

# 计算机组成原理课程设计

## Bochs 虚拟机——测试与分析

班级：2016211310

组长（学号）：刘佳玮 2016211396

组员（学号）：陈博 2016211503 张绍磊 2016211392 刘含宇 2016211390

张明睿 2016211364 关磊 2016211400

日期：2018.6

## 目录

一、初识 Bochs 虚拟机 .....	4
1.1 Bochs 虚拟机简介 .....	4
1.1.1 Bochs 综述 .....	4
1.1.2 Bochs 功能描述 .....	5
1.2 下载与安装步骤 .....	6
二、使用 Bochs 虚拟机 .....	9
2.1 初步使用——运行内置的 DLX Linux Demo.....	9
2.1.1 DLX 家族简介.....	9
2.1.2 Bochs 虚拟机的启动.....	10
2.2 进一步使用——自定义操作系统的运行&调试.....	11
2.2.1 编写最简单的操作系统（.asm 文件） .....	11
2.2.2 使用 NASM 汇编器（.asm->.bin） .....	12
2.2.3 使用 dd 命令创建软盘映像并将引导扇区写入软盘（.bin->.img） ..	12
2.2.4 编写配置文件、加载并运行.....	14
2.2.5 使用 bochsdbg 进行调试.....	16
2.2.6 测试 Bochs 的防御机制.....	16
三、各类虚拟机比较.....	20
3.1 Bochs 虚拟机 VS Zero 虚拟机.....	20
3.1.1 上层设计与用户界面比较.....	20
3.1.2 开发环境与实现方法比较.....	22
3.1.3 总结与展望.....	23

3.2 Bochs 虚拟机 VS Chip 8 虚拟机.....	25
3.2.1 功能比较 .....	25
3.2.2 实现比较 .....	25
3.2.3 总结与展望 .....	30
3.3 Bochs 虚拟机 VS Prim 虚拟机.....	31
3.3.1 硬件框架比较 .....	31
3.3.2 软件功能比较 .....	32
3.3.3 具体实现比较 .....	34
3.3.4 总结与展望 .....	36
3.4 Bochs 虚拟机 VS 星河一号虚拟机 .....	38
3.4.1 差异比较 .....	38
3.4.2 总结与展望 .....	41
3.5 Bochs 虚拟机 VS jfvm 虚拟机.....	42
3.5.1 硬件功能比较 .....	42
3.5.2 软件功能比较 .....	43
3.5.3 总结与展望 .....	48
3.6 Bochs 虚拟机 VS DIYVM 虚拟机.....	49
3.6.1 差异比较 .....	49
3.6.2 总结与展望 .....	52
参考资料.....	54

## 一、初识 Bochs 虚拟机

### 1.1 Bochs 虚拟机简介

#### 1.1.1 Bochs 综述

Bochs<sup>1</sup>是一种使用 C++ 编写的 Intel x86 电脑模拟器（仿真器），能够以不同模式编译，仿真 386、486、Pentium、Pentium III、Pentium 4 乃至 x86-64 CPU，同时它还包含了对 MMX、SSE x 以及 3DNow!指令集的支持。

Bochs 解释从开机到重启的每一条指令，并且支持所有的标准 PC 外设的驱动模型：键盘、鼠标、VGA 显卡/显示器、磁盘、时钟芯片、网卡，这与 VMware 等虚拟机有很大的区别，在这个仿真环境中的程序会认为其运行在一个真实的计算机上，很多软件不用修改即在 Bochs 中运行，包括大部分流行的 x86 操作系统：Windows 95/98 /NT/2000/XP 和 Vista，以及所有的 Linux，所有的 BSD 等等。因此，Bochs **十分适合开发操作系统**。<sup>2</sup>借助自带的调试器 bochsdbg，调试操作系统也变得很简单。

Bochs 可以运行在很多主流平台上，但无论是何种宿主平台，Bochs 仿真的平台都是 x86。也就是说，Bochs 并不依赖于宿主机器的指令，这即是 Bochs 与很多其他 x86 虚拟软件（如 plex86, Vmware 等）的主要不同之处。Bochs 是为数不多的几个能在非 x86 平台运行 x86 软件的选择，包括 Solaris(Sparc)、GNU/Linux (PowerPC/Alpha)、MacOS(PowerPC)、IRIX(MIPS)、BeOS(PowerPC)、Digital Unix(Alpha)、以及 AIX(PowerPC)等。 [1]

<sup>1</sup> Bochs 由 Kevin Lawton 编写，现由 Source Forge 的 Bochs 项目组维护，是遵循 GNU LGPL 的开源软件。

<sup>2</sup> 鉴于 Bochs 对操作系统设计的巨大作用，《操作系统：设计与实现》所带的光盘中就有 Bochs 软件。

综上所述，Bochs 特点可归纳如下：

1. 开源，开发文档丰富。
2. 高移植性，支持多种平台，不依赖主机 ISA，只模拟 x86。
3. 可使用不同模式编译，仿真多种类型的 CPU。
4. 完全靠软件模拟整个 PC，方便调试，十分适合操作系统开发。
5. 轻量级，用户界面简洁。

### 1.1.2 Bochs 功能描述

Bochs 的主要功能如下图所示 [1]：

功能	描述
配置脚本	Bochs使用GNU autoconf配置Makefiles以及头文件。Bochs使用 Autoconf完成多个平台的编译
386,486,Pentium 仿真	Bochs根据配置可以仿真Intel CPU的多个种类。有些属于Pentium 还不支持之类型。比如 MTRRs和SMM(System Management Mode)
P6及其后续CPU 仿真	Bochs根据配置可以仿真包括MMX,SEE支持在内的所有的P6家族的CPU
Pentium4仿真	Bochs根据配置可以仿真所有Intel(CMPXCHG16B指令)或者 AMD(RDTSCP指令)的x86-64CPU
命令行调试器	包括停止执行,检查寄存器和内存内容,设置断点在内的功能强大的调试器。但是调试器缺少对 x86-64的支持并且几乎不能调试x86-64程序
浮点	使用基于SoftFloat浮点仿真库的软件浮点引擎
加强版BIOS	实现了EiTorito,EDD v3.0,基本APM功能,PCIBIOS功能以及PCI中断路径表。最新版Bochs BIOS支持32位ACPI,SMM和SMP
VGA	VGA彩色图形窗口仿真
VBE(VESA)	当前支持的最高分辨率为1024× 768 ×32bpp。你必须使用启用了VBE的基于LGPL的VGABIOS来编译Bochs。
Cirrus Logic 显卡	支持Cirrus Logic CL-GD543 ISA和Cirrus Logic CL-GD5446 PCI显卡。
软盘	在所有平台都支持虚拟软盘。2.88M 3.5",1.44M 3.5",1.2M 5.25",750K 3.5"和360K 5.25"。在Unix和Windows9x/NT/2000/XP 上可以访问物理软盘

功能	描述
多个ATA通道	最多可以模拟4个通道。最多可以安装8个虚拟设备。也就是说你可以有8个硬盘或者7个硬盘,1个CD-ROM或者4个硬盘,4个CDROM 或者7个CD-ROM,1个硬盘,等等。
硬盘	通过映像文件来模拟ATA-6/IDE硬盘。某些平台支持直接物理硬盘访问,但是出于安全因素的考虑,并不推荐使用这种方式。最大可以支持127GB,任何平台下都支持大文件访问。
CD-ROM	模拟ATAPI-4/IDE CD-ROM。CD-ROMs可以读取任何平台下的ISO文件。在Windows(9x/ME/NT/2000/XP),Linux,SunOS,FreeBSD,NetBSD,OpenBSD,Amiga/MorphOS,MacOSX 和 BeOS 上,Bochs可以直接访问物理光驱。从1.4版本,Bochs开始支持从可引导光盘或得 ISO文件启动。
键盘	模拟串行,PS/2,或USB的3键鼠标,并提供附加键支持
Sound Blaster	模拟Sound Blaster 16位声卡(ISA,非即插即用)。在 Windows,Linux,FreeBSD,MacOS 9,MacOS X 平台上,声卡可以输出到host电脑的声音系统中。
网卡	模拟NE2000兼容网卡。在Windows NT/2000,Linux,FreeBSD,和 NetBSD上,Bochs从操作系统中接受数据包并往操作系统发送数据包,因此guest操作系统即可与物理上的网络通信。但是,在某些平台上,guest操作系统可以与网络中任何计算机通信,但是却 不可以与host主机通信。启用TAP或得TUN/TAP接口的Windows 不会出现这种问题,guest操作系统一般情况通过配置之后可以连接到Internet。如果运行的是 Mac OSX,你必须从以下网址下载TUN 驱动: <a href="http://chrisp.de/en/projects/tunnel.html">http://chrisp.de/en/projects/tunnel.html</a>

功能	描述
并口	并口模拟由Volker Ruppert在Boch1.3中添加, guest OS发送到并口的数据可以存储到一个文件中或者直接发送到物理端口 (只支持Unix).
串口	GNU/Linux, FreeBSD, NetBSD, OpenBSD作为host和guest时,Bochs 支持串口(单个16550A UART 模拟).在其他平台上,虽然也可以模拟串口,但是还没实现到host平台软件或者硬件连接
游戏端口	模拟一个标准PC游戏端口. Linux或者win32支持joystick
PCI	模拟大部分i440FX芯片组. 支持Host-to-PCI桥(PMC/DBX), PCI到ISA桥以及PCI IDE控制器 (PIIX3).支持5个PCI插槽供PCI卡使用
插件	支持Linux, Mac OS X, Solaris, 以及Cygwin
16/32位寻址	16位/32位操作数, 堆栈, 地址
V8086/分页	支持可选的v8086模式扩展(VME)以及分页
PIC	支持主编程中断控制器
CMOS功能	支持CMOS功能
模拟对称多处理器	Bochs经过配置可以支持高达8个处理器. 该功能仍处于实验阶段,但是它已经能够使用SMP成功启动Linux 2.2内核. 需要注意这并不是说Bochs能够比实际的SMP运行得更快.
复制与粘贴	根据host平台, 文本模式的屏幕的文本可以输出到剪切板. 然后通过Bochs模拟击键来粘贴到 guest OS

图 1 Bochs 主要功能

## 1.2 下载与安装步骤

第一步：访问 Source Forge 的 Bochs 主页，下载 Bochs 虚拟机（目前最新版本是 2017 年 4 月发布的 2.6.9 版本）。<sup>3</sup>

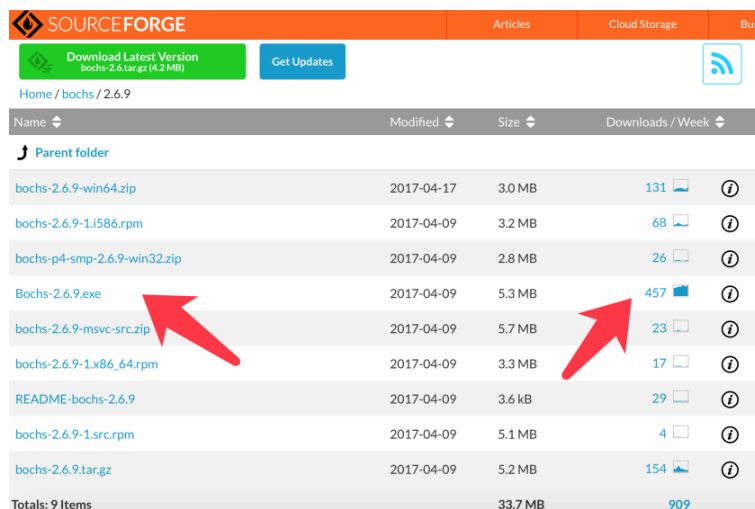


图 2 Bochs 虚拟机下载界面

第二步：安装 Bochs 虚拟机。

<sup>3</sup> Bochs 2.6.9 的下载地址为 <https://sourceforge.net/projects/bochs/files/bochs/2.6.9/>

1. 打开安装包。

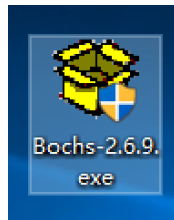


图 3 Bochs 虚拟机图标

2. 同意安装协议。

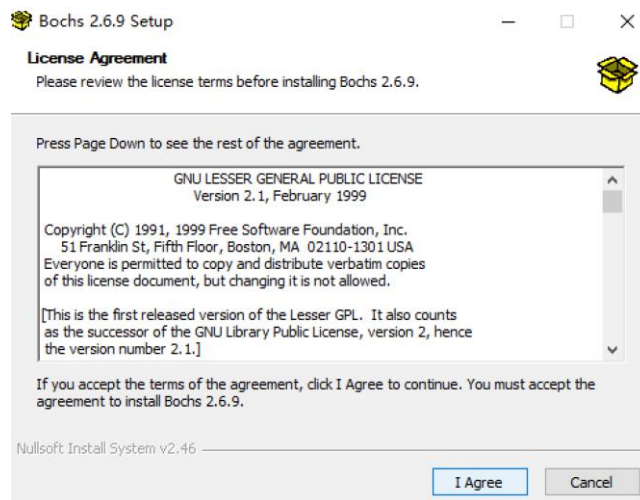


图 4 Bochs 安装协议

3. 选择使用权限（选择二者任一）。

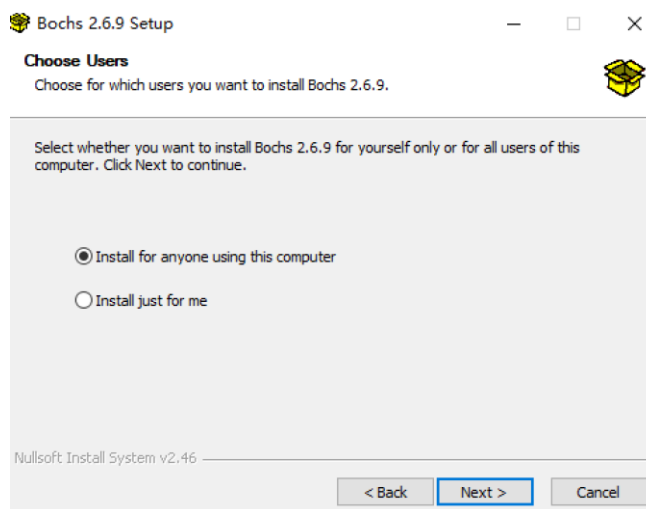


图 5 Bochs 使用权限设置

#### 4. 安装完整版（含 DLX Linux Demo）。

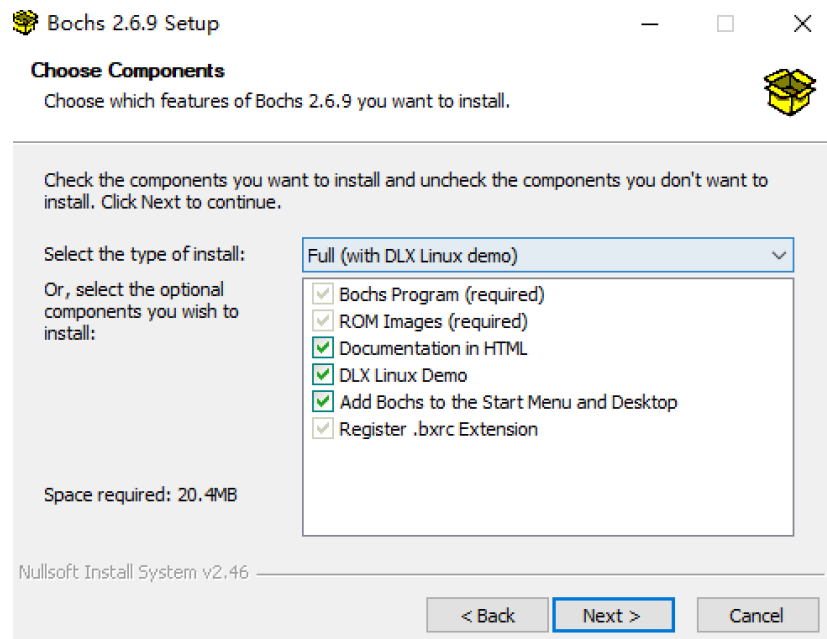


图 6 Bochs 安装选项

#### 5. 设置安装路径。

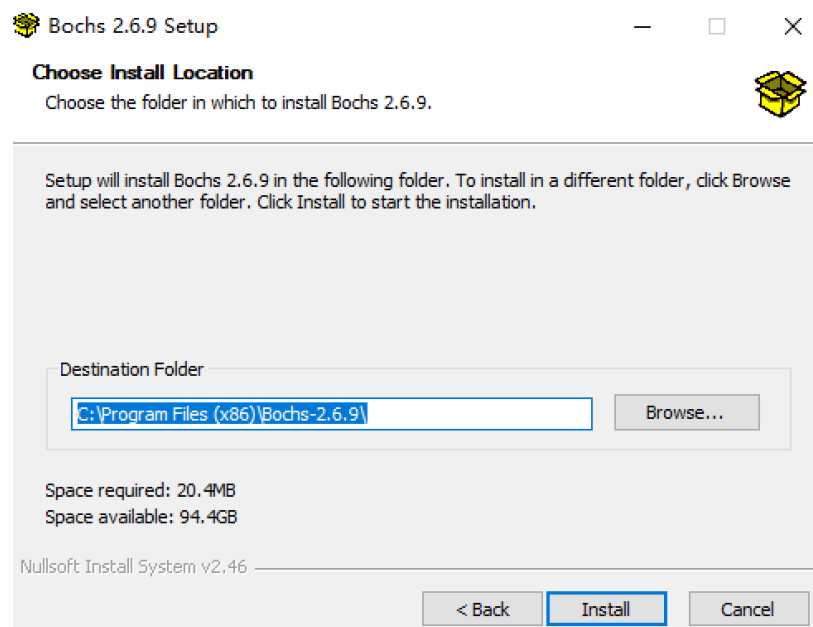


图 7 Bochs 安装路径

#### 6. 安装成功。



## 二、使用 Bochs 虚拟机

### 2.1 初步使用——运行内置的 DLX Linux Demo

Bochs 自带了一个 DLX Linux Demo，可用于初学者熟悉 Bochs 环境。本小组也首先运行了该程序，并体验到了 Bochs 的强大之处。

#### 2.1.1 DLX 家族简介

DLX 处理器是 2017 年图灵奖得主 Hennessy & Patterson 的巨著《计算机体系结构：量化研究方法》中介绍流水线处理器的一个例子，DLX ISA 是工作在 DLX 处理器上的指令系统，遵循“硬件设计的四大原则”，具有 Load/Store 结构、注重流水效率、采用定长指令等特点。 [2]

DLX OS 由马里兰州巴尔的摩县大学和加利福尼亚大学圣克鲁斯分校编写并用于教学，基于 Hennessy & Patterson 描述的 DLX 指令集和计算机的操作系统，可以在 UNIX、Linux、Solaris 环境的模拟器（如 Bochs）中以用户程序的形式运行，此时编写的所有代码与在裸露硬件上运行 DLX OS 完全相同。事实上，在行业中，系统代码在硬件上运行前，都会先在模拟环境中测试。 [3]

第一次使用 Bochs 建议安装 DLX Linux Demo，这是在 Intel 体系结构上运行的全功能 Linux 操作系统（演示版），设计简单，可以用最少的配置测试 Bochs 的功能。虽然有些老旧、本身并不完全有用，但作为初学者熟悉 Bochs 环境，或作为第二个访客系统诊断其他操作系统（如 MINIX）可能因源码被黑客攻击产生的网络问题十分有用。 [4]

## 2.1.2 Bochs 虚拟机的启动

第一步：双击 Linux Demo in Bochs 图标，或进入安装目录，点击 run.bat，即可进入演示版。

名称	修改日期	类型	大小
bochsout.txt	2018/6/9 18:30	文本文档	15 KB
bochsrc.bxrc	2017/4/9 15:33	Bochs 2.6.9 Conf...	2 KB
hd10meg.img	2018/6/9 18:30	IMG 文件	10,404 KB
readme.txt	2017/4/9 15:33	文本文档	1 KB
run.bat	2018/6/9 18:14	Windows 批处理...	1 KB
testform.txt	2017/4/9 15:33	文本文档	5 KB

图 8 run.bat 安装位置

第二步：直接输入 root 并回车即可登录。

第三步：登录后，输入 Linux 命令行语句（如 pwd 显示工作目录、ls -al 显示当前目录下所有文件/文件夹等），即可体验 Bochs 虚拟机运行 Linux 操作系统。

第四步：按 Power 键退出。

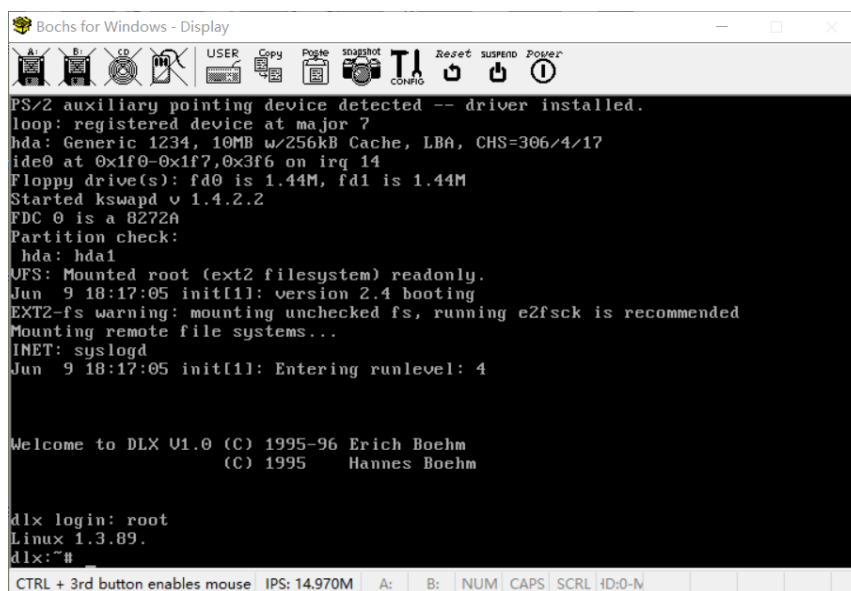


图 9 DLX Linux demo 启动界面

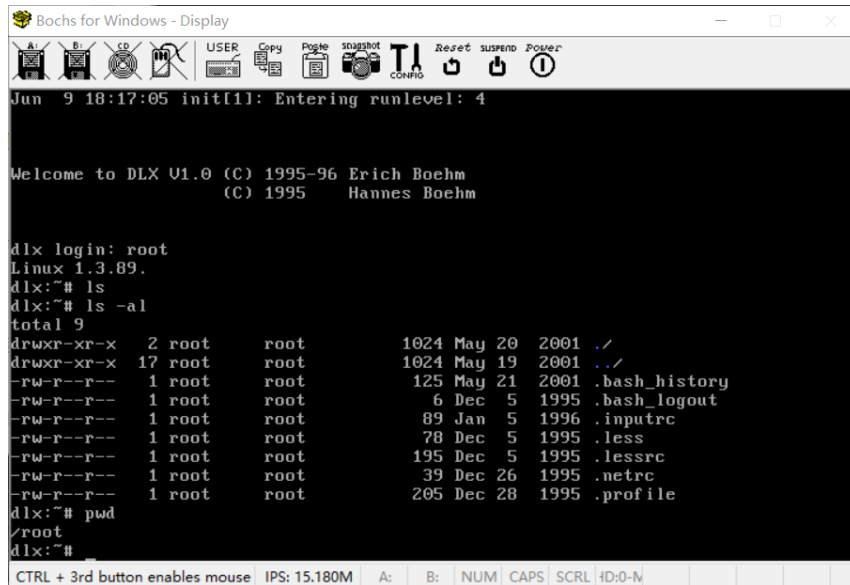


图 10 DLX Linux demo 命令测试

## 2.2 进一步使用——自定义操作系统的运行&调试

### 2.2.1 编写最简单的操作系统（.asm 文件）

在记事本中编写 NASM 风格汇编程序，实现输出“Hello”的功能，将后缀改为.asm。

```

mov ah, 0x0e ; tty mode
mov al, 'H'
int 0x10
mov al, 'e'
int 0x10
mov al, 'l'
int 0x10
int 0x10 ; 'l' is still on al
mov al, 'o'
int 0x10
jmp $ ; jump to current address = infinite loop
; padding and magic number
times 510 - ($-$) db 0
dw 0xaa55

```

图 11 Hello 汇编源码

### 2.2.2 使用 NASM 汇编器 (.asm->.bin)

NASM (The Netwide Assembler) 是一个为可移植性与模块化而设计的一个开源 80x86 汇编器<sup>4</sup>，它将汇编文件 (.s) 转换成二进制目标文件 (.o)。与 GCC 不同<sup>5</sup>，NASM 采用与 Intel 相似但更简洁的语法。

在 NASM 所在目录打开 cmd，键入命令：

```
nasm -f bin -o 二进制文件名.bin 汇编文件名.asm
```

则会在目录下看到新生成的 .bin 文件。

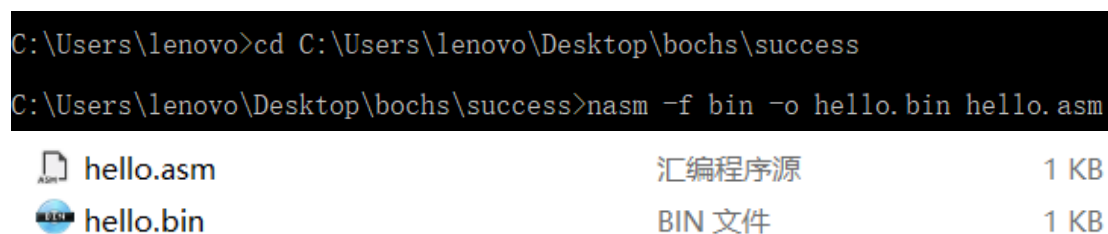


图 12 NASM 使用方法

### 2.2.3 使用 dd 命令创建软盘映像并将引导扇区写入软盘 (.bin->.img)

dd 本为 Linux 系统中的命令行语句，用于复制文件并对原文件的内容进行转换和格式化处理。值得一提的是，dd 本身可以创建并写入软盘，故不需要通过 bximage 单独创建空白软盘。

在 Windows 系统中使用 dd 命令，本小组总结了 3 种方法：

#### 1. 使用 WSL (Windows Subsystem for Linux)

WSL 是一个为在 Windows 10 上能够原生运行 Linux 二进制可执行文件 (ELF 格式) 的兼容层。WSL 提供了一个微软开发的 Linux 兼容内核接口 (不包含 Linux

<sup>4</sup> NASM 最新版 (2.13.3) 下载地址: <https://www.nasm.us/pub/nasm/releasebuilds/2.13.03/win64/>

<sup>5</sup> GCC (GNU Compiler Collection) 提供“预处理-编译-汇编-链接”全功能，汇编部分采用 AT&T 语法，通过 gcc -c 生成二进制目标文件。

代码), 来自 Ubuntu 的用户模式二进制文件在其上运行。

## 2. 使用 Cygwin

Cygwin 是一个在 windows 平台上运行的类 UNIX 模拟环境, 是把 Linux 程序交叉编译成 Win32 程序, 程序 link 到 CygwinX.dll 模拟常用的 POSIX 要求的内核 API; 而 WSL 是直接运行原生 Linux 程序, 通过 NT 内核上架设的兼容层模拟 Linux 内核的行为, 将程序的内核调用转为 NT 调用。二者原理不同, 但都能执行 Linux 命令。

## 3. 使用单独的 dd.exe<sup>6</sup>, 可直接实现该功能。

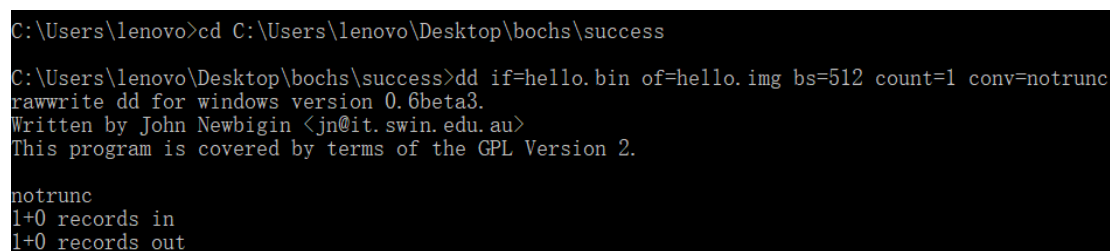
注: Windows 系统下, 也可以用 Rawrite 实现 dd 的功能, 效果相同。命令为:

```
rewrite2.exe -f boot.bin -d A
```

以方法三为例, 在命令行中键入:

```
dd if=二进制文件名.bin of=镜像文件名.img bs=512 count=1 conv=notrunc
```

则会在目录下看到新生成的.img 文件。



```
C:\Users\lenovo>cd C:\Users\lenovo\Desktop\bochs\success
C:\Users\lenovo\Desktop\bochs\success>dd if=hello.bin of=hello.img bs=512 count=1 conv=notrunc
rawwrite dd for windows version 0.6beta3.
Written by John Newbigin <jn@it.swin.edu.au>
This program is covered by terms of the GPL Version 2.

notrunc
1+0 records in
1+0 records out
```

图 13 方法三写入软盘

方法三安装简单, 操作方便, 是本小组推荐使用的方式。此外, 小组成员分

<sup>6</sup> 下载地址为 <http://www.chrysocome.net/downloads/766c72a314affe573146457be98410bb/dd-0.6beta3.zip>

别测试了方法一和方法二，效果如下：

```
coherence@DESKTOP-ISLPV6:/mnt/c/Users/Think$ cd Desktop
coherence@DESKTOP-ISLPV6:/mnt/c/Users/Think/Desktop$ dd if=boot.bin of=a.img bs=512 count=1 conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.033652 s, 15.2 kB/s
```

图 14 方法一写入软盘

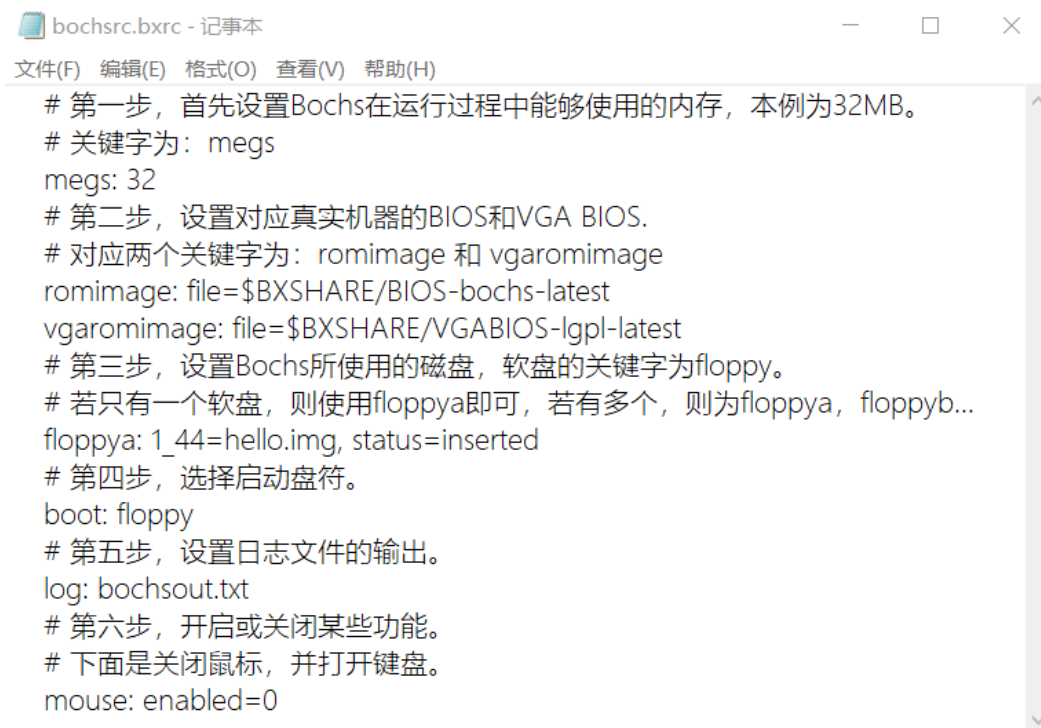
```
lenovo@LAPTOP-IIGAFPGN /cygdrive/c/Users/lenovo/AppData/Local/bin/NASM
$ dd if=boot.bin of=a.img bs=512 count=1 conv=notrunc
记录了1+0 的读入
记录了1+0 的写出
512 bytes copied, 0.00520089 s, 98.4 kB/s
```

图 15 方法二写入软盘

本小组进行了验证，证明三者的效果相同。

## 2.2.4 编写配置文件、加载并运行

编写配置文件，并命名为 bochsrc.bxrc。



```
bochsrc.bxrc - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
# 第一步，首先设置Bochs在运行过程中能够使用的内存，本例为32MB。
# 关键字为：megs
megs: 32
# 第二步，设置对应真实机器的BIOS和VGA BIOS。
# 对应两个关键字为：romimage 和 vgaromimage
romimage: file=$BXSHARE/BIOS-bochs-latest
vgaromimage: file=$BXSHARE/VGABIOS-lgpl-latest
# 第三步，设置Bochs所使用的磁盘，软盘的关键字为floppy。
# 若只有一个软盘，则使用floppya即可，若有多个，则为floppya, floppyb...
floppya: 1_44=hello.img, status=inserted
# 第四步，选择启动盘符。
boot: floppy
# 第五步，设置日志文件的输出。
log: bochsout.txt
# 第六步，开启或关闭某些功能。
# 下面是关闭鼠标，并打开键盘。
mouse: enabled=0
```

图 16 编写配置文件

打开安装好的 bochs，在启动界面点击 load，加载 bochsrc.bxrc 文件，即可

正常运行。

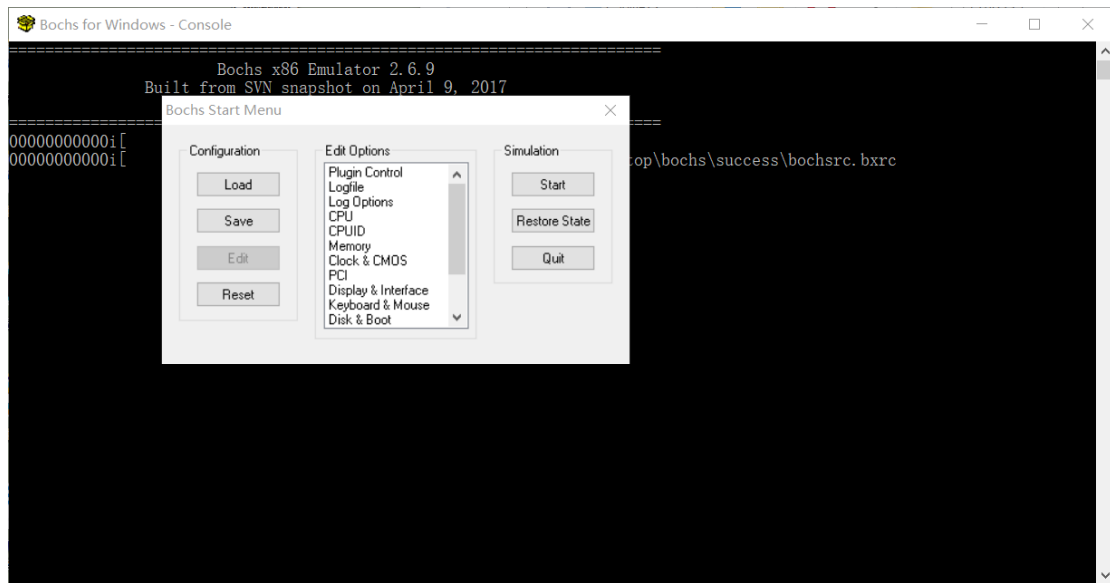


图 17 Bochs 加载界面

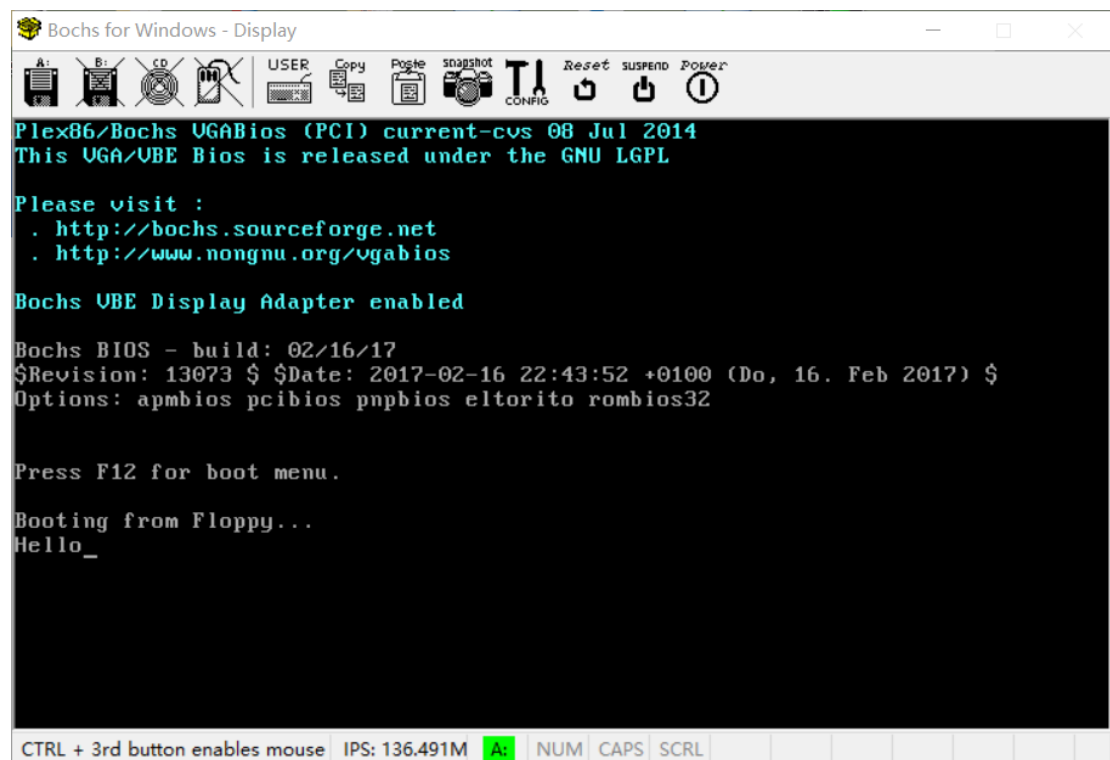


图 18 运行输出 Hello 的“操作系统”

## 2.2.5 使用 bochsdbg 进行调试

Bochs 提供了强大的调试功能，为操作系统的开发提供了极大的方便。

命令	功能描述
<b>c</b> <b>cont</b> <b>continue</b>	继续执行
<b>s [count]</b> <b>step [count]</b>	执行count条指令，默认为1
<b>Ctrl+C</b>	停止执行，返回到命令行提示符
<b>Ctrl+D</b>	如果命令行处于空行状态，退出
<b>q</b> <b>quit</b> <b>exit</b>	退出调试和执行界面

图 19 bochsdbg 的执行控制语句

我们对使用这些命令进行了测试，结果如下：

```

Bochs x86 Emulator 2.6.9
Built from SVN snapshot on April 9, 2017
Compiled on Apr  9 2017 at 09:49:25
=====
000000000000i[ ] reading configuration from bochsrc.bxrc
000000000000i[ ] reading configuration from C:\Users\lenovo\Desktop\bochs\success\bochsrc.bxrc
000000000000i[ ] installing win32 module as the Bochs GUI
000000000000i[ ] using log file bochsout.txt
Next at t=0
(0) [0x0000ffff0] f000:fff0 (unk. ctxt): jmpf 0xf000:e05b      ; ea5be00f0
<bochs:1> s
Next at t=1
(0) [0x000000fe05b] f000:e05b (unk. ctxt): xor ax, ax          ; 31c0
<bochs:2> step 4
Next at t=5
(0) [0x000000fe063] f000:e063 (unk. ctxt): out 0xd6, al         ; e6d6
<bochs:3> step 100
Next at t=105
(0) [0x000000fe0c4] f000:e0c4 (unk. ctxt): rep stow word ptr es:[di], ax ; f3ab
<bochs:4> c

```

图 20 调试功能测试

## 2.2.6 测试 Bochs 的防御机制

Bochs 提供了强大的调试功能，为操作系统的开发提供了极大的方便。以输出 Hello 的简单程序为例，测试 Bochs 面对非法指令时的防御机制。



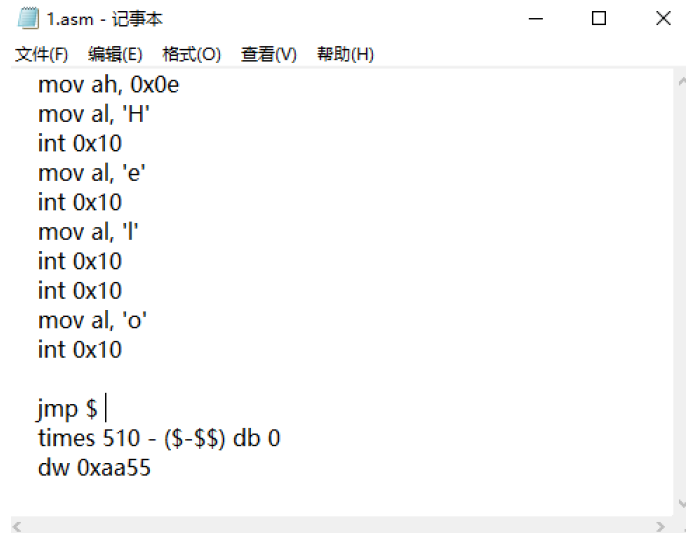


图 21 可正常运行的程序代码

面对错误指令代码：

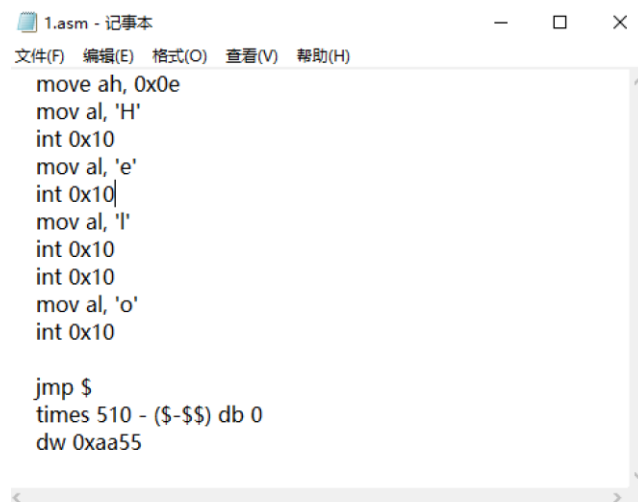


图 22 无法通过语法检查的程序代码

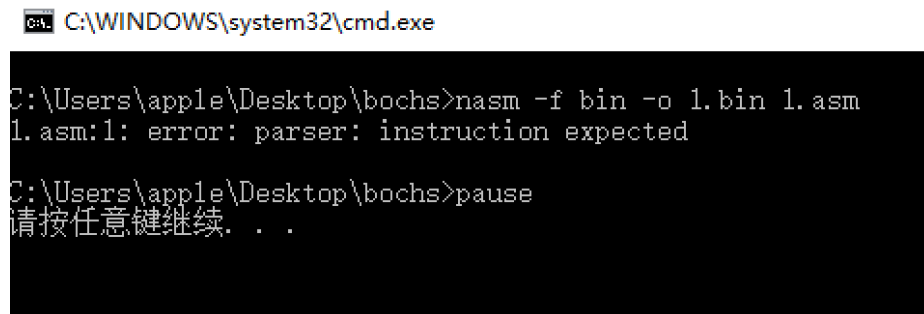


图 23 语法检查结果

结果为：无法通过语法检查。

面对非法地址：

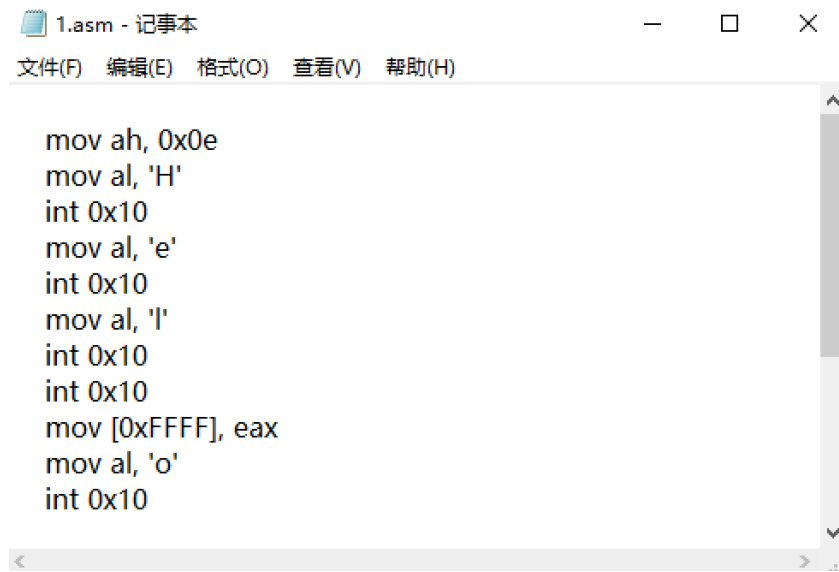


图 24 企图访问非法地址的程序代码

该程序在输出“Hello”的程序中，添加了一句将 eax 中的数据移动到内存 0xFFFF 地址的指令。

结果如下：

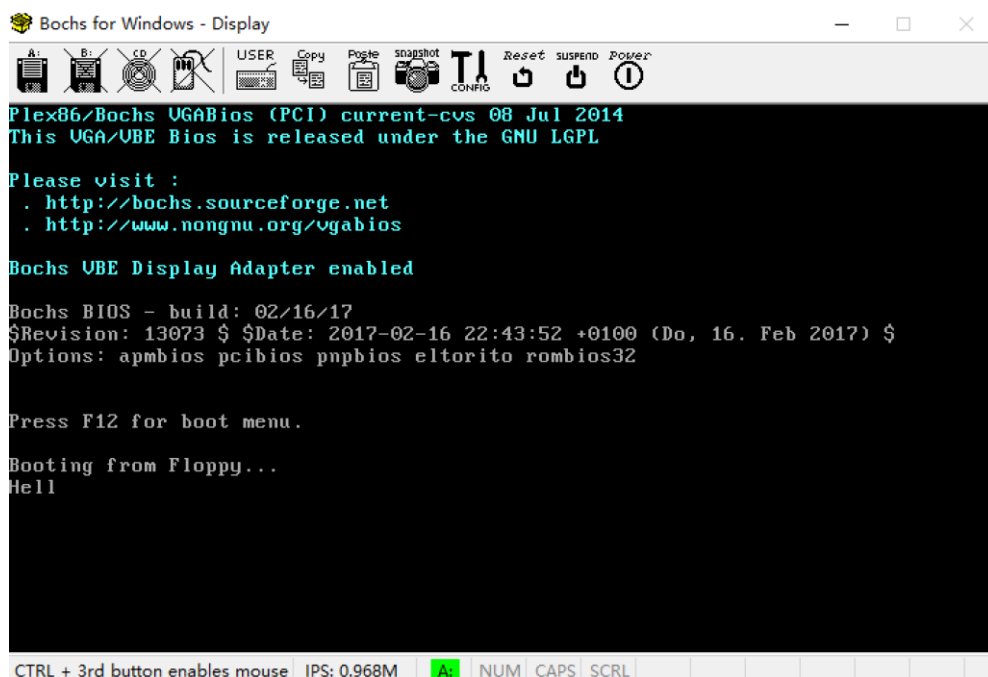


图 25 带有非法指令的程序的运行结果

Bochs 在寻址到非法地址时, 直接停止运行了, 因此无法运行后面输出字符“o”的指令。程序输出至“hell”就不再输出了。

错误分析:

要想测试 Bochs 面对非法指令时的防御机制, 需要提供一组能通过语法检查的汇编程序代码, 但这段代码却又能引起程序错误。

在上面示例的程序中, `mov [0xFFFF], eax` 指令将 `eax` 寄存器中的数值移动到内存 `0xFFFF` 开始的一段内存中。

在 x86 平台上, 数据和地址总线均为 32 位。一个寄存器中的数值为 32 位, 可寻址最大内存编号为 32 位可表示的最大数值 `0xFFFF`。32 位数需要用 4 个字节的寄存器空间或内存空间来存储, 而上述指令理论上将从 `0xFFFF` 地址开始, 将 32 位数移动到 `0xFFFF`、`0x10000`、`0x10001`、`0x10002` 四个内存地址, 但是在 32 位架构下, 后三个内存地址在物理上和逻辑上都是不存在的。或许是由于 Bochs 从硬件层级开始模拟的原因, 或许是由于设计上的漏洞, 这里 Bochs 在面对非法的地址时, 直接停止了运行。因此可以推测, Bochs 在面对一些错误, 如非法内存地址访问时, 采取的是直接停机的防御措施。

### 三、各类虚拟机比较

本小组将 Bochs 虚拟机与自己上半学期做的虚拟机进行了比较，汇总如下：

#### 3.1 Bochs 虚拟机 VS Zero 虚拟机

##### 3.1.1 上层设计与用户界面比较

作为指令集虚拟机，Bochs 虚拟机是一个专门用于 X86 仿真的虚拟机，Zero 虚拟机是一个专门用于 Pep/9 仿真的虚拟机，二者都不依赖宿主机的 ISA，同时都很注重系统级设计，在很多方面有相似之处。

首先，在上层设计上，二者比较如下：

设计点	Bochs 虚拟机	Zero 虚拟机
虚拟机类型	都是指令集虚拟机	
跨平台性	都能够在 Windows、Linux 等多平台运行，Bochs 比 Zero 虚拟机支持的宿主操作系统类型更加丰富	
设计原则	都遵从“计算机分层设计理念”	
体系结构	冯·诺依曼体系结构	
是否开源	是，但 Bochs 设计之初不是开源，而 Zero 虚拟机一开始就是面向开源进行设计的	
内存	支持 4MB 的内存扩展，通过 -enable-4meg-pages 设置	通过按键选择内存拓展规模，最大可拓展至 1MB
CPU	支持 x86 架构的多版本 CPU，如 386、486、Pentium 等	只支持 Pep/9，不向下兼容 Pep/8、Pep/7 等
外设	支持键盘、鼠标、VGA 显卡/显示器、磁盘、时钟芯片、网卡等，十分齐全	支持键盘、鼠标、显示器、磁盘、打印机等，满足简单需求
操作系统	提供了 DLX Linux Demo 可直接使用；同时提供大量软/硬盘映像可供加载，用户需自行写入，可自行修改，灵活性强	提供 Pep 操作系统，目前对其他操作系统不支持，不可由用户定制
调试功能	支持单步执行、设置断点等，调试功能丰富	支持单步执行

亮点	小巧、简单、全功能，界面有关机、重启键等，调试功能强大，尤其适合操作系统开发	界面友好、集成源码编辑、显示、模式切换等功能于一体，适合新手学习计算机系统
----	--	---------------------------------------

再来比较二者的用户界面：

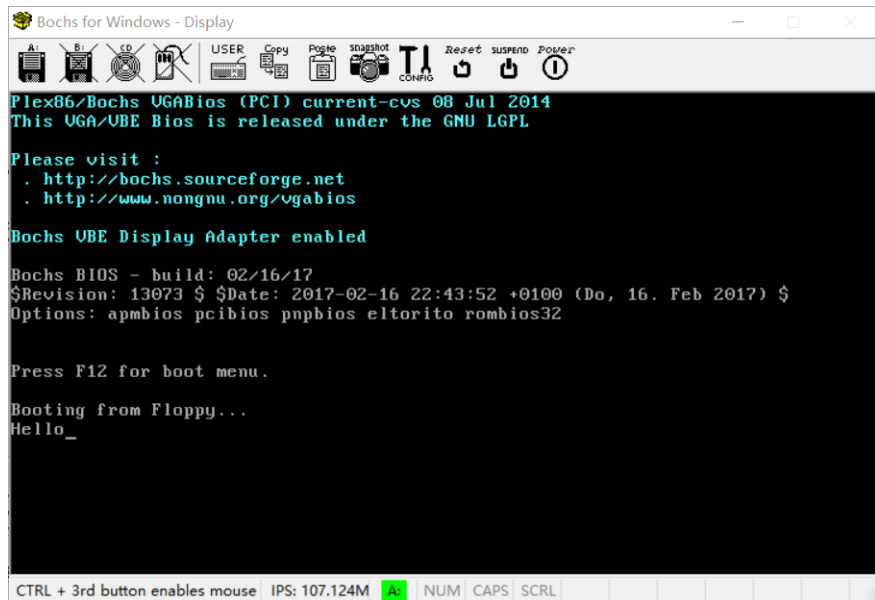


图 26 Bochs 虚拟机

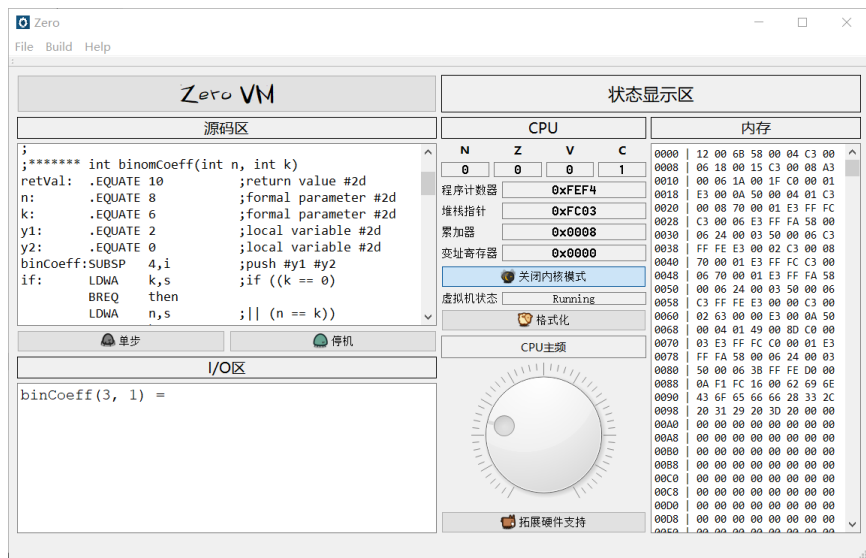


图 27 Zero 虚拟机

可以发现，虽然二者都是“开源的、系统级的指令集虚拟机”，但二者在功能上有一定的差异，在用户界面上更是完全不同。其原因如下：

首先，**架构和工程量决定了二者仿真程度的不同**。Bochs 虚拟机在 PC 机的整体仿真程度上明显高于 Zero 虚拟机，一方面是由于 x86 架构本身就比较复杂，另一方面是由于 Bochs 有团队进行开发，且经历了长时间的开源，已经十分成熟，而 Zero 虚拟机目前是 1.0 版本，无论是代码量还是设计难度上，都远低于 Bochs 虚拟机。

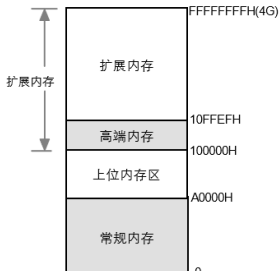
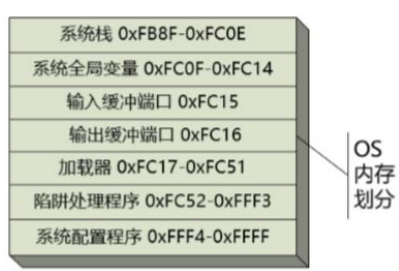
其次，**产品定位决定了二者用户界面的不同**。Zero 虚拟机的定位是“对用户友好、易上手的计算机系统教学虚拟机”，其目标用户是学生，而且是全新打造的虚拟机，因此在用户界面上将计算机的功能呈现地十分直观，包括“一键切换模式”、“磁盘扩容”、“CPU 主频调节”等功能，更多的是为了演示和学习的方便；而 Bochs 虚拟机不仅用于教学，而且用于操作系统的开发和仿真 Windows 系统（如在 Linux 系统、Android 系统中），这就要求更强的隔离性、逼真性，因为目标用户可能不了解计算机，所以将全部功能放置在用户界面上既没有必要、也不安全，而电源键等功能反而更有意义。

### 3.1.2 开发环境与实现方法比较

作为使用 C++ 开发的指令集虚拟机，Bochs 和 Zero 虚拟机在硬件虚拟和 ISA 仿真上都有很多可比之处。

设计点	Bochs	Zero
实现原则	都遵从“硬件设计四大原则”	
开发语言	C++（Zero 虚拟机借助 Qt 进行开发）	
硬件仿真	都采用面向对象的方式，Zero 虚拟机利用“信号与槽”机制	
用户界面开发	调用 Win32 GUI	利用 Qt 控件

除此以外，由于 x86 和 Pep/9 架构自身的区别，Bochs 和 Zero 虚拟机在实现上也有很大的差异：

设计点		Bochs (x86)	Zero (Pep/9)
CPU	工作模式	实模式+保护模式+系统管理模式+虚拟 8086 模式	用户模式+内核模式，实质是两层访问权限
	结构	ALU+控制器+辅助设备 (MMU, FPU, Cache 等)	ALU+控制器
	寄存器	通用寄存器+段寄存器+控制寄存器+状态寄存器+程序计数器 etc	没有通用寄存器，其他基本相同
	ISA	CISC 指令集，指令不定长，十分复杂	类 RISC 指令集，采用 load-store 结构
	加速技术	流水线、乱序执行等	无
存储器	内存编址		

### 3.1.3 总结与展望

Zero 虚拟机和 Bochs 虚拟机都是指令集虚拟机，但无论是定位还是实现方式都有很大的不同，各有特色。

用于初学计算机系统，Zero 虚拟机凭借良好的用户界面、简单的 Pep/9 指

令集以及各种拓展性设计、直观透明的变量显示，比 Bochs 更加对用户友好，也更简单易学，尤其是 Zero 提供了完整的中文文档，这给国内学生提供了很大的便利；更重要的是，Zero 将源码编辑集成到虚拟机的主界面中，并提供了丰富的 Pep/9 汇编指令参考表，对于学习汇编语言也有很大的帮助。

对于开发操作系统，Bochs 有得天独厚的优势，Bochs 解释从开机到重启的每一条指令，并且支持所有的标准 PC 外设的驱动模型，借助 bochsdbg 更方便地进行调试。此外，Bochs 也支持更多的宿主平台，具有极强的兼容性。

鉴于 Bochs 的成熟性，在未来，我们也可以对其进一步挖掘和应用。如尝试在 Android、Linux、Mac OS 等平台安装 Bochs，并借助 Bochs 官网上的开源镜像体验安装其他操作系统等。

在设计 Zero 虚拟机时，我认为可以在 7 个维度进行进一步的挖掘和应用：

1. 是否可以支持更多的 RISC/CISC 指令集，将其打造成强大的汇编学习工具？
2. 是否可以增加更精确的调试功能，将其打造成汇编语言的集成开发环境？
3. 是否可以增加更多的拓展接口，将其打造成更强大的接口仿真工具？
4. 是否可以增加显存、音频设备和应用层，呈现出更丰富的可视化效果？
5. 是否可以改造和丰富 Pep OS，将其打造成操作系统的入门学习工具？
6. 是否可以利用简化助记符和语法糖的思想，进一步降低 Pep/9 的学习门槛？
7. 是否可以添加编译和链接功能，集成高级语言转化为 Pep/9 汇编语言的工具？

通过对 Bochs 的学习，我发现 Bochs 已经成为 x86 强大的学习工具，具备精准的调试功能，支持所有的标准 PC 外设的驱动模型，支持丰富的操作系统，借助 NASM 风格的 x86 汇编提供了简化，与我当时对 Zero 的期待不谋而合，这也充分说明了我当时的展望是有价值的，更是可行的。

作为指令集虚拟机，Bochs 和 Zero 都存在运行效率低的问题，但这不影响二者在适合的领域发挥不可替代的作用。Bochs 借助开源优势，已经得到了全世



界广泛的支持,这也为 Zero 虚拟机未来的发展提供了很好的参考。与 x86 相比, Pep/9 是更适合新手学习的, 结合其独特的定位和设计, 我对未来 Zero 虚拟机的改进和应用充满了期待。

## 3.2 Bochs 虚拟机 VS Chip 8 虚拟机

### 3.2.1 功能比较

Bochs 是一款用 C++ 编写的开源 IA-32(x86)电脑模拟器,它仿真英特尔 x86 CPU、常见的 I/O 设备、和定制的 BIOS, Bochs 使用指令丰富的 CISC 指令集, 作为一款虚拟机, 在 Bochs 仿真环境里能够运行许多操作系统, 比如 Linux、DOS、Windows 95/98/NT/2000/XP 或者 Windows Vista。Chip8 是一款针对游戏的虚拟机, 指令集十分精简, 只有 35 条汇编指令, 都是针对简易游戏开发中常用的操作设计的指令。

Bochs 可以运行操作系统, 实验中我们运行了一个简易的操作系统并且实现了一个 HELLO WORLD 程序, chip8 功能相对单一, 主要是运行游戏程序, 测试时运行了俄罗斯方块等游戏程序。

Bochs 提供了调试功能, 可以在程序运行时对程序进行调试, 输出程序执行时的一些数据。CHIP8 虚拟机在我编写时, 在旁边加上了一个输出区域, 在输出区域可以随时查看 PC, OP, 还有各个寄存器的值。

### 3.2.2 实现比较

#### 编程结构:

Chip8 结构简洁, 输入输出功能都很简单, 所以在编写 chip8 虚拟机时, 我

将 chip8 虚拟机封装成了一个类，这样实现起来更方便，绘制图形界面时也更方便调用。但是在 Bochs 中，要实现一个功能完整的机器，一个类过于臃肿，所以 Bochs 对于每一部分使用一个类来进行模拟。

```
class chip8 {
public:
    chip8();
    ~chip8();

    bool drawFlag;
    bool opcodeFlag;
    unsigned short opcode;           // 当前执行的代码
    unsigned short pc;               // 程序计数器
    unsigned char V[16];            // 寄存器V0-VE，最后一个VF用于储存进位
    void emulateCycle();
    bool loadApplication(const char * filename);
    int run() { // 运行

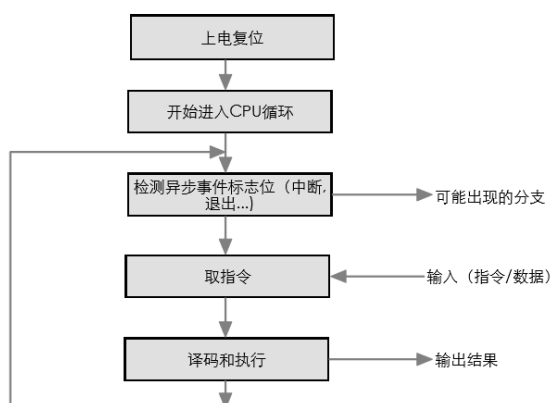
```

图 28 虚拟硬件的 Chip 8 类

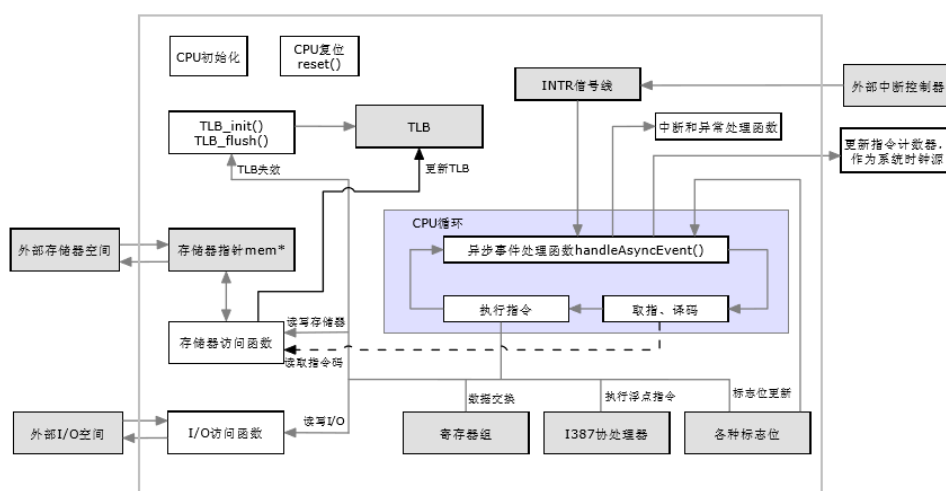
Bochs 总共有 VM 控制台界面类，CPU 模拟，Memory 模拟，I/O device 模拟，四个类，在分类上遵循了冯诺依曼计算机体系结构的基本原则。

程序运行：和实际 CPU 的原理一样，Bochs 虚拟机 CPU 的运行方式为：取指->执行->输出结果。CPU 复位后进入实模式，EIP = 0x0000FFF0；CS 的 cache 内的 Base= 0xFFFF0000，两种相加得到 CS:EIP=0xFFFFFFFF0，该位置是 BIOS ROM 所在位置。用户程序从该处开始运行。进入 CPU 循环后，除了取指和执行工作外，每次还要检测标志位，如果没有复位或退出命令，CPU 循环将会一直进行下去。

我在设计 Chip8 也是按照计算机实际运行时的执行逻辑，开机时，由于 chip8 的解释器存放在 0x000-0x200 之间的内存中，所以开机时直接引导程序存放在 0x200 往后的地址中，开始执行程序时，Chip8 会自动从这里开始执行。



### CPU 模拟:



Bochs 是模拟了计算机 CPU 处理事件的逻辑，处理的事件比较复杂，有很多特殊事件比如浮点指令都会申请其他处理器协助。Chip8 作为一款游戏模拟器，CPU 功能比较少，都是简单的指令，大部分都由 CPU 自己执行完成。少部分设计读入和输出的指令会调用 I/O 区协助完成。

## 通用寄存器：

Bochs 总共定义了如下八类寄存器：

```

// General register set
// eax: accumulator
// ebx: base
// ecx: count
// edx: data
// ebp: base pointer
// esi: source index
// edi: destination index
// esp: stack pointer

```

图 31 Bochs 的 8 类寄存器

编写 Chip8 虚拟机时总共设计了六种寄存器如下：

```

unsigned short opcode;      // 当前执行的代码
unsigned short pc;          // 程序计数器
unsigned char V[16];        // 寄存器V0-VE，最后一个VF用于储存进位

unsigned short I;           // 引导寄存器，用于引导一些整片读入输出的操作
unsigned short sp;          // 栈指针

unsigned short stack[16];   // 栈大小为16

```

图 32 Chip 8 的 6 类寄存器

Bochs 在设计时针对不同的需要设计了可以切换的大端和小端存放模式。

Chip8 主要采用小端模式进行存放。

## CPU 主循环：

```

while (1) //循环执行以下所有的操作
{
    //处理异步事件、中断
    if (BX_CPU_THIS_PTR async_event) { //检查是否有异步事件
        if (handleAsyncEvent()) {
            // If request to return to caller ASAP.
            return;
        }
    }
}

```

图 33 Bochs 主循环

Bochs 的主循环主要进行中断等特殊事件处理，和常规 CPU 周期的运行。

Chip8 的主循环就是模拟一个指令周期，执行一条指令。

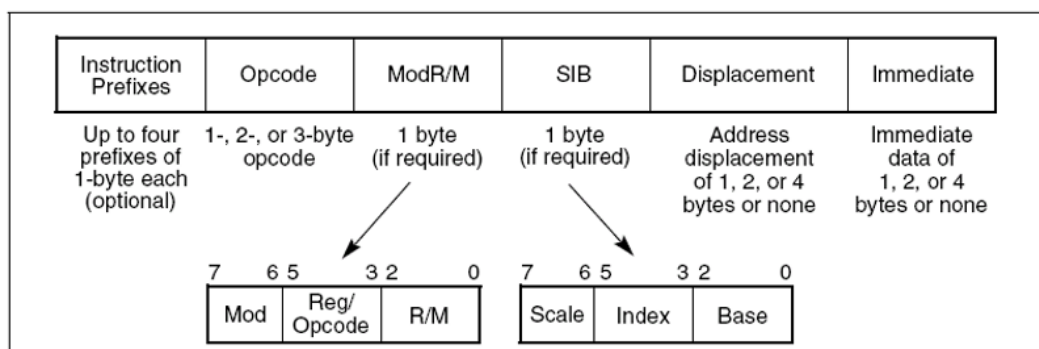
**指令：**

图 34 Bochs 指令结构

从上图可以看到，最长的 x86 指令总共有六部分组成，但很多指令只含有其中的部分内容。

指令前缀：总共可以分成 4 组

组 1 锁定和重复；组 2 段重载；组 3 操作数重载前缀；组 4 地址重载前缀。

操作码：opcode；Mod R/M 和 SIB；DISPLACEMENT 和立即数字节；

Chip8 指令集比较简单，指令由基础的操作码和操作数组成，操作数分为直接和间接。共 35 条指令。

**存储器：**

Bochs 内存由四部分组成：常规内存，上位内存区，高端内存区，扩展内存。

Chip8 内存主要分为系统区和工作区，Bochs 由大量虚拟内存机制，可扩展大量内存。Chip8 没有虚拟内存功能，内存大小固定。

**外设模拟：**

Bochs 提供了“设备集合类 `bx_devices_c`”，通过这个类可以查看调用外设的各种信息，通过不同类之间的相互调用，实现类似硬件接口的功能。

Chip8 的外设主要是输入输出的模拟，输入输出通过读入虚拟的外设以后，

会由 chip8 的指令进行读取和发送。

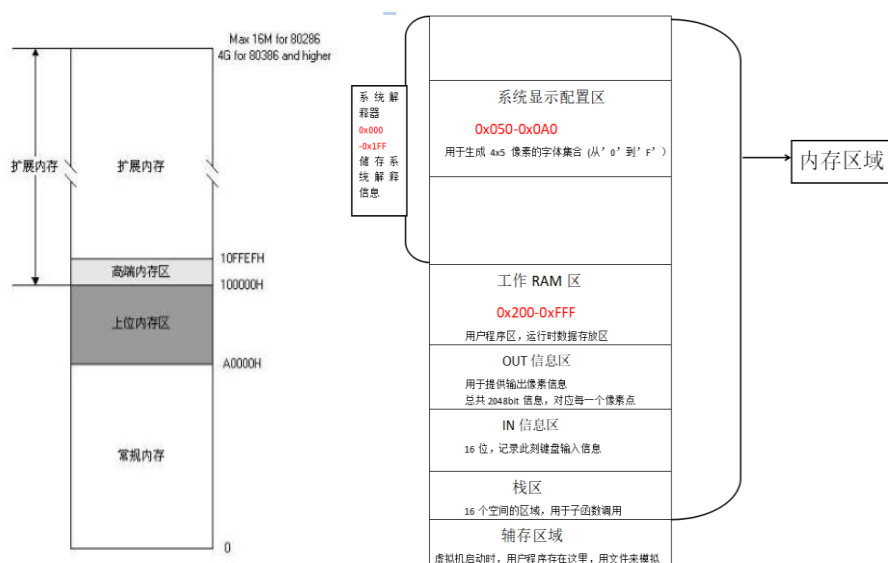


图 35 Bochs 和 Chip 8 的内存分布

### 3.2.3 总结与展望

总结：通过这次 chip8 虚拟机的编写，测试，应用，到最后使用 boch 虚拟机，深刻的理解和体会了计算机的体系结构。多层次的体会计算机冯诺依曼体系下的计算机结构，Chip8 作为一款专用计算机，在实现上简洁但是在结构上比较完整。Bochs 模拟量 x86 机器，内容十分丰富，有很多细节，帮助我更细致的研究计算机结构功能的实现方式。

展望：通过这次虚拟机设计，加深了对计算机体系的理解，深入研究了细节上的知识，但更能感受到计算机体系的复杂，还有很多计算机功能的具体实现并不甚清晰，希望在今后的学习中深入了解，同时，希望将硬件和软件结合，学习更多高效执行程序的方法。

## 3.3 Bochs 虚拟机 VS Prim 虚拟机

### 3.3.1 硬件框架比较

Bochs 是一个 x86 硬件平台的开源模拟器。它可以模拟各种硬件的配置。Bochs 模拟的是整个 PC 平台，包括 I/O 设备、内存和 BIOS。更为有趣的是，甚至可以不使用 PC 硬件来运行 Bochs。事实上，它可以在任何编译运行 Bochs 的平台上模拟 x86 硬件。通过改变配置，可以指定使用的 CPU(386、486 或者 586)，以及内存大小等。换言之，Bochs 是电脑里的“PC”。根据需要，Bochs 还可以模拟多台 PC。此外，它甚至还有自己的电源按钮。<sup>[5]</sup>

而我制作的 Prim 虚拟机（以下简称 Prim 虚拟机）通过软件模拟的具有完整硬件系统功能的、运行在一个完全隔离环境中的完整计算机系统。虚拟机的软件从电脑资源中分出一部分的 CPU、内存、硬盘存储...等等，然后虚拟机软件把这些资源整合，组成了一台电脑。我的虚拟机不包含物理机器中的 BIOS 和操作系统。虚拟机包括逻辑处理、寄存器堆、鼠标和键盘外部设备的 I/O、SP 和内存，其中内存包括虚拟机测试程序、数据区和 I/O 区。

Bochs 和 Prim 虚拟机硬件区别大致如下图所示，Bochs 包含 BIOS，而 Prim 虚拟机不包含 BIOS。

BIOS 是一组固化到计算机内主板上一个 ROM 芯片上的程序，它保存着计算机最重要的基本输入输出的程序、开机后自检程序和系统自启动程序，它可从 CMOS 中读写系统设置的具体信息。其主要功能是为计算机提供最底层的、最直接的硬件设置和控制。此外，BIOS 还向作业系统提供一些系统参数。系统硬件的变化是由 BIOS 隐藏，程序使用 BIOS 功能而不是直接控制硬件。

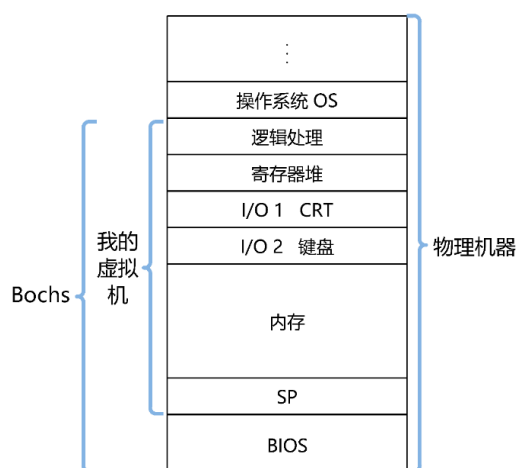


图 36 硬件框架区别

### 3.3.2 软件功能比较

Bochs 是一个 X86 PC 的模拟器，它可以模拟几乎所有类型的 X86 CPU，包括 16 位，32 位和 64 位(X86\_64)，内存以及 I/O 设备，在其上可以运行 Linux，Windows 等操作系统。Bochs 基本构成及功能如下图所示<sup>[6]</sup>：

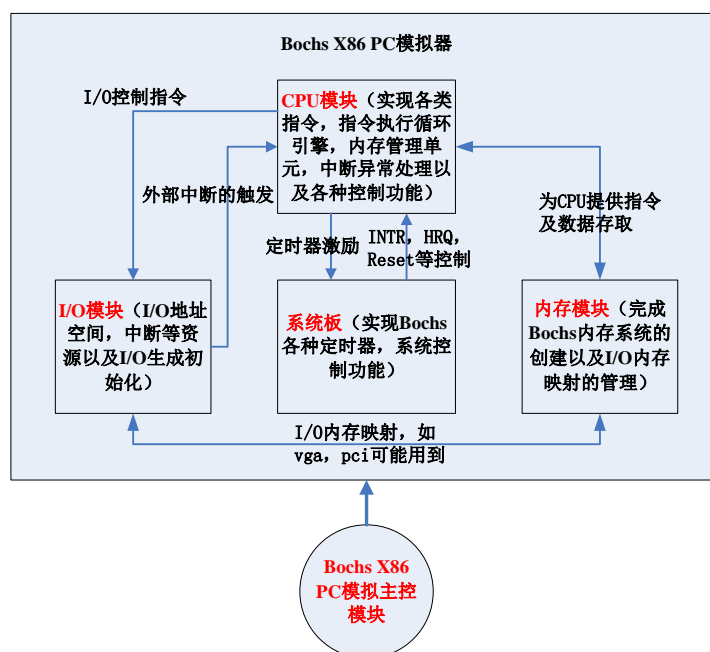


图 37 Bochs X86 系统构成



而 Prim 虚拟机包含 64 位运算器, 64 位存储器, 64M 内存。并且基于冯诺依曼体系结构, 通过软件实现物理机器中的运算器、存储器、控制器、数据通路、I/O 设备、桥、总线等硬件模块。自行设计基本指令集, 通过软件仿真仿照计算机硬件模拟虚拟机的构造, 使虚拟机程序运行流程及指令执行过程基本, 与物理机器中的微程序控制器作用、逻辑基本相同。

指令集使用 RISC 精简指令。参考 x86 汇编指令, 在其基础上做了一些简化, 有所改动。指令集包含了 38 条指令, 能完成计算机基本操作。每一条指令不超过 12 位, 前 3 位 (4 位) 表示操作数, 后面是参与操作的寄存器或者内存地址。

同时, 虚拟机支持中断操作, 能保护现场、执行中断子程序、恢复现场。虚拟机支持 I/O, 能从外部设备接受鼠标和键盘的信号, 进行交互。虚拟机支持 CMD 手动输入指令, 方便进行逐条指令的调试。并且有较清晰的交互界面。

两者主要相似和差异主要在于:

	Bochs 虚拟机	Prim 虚拟机
相似	均按照冯诺依曼体系。具有相似的硬件构成, 如 CPU、内存以及 I/O 设备。以及相同的指令执行流程和程序运行流程。	
	均能完成中断、I/O、特权指令等操作及指令。	
	具有从开机到关机每一步操作, 充分仿真模拟一台计算机。	
差异	Bochs 是一个 X86 PC 的模拟器, 它可以模拟几乎所有类型的 X86 CPU, 包括 16 位, 32 位和 64 位(X86_64)。	只可模拟单一特定的 CPU, 仅支持仿真 64 位 CPU。

	在 Bochs 上可以运行 Linux, Windows 等操作系统。	指令集使用 RISC 精简指令。参考 x86 汇编指令,在其基础上做了一些简化及优化,有所改动。故不能与 Linux, Windows 等现有操作系统严格匹配。
	Bochs 有丰富的调试指令,如 'b' 设置断点、'c' 连续执行、's' 单步执行、'step N' 执行 N 条指令、寄存器查询等。	我设计的虚拟机有详尽的显示交互界面,可实时显示各寄存器、内存的值和操作的结果。并且可选择单拍或者连续执行。故调试方法简单快捷,直观明了。

3.3.3 具体实现比较

Bochs 为一个 x86 CPU 的模拟器,它是一个用 C++语言写成的高移植性的开源 IA-32 架构的 PC 模拟器,可以在很多的流行操作系统上运行。它模拟了 Intel x86 CPU, 常见 IO 设备以及一个 BIOS<sup>[6]</sup>。

<i>Bios</i>	系统 BIOS 代码
<i>Bx_debug</i>	Bochs 调试器的代码
<i>Cpu</i>	模拟 x86 CPU
<i>Dasm</i>	反汇编,用于 bochs 调试版本中
<i>Docs-html</i>	Html 格式的部分文档
<i>Doc</i>	部分文档
<i>Font</i>	VGA 字体点阵
<i>Fpu</i>	浮点处理器
<i>Gui</i>	图形界面,和 OS 相关
<i>Instrument</i>	辅助插件
<i>Iodev</i>	IO 设备
<i>Memory</i>	模拟存储器源代码
<i>Misc</i>	一些其它的部件源代码

图 38 Bochs 工程文件

Prim 虚拟机包含 CPU、Debug、Doc、Gui、Iodev、Memory 等模块，相比 Bochs 源码中的工程文件缺少了 BIOS、Dasm 反汇编、Font、Fpu 浮点处理、Instrument 等功能及源文件。

Prim 虚拟机也是用 C++ 写成的高移植性的仿真模拟器。Prim 虚拟机在指令执行和程序运行流程中，读取指令、执行指令、中断、I/O 的逻辑处理，基本和 Bochs 实现方式相似。但是在各组成部分具体实现的方法上，有所差别：

**1. 指令集：**Bochs 指令集集成了复杂指令集 (CISC) 和精简指令集 (RISC) 的优点<sup>[7]</sup>，而 Prim 虚拟机主要是用精简指令集 (RISC)。CISC 包含了一套复杂的指令集。往往一条指令就可以完成一套复杂的操作，而在 RISC 中，这些操作需要几条指令才能完成。CISC 功能强大但译码器复杂，RISC 简洁但需要更多的指令完成相同的操作。

**2. 内存、寄存器实现：**Bochs 中内存和寄存器均使用一个类表示，而 Prim 虚拟机实现过程中，内存和寄存器分别通过二维数组和一维数组实现。

**3. 特殊寄存器：**Bochs 包含更多种类的特殊寄存器，例如段寄存器、全局寄存器 GDI 和 IDT。Prim 虚拟机只实现了常用的 PC、PSW、SP、IR、IAR 等常用特殊寄存器，没有实现 GDI 和 IDT 这两种寄存器。

**4. 交互界面：**Bochs 的交互界面类似于 CMD 命令行。而 Prim 虚拟机的交互界面使用 Qt 5.9.2，将虚拟机各个组建图形化显示，如下图所示：



图 39 Prim 虚拟机交互界面

### 3.3.4 总结与展望

#### 总结:

目前常用的虚拟机种类繁多。它们实现的功能也不尽相同： JAVA 虚拟机实现了代码的跨平台运行， Virtual PC 提供了一台虚拟的 PC 机供人们使用， VMware ESX server 则将一台功能强大的服务器划分称为若干网络虚拟主机。还有一些实际上可以被称为模拟器的虚拟机则仿真了某种 CPU 及系统的功能。而 Bochs 是一个 x86 CPU 的模拟器，一个用 C++ 语言写成的高移植性的开源 IA-32 架构的 PC 模拟器，可以在很多的流行操作系统上运行。

而 Bochs 一大特色就是其调试功能极其强大。在使用过程中，我感觉 Bochs 的调试和 gdb 对汇编的调试很类似，调试指令较多，用起来十分方便。而且对于开发操作系统的人来说，可以直接看到 CPU 的执行情况，以及各个寄存器和

内存单元的内容，这对于调试程序、掌握程序的运行情况是很有帮助的。

在比较 Bochs 和我制作的 Prim 虚拟机的过程中，让我发现了 Prim 虚拟机上可以完善的一些部分，例如可以将内存和寄存器分别打包成类，并且记录一下基本信息，方便调试时对该部分进行查询等等。

通过这次 Bochs 的安装与比较，让我对计算机组成尤其是 X86 体系结构有了更深的认识。更加清晰理解了计算机各部件是如何通过软件实现的。让我对计算机组成原理有了更深层次的理解。

### **展望：**

同时，对未来继续研究 Bochs 有一些展望：

1. 这次实验，将 Bochs 安装在了 Windows 系统下。未来可尝试将 Bochs 安装在 Linux 或者 Android 系统下，并且进行调试。
2. 可以再测试单条指令的基础上，在 Bochs 上模拟安装 Windows 等操作系统。
3. 详细分析 Bochs 指令集中的指令特点，及指令集优势。
4. 继续熟练掌握 Bochs 调试操作，并且应用在日常使用中对各种操作系统的调试与学习中。

### 3.4 Bochs 虚拟机 VS 星河一号虚拟机

#### 3.4.1 差异比较

	Bochs	星河一号
运行平台	多平台	多平台，图形界面主要支持 Windows10
编写语言	C++	C++
架构设计	选用冯·诺依曼结构，将数据和指令统一存放在内存中，由控制器、运算器、存储器、输入设备和输出设备五大部件组成	选用冯·诺依曼结构，将数据和指令统一存放在内存中，由控制器、运算器、存储器、输入设备和输出设备五大部件组成
虚拟硬件	以 x86 硬件平台为基础的全套计算机设备	通用计算机
外设配置	支持模拟键盘、鼠标、VGA 显卡/显示器、磁盘、时钟芯片、网卡等标准 PC 外设，并自带驱动	需要程序员自行编写驱动程序
指令系统	支持运行 x86 平台的汇编语言程序，并可借助外设运行整个操作系统	使用自主设计的类汇编语言 (Assembloid) 指令系统，采用精简指令集结构，指令结构简单，对程序员编程更为

		友好，指令运行效率高，指令利用率高
操作系统	可运行基于 x86 结构的几乎所有操作系统，包括 DOS、Microsoft Windows 的多个版本、BSDs、Linux、Xenix、Rhapsody (Mac OS X 的前身)[8]	程序员可自由编写各种各样的操作系统，供虚拟机运行
交互界面	拥有终端版本的显示界面，也可以使用内置的虚拟显示器显示所运行的操作系统提供的图形化界面	在需要交互界面作调试或展示等用途时，可以使用内置的星河一号图形化展示界面；当追求高性能时，可以使用终端版本显示界面
调试方法	可以使用调试命令 “s” (step) 进行单步调试；也可以使用调试命令 “c” (continue) 进行连续调试；使用 Ctrl+C 停止执行；使用 Ctrl+D 退出命令行空行状态；使用 q(quit) 退出调试和执行界面	可以使用图形界面内置的输出设备实时查看系统关键寄存器和数个通用寄存器的值，也可以提供中断信号，随时进入命令行模式，查看任何一个值，以此来做具体的调试工作

程序加载	在开机前可通过操作系统自带的浏览并打开文件的方式，加载计算机中任意一个虚拟机程序文件或操作系统镜像	默认方式为运行与虚拟机同目录下的程序文件“program”，也可以自行调整默认加载的程序文件名，所加载的程序为类汇编语言源代码
程序特征	需要提前编辑好程序映像，并提前指定虚拟机启动时的各项参数	由虚拟机自动读取虚拟机所运行的程序，在虚拟机内部由编译器将程序编译为可执行的指令代码，存入内存，待 CPU 取指令阶段取出
应用功能	面向已有的架构制作的虚拟机，因此不需要程序员对程序和操作系统做太多更改，便可以在虚拟机上直接运行 x86 汇编程序和操作系统	采用自主设计的全新架构，需要程序员为该种架构特别编写程序和操作系统
虚拟机实现	从底层硬件开始模拟，支持处理器（包括保护模式），内存，硬盘，显示器，以太网，BIOS，IBM	针对虚拟机“虚拟”的特性，即全部数据存储在主机的内存中的特性，对真实计算机结构做了精简和特别优



	PC 兼容机的常见硬件外设 的仿真[8]	化，使其更适合作为虚拟计算机的结构，并在此基础上使用软件实现该种结构
应用范围	主要用于操作系统开发（当一个模拟操作系统崩溃，它不崩溃主机操作系统，所以可以调试仿真操作系统）和在主机操作系统运行其他来宾操作系统，它也可以用来运行不兼容的旧的软件（如计算机游戏）	主要用于向计算机组成原理学习者提供一个学习计算机基础架构的平台，并为后续在虚拟机上实现高级语言和操作系统提供无限可能

### 3.4.2 总结与展望

我所使用过的虚拟机软件，除了我所设计的星河一号虚拟机之外，还有 VMware、Virtual Box、Parallels Desktop 等，而这些虚拟机在生活中的应用范围几乎都是使用其他操作系统、在封闭的环境中做测试等。它们的使用方式都是直接运行所需要运行的操作系统的映像文件。这次的 Bochs 虚拟机是我所深入接触的第一个可以对计算机系统硬件进行模拟和操作的虚拟机软件。

这次对 Bochs 虚拟机的研究，带给了我以下几点收获：

- ① 尝试了更多种类的虚拟机软件，了解了市面上用来模拟操作系统的虚拟机软件的工作原理。

- ② 通过动手实践的方式，较为深入地学习了现今用途广泛的 x86 平台的硬件结构和使用方式。
- ③ 和自主设计制作的星河一号虚拟机进行对比，学习其他虚拟机软件的长处。
- ④ 初步了解了 x86 汇编语言，并独立编写了自己的首个汇编语言程序，开启了汇编语言的世界，为今后的汇编语言课程打下了基础。

今后也可以在星河一号虚拟机的基础上，继续进行如下工作：

- ① 利用词法和语法分析知识，编写一个支持数组、指针等功能的高级语言。
- ② 利用操作系统的知识，使用上述高级语言编写一个简单的操作系统。
- ③ 将星河一号虚拟机进行升级，使其更适合计算机初学者的学习。

## 3.5 Bochs 虚拟机 VS jfvm 虚拟机

### 3.5.1 硬件功能比较

Bochs 虚拟机是可移植、开放源代码的 x86、x86-64 IBM PC 兼容机模拟器和调试工具。它支持处理器（包括保护模式），内存，硬盘，显示器，以太网，BIOS，IBM PC 兼容机的常见硬件外设的仿真。Bochs 模拟的是整个 PC 平台，包括 I/O 设备、内存和 BIOS。事实上，它可以在任何编译运行 Bochs 的平台上模拟 x86 硬件。Bochs 主要用于操作系统开发和在主机操作系统运行其他来宾操作系统。当一个模拟操作系统崩溃，它不会崩溃主机操作系统，所以可以调试仿真操作系统。它也可以用来运行不兼容的旧软件，如计算机游戏。

Bochs 的优点在于能够模拟跟主机不同的机种，但缺点是它的速度却慢得多。

与此对比, 我制作的 jfvm 虚拟机是通过软件模拟的, 具有完整硬件系统功能的、可以运行在完全隔离环境中的计算机系统中。Jfvm 虚拟机作为用户程序运行在操作系统之上, 利用软件从电脑资源中分配得到的 CPU、内存、磁盘等运算、存储资源, 整合后通过软件模拟的方式, 组成了一台电脑。Jfvm 虚拟机包含逻辑处理、通过显示器和键盘鼠标等输入输出设备、在硬件内存中模拟的寄存器堆、堆栈指针和内存区域, 但不包含操作系统和基本输入输出系统(BIOS)部分。由于系统硬件的变化是由 BIOS 隐藏, 程序使用 BIOS 功能而不是直接控制硬件, 所以现代操作系统会忽略 BIOS 提供的抽象层并直接控制硬件组件。

### 3.5.2 软件功能比较

Bochs 是一种平台仿真器, 这意味着它其实并不是现代意义上的虚拟化, 而是模拟。在虚拟化中, 虚拟机监控程序和虚拟机(VM)在裸机硬件上执行(通常通过硬件指令, 虚拟机监控程序创建环境在 VM 之间共享硬件)。因为通过主机处理器直接执行指令, 所以此过程通常被称为直接执行。此种类型的虚拟化使用来自现代处理器的支持。仿真通常在 VM 和底层硬件和操作系统之间提供层以便创建如下图所示的预期的平台环境的幻想。

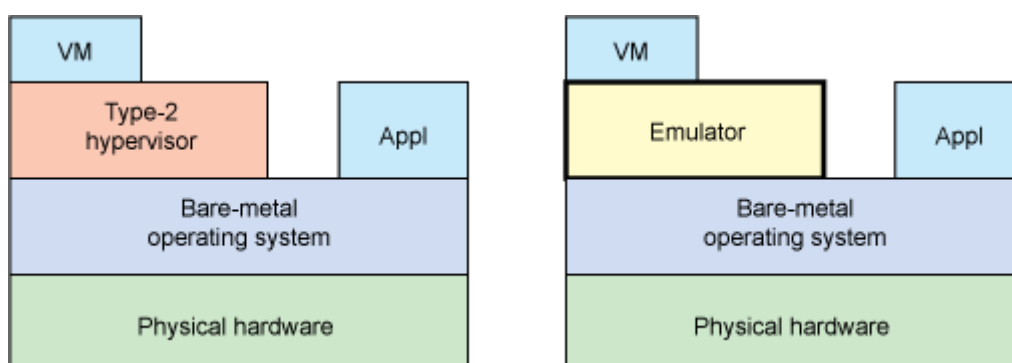


图 40 Bochs 的仿真层次

仿真的优势是对环境的完全控制，即更容易暂停整个机器来检查其状态以及将任何类型的故障引入仿真以测试操作系统的错误树。虽然此功能以性能为代价，但是它才是此类环境的真正好处。此外，仿真器提供了来宾机器与主机的真正隔离。这样，仿真器可以提供完全透明的环境。

### CPU 仿真：

Bochs 在其仿真方法方面很独特。Bochs 实现了 CPU ISA 的纯仿真，提供了真正 CPU 的仿真，甚至达到仿真 CPU 的提取-解码-执行流的水平。

在流阶段中，Bochs 执行权限检查，然后提取操作码。在解码阶段，Bochs 将已提取的 x86 指令解码为内部表示。它还通过存储原始 x86 系统的“微操作”来维护指令缓存(其后可在没有解码开销的情况下提取)，进而改进性能。最后，在执行阶段，Bochs 执行一些围绕特定指令执行的操作(包括操作数的有效地址计算，然后针对特定指令执行方法执行间接调用)。在执行指令时，可(适当地)更新任何受影响的寄存器和标志。Bochs 在此处提供名为迟缓标志更新的优化以便仅在需要算术标志时才计算这些标志(而不是在每一步都进行更新)。

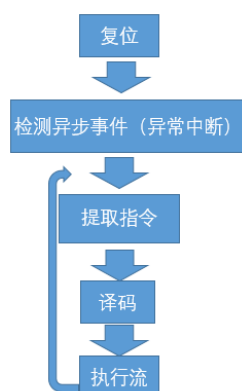


图 41 Bochs 指令执行步骤

在提取-解码-执行流之外, Bochs 还管理着诸如设备中断等外部事件和其他必要功能, 如字节交换和其他特权检查。除了通用 x86 CPU 以外, Bochs 实现了许多 CPU 功能, 如使用 MMX™ 技术的 Intel Pentium® 处理器; Intel 的流式单指令, Multiple Data Extensions 2 (Multiple Data Extensions 2, SSE2); 以及 AMD 3DNow! 指令。

Jfvm 的 CPU 仿真方式为先初始化、启动机器, 每次循环先检测是否中断, 如果有: 保存现场、修改 pc、执行中断服务程序、改回 pc、恢复现场。如果没有就继续执行。

取指: 取指阶段从存储器读入指令, 地址为程序计数器(PC)的值。从指令中抽取出指令指示符字节的两个四位部分, 称为 icode 和 ifun。再取出一个寄存器指示符字节, 指明一个或两个寄存器操作数指示符 rA 和 rB; 或是取出一个四字节常数字 valC, 并按顺序方式计算当前指令的下一条指令的地址 valP。  
(valP 等于 PC 的值加上已取出指令的长度)

执行: 在执行阶段, ALU 要么执行指令指明的操作(根据 ifun), 计算存储器引用的有效地址, 要么增加或减少栈指针, 得到 valE。在此, 也可能设置条件码。对一条跳转指令来说, 这个阶段会检验条件码和(ifun 给出的)分支条件, 看是不是应该选择分支。访存阶段可以将数据写入存储器, 或者从存储器读出数据。读出的值为 valM。写回阶段最多可以写两个结果到寄存器文件。最后更新 PC, 将 PC 设置成下一条指令的地址。

总之，处理器无限制地循环执行这些阶段，只有在遇到 halt 指令或一些错误情况时，才会停下来。错误情况包括非法存储器地址(程序地址或数据地址)，以及非法指令。

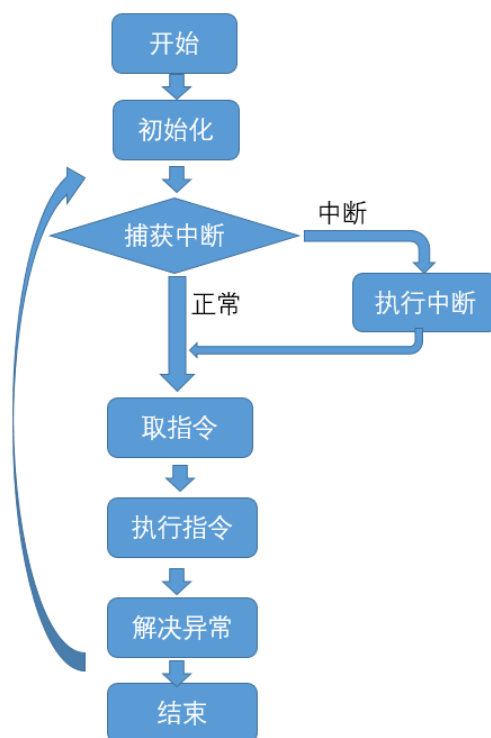


图 42 Jfvm 虚拟机的指令执行步骤

### 设备仿真:

通过基于设备类型的设备仿真，Bochs 提供了对 PC 平台的仿真。例如，虽然键盘是主机和虚拟机之间共享的物理设备，但是磁盘是通过仿真共享的，在这里来宾磁盘是主机磁盘文件系统中的文件。

Bochs 提供了一系列有用的仿真硬件，如内存、网卡、显卡、芯片组、集线器、声卡、磁盘和软盘控制器。如同 CPU，这些设备也被仿真，这在整个过程中会引入相当大的开销。例如，在驱动程序与硬件进行通信时，它操纵寄存器以便请求一些所需的设备操作。在仿真平台中，代表设备的寄存器和内存将

被仿真。在来宾操作系统的驱动程序窥探内存和硬件寄存器时，软件存在于下面来仿真设备。

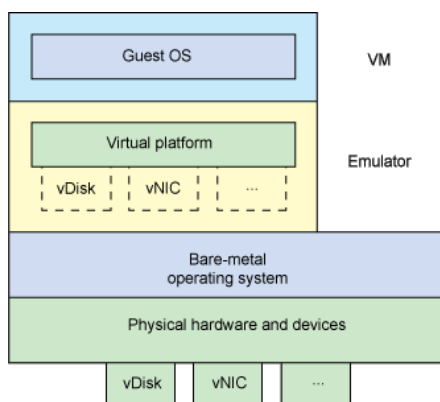


图 43 Jfvm 虚拟机的虚拟层次

Jfvm 的设备仿真通过利用高级语言中变量和结构体的概念，对寄存器进行了抽象模拟。该虚拟机的寄存器和存储器数据全部通过操作系统映射，存放在了宿主机的内存中，因此虚拟机中读取/存取寄存器的速度和操作内存的速度一致。当然为了区分，虚拟机通过高级语言中变量和结构体的概念，对寄存器进行了抽象模拟。通过数组下标代替内存地址，用数组数据代替内存数据。虚拟机使用高级语言中的数组模拟内存，使得虚拟机的内存调用和宿主机的内存调用速度一致。

本虚拟机内存将字符串当成存放的数据，通过指令的分成四类，从而有效地合成一条完整的指令。

### 用户界面比较：

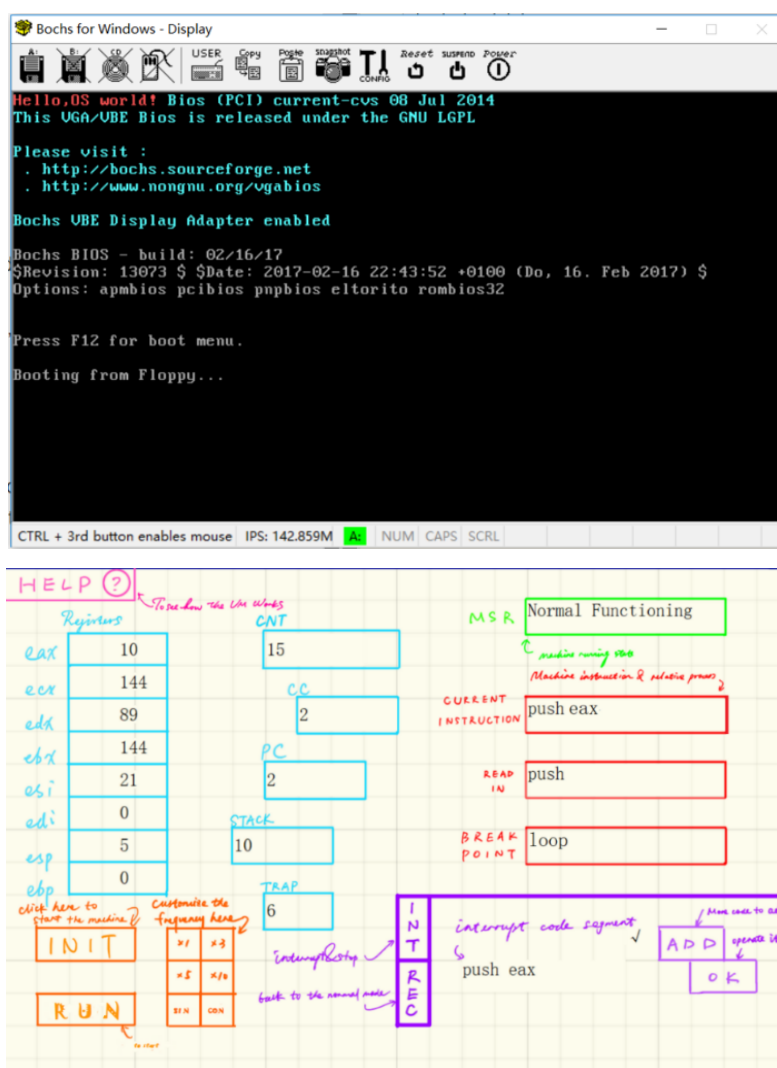


图 44 Bochs 与 Jfvm 虚拟机的用户界面比较

### 3.5.3 总结与展望

总结：从上学期用 Proteus 硬件仿真平台，利用冯诺依曼结构实现计算机的小型模型机之后，再到本学期利用了软件仿真的方式，通过高级语言来模拟虚拟机的编写，测试，应用，到最后使用 boch 虚拟机，深刻的理解和体会了计算机的体系结构。多层次的体会计算机冯诺依曼体系下的计算机结构。

Jfvm 虚拟机作为运行在操作系统上的一个程序，本质上虚拟机软件像操作系统申请的内存空间，因此虚拟机中的变量寄存器、程序内存的调用和访问速度是



没有差别的。而利用这一点，我们将逻辑上的硬件结构做了简化，减少了软件实际运行指令的条数，增强了性能。而 Bochs 是一种 x86 PC 仿真器和调试器，提供对整个 PC 平台的仿真是一个适用于操作系统开发的理想应用程序。虽然 Bochs 通过仿真完成此操作，但是却以性能为代价。

展望：由于 Bochs 跨平台的特性，这次我们是安装在 Windows 系统下。未来可尝试将 Bochs 安装在 Linux 或者 Android 系统下并进行调试。同时我们也可以仔细研究 Bochs 的指令集，并分析其指令集设计的特点及优势。更重要的是，通过对 Bochs 的深入学习，也是对以后在不同操作系统上调试和学习打下了基础。

## 3.6 Bochs 虚拟机 VS DIYVM 虚拟机

### 3.6.1 差异比较

#### 1. 开发背景

我的虚拟机是为进一步学习计算机组成原理而设计，在各个环节注重理论知识的具体实现，而没有把虚拟机的实用性作为重点，也没有进行后续开发维护的打算。由于知识水平的局限性，我的虚拟机必然是有许多不足的。

Bochs 作为成熟的可商用虚拟机，在各个方面都已十分成熟，在许多应用场景具有一定的实用性和竞争力，且有专业人员持续更新维护。

#### 2. 安装配置

我的虚拟机使用 C++ 语言和 Qt 4.8.1 开发环境，Qt 生成的可执行程序依赖 Qt 的一系列动态链接库文件和源代码文件，还有所执行的汇编程序，都需要放在同一目录下，如图：

名称	修改日期	类型	大小
DIYVM.exe	2018/5/17 16:22	应用程序	1,308 KB
libgcc_s_dw2-1.dll	2009/6/23 10:42	应用程序扩展	42 KB
main.o	2018/5/17 15:09	O 文件	185 KB
mainwindow.o	2018/5/6 15:39	O 文件	123 KB
mingwm10.dll	2009/1/11 3:32	应用程序扩展	12 KB
moc_mainwindow.cpp	2018/5/6 15:42	C++ Source file	3 KB
moc_mainwindow.o	2018/5/6 15:42	O 文件	131 KB
moc_myinfo.cpp	2018/5/17 15:09	C++ Source file	4 KB
moc_myinfo.o	2018/5/17 15:09	O 文件	171 KB
moc_myvm.cpp	2018/5/17 15:09	C++ Source file	4 KB
moc_myvm.o	2018/5/17 15:09	O 文件	118 KB
myinfo.o	2018/5/17 15:09	O 文件	198 KB
myvm.o	2018/5/17 16:22	O 文件	136 KB
QtCored4.dll	2018/4/30 12:29	应用程序扩展	42,670 KB
QtGui4.dll	2012/3/23 21:29	应用程序扩展	175,369 KB
testProgram.txt	2018/5/7 20:30	文本文档	1 KB

图 45 DIYVM 虚拟机的配置文件

Bochs 则将源代码封装到可执行程序内部，安装十分轻便，但也需要读取虚拟机的配置文件并指定磁盘镜像文件。在测试过程中，若这些文件不放在同一目录下则可能会发生 “No Bootable Device” 的问题，因此建议放在同一目录下，如图：

1.img	512	47	IMG 文件	2018/6/3 11:00
bochs.exe	3,924,9...	981,060	应用程序	2017/4/9 15:32
bochsdbg.exe	4,114,4...	1,082,5...	应用程序	2017/4/9 15:49
bochsrc.bxrc	684	477	BXRC 文件	2018/6/5 10:56

图 46 Bochs 虚拟机的配置文件

### 3.指令集

我的虚拟机的指令集参考 Mips 精简指令集，选取了实现功能必须的二十几条指令，整体来讲比较简单，除了运算指令外，转移类指令只有无条件跳转指令 JUMP 和条件转移指令 JC，比较类指令只有基本比较指令 CMP，因此编写汇编程序时部分功能的实现比较麻烦，比如想实现高级语言中的条件判断语句时，可能需要多次比较指令和多处条件转移指令，如图，为判断两寄存器相等则执行一

段代码，需要两次比较互相均不大于，则跳过 else 代码段，否则通过 JC 跳过 JUMP 指令，不跳过，即执行 else 代码段。

```
CMP 0 2↵
JC 0000002C↵
CMP 2 0↵
JC 0000002C↵
JUMP 00000044
```

图 47 DIYVM 虚拟机的跳转指令

Bochs 则实现了 x86 指令集，较为复杂但更加实用，网上有大量介绍，此处不再赘述。

#### 4.编译和存储

我的虚拟机从编译到存储再到运行均由其一体完成，存储通过 C 语言定义变量的方式开辟空间，然后通过赋值写入。由于编写时还没有学习操作系统课程中的内存管理章节，故内存分配比较简单，给编写汇编程序造成了一定麻烦。

Bochs 则采取了虚拟机与编译存储分离的策略，使得虚拟机本身更加轻便。但是从汇编程序到和加载到虚拟机中的.img 镜像文件，还需要额外的工具和操作。这部分网上教程不多，而且大部分是面向 Linux 系统的，学习时给我造成了一点困扰。

#### 5.I/O 操作

我的虚拟机仅支持输入无符号整数的操作，输出则为在程序结束后输出 0 号寄存器的值，相应的，汇编测试程序也仅能读取到无符号整数输入，输出则需要把结果装入其中。因为 I/O 操作设计起来难度较大，作为课程设计不需要在这里投入过多精力。

Bochs 则支持 PC 可支持的全部 I/O 功能，十分强大。

## 6.交互界面

我的虚拟机没有将重点放在图形界面的设计上，实现得有些简陋，但可以实时显示虚拟机各个部分的数值，操作上允许单步执行，方便调试和分析，如图所示：

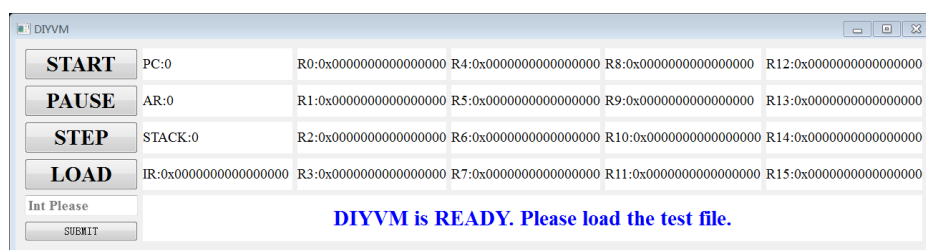


图 48 DIYVM 虚拟机的用户界面

Bochs 则具有简洁而成熟的图形界面，可以对虚拟机进行初始化配置，执行时进入命令行模式，在调试模式中也可以单步执行。这部分在本文档前面部分已有介绍。

### 3.6.2 总结与展望

通过此次对 Bochs 虚拟机的尝试，我更加清楚地观测到不同虚拟机之间的异同特点，在比较之中有了更深的理解。从编译到运行，虚拟机可以是一体完成的，也可以利用现有大量工具完成编译工作，使虚拟机本身更加的“高内聚，低耦合”；虚拟机可以通过规定并读取配置文件来灵活实现更多功能；虚拟机是强大的开发测试工具，软硬件开发均离不开虚拟机的使用。

Bochs 作为已经很成熟的虚拟机，是我学习虚拟机设计、计算机体系结构和 x86 架构很好的参考。下一步将利用 Bochs 进行更多测试学习，包括：

尝试 x86 复杂汇编程序的实现与调试，深入理解 x86 架构的特点；

尝试在 Bochs 上安装其他操作系统，并在其中安装运行正常程序，体会虚拟机的性能；

浏览 Bochs 开源社区，寻找更多学习资料，学习更多 Bochs 的功能和使用技巧。

## 参考资料

- [1] <http://bochs.sourceforge.net/doc/docbook/user/index.html>.
- [2] J. L.Hennessy and D. A. Patterson, Computer architecture-A Quantitative Approach, China Machine Press, 2012.
- [3] S. A. Brandt, “CMPS 111: Introduction to Operating Systems,” Computer Science Department, 2013.
- [4] <https://www.barik.net/archive/2005/01/16/201220/>.
- [5] Bochs-X86-PC 模拟器原理与启动流程分析 ——熊海泉
- [6] Bochs 项目源码分析与注释 ——喻强
- [7] 基于 Bochs 的 X86 小核前端时序模拟器的设计及优化 ——肖荣荣
- [8] <https://en.wikipedia.org/wiki/Bochs>