运行时，时常会有硬件中断（比如，可编程间隔定时器(PIT, Programmable Interval Timer)产生的中断IRQ0用于记录时间）。在实模式，这些中断由实模式下的Interrupt handler处理。

进入保护模式前，我们准备好了全局描述符表GDT。将准备好的信息加载到GDTR和IDTR。IDT寄存器的limit部分（标明IDT的长度）首先会被写成0，这是刻意为了避免实模式的BIOS Interrupt Call（参考：https://en.wikibooks.org/wiki/X86_Assembly/Protected_Mode#Entering_Protected_Mode）。这时进入保护模式。像IRQ0这样的硬件中断会使CPU产生异常（因为limit部分为0，相当于想被执行的interrupt handler超过了limit，引发保护模式的异常），紧接着又会产生#DF(Double Fault)、#TF.....所以需要暂时关中断。

什么是实模式

Real mode, also called **real address mode**, is an operating mode of all x86-compatible CPUs. Real mode is characterized by a 20-bit segmented memory address space (giving exactly 1 MiB of addressable memory) and unlimited direct software access to all addressable memory, I/O addresses and peripheral hardware. Real mode provides no support for memory protection, multitasking, or code privilege levels. Before the release of the 80286, which introduced protected mode, real mode was the only available mode for x86 CPUs. In the interest of backward compatibility, all x86 CPUs start in real mode when reset, though it is possible to emulate real mode on other systems when starting on other modes.

VGA-Compatible text mode

此即为VGA显存里存放文本内容的格式。

格式：

Attribute								Character							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Blink[n 1]	Background color			Foreground color[n. 2][n. 3]				Code point							

其中Character部分为ASCII码。

前景色和背景色的数码均由VGA主要颜色指定。

实模式为什么要初始化段寄存器

在Intel设计的内存模型中，CS、DS、SS三个段寄存器分别指明了代码所在段、数据所在段、栈所在段。如果不先把这些段寄存器初始化为0，执行的代码就可能不在0x7C00处，而是要加上16xCS。

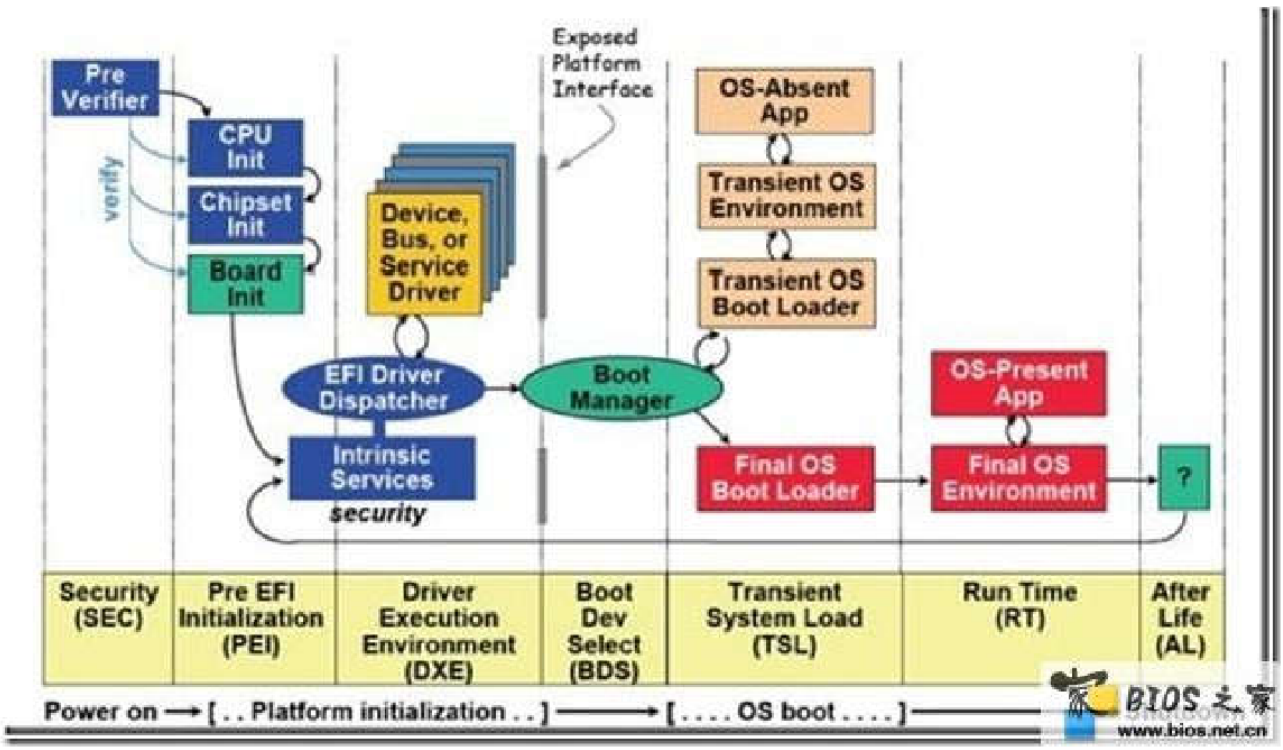
ld文件

ld文件是链接器ld的脚本文件，主要用来将汇编代码中的各个段布局到二进制可执行文件的各具体位置。

具体用法可参考：https://ftp.gnu.org/pub/old-gnu/Manuals/ld-2.9.1/html_node/ld_6.html#SEC6

ld文件和汇编源代码都用实心点来表示当前代码的位置，通过对实心点赋值来控制代码的布局。

上电后的启动步骤



上电后，首先会进行硬件的自检与CPU、芯片组和主板的初始化。

我们所写的代码是位于Transient System Load阶段的独立于OS的程序。

用gdb调试启动代码

由于是第一次自己写汇编程序，难免出现一些想当然的错误。

比如：

将：

```
movb latter, %dl
```

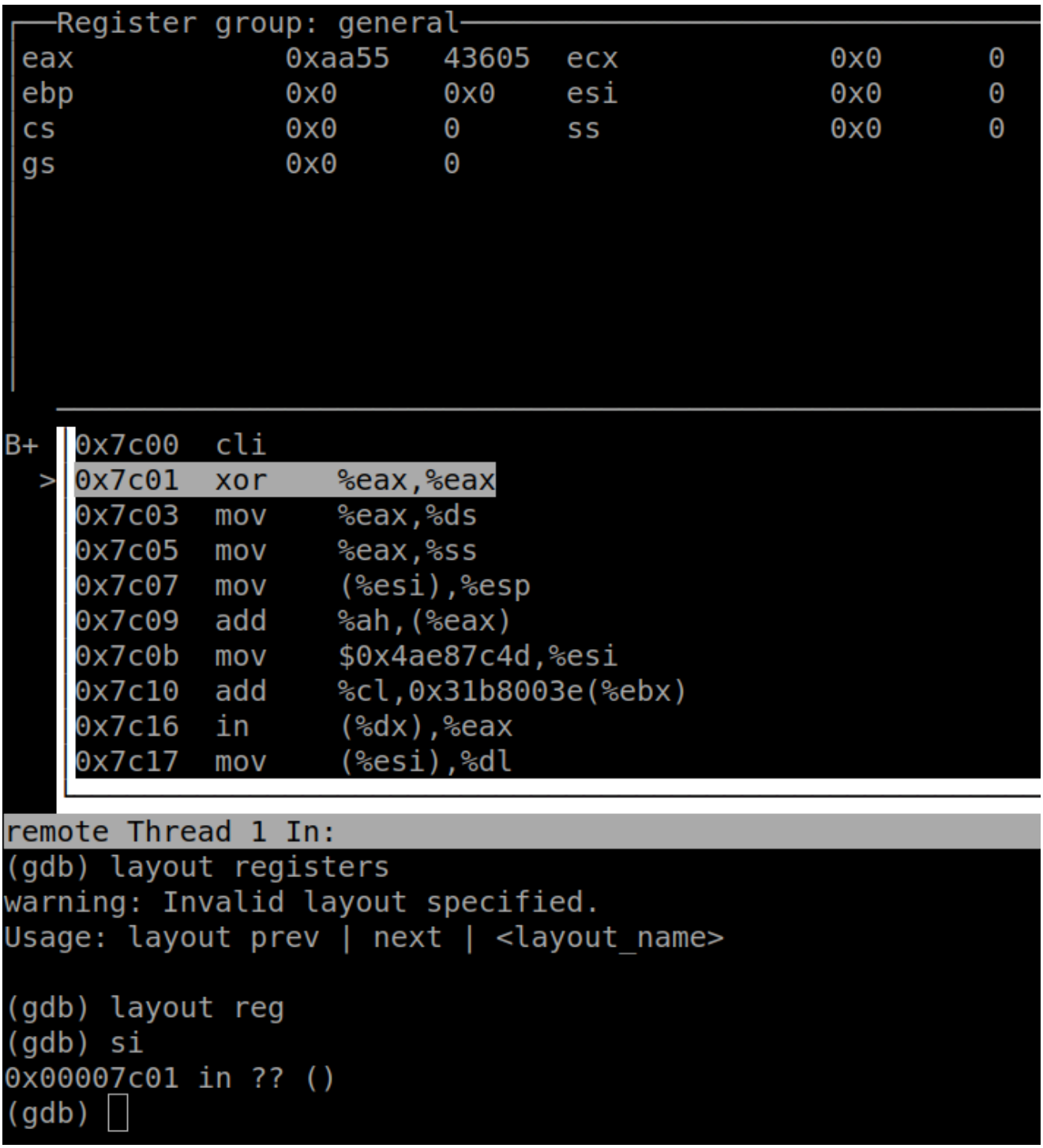
写成：

```
movb (latter), %dl
```

故适合用步进的方法查看执行步骤，具体操作如下（已经启动了等待调试的qemu）：

```
$ gdb

(gdb) target remote localhost:1234
(gdb) break *0x7c00 #在启动代码处设置断点
(gdb) layout asm
(gdb) layout reg #分别显示汇编指令窗口和寄存器窗口
(gdb) cont
(gdb) si #步进调试
```



部分原理说明

- 为什么要用objcopy

  ELF文件格式并不是我们想要的，这是Linux系统使用的可执行文件格式。它的文件头是包含各种段信息的固定格式，会由操作系统识别，这在实模式被执行时完全是多余且有害的。

  我们需要的是一个仅包含x86指令的二进制文件，这就要用到objcopy，它能从elf文件中把x86指令提取出来。

- 汇编指令中地址的表示

- 被运行的程序入口地址是0x7C00，但在启动扇区里程序代码是从文件的0x0000开始的。所以我们在制作elf文件时，就用ld脚本将o文件布局到0x7C00处，这时链接器会以0x7C00为起点重新计算各操作数地址（reallocation）。elf文件中0x7C00地址之前的空白部分不是我们想要的。于是objcopy又原封不动地将指令拷贝到bin文件。这就实现了把代码放到开始处，而代码本身又是按照0x7C00开始来生成的。
- 也可以用直接计算的方法来得到正确的地址（此方法来自grub源代码里的stage1.s，本人并未采用）：

  本段代码将label处的地址装载到%si寄存器内。

```
#define ABS(x) (x-_start+0x7c00) #_start为程序入口地址标签
movw $ABS(label), %si
#等效的做法：
lea label, %si
```

前两行，把label标签与_start标签相减，得到label处与程序入口处的地址差。而程序入口地址我们已经知道在执行时会是0x7C00。故依此得到立即数$ABS(label)。该操作数因为是立即数，不会参与ld的reallocation。经过gdb调试，可以发现执行第四行与执行第1、2行后，%si的值是一样的。

遇到的问题

汇编时提示：display(%ax)' is not a valid base/index expression

Answer:

实模式下 %ax寄存器不能作为base/index的用途。可考虑用其他寄存器作为偏移量。

Reference: http://stackoverflow.com/questions/34345583/invalid-base-index-expressions

".signature" section的aa55在ELF文件有，而在BIN文件总是缺失

可能的解释：The AT&T assembler does not provide support for multiple section programs. Unix style systems expect all programs to be single sections.

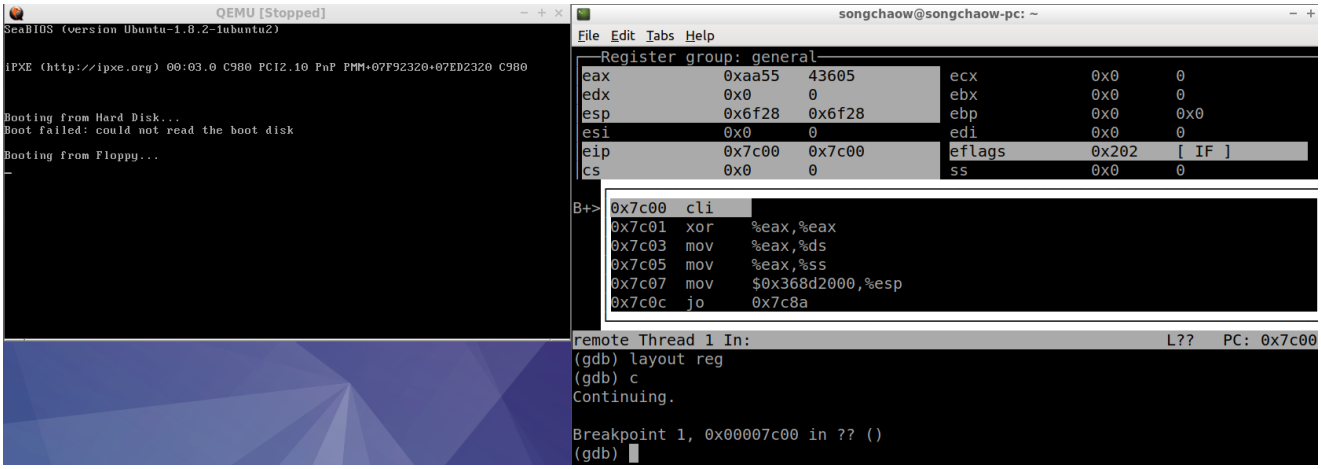
来自: ftp://ftp.gnu.org/old-gnu/Manuals/gas/html_node/as_196.html

> 不一定正确。因为群里貌似有人使用AT&T的汇编格式也成功生成了"aa55"

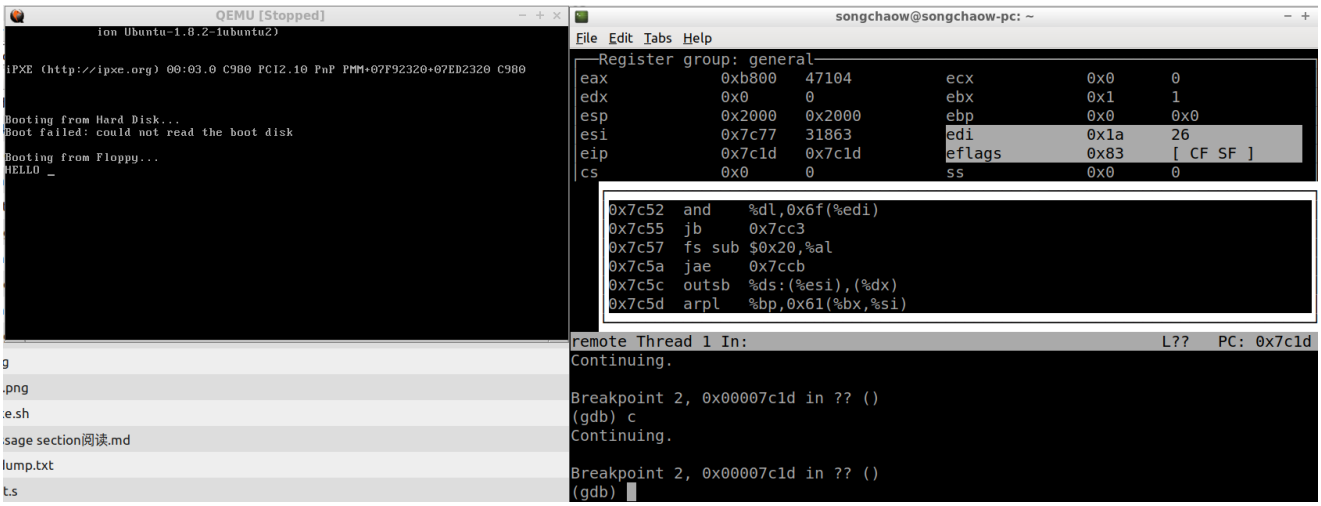
最后暂时的解决办法是将aa55的位置直接在S文件中指定。

实验结果

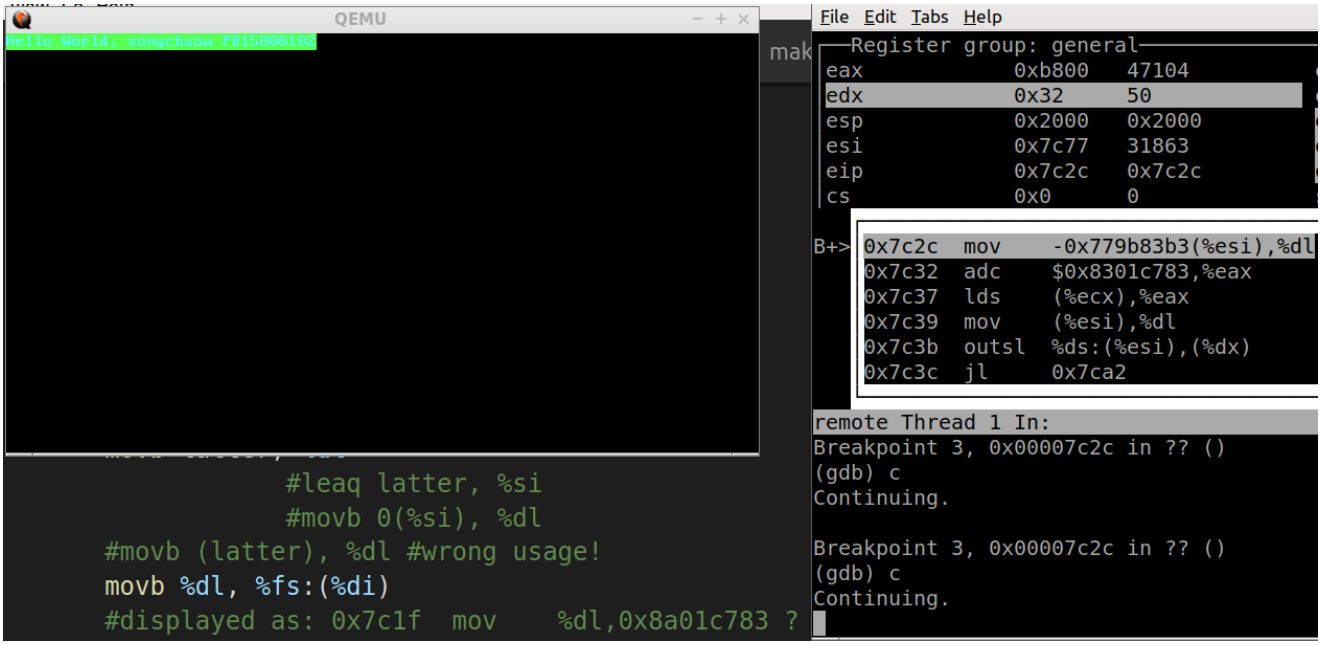
boot开始时：



清屏中(后面输出的HELLO是由BIOS中断输出用于测试，不是重点)：



输出HELLOWORLD（自己使用了浅绿色的背景色和天蓝色的前景色）：


QEMU
Hello World, songchaow PB15000102
#leaq latter, %si
#movb 0(%si), %dl
#movb (latter), %dl #wrong usage!
movb %dl, %fs:(%di)
#displayed as: 0x7c1f mov %dl,0x8a01c783 ?
File Edit Tabs Help
Register group: general
eax 0xb800 47104
edx 0x32 50
esp 0x2000 0x2000
esi 0x7c77 31863
eip 0x7c2c 0x7c2c
cs 0x0 0
B+> 0x7c2c mov -0x779b83b3(%esi),%dl
0x7c32 adc $0x8301c783,%eax
0x7c37 lds (%ecx),%eax
0x7c39 mov (%esi),%dl
0x7c3b outsl %ds:(%esi),(%dx)
0x7c3c jl 0x7ca2
remote Thread 1 In:
Breakpoint 3, 0x00007c2c in ?? ()
(gdb) c
Continuing.
Breakpoint 3, 0x00007c2c in ?? ()
(gdb) c
Continuing.