***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： | CS1601 |
| 学 号： | U201614526 |
| 姓 名： | 田志伟 |
| 电 话： | 15827451200 |
| 邮 件： | tian\_zw@qq.com |
| 完成日期： | 2020-1-1 |

目 录

[1 课程设计概述 1](#_Toc1147038851)

[1.1 课设目的 1](#_Toc321269638)

[1.2 实验环境 1](#_Toc804917308)

[2 实验过程 2](#_Toc1245703103)

[2.1 PA0 2](#_Toc713977518)

[2.1.1 环境配置过程 2](#_Toc1829009237)

[2.2 PA1 2](#_Toc1077323101)

[2.2.1 总体设计 2](#_Toc1134730650)

[2.2.2 详细设计 2](#_Toc1763249190)

[2.2.3 运行结果 5](#_Toc1802291377)

[2.2.4 问题解答 7](#_Toc854332304)

[2.3 PA2 8](#_Toc687369486)

[2.3.1 总体设计 8](#_Toc1304413989)

[2.3.2 详细设计 8](#_Toc1772844547)

[2.3.3 运行结果 11](#_Toc2010182951)

[2.3.4 问题解答 13](#_Toc932122853)

[2.4 PA3 13](#_Toc295585506)

[2.4.1 总体设计 13](#_Toc775923831)

[2.4.2 详细设计 14](#_Toc1431123033)

[2.4.3 运行结果 16](#_Toc1067969231)

[3 设计总结与心得 17](#_Toc690543791)

[3.1 课设总结 17](#_Toc967866949)

[3.2 课设心得 18](#_Toc1898669465)

[参考文献 19](#_Toc1478187482)

# 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:

* 简易调试器
* 冯诺依曼计算机系统
* 批处理系统
* 分时多任务
* 程序性能优化

## 实验环境

* CPU架构：x64
* 操作系统：GNU/Linux
* 编译器：GCC
* 编程语言：C

# 实验过程

## PA0

### 环境配置过程

首先安装docker，按照pdf中所讲，编写Dockerfile，生成镜像，创建容器。在dokcer中安装相关工具包，练习使用vim。

下载实验框架代码，修改makefile中的学号姓名信息，git submit，并在git origin添加个人git hub，方便个人代码管理和分享。

在后续实验中发现，macOS中的xquartz有问题未解决，无法在docker中运行GUI程序，将实验环境更换为Ubuntu虚拟机，将已完成的代码从github上git clone下来，PA0环境配置到此结束。

## PA1

### 总体设计

利用C语言的联合在reg.h中实现寄存器结构体，之后实现简易调试器。

在nemu/src/monitor/debug目录中实现如下任务。

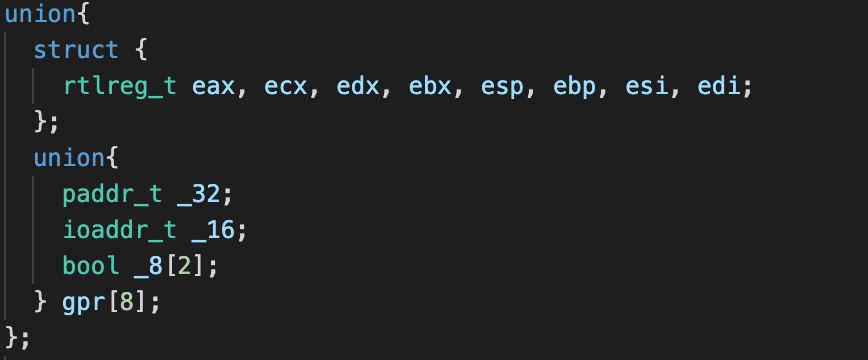
Task PA1.1：实现单步执行cmd\_si，打印寄存器状态cmd\_info，扫描内存cmd\_x

Task PA1.2：实现算数表达式求值cmd\_p

Task PA1.3：实现所有要求cmd\_w，cmd\_d

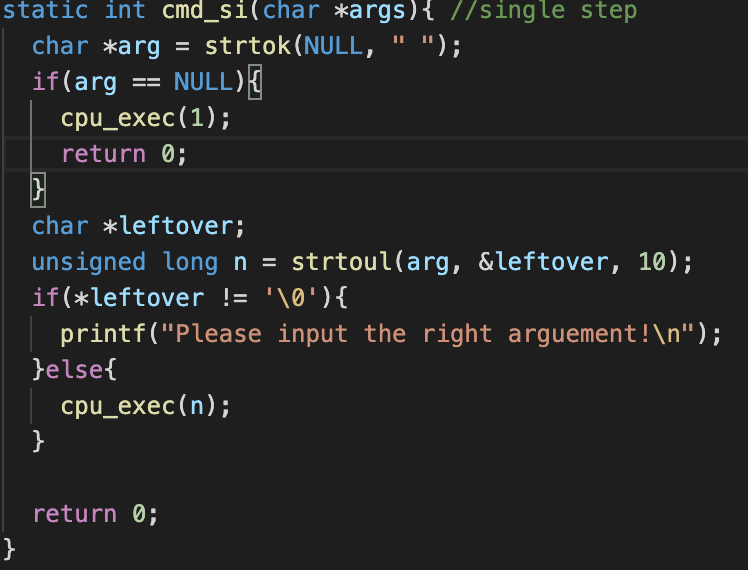
### 详细设计

1. 寄存器



使用联合来公用32位寄存器的低16位和16位寄存器和两个8位寄存器。

1. 单步执行



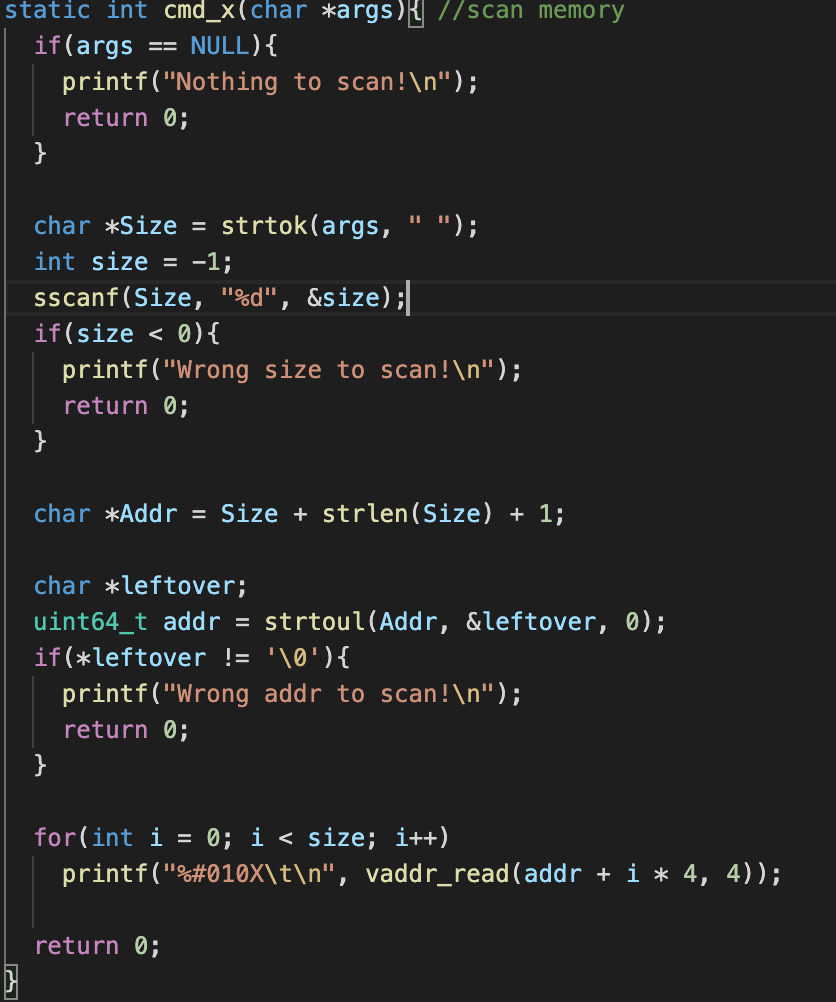
首先判断输入参数是否为空，空则单步执行1，不为空则判断参数是否为数字，为数字则执行n条指令，不为数字则输出报错信息。

1. 打印寄存器状态



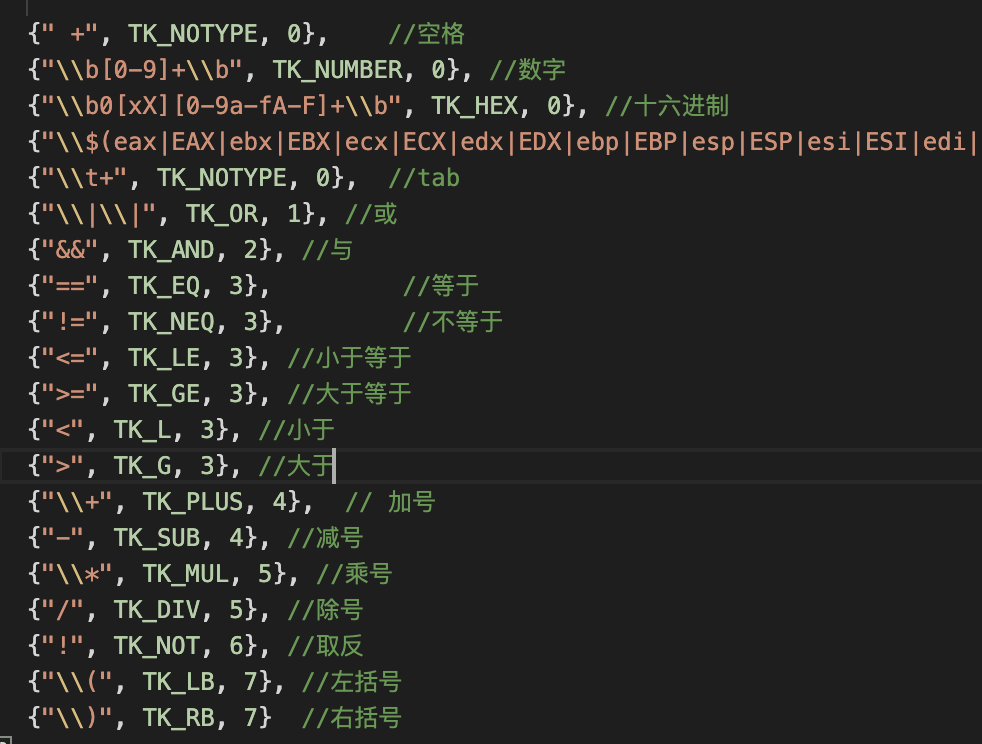
完成打印单个寄存器信息的函数，之后遍历调用即可。

1. 扫描内存



首先读取参数，读取初始地址和读取的长度，若不符合标准则输出报错信息，之后利用vaddr\_read()函数遍历输出内存信息。

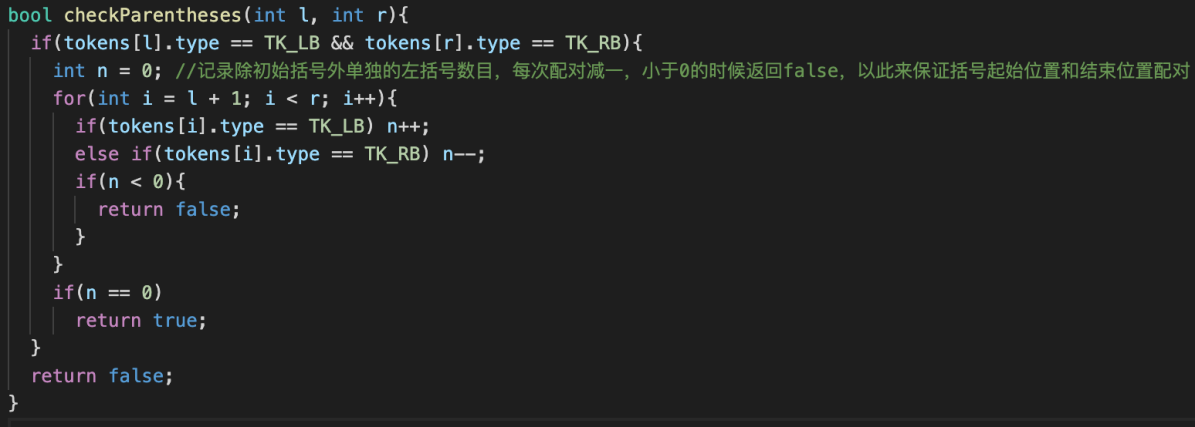
1. 算数表达式求值

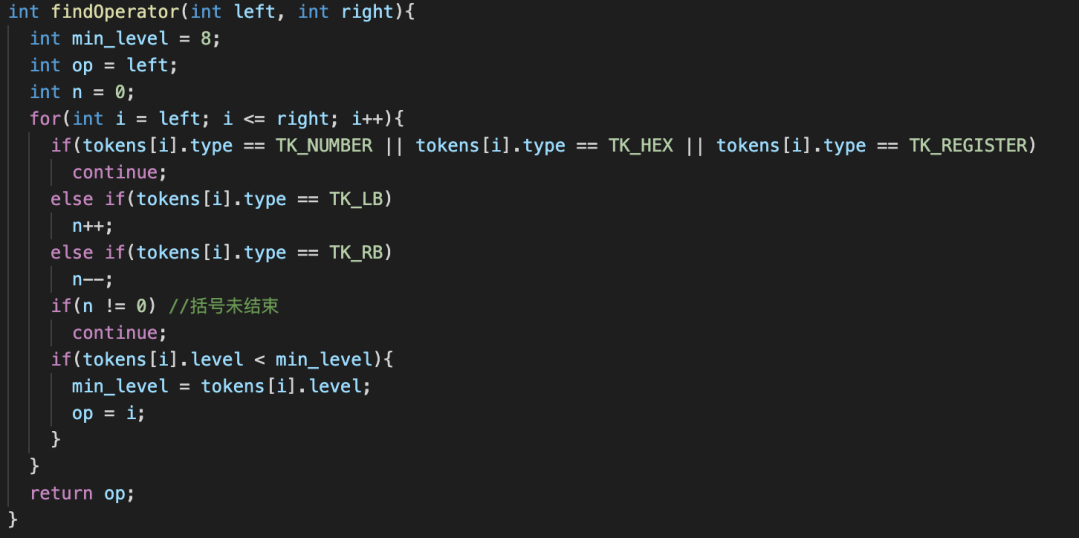


表达式求值过程主要难点在于正则表达式匹配，填写正则表达式中所需要匹配的字符，规则以及算数优先级。

并在expr函数中判断出\*为指针还是乘号，-为负号还是减号。

之后调用caculation函数进行计算，caculation函数递归求值的主题框架指导手册里已经给出，只要按照需求实现判断括号的checkParentheses()函数和寻找主运算符的findOperator()函数即可。



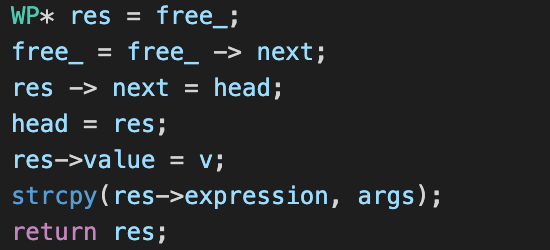


整个计算过程需要注意的是，在判断\*和-时，注意判断寄存器和括号，另外在正则匹配>和>=号时，需要优先匹配>=号，否则会被忽略掉。

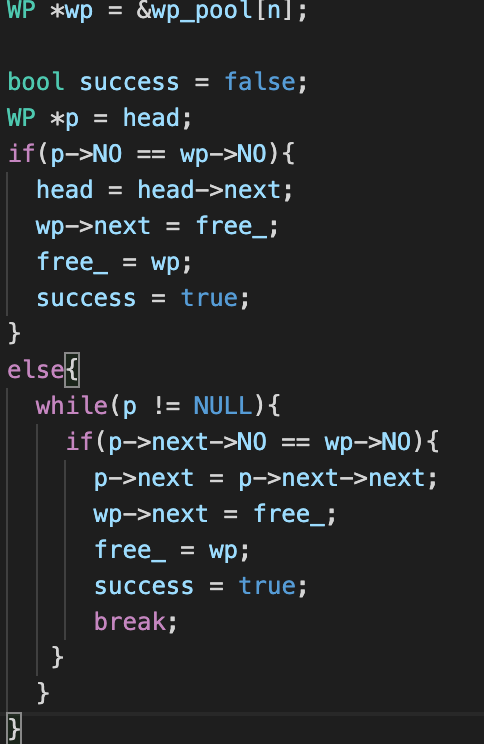
1. 监视点设置和删除

监视点设置和删除主要实现的即链表的增删查功能，比较简单。

增加部分如下：



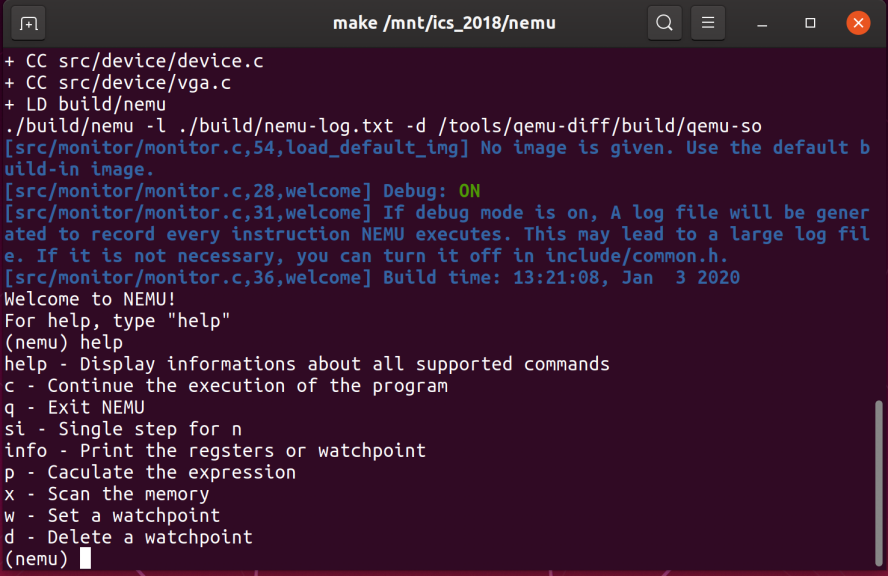
删除部分如下：



到此为止，PA1的主体功能实现完毕。

### 运行结果

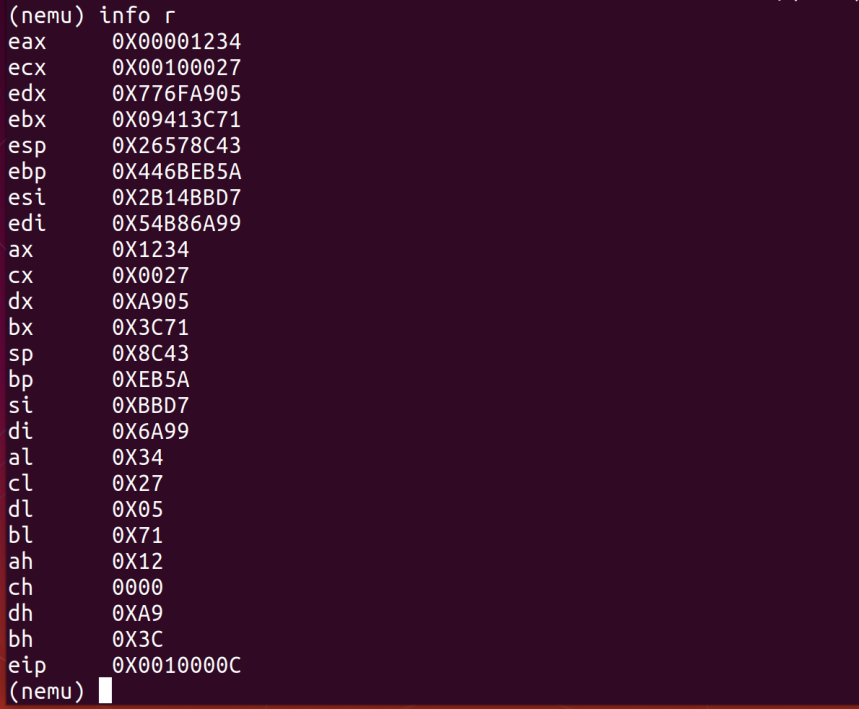
1）初始化运行并执行help



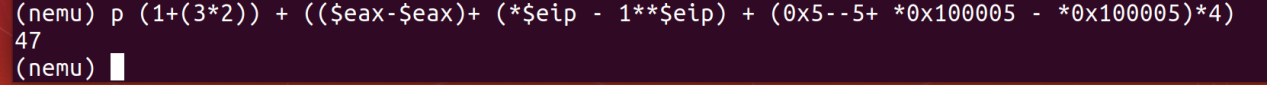
2）单步执行



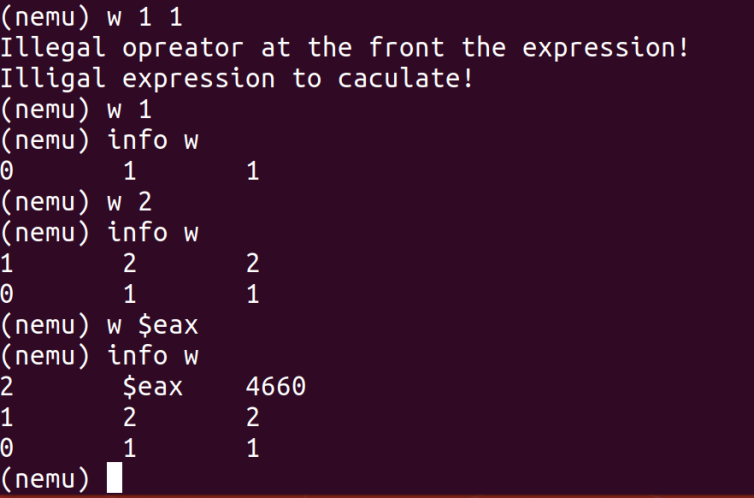
3）打印寄存器信息

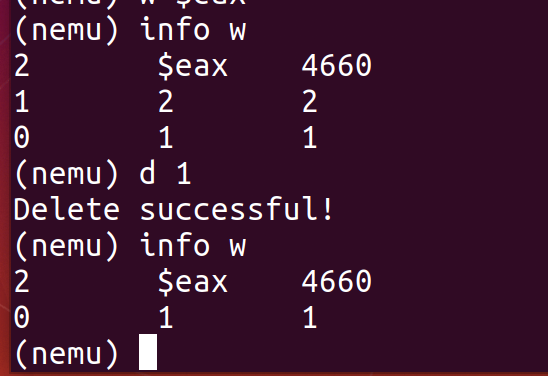


4）表达式求值



5）监视点的设置和删除





所有的功能均实现完毕。

### 问题解答

1. 理解基础设施

500\*90%\*30\*20=270000s=75h

一学期将要花费75个小时的时间在调试上。

500\*90%\*20\*20=180000s=50h

一学期可以节省50个小时调试的时间。

1. 查阅i386手册

CF为进位标志位。

ModR/M字节对于不定长指令用于指定操作的寄存器号和内存。

mov指令eg：88 01

88位opcode指定指令为mov指令，01位ModR/M字节00000001，Mod=00指定1字节，Reg=000即eax/ax/al，R/M=001即ecx，最后的指令即为

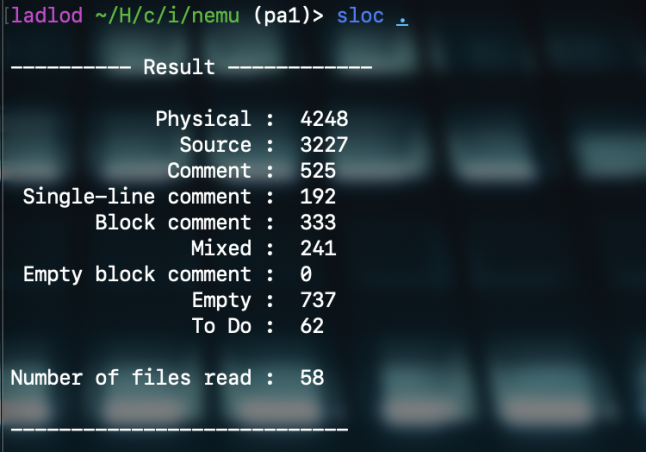
mov ecx，al

1. shell命令

执行find . "(" -name "\*.h" -or -name "\*.c" ")" -print | xargs wc -l指令，即可统计代码总行数，统计结果如下：



和统计代码工具包统计结果基本一致



## PA2

### 总体设计

实现新指令，在opcode\_table中填写正确的译码，执行函数以及操作数宽度，用RTL实现正确的执行函数，完成以下任务：

Task PA2.1：在NEMU中运行第一个c程序dummy

Task PA2.2：实现更多指令，在NEMU中运行所有cputest

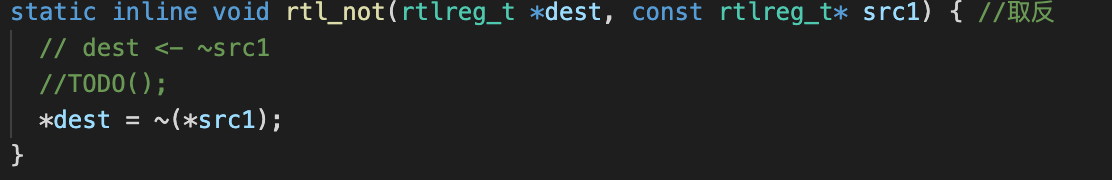
通过内存映射完成io功能，完成以下任务：

Task PA2.3：运行打字游戏，ppt播放，超级玛丽

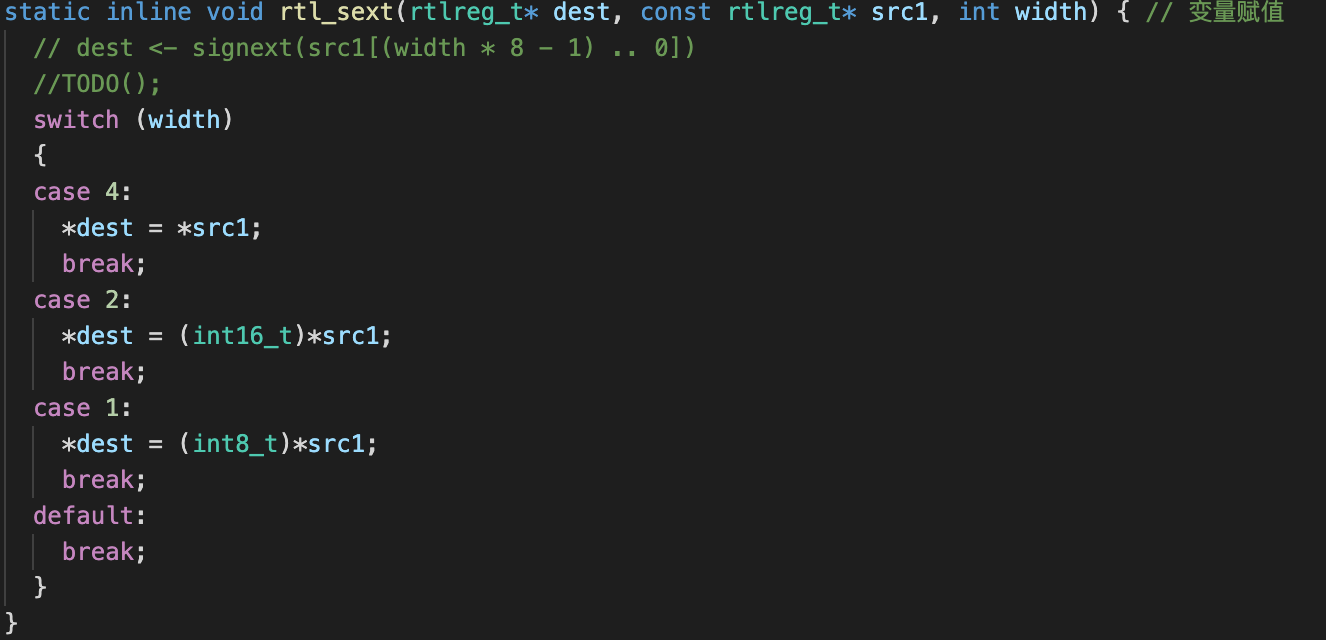
### 详细设计

1）首先实现未实现的rtl：

取反



变量赋值



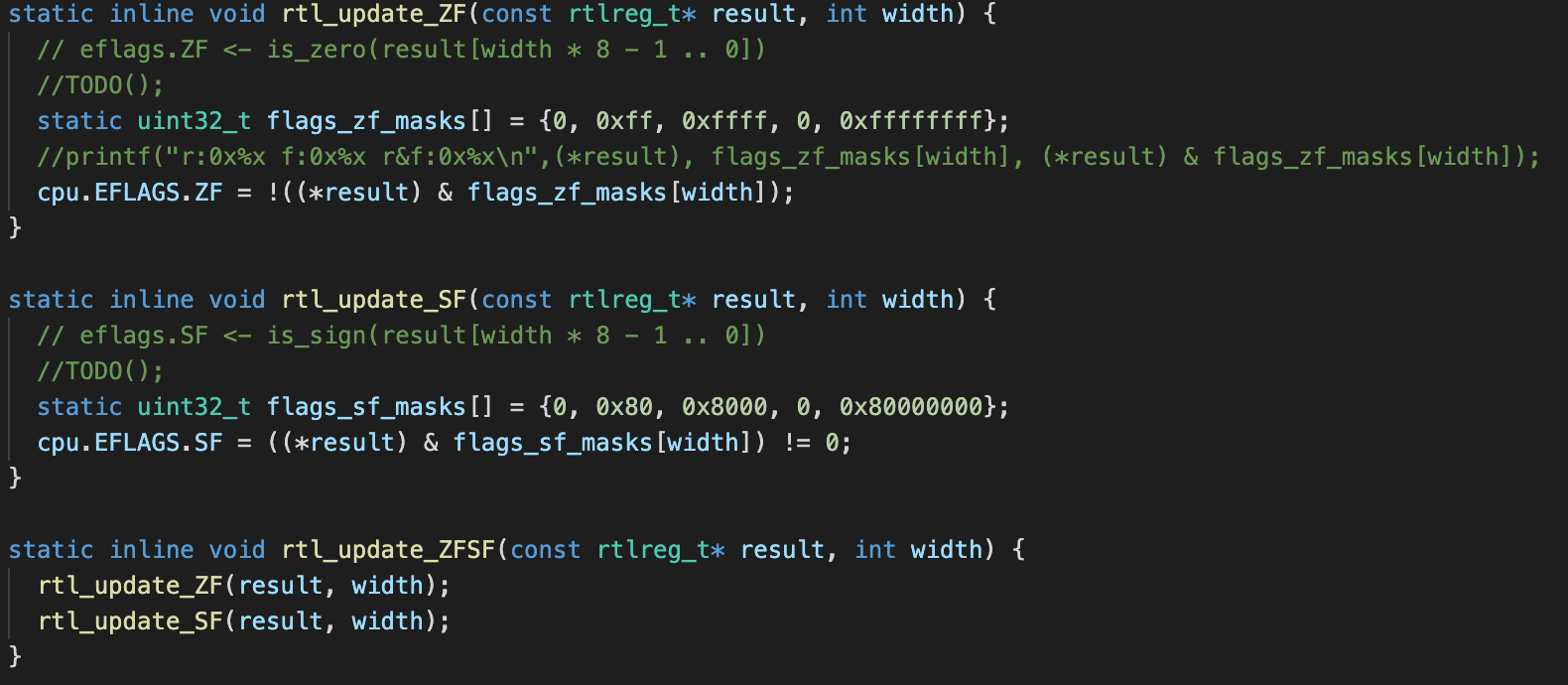
入栈



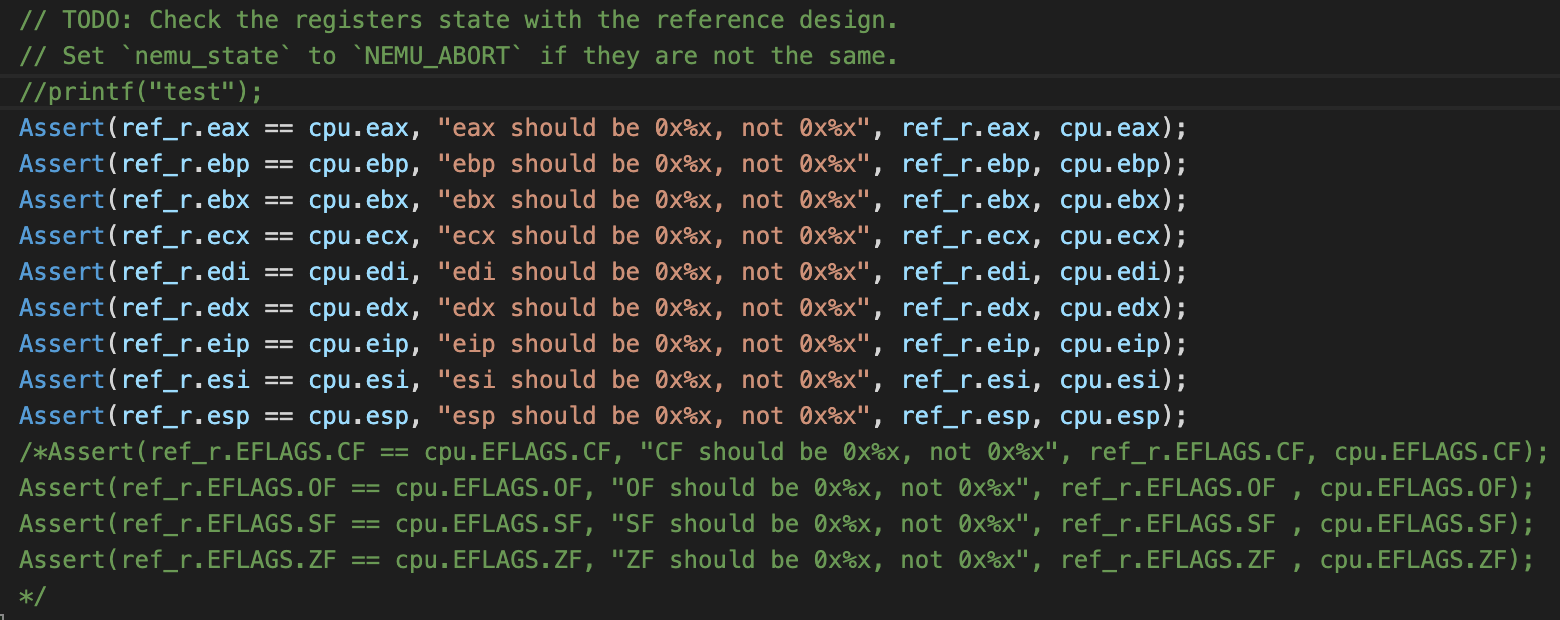
出栈



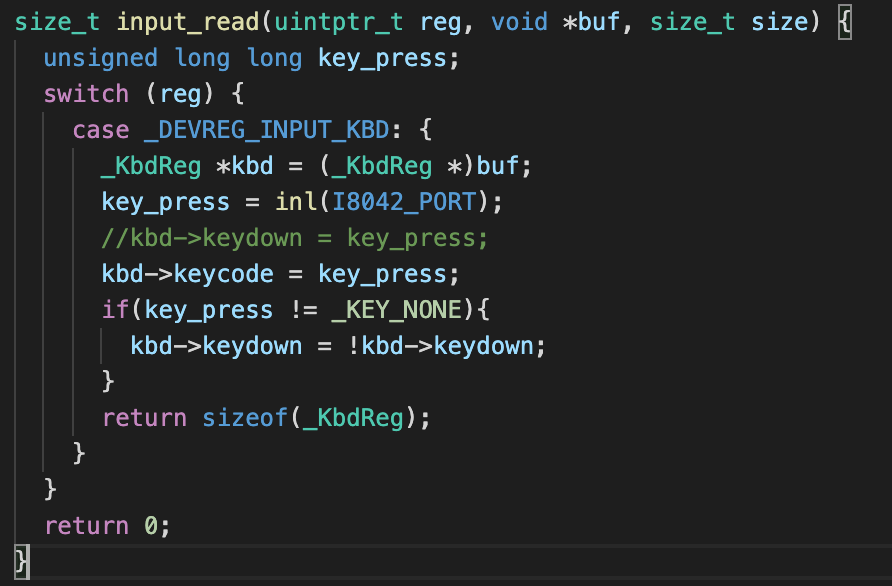
更新标志位

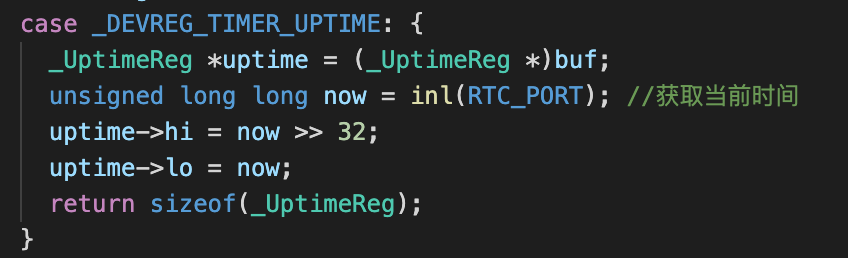


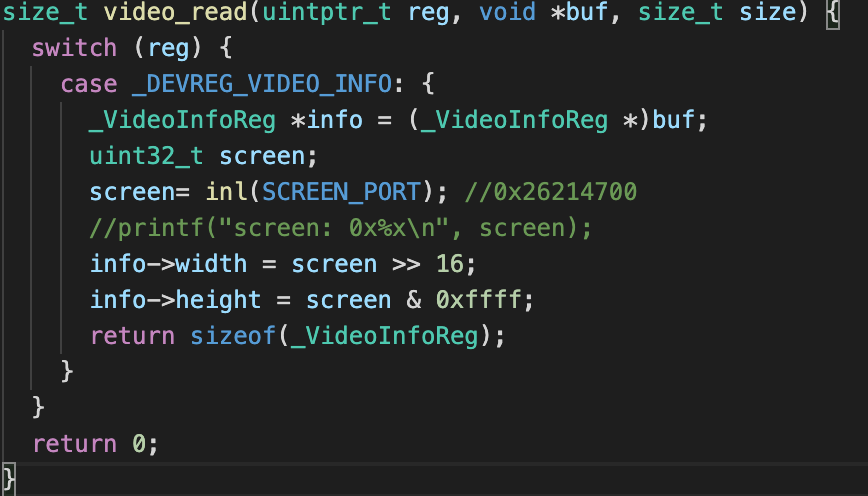
1. 参照i386手册，在opcode table中填写需要用到的指令并在all-instr.h头中添加需要实现的执行函数，分别在exec目录下的各个文件中实现这些执行函数。
2. 完成klib中的printf函数和string.c中的函数
3. 在上述过程中，极易出现译码填写或执行函数编写错误的情况，由于一开始没有完成difftest模块，每个bug调试起来都需要几个小时的时间，所以完成difftest模块，在common.h中定义DIFFTEST，在diff-test.c中实现difftest，代码如下：



1. 完成src/device目录下的input.c,timer.c,video.c中的功能函数，将设备抽象为IOE





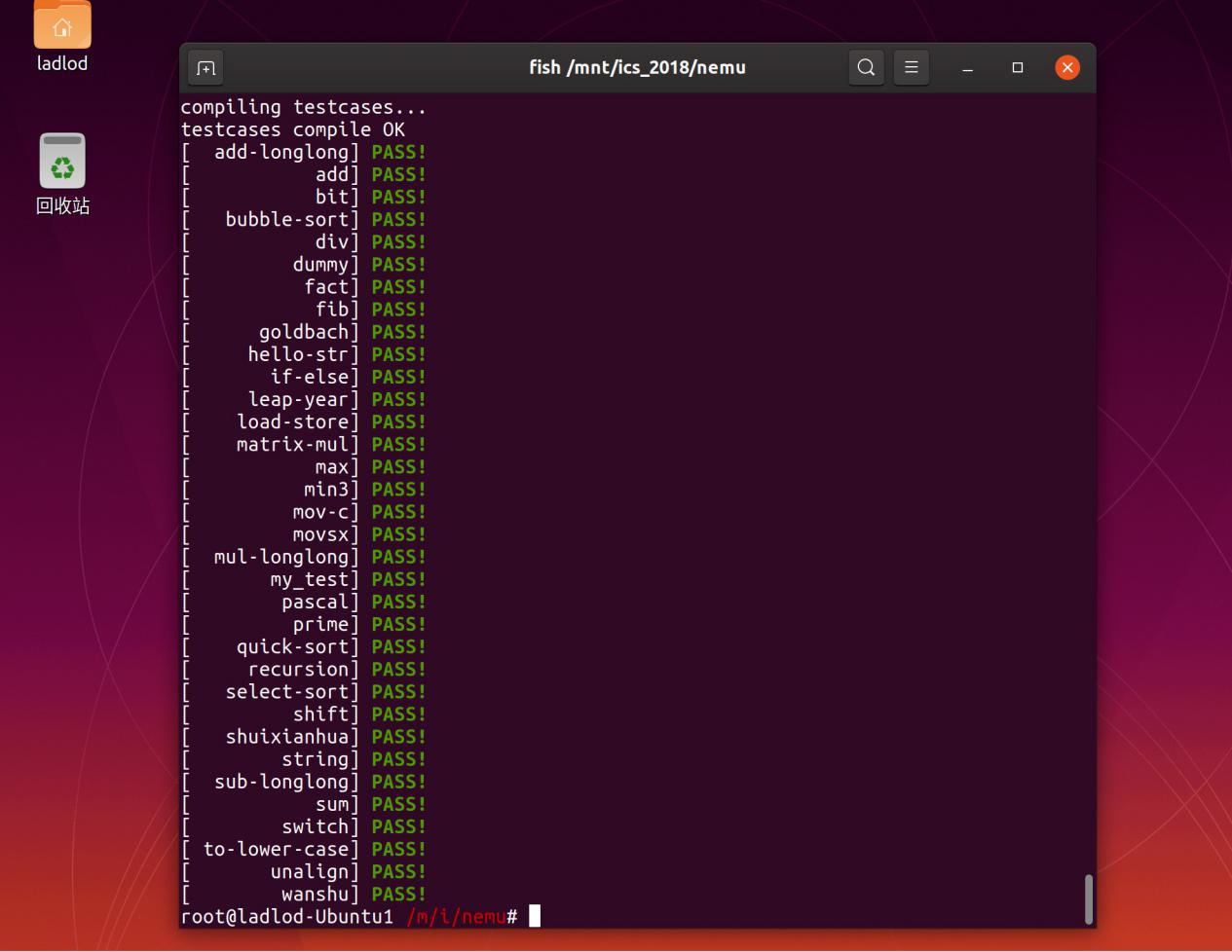


完成以上功能之后，即可调试运行microbanch以及打字游戏等测试程序。

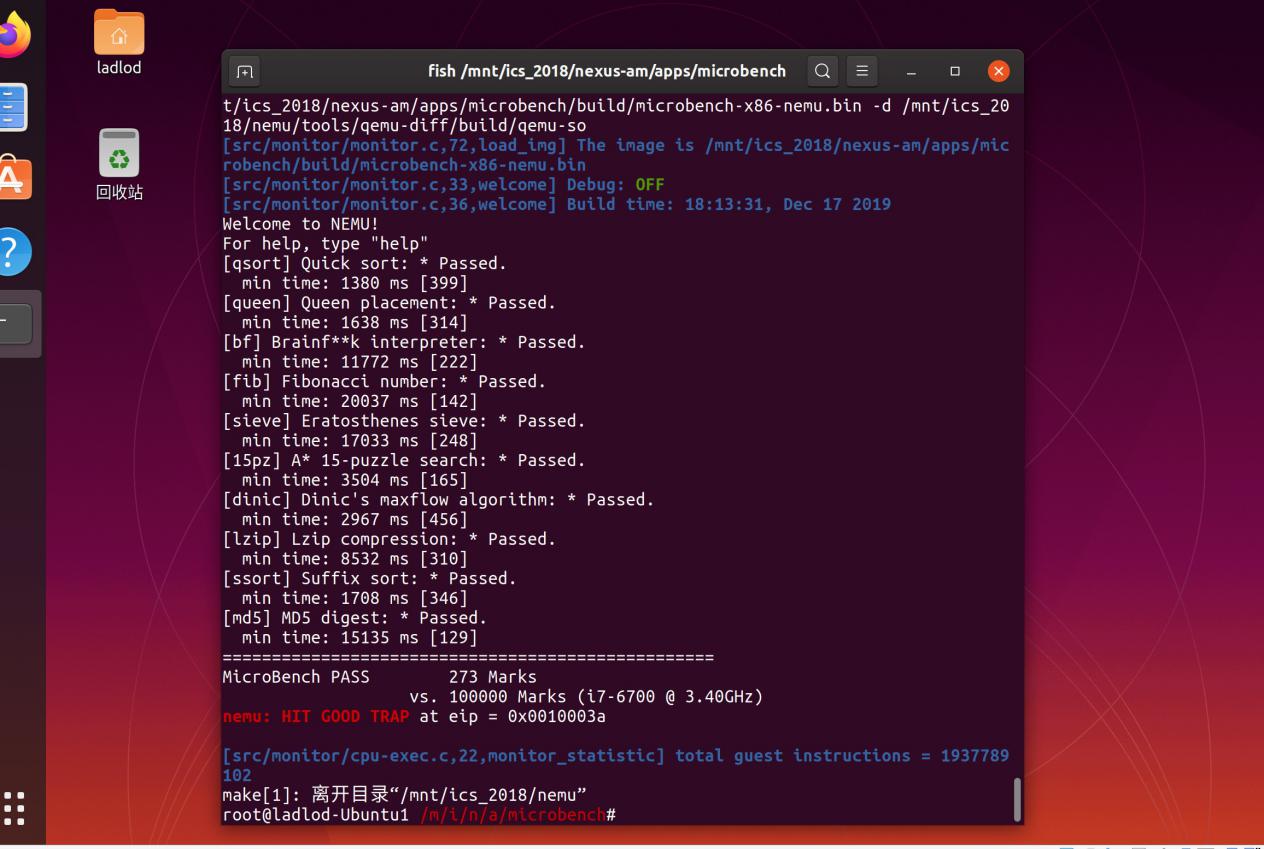
到此为止，PA2的主体功能实现完毕。

### 运行结果

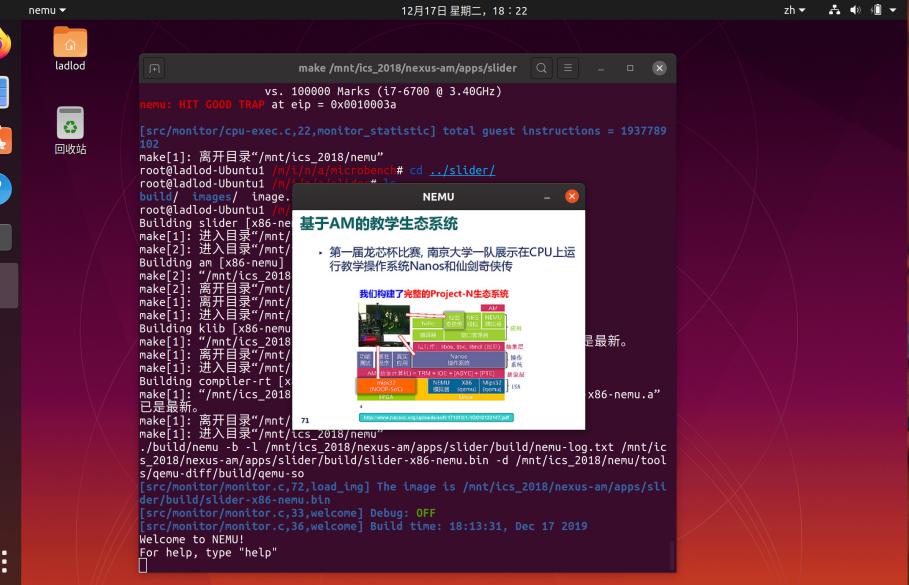
1. Runall



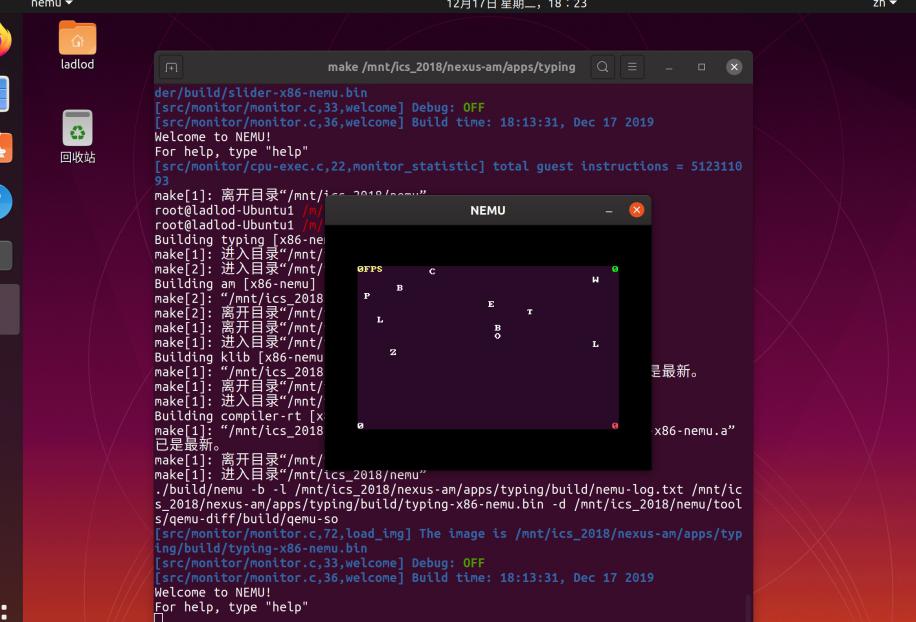
1. Microbanch



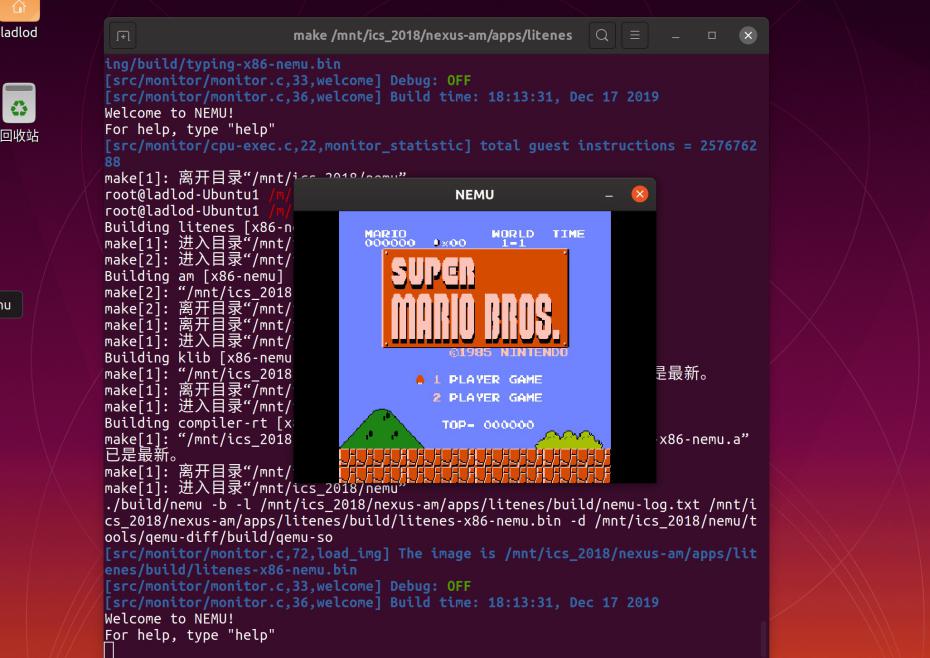
1. ppt演示



1. typing游戏



1. 超级玛丽



PA2全部测试样例通过。

### 问题解答

1. 编译与链接：inline关键字表示建议编译器进行函数内联，但并不强制内联，非内联函数在多个c文件中引用会导致重定义，所以不可以去掉inline，在inline前面加上static表示让该函数只在本文件中可以被识别，在函数没有被inline内联时可以防止重定义，代码健壮性更高。
2. 编译与链接：
3. 一个
4. 两个

static关键字定义表示只有在本文件中可以被识别，而volatile表示此处不被编译器优化，所以重新声明的变量不会覆盖之前的dummy。

1. 出现重定义



1. 了解Makefile：
2. 首先依次读取变量“MAKEFILES”定义的makefile文件列表
3. 读取工作目录下的makefile文件
4. 一次读取工作目录下的makefile文件指定的include文件
5. 查找重建所有已读的makefile文件的规则
6. 初始化变量值并展开那些需要立即展开的变量和函数并根据预设条件确定执行分支
7. 建立依赖关系表
8. 执行rules

## PA3

### 总体设计

继续实现指令，重新组织arch.h中的\_Context结构体，完成int事件分发，实现系统调用，完成以下任务。

PA 3.1：实现自陷操作\_yield()及其过程

PA 3.2：实现用户程序的加载和系统调用

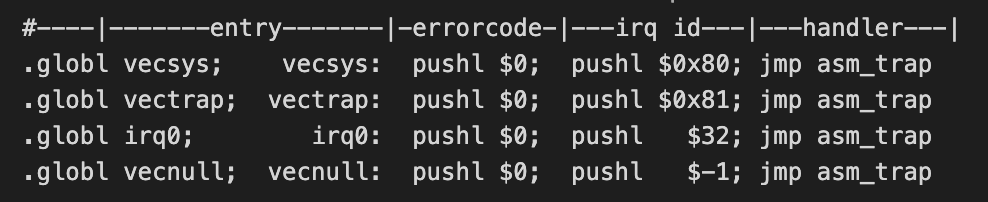
实现简易文件系统，完成以下任务。

PA 3.3：运行仙剑奇侠传并展示批处理系统（未完成）

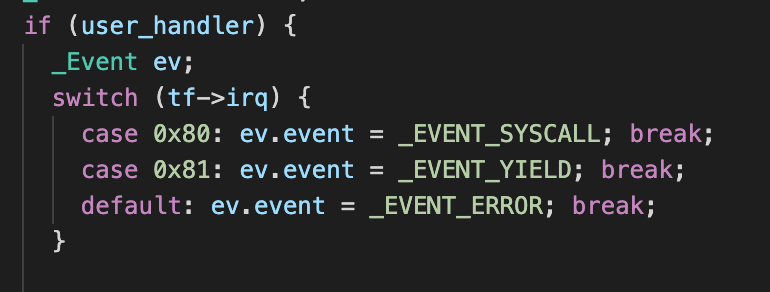
### 详细设计

重新组织arch.h中的Context结构体使之与trap.S中的上下文对应

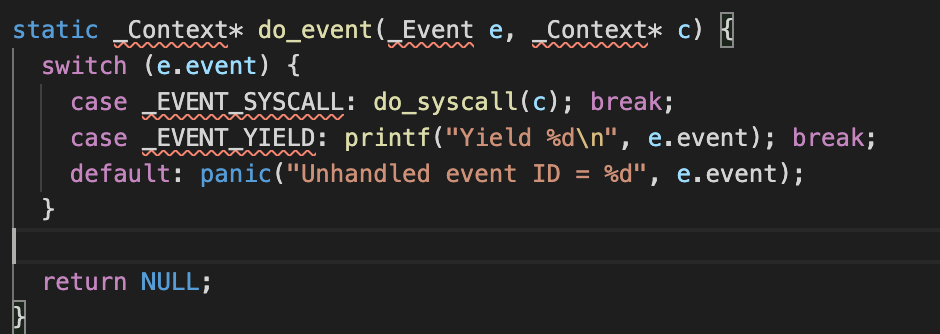




在cte.c中完成irq.c实现对0x80系统调用和0x81自陷的事件分发

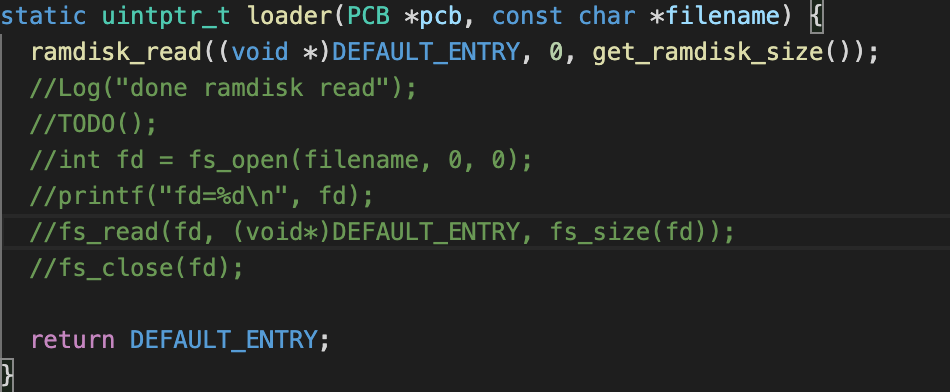


在irq.c中完成对自陷的处理



到此完成PA3.1

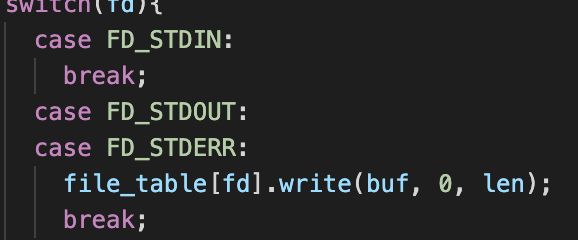
实现loader函数，将从0开始，size为ramsize的内存读取到0x4000000位置即可。



运行结果即为从ramdisk中读取的dummy程序。

在syscall.c中实现sys\_write系统调用，将从0开始len长的字符串写入到输出串口即可。

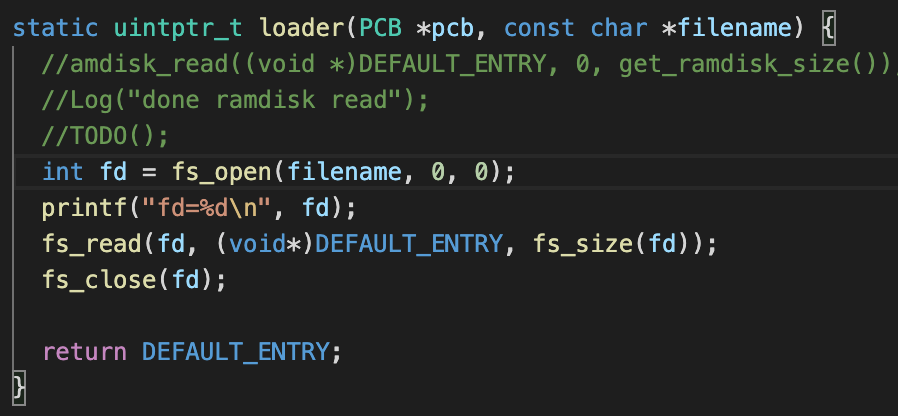




修改makefile，从ramdisk中读取helloworld程序进调试。

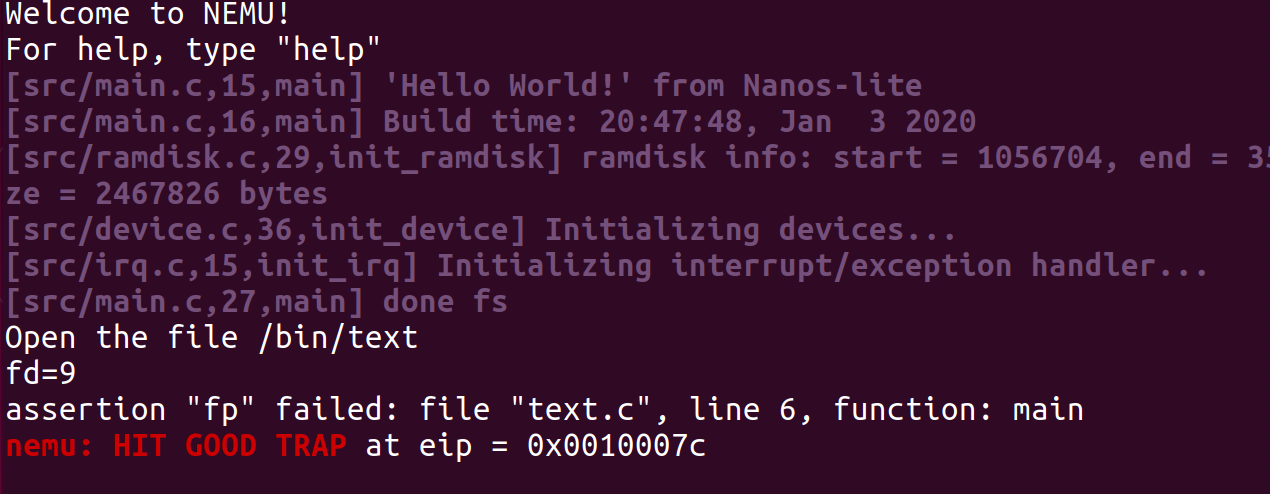
PA2.2到此结束。

实现fs\_open，fs\_read，fs\_write等函数，修改loader读取文件。



修改makefile，从ramdisk中读取text程序进行调试。

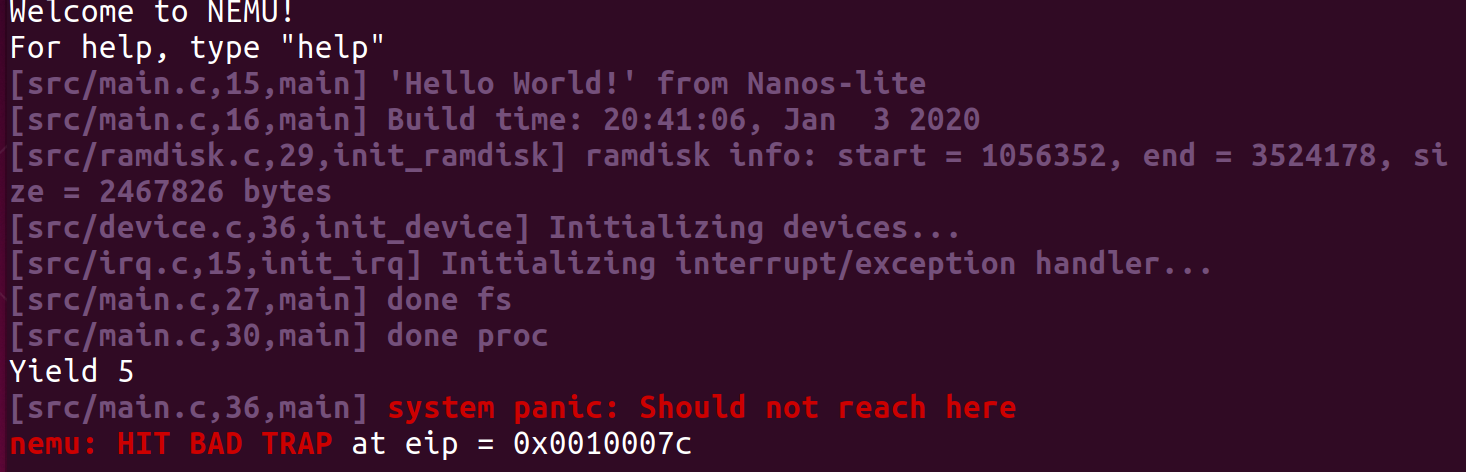
结果在text程序中出现以下错误



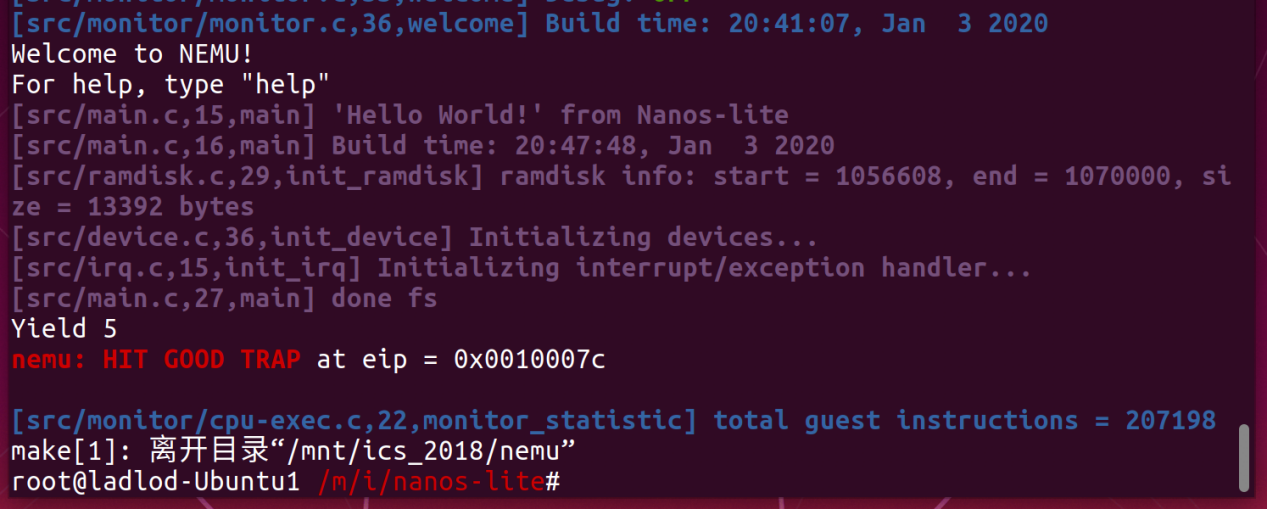
未找到解决方法。

### 运行结果

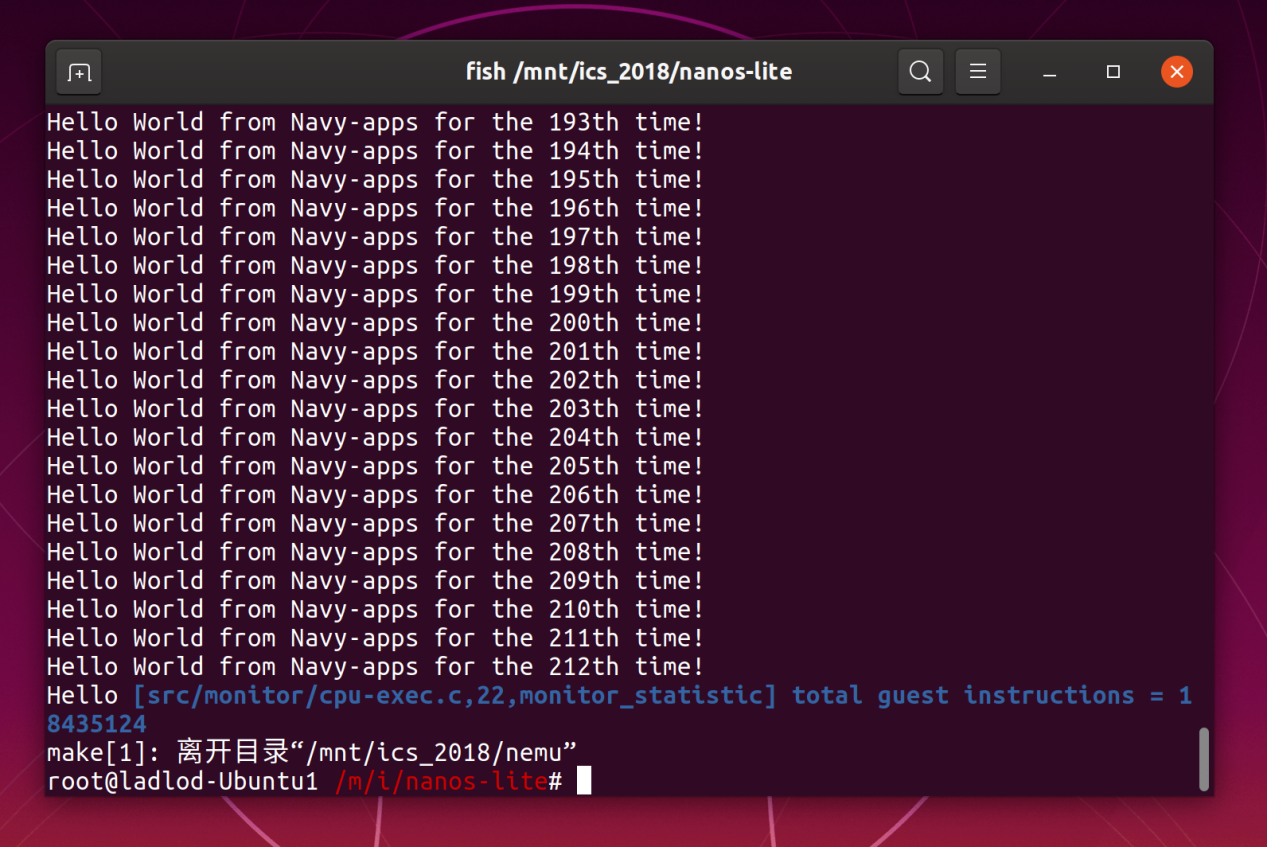
yield测试：



dummy测试：



系统调用输出hello测试：



后续实验没有完成。

# 设计总结与心得

## 课设总结

本次课程设计总体难度较高，耗时较长，编写过程中遇到过如下问题：

在PA1的编写过程中，主要难点在于运算器部分，运算器主要采用的的想法是递归，在递归之前需要先对字符串进行正则匹配处理，在递归时需要对运算符进行括号判断，主运算符选取。在正则匹配过程中，需要注意的是\*可以代表乘号也可以代表指针，-可以代表减号也可以代表负号，需要单独处理，处理的想法是如果这两个运算符前面是数字则为乘号或减号。但实际处理过程中需要考虑到，右括号，寄存器，也可以处理为数字，在实验开始时未考虑到这一点，导致实验出错。另外，在正则匹配过程判断>号和>=号时，需要注意优先判断>=号，如果先判断>号则会将>=号判断为一个>号和一个未知类型的=号，导致程序出错。在括号是否匹配判断过程的主要想法是判断左括号个数是否与右括号相等以及过程中是否出现过右括号数量大于左括号的情况，这种想法没有考虑到() + ()的情况，会将第一个括号和最后一个括号判断为匹配，所以将判断条件更改为，处理过程中左括号至少比右括号多一个，直到最后一个再跳出判断。

在PA2的编写过程中，主要难点在于opcode table的填写和rtl中关于符号位寄存器更新的实现。opcode table的填写比较耗费耐心，尤其注意操作数位数的选择，因为在选错操作数位数时，可能当时的一条指令没有出现问题，在后续过程出现了问题导致很难找出bug。rtl中关于负号位更新主要难点在于进位标志位和溢出标志位的判断，溢出主要针对有符号数，进位主要针对无符号数，在有符号数的溢出判断中，尤其需要注意对0的判断，在进行0 - 128运算时，结果为正数，应该发生了溢出。在调试过程中，一开始没有完成difftest部分，出现opcode table填写错误的bug时很难找出bug，甚至出现了一个bug找一天的情况，后来实现了difftest，大幅度提高了改代码的效率。在调试死循环过程中，我在jmp和j指令中将跳转信息打印出来，也对调试过程有很大帮助。

在PA3的编写过程中，主要难点在于对代码框架的理解，理解了代码结构之后，编写代码的过程进行的很快。但在读ramdisk时因为编译器的原因disk过大，导致实验停滞了几天才继续进行。在后续PA3.3的文件系统调试text的过程中，出现了函数fopen()返回NULL的问题，经过测试，问题出现在fopen.c中，\_fopen\_r()函数的一个判断条件，\_sfp(ptr)==NULL，而ptr为框架代码中的\_REENT，所以怀疑问题依然出现在ramdisk的生成过程中，问题没办法解决，实验停滞在了这一步。

## 课设心得

本次实验是对我们整个大学四年所学知识的综合性应用考察，考察了我们对汇编语言，编译原理，组成原理，操作系统，还有c语言的掌握理解程度，是我们第一次进行综合性的大型实验。实验材料来自南京大学，实验进度安排科学合理，指导手册清晰，老师的指导也十分详细负责，实验过程非常愉快。我在实验的前半阶段进度较快，而在完成PA2之后有了些懈怠，导致PA3最终没有完成，比较遗憾。

总的来说，本次实验是对我们四年所学的综合性检验，也是对我们动手能力，学习能力的检验，在实验中使用到的工具比如vim，git，make都会对我们未来的工作生活有巨大的帮助。

# 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.