用汇编和 C 编写引导加载程序 - 第 2 部分

阿什奇兰·巴特

2013 年 10 月 17 日 警察

使用 BIOS 中断和服务读取软盘的内容。

介绍

在上一篇文章中，我试图简要介绍引导、如何用 C 和汇编编写可引导代码，以及如何在 C 程序中嵌入汇编语句。我们还尝试编写一些程序来检查我们注入设备引导扇区的代码是否有效。在本文中，我将尝试解释在启动过程中从软盘中分段和读取数据并将其显示在屏幕上。我们将尝试编写一些程序并检查我们是否可以读取数据并尝试将其显示在屏幕上。

文章的范围是什么？

我将文章的范围限制在如何用汇编编写程序代码，如何将其复制到 3.5 英寸软盘映像的引导扇区，然后如何测试软盘以使用我们的代码启动它使用像 bochs 这样的 x86 模拟器编写。我将借助BIOS服务来实现从软盘读取数据的任务。通过这样做，我们可以探索更多关于 BIOS 例程的信息，并在实模式下感到舒适。

主题细分

- 细分简介
- 编程环境
- 从 RAM 存储器读取数据
- 存储设备介绍
- 软盘的架构
- 与软盘交互

分割简介

在我继续写一些关于如何读取软盘的示例之前，我想刷新一下分段的主题及其需求，如下所示。

什么是分段?

主存储器被分成多个段，这些段由一些称为段寄存器（CS、DS、SS 和 ES）的特殊寄存器索引。

分段有什么用?

当我们指定一个 16 位地址时，CPU 会自动计算相应段的起始地址。但是，程序员有责任指定每个段的起始地址，尤其是在编写引导加载程序等程序时。

有哪些不同类型的细分市场?

我现在只提到四种类型，因为它们对我们理解很重要。

- 代码段
- 数据段
- 堆栈段
- 扩展段

代码段

它是内存中包含可执行指令的程序部分之一。如果您参考我之前的文章，您将看到标签 .text，我们打算在其中放置要执行的指令。当程序加载到内存中时，.text 段下的指令被放入代码段。在 CPU 中，我们使用 CS 寄存器来引用内存中的代码段。

数据段

它是内存中程序的一部分，其中包含程序员的静态和全局变量。我们使用 DS 寄存器来引用内存中的数据段。

堆栈段

程序员可以在他编写的程序范围内使用寄存器来存储、修改和检索数据。由于在程序运行期间可供程序员使用的寄存器很少，因此程序逻辑总是有可能变得复杂，因为只有少数寄存器可供临时使用。因此，程序员可能总是觉得需要一个更大的地方，在存储、处理和检索数据方面更灵活。CPU 设计者提出了一个称为堆栈段的特殊段。为了在堆栈段上存储和检索数据，程序员使用push 和pop 指令。我们也使用推送指令将参数传递给函数。我们使用 SS 寄存器来引用内存中的堆栈段。还要记住堆栈向下增长。

扩展段:

扩展段通常用于加载远大于数据段中存储的数据大小的数据。您将进一步看到我将尝试从扩展段上的软盘加载数据。我们使用 ES 寄存器来引用内存中的扩展段。

如何设置段寄存器?

程序员没有直接设置任何段寄存器的自由，而是我们遵循这种方式。

自动售货机 复制代码

```
movw $0x07c0, %ax
movw %ax, %ds
```

上面的步骤是什么意思?

- 将数据复制到通用寄存器。
- 然后将其分配给段寄存器。

我们将 AX 寄存器设置为 0x07c0

当我们将 AX 的内容复制到 DS 时。绝对地址计算如下。

复制代码

```
DS = 16 * AX
So DS = 0x7c00
```

我们进一步使用偏移量从这里开始遍历。为了到达数据段中的某个位置，我们使用偏移量。

编程环境

- 操作系统 (GNU Linux)
- 汇编程序（GNU 汇编程序）
- 编译器（GNU GCC）
- 链接器（GNU 链接器 ld）
- 用于我们的测试目的的 x86 模拟器（bochs）。

从 RAM 读取数据

笔记

现在，如果您观察 BIOS 在 0x7c00 加载我们的程序并开始执行它，然后我们的程序开始一个一个地打印值。我们通过直接指定偏移量并将数据段设置为 0x7c00 来访问 RAM 上的数据。

示例 1

一旦我们的程序在 0x7c00 处被 BIOS 加载，让我们尝试从偏移量 3 和 4 读取数据，然后将它们打印到屏幕上。

程序：test.S

自动售货机 复制代码

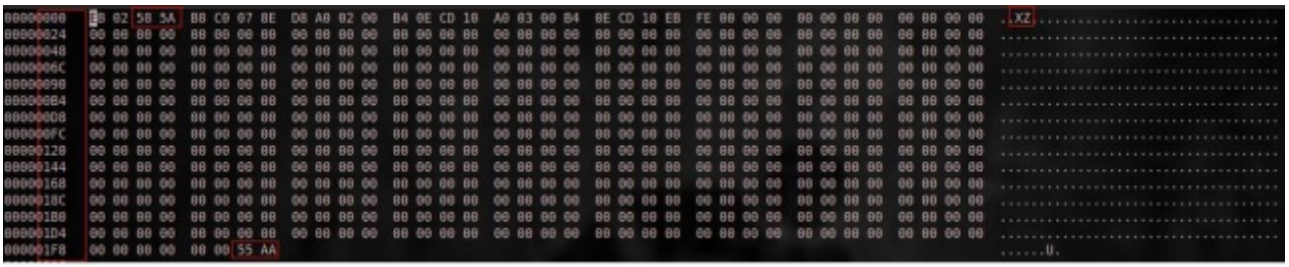
```
.code16                     #generate 16-bit code
.text                       #executable code location
     .globl _start;
_start:                     #code entry point
     jmp  _boot             #jump to boot code
     data : .byte 'X'       #variable
     data1: .byte 'Z'       #variable
_boot:
     movw $0x07c0, %ax      #set ax = 0x07c0
     movw %ax    , %ds      #set ds = 16 * 0x07c0 = 0x7c00
     #Now we will copy the data at position 3 from 0x7c00:0x0000
     # and then print it onto the screen
     movb 0x02   , %al      #copy the data at 2nd position to %al
     movb $0x0e  , %ah
     int  $0x10
    #Now we will copy the data at position 4 from 0x7c00:0x0000
    # and then print it onto the screen
     movb 0x03   , %al      #copy the data at 3rd position to %al
     movb $0x0e  , %ah
     int  $0x10
#infinite loop
_freeze:
     jmp _freeze
     . = _start + 510       #mov to 510th byte from 0 pos
     .byte 0x55             #append boot signature
     .byte 0xaa             #append boot signature
```

现在在命令行上，要生成二进制文件并将代码复制到软盘的引导扇区，请键入以下内容，然后按回车键。

- 作为 test.S –o test.o
- ld –Ttext=0x7c00 –oformat=binary boot.o –o boot.bin
- dd if=/dev/zero of=floppy.img bs=512 count=2880
- dd if=boot.bin of=floppy.img

笔记

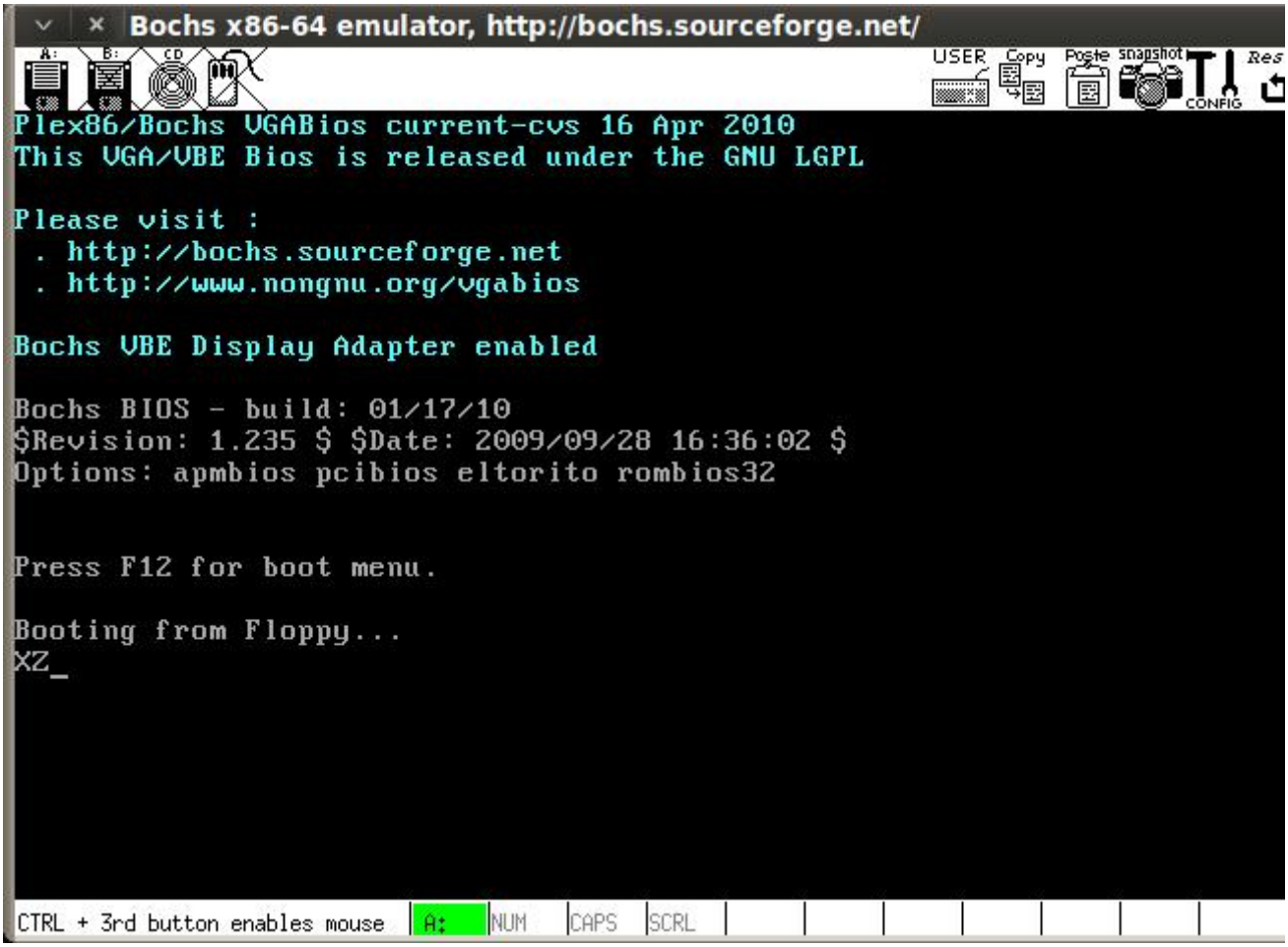
如果你在十六进制编辑器中打开*boot.bin*文件，你会看到类似下面的内容。



您将看到 X 和 Y 位于距 0x7c00 的第 0 个偏移量的第三和第四个位置。

要测试代码，请键入以下命令，如下所示。

- 博克斯



例子：

一旦我们的程序在 0x7c00 处被 BIOS 加载，让我们从偏移量 2 读取一个空终止字符串，然后打印它。

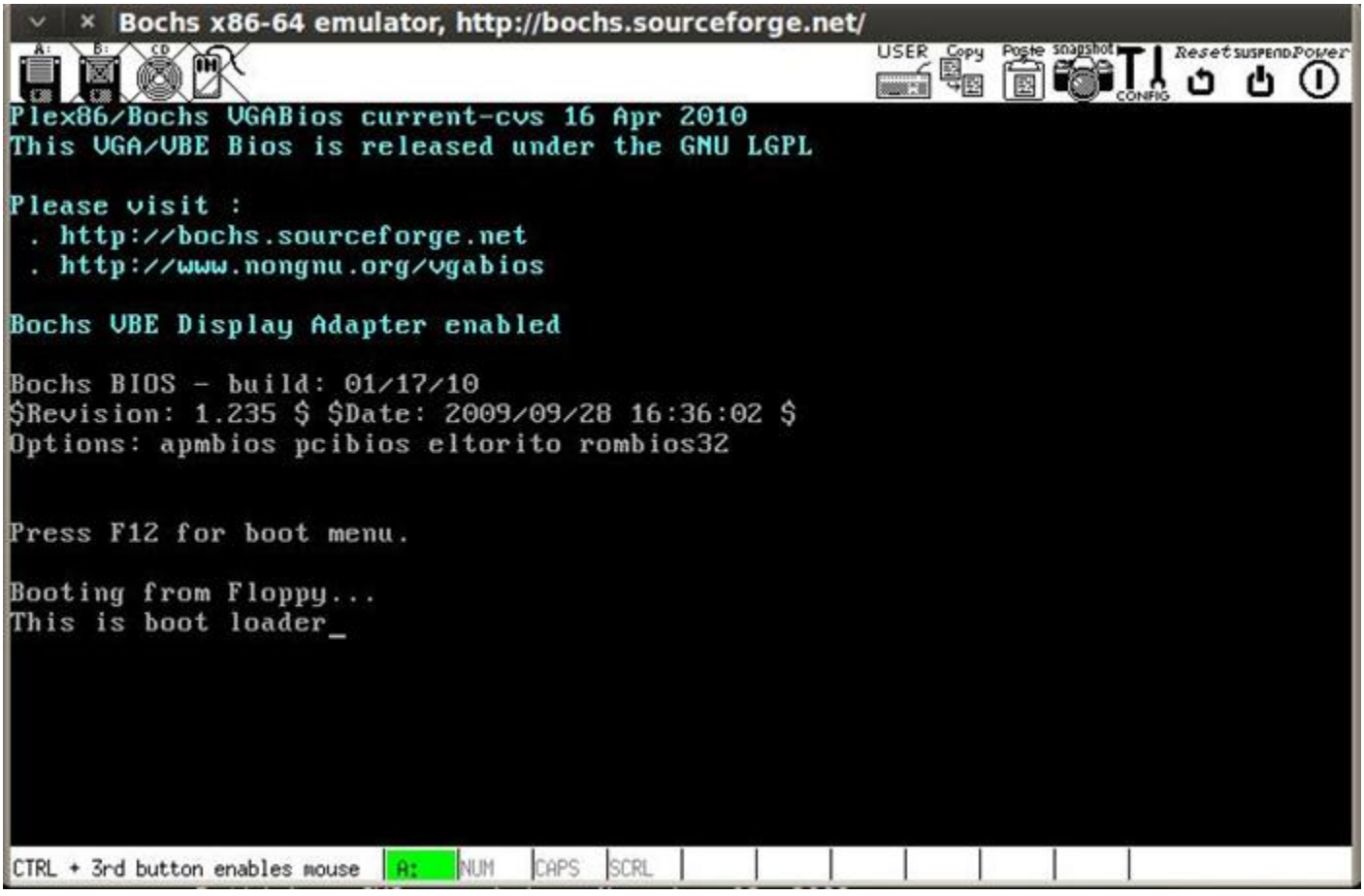
程序：test2.S

自动售货机 缩小▲ 复制代码

```
.code16                                     #generate 16-bit code
.text                                       #executable code location
     .globl _start;
_start:                                     #code entry point
     jmp  _boot                             #jump to boot code
     data : .asciz "This is boot loader"    #variable
     #calls the printString function which
     #starts printing string from the position
```

```
     .macro mprintString start_pos          #macro to print string
          pushw %si
          movw  \start_pos, %si
          call  printString
          popw  %si
     .endm
     printString:                           #function to print string
     printStringIn:
          lodsb
          orb %al   , %al
          jz  printStringOut
          movb $0x0e, %ah
          int  $0x10
          jmp  printStringIn
     printStringOut:
     ret
_boot:
     movw $0x07c0, %ax                      #set ax = 0x07c0
     movw %ax    , %ds                      #set ds = 16 * 0x07c0 = 0x7c00
     mprintString $0x02
_freeze:
     jmp _freeze
     . = _start + 510                       #mov to 510th byte from 0 pos
     .byte 0x55                             #append boot signature
     .byte 0xaa                             #append boot signature
```

如果您编译程序并在十六进制编辑器中打开二进制文件，您可能会在输出中看到字符串“This is boot loader”。



存储设备介绍

什么是存储设备？

它是一种用于信息存储、检索的设备。它也可以用作可启动媒体。

为什么计算机需要存储设备?

我们主要使用计算机来存储信息，检索信息并对其进行处理，因此作为信息存储和检索的一部分，制造商提出了一种新设备，称为存储设备，具有多种类型。

有哪些不同类型的存储设备可用?

根据数据的大小，我将尝试将它们列出如下。

- 软盘
- 硬盘
- U盘

和更多...

什么是软盘?

它是一种用于信息存储、检索的设备。它也用作可启动媒体。

软盘设计用于存储少量数据，其最大大小可以限制为几兆字节。

什么是兆字节?

在计算中，数据的大小是通过以下方式之一来衡量的：

- **位**：它可以存储 1 或 0 的值
- **半字节**：4 位
- **字节(B)** : 8 位
- **千字节(KB)** : 1024 字节
- **兆字节（MB）** ： 1 千字节 * 1 千字节 = 1,048,576 字节 = 1024 千字节 = 1024 * 1024 字节
- **Giga Byte(GB)** : 1,073,741,824 Bytes= 2^30 Bytes = 1024 Mega Bytes = 1,048,576 Kilo Bytes = 1024 * 1024 * 1024 Bytes
- **Tera Byte(TB)** : 1,099,511,627,776 Bytes= 2^40 Bytes = 1024 Giga Bytes = 1,048,576 MB = 1024 * 1024 * 1024 * 1024 Bytes

还有更多.....但我们会将自己限制在上述范围内。

软盘是什么样子的?

我可以在软盘上存储多少数据?

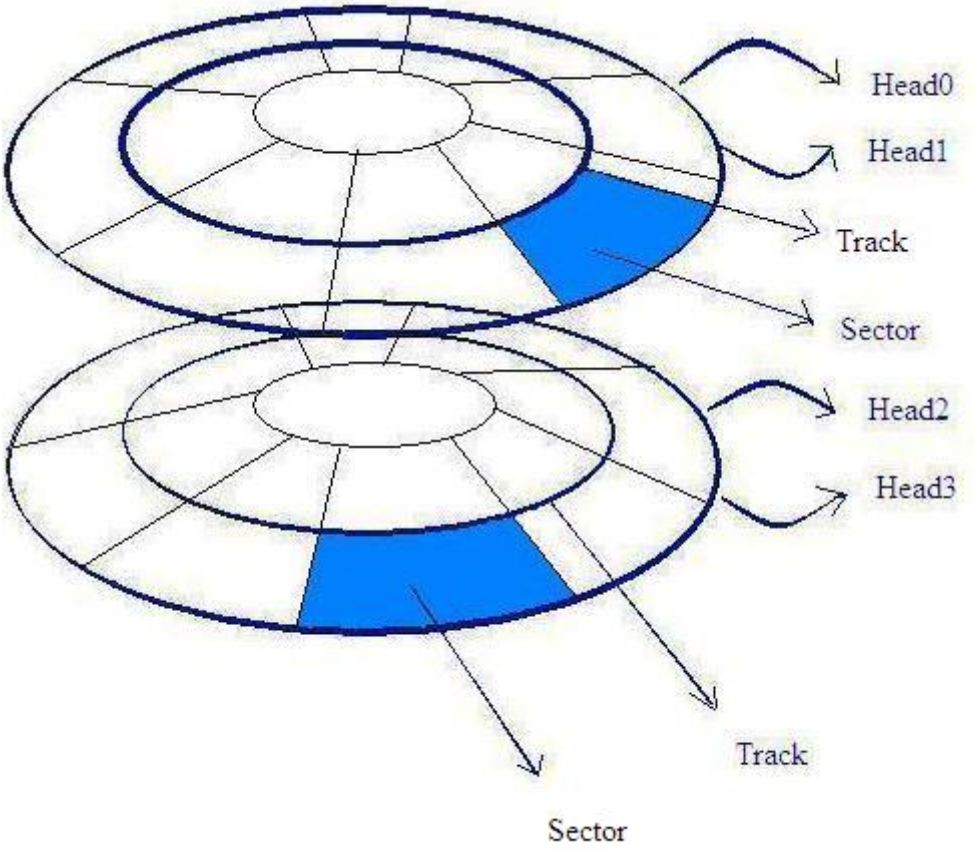
这完全取决于制造商提供的各种类型的软盘及其各自的大小。

我将简单地列出我一生中使用过的3.5英寸软盘的详细信息。

复制代码

```
Name            Description              Size(mm)            Volume(mm2)     Capicity(MB)
3.5 inches   3.5 inch Floppy Disk   93.7 x 90.0 x 3.3     27,828.9         1.44 MB
```

典型软盘的架构



上图展示了一个典型软盘的架构，让我们关注一下3.5英寸软盘，我将在下面详细解释。

如何描述一张3.5英寸的软盘?

- 它有 2 个面
- 它的侧面称为头部。
- 每面包含 80 条轨道。
- 每个磁道包含 18 个扇区。
- 每个扇区包含 512 个字节。

如何计算软盘的大小?

- 以字节为单位的总大小：总边数 * 总轨道数 * 每个轨道的总扇区数 * 每个扇区的总字节数。

总大小（字节）= 2 * 80 * 18 * 512 = 1474560 字节。

- 以千字节为单位的总大小：（总边数 * 总轨道数 * 每条轨道的总扇区数 * 总字节数）/1024。

总大小(KB) = (2 * 80 * 18 * 512)/1024 = 1474560/1024 = 1440 KB。

- 以兆字节为单位的总大小：（（总边数 * 总轨道数 * 每条轨道的总扇区数 * 总字节数）/1024）/1024

总大小(MB) = ((2 * 80 * 18 * 512)/1024)/1024 = (1474560/1024)/1024 = 1440/1024 = 1.4 MB

软盘上的引导扇区位于何处?

它位于磁盘的第一个扇区。

与软盘交互

如何从软盘读取数据?

由于我们在本文中的任务是从软盘读取数据，因此目前唯一的选择是在我们的程序中使用 BIOS 服务，因为在启动时我们处于实模式以与软盘进行交互。我们需要使用 BIOS 中断来完成我们的任务。

我们将使用哪些中断?

复制代码

```
Interrupt 0x13
Service code 0x02
```

如何使用中断 0x13 访问软盘?

- 要请求 BIOS 读取我们下面使用的软盘上的扇区。

复制代码

```
AH = 0x02
```

- 要请求 BIOS 从我们下面使用的第 N 个柱面读取。

复制代码

```
CH = ‘N’
```

- 要请求 BIOS 从我们下面使用的第 N 个磁头读取。

复制代码

```
DH = ‘N’
```

- 要请求 BIOS 读取我们在下面使用的第 N 个扇区。

复制代码

```
CL = ‘N’
```

- 要请求 BIOS 读取我们在下面使用的“N”个扇区。

复制代码

```
AL = N
```

- 要中断 CPU 以执行此活动，我们将在下面使用。

复制代码

```
Int 0x13
```

从软盘读取数据

让我们编写一个程序来显示几个扇区的标签。

程序：*test.S*

自动售货机 缩小▲ 复制代码

```
.code16                         #generate 16-bit code
.text                           #executable code location
.globl _start;                  #code entry point
_start:
     jmp _boot                  #jump to the boot code to start execution
     msgFail: .asciz "something has gone wrong..." #message about erroneous operation
      #macro to print null terminated string
      #this macro calls function PrintString
     .macro mPrintString str
          leaw \str, %si
          call PrintString
     .endm
     #function to print null terminated string
     PrintString:
          lodsb
          orb  %al  , %al
          jz   PrintStringOut
          movb $0x0e, %ah
          int  $0x10
          jmp  PrintString
     PrintStringOut:
     ret
     #macro to read a sector from a floppy disk
     #and load it at extended segment
     .macro mReadSectorFromFloppy num
          movb $0x02, %ah     #read disk function
          movb $0x01, %al     #total sectors to read
          movb $0x00, %ch     #select cylinder zero
          movb $0x00, %dh     #select head zero
          movb \num, %cl      #start reading from this sector
          movb $0x00, %dl     #drive number
          int  $0x13          #interrupt cpu to get this job done now
          jc   _failure       #if fails then throw error
          cmpb $0x01, %al     #if total sectors read != 1
          jne  _failure       #then throw error
     .endm
     #display the string that we have inserted as the
     #identifier of the sector
     DisplayData:
     DisplayDataIn:
          movb %es:(%bx), %al
          orb  %al      , %al
          jz   DisplayDataOut
          movb $0x0e    , %ah
          int  $0x10
          incw %bx
          jmp  DisplayDataIn
     DisplayDataOut:
     ret
_boot:
     movw  $0x07c0, %ax       #initialize the data segment
     movw  %ax    , %ds       #to 0x7c00 location
     movw  $0x9000, %ax       #set ax = 0x9000
     movw  %ax    , %es       #set es = 0x9000 = ax
     xorw  %bx    , %bx       #set bx = 0
     mReadSectorFromFloppy $2 #read a sector from floppy disk
     call DisplayData         #display the label of the sector
     mReadSectorFromFloppy $3 #read 3rd sector from floppy disk
     call DisplayData         #display the label of the sector
_freeze:                      #infinite loop
     jmp _freeze              #
_failure:                     #
     mPrintString msgFail     #write error message and then
     jmp _freeze              #jump to the freezing point
     . = _start + 510         #mov to 510th byte from 0 pos
```

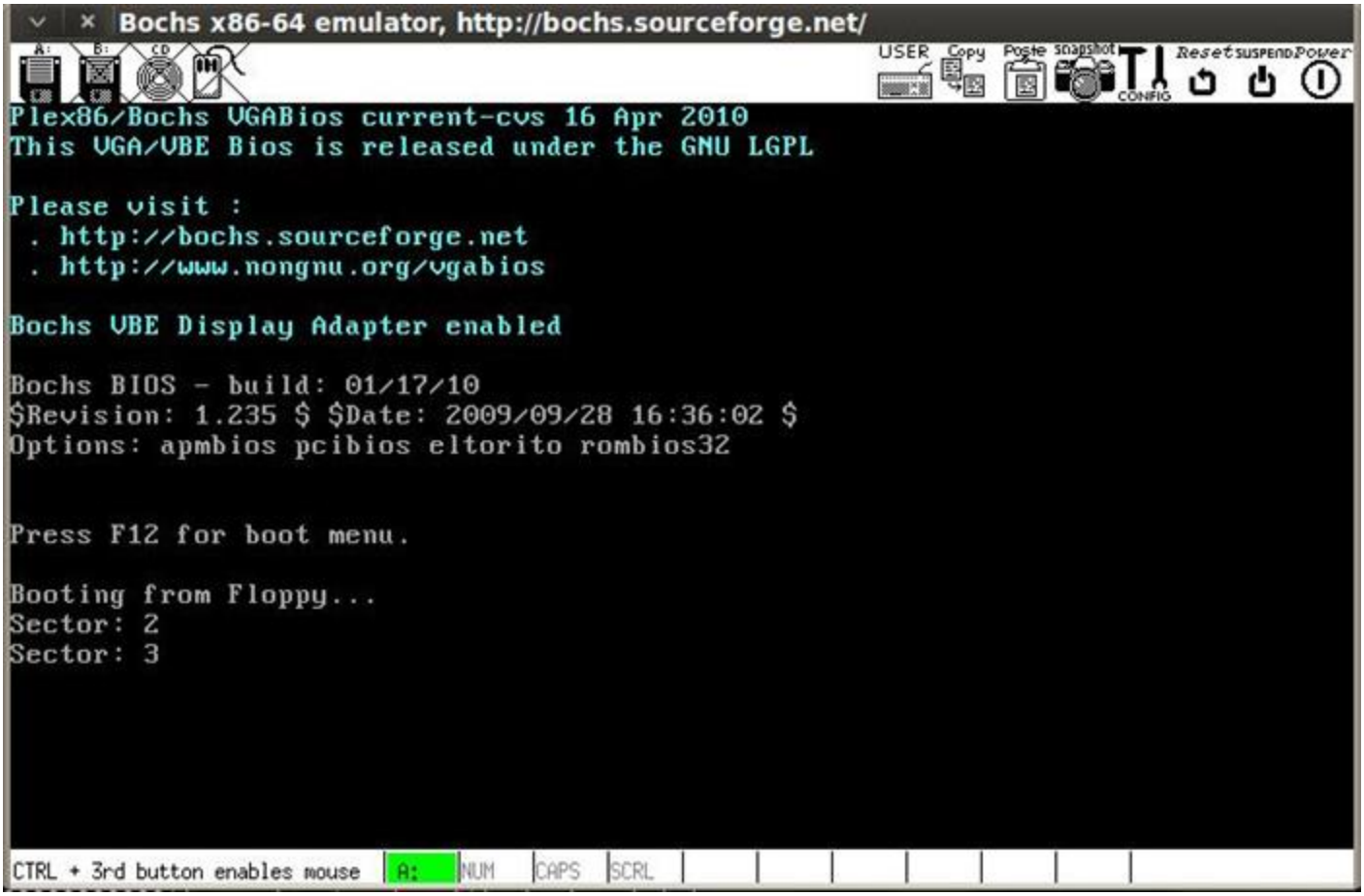
```
     .byte 0x55                #append first part of the boot signature
     .byte 0xAA                #append last part of the boot signature
_sector2:                      #second sector of the floppy disk
     .asciz "Sector: 2\n\r"    #write data to the begining of the sector
     . = _sector2 + 512        #move to the end of the second sector
_sector3:                      #third sector of the floppy disk
     .asciz "Sector: 3\n\r"    #write data to the begining of the sector
     . = _sector3 + 512        #move to the end of the third sector
```

现在编译代码如下

- 作为 test.S -o test.o
- ld -Ttext=0x0000 --oformat=binary test.o -o test.bin
- dd if=test.bin of=floppy.img

如果您在十六进制编辑器中打开*test.bin*，您会发现我在扇区 2 和 3 中嵌入了一个标签。我在下面的快照中突出显示了它们。

如果您通过在命令提示符下键入 bochs 来运行该程序，您将看到如下所示。



上述程序的功能是什么?

在上面的程序中，我们定义了宏和函数来读取和显示嵌入在每个扇区中的字符串的内容。

让我向您介绍每个宏和功能。

自动售货机 复制代码

```
#macro to print null terminated string
#this macro calls function PrintString
.macro mPrintString str
  leaw \str, %si
  call PrintString
.endm
```

这个宏被定义为接受一个字符串作为参数，并在内部调用另一个函数`PrintString`，该函数 执行在屏幕上逐个字符显示的工作。

自动售货机 复制代码

```
#function to print null terminated string
PrintString:
  lodsb
  orb  %al  , %al
  jz   PrintStringOut
  movb $0x0e, %ah
  int  $0x10
  jmp  PrintString
PrintStringOut:
Ret
```

这是宏调用的函数，`mPrintString`用于在屏幕上显示空终止字符串的每个字节。

MSIL 复制代码

```
#macro to read a sector from a floppy disk
 #and load it at extended segment
 .macro mReadSectorFromFloppy num
      movb $0x02, %ah     #read disk function
      movb $0x01, %al     #total sectors to read
      movb $0x00, %ch     #select cylinder zero
      movb $0x00, %dh     #select head zero
      movb \num, %cl      #start reading from this sector
      movb $0x00, %dl     #drive number
      int  $0x13          #interrupt cpu to get this job done now
      jc   _failure       #if fails then throw error
      cmpb $0x01, %al     #if total sectors read != 1
      jne  _failure       #then throw error
 .endm
```

这个宏 mReadSectorFromFloppy 将一个扇区读入扩展段并放置在那里以供进一步处理。这个宏需要一个扇区号作为参数来读取。

自动售货机 复制代码

```
#display the string that we have inserted as the
#identifier of the sector
DisplayData:
DisplayDataIn:
  movb %es:(%bx), %al
  orb  %al      , %al
  jz   DisplayDataOut
  movb $0x0e    , %ah
  int  $0x10
  incw %bx
  jmp  DisplayDataIn
DisplayDataOut:
Ret
```

此函数显示从开始到遇到空终止字符的数据的每个字节。

自动售货机 复制代码

```
_boot:
     movw  $0x07c0, %ax       #initialize the data segment
     movw  %ax    , %ds       #to 0x7c00 location
     movw  $0x9000, %ax       #set ax = 0x9000
     movw  %ax    , %es       #set es = 0x9000 = ax
     xorw  %bx    , %bx       #set bx = 0
```

这是执行的主要引导代码。在开始打印磁盘内容之前，我们将数据段设置为 0x7c00，并将扩展段设置为 0x9000。

设置扩展段的目的是什么?

首先，我们将一个扇区读入程序存储器的 0x9000 处，然后开始显示该扇区的内容。

这就是我们将扩展段设置为 0x9000 的原因。

自动售货机 复制代码

```
mReadSectorFromFloppy $2 #read a sector from floppy disk
call DisplayData         #display the label of the sector
mReadSectorFromFloppy $3 #read 3rd sector from floppy disk
call DisplayData         #display the label of the sector
```

我们调用宏读取扇区2，然后显示其内容，然后再次调用宏读取扇区3，然后显示其内容。

自动售货机 复制代码

```
_freeze:                      #infinite loop
jmp _freeze              #
```

显示扇区内容后，我们进入一个无限循环挂起我们的程序。

自动售货机 复制代码

```
_failure:                       #
mPrintString msgFail        #write error message and then
jmp _freeze                 #jump to the freezing point
```

我们将此部分定义为在出现任何错误情况时跳转到此标签，然后再次挂起程序。

自动售货机 复制代码

```
. = _start + 510            #mov to 510th byte from 0 pos
.byte 0x55                  #append first part of the boot signature
.byte 0xAA                  #append last part of the boot signature
```

我们移动到扇区的第510个字节，然后附加引导签名，这是将软盘识别为可引导设备所必需的，否则系统将错误视为无效磁盘。

自动售货机 复制代码

```
_sector2:                       #second sector of the floppy disk
     .asciz "Sector: 2\n\r"     #write data to the begining of the sector
     . = _sector2 + 512         #move to the end of the second sector
_sector3:                       #third sector of the floppy disk
     .asciz "Sector: 3\n\r"     #write data to the begining of the sector
     . = _sector3 + 512         #move to the end of the third sector
```

上述步骤执行在扇区 2 和 3 的开头附加一个字符串。

这就是本文的全部内容:)

玩得开心，尝试探索在实模式下读取软盘并将该功能嵌入到您的引导加载程序中。

在以下文章中，我将尝试简要介绍文件系统及其重要性。我们还将编写一个最小的引导加载程序来解析 fat12 格式的软盘，如何对其进行读写以及编写第二阶段引导加载程序及其重要性。

暂时再见：)

执照

分享

关于作者

阿什奇兰·巴特
软件开发人员
美国
手表该会员

Ashakiran 来自印度海得拉巴，目前在美国担任软件工程师。他是一名业余程序员，喜欢编写代码。

评论和讨论

添加评论或问题 | 电子邮件提醒 | Search Comments

第一 页上一页 下一页

如何使我们的引导加载程序与连接到我们计算机的网络接口卡和其他类型的硬件和驱动程序交互?
Member 14509631 23-Jun-19 19:37

使用 ASM 引导加载程序
c0derM 31-Oct-14 13:50

回复：使用 ASM 引导加载程序
AshakiranBhatter 31-Oct-14 22:18

示例 1 的错误
mnjrupp 20-Dec-13 2:39

回复：示例 1 的错误
AshakiranBhatter 24-Dec-13 21:23

回复：示例 1 的错误
mnjrupp 24-Dec-13 23:07

回复：示例 1 的错误
mnjrupp 29-Dec-13 13:40

回复：示例 1 的错误
AshakiranBhatter 29-Dec-13 13:50

优秀!
R. Hoffmann 4-Nov-13 17:15

好文章
BillW33 30-Oct-13 21:07

等不及要读下一篇了
Nicolas Dorier 28-Oct-13 20:06

优秀
Sergio Andrés Gutiérrez Rojas 28-Oct-13 3:22

关于编程工具

Super Flanker **17-Oct-13 6:47**

Re: 关于编程工具
AshakiranBhatter 17-Oct-13 13:45

我的4票
M.Ebrahimi.Ahouei **15-Oct-13 13:52**

刷新 1

一般 新闻 建议 问题 错误 答案 笑话 赞美 咆哮 管理员

使用Ctrl+Left/Right 切换消息，Ctrl+Up/Down 切换主题，Ctrl+Shift+Left/Right 切换页面。

永久链接
广告
隐私
Cookie
使用条款

布局：固定 | 体液