***2019***



**系统能力综合训练 课程设计报告**

|  |  |
| --- | --- |
| 题 目： | X86模拟器设计 |
| 专 业： | 计算机科学与技术 |
| 班 级： | CS1607 |
| 学 号： | U201614702 |
| 姓 名： | 樊俊超 |
| 电 话： | 17607117148 |
| 邮 件： | [616415398@qq.com](mailto:616415398@qq.com) |
| 完成日期： | 2020-01-06 |

目 录

[1 课程设计概述 1](#_Toc29284978)

[1.1 课设目的 1](#_Toc29284979)

[1.2 课设任务 1](#_Toc29284980)

[1.3 实验环境 1](#_Toc29284981)

[2 实验过程 2](#_Toc29284982)

[2.1 PA1 最简单的计算机 (\*^\_^\*) 2](#_Toc29284983)

[2.1.1 实现目标 2](#_Toc29284984)

[2.1.2 实现过程 3](#_Toc29284985)

[2.1.3 测试结果 4](#_Toc29284986)

[2.1.4 必答题 4](#_Toc29284987)

[2.2 PA2 简单(?)复杂(!)的机器 (/(ㄒoㄒ)/~~) 5](#_Toc29284988)

[2.2.1 实现目标 5](#_Toc29284989)

[2.2.2 实现过程 5](#_Toc29284990)

[2.2.3 测试结果 8](#_Toc29284991)

[2.2.4 必答题 8](#_Toc29284992)

[2.3 PA3 穿越时空的旅程 (((φ(◎ロ◎;)φ))) 9](#_Toc29284993)

[2.3.1 实现目标 9](#_Toc29284994)

[2.3.2 实现过程 9](#_Toc29284995)

[2.3.3 测试结果 10](#_Toc29284996)

[2.3.4 必答题 10](#_Toc29284997)

[3 设计总结与心得 11](#_Toc29284998)

[3.1 课设总结 11](#_Toc29284999)

[3.2 课设心得 11](#_Toc29285000)

[4 参考文献 13](#_Toc29285001)

# 课程设计概述

## 课设目的

理解"程序如何在计算机上运行"的根本途径是从"零"开始实现一个完整的计算机系统. 南京大学计算机科学与技术系计算机系统基础课程的小型项(Programming Assignment, PA)将提出x86架构的一个教学版子集n86, 指导学生实现一个功能完备的n86模拟器NEMU(NJU EMUlator), 最终在NEMU上运行游戏"仙剑奇侠传", 来让学生探究"程序在计算机上运行"的基本原理. NEMU受到了[QEMU](http://www.qemu.org/" \t "_blank)的启发, 并去除了大量与课程内容差异较大的部分. PA包括一个准备实验(配置实验环境)以及5部分连贯的实验内容:

## 课设任务

* PA1
  + PA1.1: 实现单步执行, 打印寄存器状态, 扫描内存
  + PA1.2: 实现算术表达式求值
* PA2
  + PA2.1: 在NEMU中运行第一个C程序 dummy
  + PA2.2: 实现更多的指令, 在NEMU中运行所有 cputes
  + PA2.3: 运行打字小游戏, 提交完整的实验报告
* PA3
  + PA3.1: 实现自陷操作 \_yield() 及其过程
  + PA3.2: 实现用户程序的加载和系统调用, 支撑TRM程序的运行
  + PA3.3: 运行仙剑奇侠传并展示批处理系统, 提交完整的实验报告

## 实验环境

docker | debian container ：fjc-vm，Linux 5870cb95368b 4.9.184-linuxkit #1 SMP Tue Jul 2 22:58:16 UTC 2019 x86\_64 GNU/Linux

实验文件：

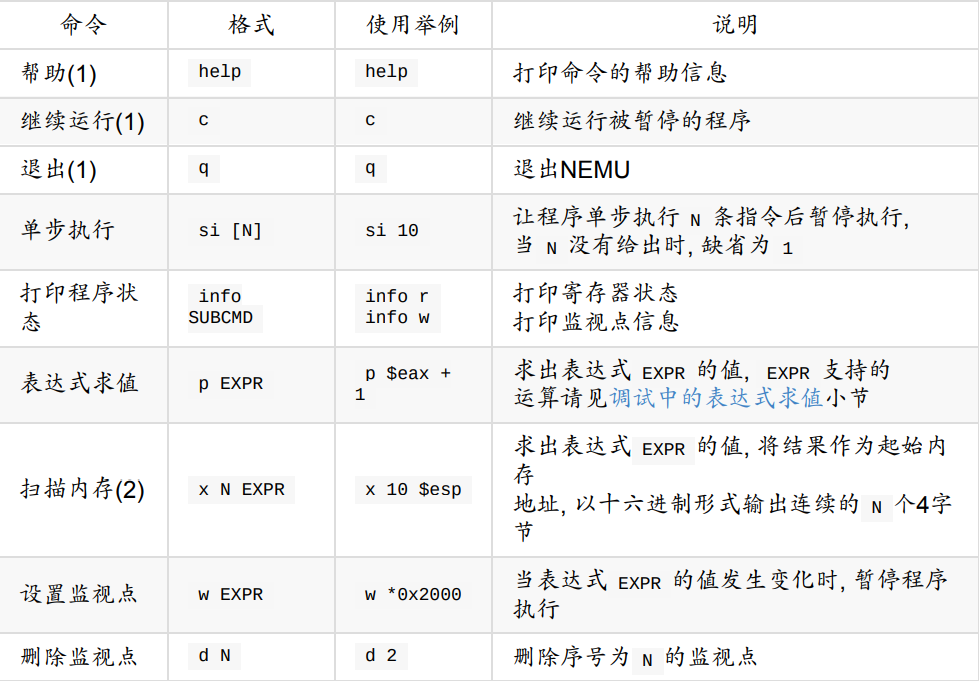
nemu（简化x86系统模拟器） Nanos-lite（简化操作系统）

# 实验过程

## PA1 最简单的计算机 (\*^\_^\*)

### 实现目标

* **简易调试器**

备注：NEMU启动时会在用户界面主循环函数ui\_mainloop()函数中输出NEMU的命令提示符：（NEMU）等待用户的输入，要求实现的可操作命令如上图，已经实现的命令是help（待完善）、c、q，其他命令待完成。

* 表达式求值

在简易调试器中，为了计算一些带有寄存器和内存的表达式，需要添加表达式求值的功能，整个流程大致为词法分析（设计正则表达式）、递归求值（分治法）、

实现带有负数的算术表达式求值。

* 监视点

监视点的作用是监视一个表达式的值何时发生变化，在简易调试器的设置监视点，删除监视点等过程中均需要首先实现监视点（要求扩展表达式求值的功能）。

为了方便同时操作多个监视点，监视点的信息采用链表形式，其结构体包括监视点序号，和指向下一个监视点的指针，要求实现监视池管理。

### 实现过程

* **简易调试器实现**

仿照已实现的命令，在cmd\_table里添加剩下的命令以及对应的信息，帮助完善help函数，同时完成对应的函数，如下。  
static int cmd\_si(char \*args);  
static int cmd\_info(char \*args);  
static int cmd\_x(char \*args);  
static int cmd\_w(char \*args);  
static int cmd\_d(char \*args);

由于大部分函数与后面的表达式求值以及监视点实现相关，不再赘述。

* **表达式求值实现**

词法分析

词法分析要求能够识别出表达式中的单元(token)，所允许出现token类型如下

十进制整数，+，-，\*，/，（，），空格串（一个或多个空格），-，= =。

-1-实现时采用Token结构体来记录token信息



-2-使用position变量来定位token的位置，tokens数组用来按顺序存放已被识别出的token信息，nr\_token指示已被识别出的token数目。

-3-特别注意由于负号与减号、指针与乘号均为同一个符号，在实现过程中要优先根据该符号的前后符号进行区分，优先识别指针与负号，修改优先级。

递归求值

当代求表达式中的token成功识别后，对tokens数组进行处理，具体方法如下：

-1-寻找主运算符，其首先为运算符，并且优先级在表达式中最低，具体设计为：

‘= =’<’+’<’-’<’\*’<’-’(负号)<’\*’（指针）

-2-对分裂出的两个子表达式进行递归求值，再根据主运算符的类型对两个子表达式的值进行运算。

-3-对于异常的表达式不应该使用assert，应在无法识别后输出提示信息，等待下步输入。

* **监视点**

监视点的信息全部存放在监视点池中，采用双链表分别记录(WP\* free\_)空闲监视点结构和已被调用监视点结构(WP\* head)。

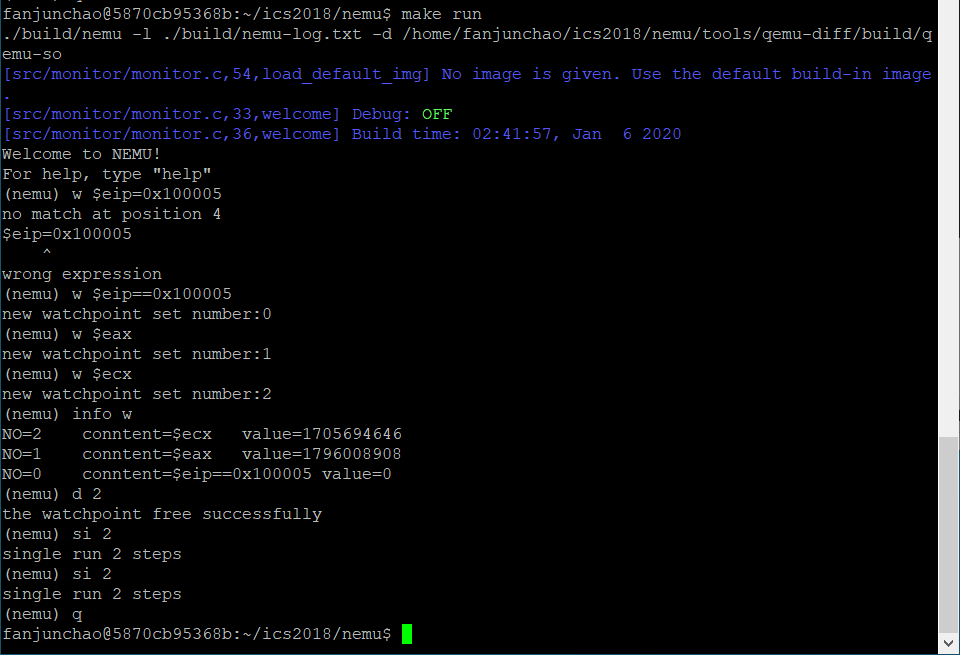
-1-当用户给出待监视表达式时，从free\_链表中申请空闲监视点结构，记录表达式值变化，当发生变化时，输出变化信息，等待用户反馈。

-2-当用户删除监视点时，从head链表中删除相关监视点，挂在free\_链表下等待下次调用。

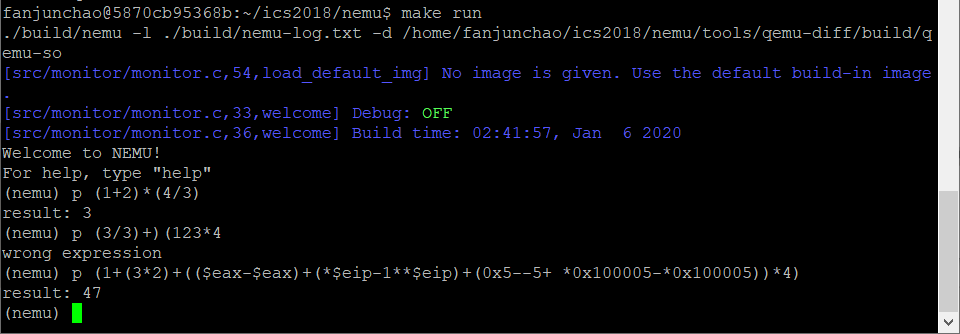
-2-当用户通过info w命令打印监视点时，输出head链表信息。

### 测试结果

* 各种命令测试



* 表达式求值及错误表达式反馈



### 必答题

见提交网站的pa1.pdf

## PA2 简单(?)复杂(!)的机器 (/(ㄒoㄒ)/~~)

### 实现目标

* CPU执行一条指令的全流程
* **取指**

进行一次内存访问，将$eip指向指令从内存中读入到CPU中

* **译码**

查表获知当前指令的操作数和操作码

* **执行并更新$eip**

真正执行指令，并更新$eip，使其指向下一条指令

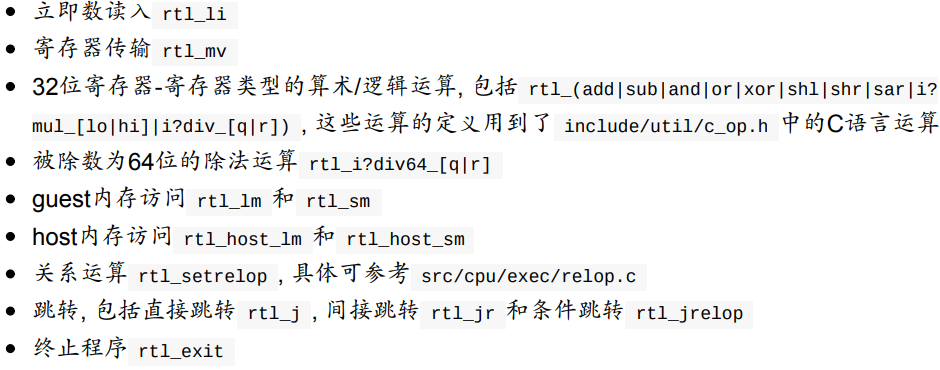
* **输入输出**

### 实现过程

* **RTL寄存器和指令**

-1- 在NEMU中, RTL寄存器统一使用 rtlreg\_t 来 定义, 而 rtlreg\_t (在 nemu/include/common.h 中定义)其实只是一个 uint32\_t 类型。分八个通用寄存器，源操作数1，源操作数2，目的操作数，临时寄存器

-2- rtl指令如下



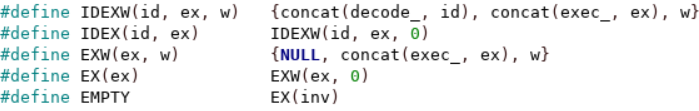
* **opcode\_map数组说明示例**

-1- 实现mov指令，如下：

C:\Users\fanchao\Documents\Tencent Files\616415398\Image\C2C\[ZJ]ES`X4Q@XC%5Z3BFW5U7.png

-2- 在opcode\_table数组中找到0xbd，根据定义好的宏填入对应位置

C:\Users\fanchao\Documents\Tencent Files\616415398\Image\C2C\[ZJ]ES`X4Q@XC%5Z3BFW5U7.png



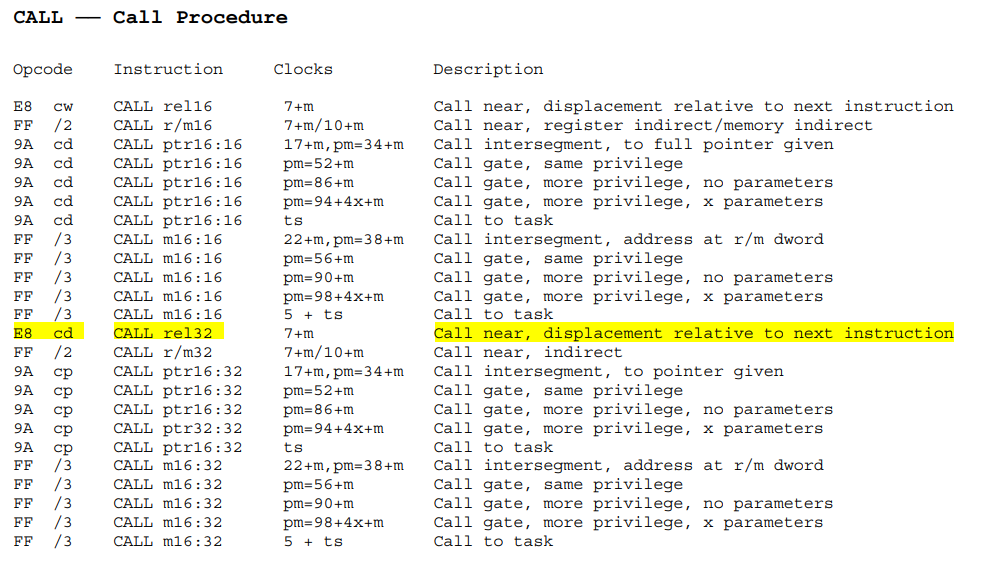
* **译码示例**

-1- 实现call指令

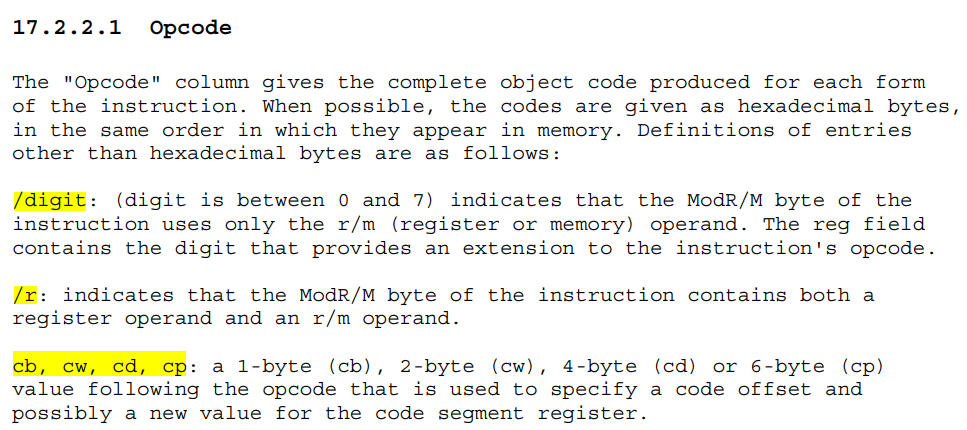
-2- 查阅i386手册的opcode map，得到对应信息CALL Av，参考已实现的mov指令填写opcode\_table数组0xe8位置。

-3- 在decode.h中找到所有的make\_Dhelper函数，并参照手册及实验文档理解opcode\_table数组中应该填写IDEX(id,ex)。

-4- 找到i386手册的call指令



同时找到opcode为cd时的说明



-5- 得知此处id应该为J，填写make\_EHelper函数，在此之前，发现很多同文件夹下的文件都包含TODO（）的make\_EHelper函数，将其全部声明在all-instr.h文件内，注明对应的实现文件位置，方便后面查找。同时得知ex为call。

-6- 在control.c文件中实现make\_EHelper（call）函数。

-7- 重复此过程，部分指令需要用到group或者ModR/M，先查看手册。

* **串口实现**

串口是最简单的输出设备. nemu/src/device/serial.c 模拟了串口的功能. 其大部分功能也被简化, 只保留了数据寄存器和状态寄存器。由于NEMU串行模拟计算机系统的工作, 串口的状态寄存器可以一直处于空闲状态; 每当CPU往数据寄存器中写入数据时, 串口会将数据传送到主机的标准输出。

实现in、out指令并在他们的Ehelper函数中分别调用pio\_read\_[l|w|b] () 和 pio\_write\_[l|w|b]() 函数

* **时钟实现**

nemu/src/device/timer.c 模拟了i8253计时器的功能. 计时器的大部分功能都被简化, 只保留了"发起时钟中断"的功能(目前我们不会用到). 同时添加了一个自定义的RTC(Real Time Clock), 初始化时将会注册 0x48 处的端口作为RTC寄存器, CPU可以通过I/O指令访问这一寄存器, 获得当前时间(单位是ms)。

在nexus-am/am/.../src/devices/timer.c 中实现\_DEVREG\_TIMER\_UPTIME 的功能，记录初始时间和当前时间，两者相减。

* **键盘实现**

键盘是最基本的输入设备. 一般键盘的工作方式如下: 当按下一个键的时候, 键盘将会发送该键 的通码(make code); 当释放一个键的时候, 键盘将会发送该键的断码(break code)。i8042初始化时会注册 0x60 处的端口作为数据寄存器. 每当用户敲下/释放 按键时, 将会把相应的键盘码放入数据寄存器, CPU可以通过IO端口访问数据寄存器, 获得键盘码; 当无按键获取时, 将会返回 \_KEY\_NONE . 在AM中, 我们约定通码的值为断码 | 0x8000 。

根据提示，每次读取0x60端口存放的键盘码，同时把记录键盘信息的结构体中的keydown记为1，输出键盘信息。

* **VGA实现**

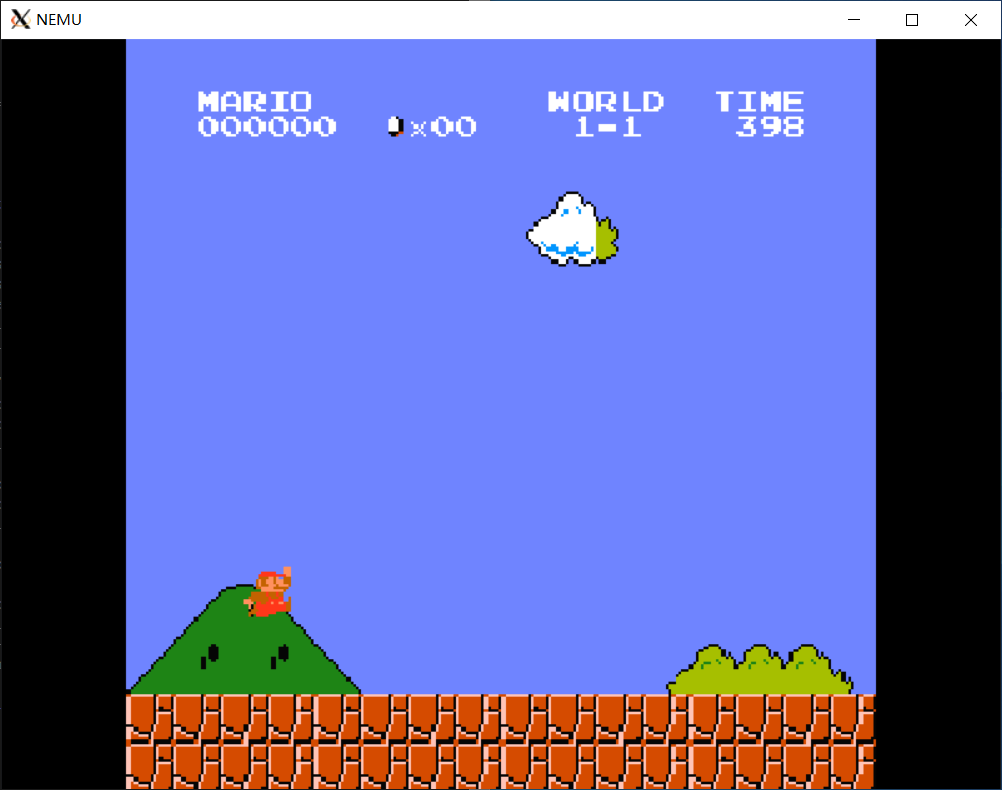
VGA初始化时注册了从 0x40000 开始的一段用于映射到video memory的物理内存。在 NEMU中, video memory是唯一使用内存映射I/O方式访问的I/O空间. 代码只模拟了 400x300x32 的图形模式, 一个像素占32个bit的存储空间。

-1-在NEMU中的 paddr\_read() 和 paddr\_write() 中加入对内存映射I/O的判断. 通过is\_mmio() 函数判断一个物理地址是否被映射到I/O空间, 如果是, is\_mmio() 会返回映射号, 否则返回 -1 。 内存映射I/O的访问需要调用 mmio\_read() 或 mmio\_write() , 调用时需 要提供映射号. 如果不是内存映射I/O的访问, 就访问 pmem。

-2-实现正确的draw\_rect()函数，无需考虑越界影响，由于采用行优先方式，每次画一行即可。

### 测试结果

* 马里奥测试



### 必答题

见提交网站的pa2.pdf

## PA3 穿越时空的旅程 (((φ(◎ロ◎;)φ)))

### 实现目标

* **异常处理**

实验采用门描述符（简化版）实现中断处理中的位置跳跃，为了方便管理各个门描述符, i386把内存中的某一段数据专门解释成一个数组, 叫IDT(Interrupt Descriptor Table, 中断描述符表), 数组的一个元素就是一个门描述符。操作系统的代码事先把IDT准备好, 然后执行一条特殊 的指令 lidt , 来在IDTR中设置好IDT的首地址和长度, 这一中断处理机制就可以正常工作了。

* **用户程序和系统调用**

用户程序的入口位于 navyapps/libs/libc/src/platform/crt0.c 中的 \_start() 函数, 它会调用用户程序的 main() 函数, 从 main() 函数返回后会调用 exit() 结束运行。可执行文件位于ramdisk偏移为0处, 访问它就可以得到用户程序的第一个字节。

完善loader函数，将ramdisk中从0开始的所有内容放置在 0x4000000 , 并 把这个地址作为程序的入口返回即可。

识别并通过系统调用处理函数do\_syscall()进行处理，传参使用宏定义GRP1、GPR2、GPR3、GPR4，系统调用返回值存放在GPR1中返回。

* **文件系统**

为了方便管理文件，采用文件记录表结构

typedef struct {

char \*name; // 文件名

size\_t size; // 文件大小

off\_t disk\_offset; // 文件在ramdisk中的偏移

} Finfo;

在Nanos-lite中, 由于简易文件系统中的文件数目是固定的, 我们可以简单地把文件记录表的下 标作为相应文件的文件描述符返回给用户程序，同时为每一个已经打开的文件引入偏移量属性 open\_offset , 来记录目前文件操作的位置。

### 实现过程

* **异常处理**

触发异常后按照下列过程处理

1. 依次将EFLAGS, CS(代码段寄存器), EIP寄存器的值压栈

2. 从IDTR中读出IDT的首地址

3. 根据异常号在IDT中进行索引, 找到一个门描述符

4. 将门描述符中的offset域组合成目标地址

5. 跳转到目标地址整个过程需要实现cpu结构的cs寄存器，实现lidt和int指令，在restart()函数中对eflags、cs初始化，重新组织排列\_Context结构体成员，实现正确事件分发。

* **用户程序和系统调用**

-1- 在 nanos-lite/src/main.c 中开始进行程序加载，可以看到除了 Log 信息，有几个重要步骤，即 init\_ramdisk(), init\_device(), init\_irq(), init\_fs(), loader()，其中 需要实现的函数为loader: 将ramdisk中从0开始的所有内容放置在 0x4000000 , 并把这个地址作为程序的入口返回即可。

-2- 实现后, 在 init\_proc() 中调用 naive\_uload(NULL, NULL)。

-3- 使用通用寄存器和已经准备好的系统调用接口\_syscall\_函数打包成一个系统调用事件并交给Nanos-lite调出系统调用处理函数 do\_syscall() 进行处理。

-4- 实现标准输出和堆区管理。

* **文件系统**

-1- 由于在Nanos-lite中，简易文件系统的文件数目是固定的，用文件在文件记录表中的下标作为文件描述符来进行文件操作即可。

-2- 添加open\_offset属性记录当前文件的操作位置，利用lseek函数调整属性。

-3- 重写loader()函数，使用已经实现的fs\_open() , fs\_read() 和 fs\_close() 函数。

-4- 把串口、VGA和设备抽象成文件。

### 测试结果



上图为读档后的截图。

### 必答题

见提交网站的pa2.pdf

# 设计总结与心得

## 课设总结

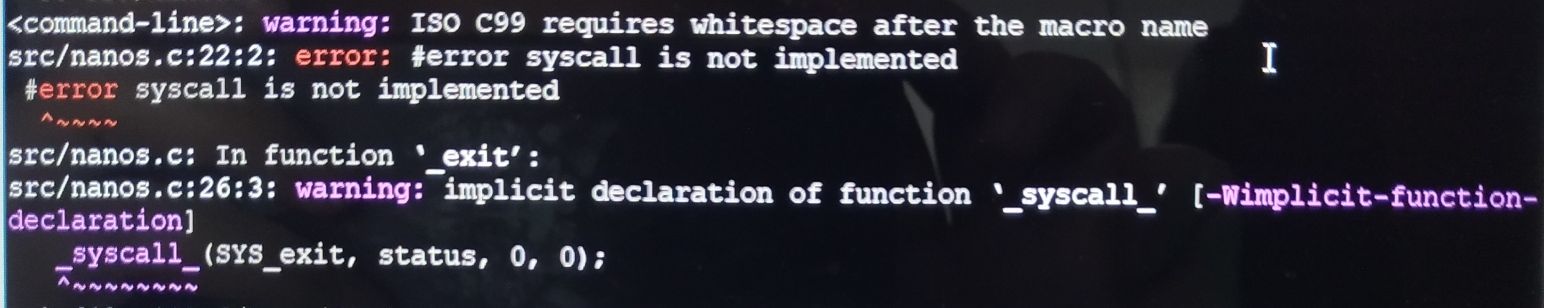
pa1实验过程中有很多问题，但pa1比较简单，遇到问题都能很快解决。

较典型的问题是负号及指针识别和相应处理问题，在实现括号识别函数时也很有趣，一开始没想到先出现‘)’的问题，后来想到需要处理异常表达式才加上。

pa2实验的问题，主要就是手册问题。

第一次写的时候一筹莫展，把文档看了好几遍，摸摸索索地做。也知道需要用到i386手册，就找了一个英文原版的，开始硬着头皮看。最开始时的问题主要是不知道在哪实现，后来发现定义好的makeEHelper函数分放在各个文件里，就做了个英明的决定，先把all-instr.h文件给填了； ModR/M和Group这个手册里有，但还是靠着stfw看理解方便点，但手册也是要读的。仿照老师写好的adc等指令的Helper函数开始写标志位的set。

pa3很简单，但是很尴尬，我在创建pa3后发现pa2的string函数有问题，在修改完毕后pa3中的数据并没有更新成功（我确实又执行了一遍，但输的什么指令我也忘了），以至于我在pa3的一开始就跑不通，我也很头疼，还麻烦老师帮我dubug（确实感谢万老师，给了我信心），老师也没找到错。那我就开始怀疑我是不是有pa2隐藏的bug，回头跑了一下pa2的测试程序，果然有问题，(ㄒoㄒ)/~~



如上图根本没进\_syscall\_函数调用中，修改后就过了。

## 课设心得

课设的确好，PA1到PA3下来我瘦了两公斤，五星好评。主要是我写实验，一旦开始就不想停下来（遇到bug除外，尤其是pa3那个让人没有头绪、一筹莫展、一无所知、知难而上、上蹿下跳、跳的贼高的问题），就开始每天早九点到晚十二点的日子，听着歌没想到这么快就实验结束了，也确实没去过教室，毕竟还有其他课的作业，最主要的原因是相比教室不能穿睡衣，不能在开心的时候唱歌，上个厕所跑那么远，没有热水喝的可怜，窝在宿舍用着16G运存6G独显电脑，穿着睡衣，喝着奶茶它不香吗的悠闲，那我当然是选择pa3了。当时想着只要我实验做得够快，考勤就撵不上我，233结果还是得靠老师的接济（允许截止到1月7号前检查）才写完实验。考研的事也搞的我心烦意乱，本来就是仓促做的决定，两个月的准备时间，又选了一门编程课，再加上能力培养，果断放弃考研（其实是没信心）连准考证都没打印，考试也没去。

现在想想，虽然pa3的顺利运行是在1月6号上午（那个自食其果的无语bug浪费了我至少5个小时，时间嗖的一下就过去了），但是的确是在标准时间之后过的，感觉自己太浪了，但凡考勤是满勤，那也不可能写pa3啊（开个玩笑，准备在放假期间把pa4和5也做了，弥补下愧疚，顺便治下强迫症）。

总的来说，这门课实验材料满分，因为老师的文档充满了说话的艺术，细节都藏在老师的调侃里，不一定你就错过了某些要点。老师也给满分，毕竟调侃归调侃，老师很热心，帮我们解决了很多问题，也是老师没找到问题所在我才想到是不是pa2没有更新。

# 参考文献

1. DAVID A.PATTERSON(美).计算机组成与设计硬件/软件接口(原书第4版).北京：机械工业出版社.
2. David Money Harris(美).数字设计和计算机体系结构（第二版）. 机械工业出版社
3. 秦磊华，吴非，莫正坤.计算机组成原理. 北京：清华大学出版社，2011年.
4. 袁春风编著. 计算机组成与系统结构. 北京：清华大学出版社，2011年.
5. 张晨曦，王志英. 计算机系统结构. 高等教育出版社，2008年.