***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](#_Toc2129209347)

[1.1 课设目的 1](#_Toc2101485068)

[1.2 实验环境 1](#_Toc2143479314)

[2 实验过程 2](#_Toc1418651973)

[2.1 PA0 2](#_Toc1920261217)

[2.1.1 环境配置过程 2](#_Toc1446027003)

[2.2 PA1 2](#_Toc303406322)

[2.2.1 总体设计 2](#_Toc1223875876)

[2.2.2 详细设计 2](#_Toc1083476966)

[2.2.3 运行结果 5](#_Toc1483524649)

[2.2.4 问题解答 7](#_Toc1313634073)

[2.3 PA2 8](#_Toc2115973751)

[2.3.1 总体设计 8](#_Toc841638737)

[2.3.2 详细设计 8](#_Toc2094953617)

[2.3.3 运行结果 8](#_Toc1891048354)

[2.3.4 问题解答 8](#_Toc91710078)

[2.4 PA3 8](#_Toc1625506047)

[2.4.1 总体设计 8](#_Toc1740658442)

[2.4.2 详细设计 8](#_Toc76711613)

[2.4.3 运行结果 8](#_Toc801891491)

[2.4.4 问题解答 8](#_Toc1930404312)

[3 设计总结与心得 9](#_Toc122332908)

[3.1 课设总结 9](#_Toc907334577)

[3.2 课设心得 9](#_Toc290858292)

[参考文献 10](#_Toc782533072)

# 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。

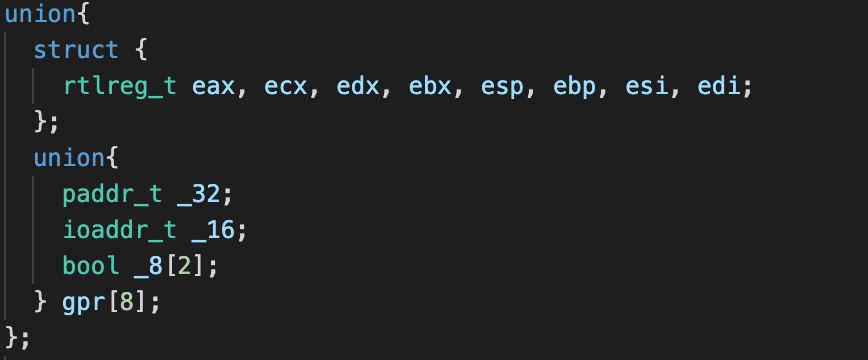
Task PA1.1：实现单步执行cmd\_si，打印寄存器状态cmd\_info，扫描内存cmd\_x

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

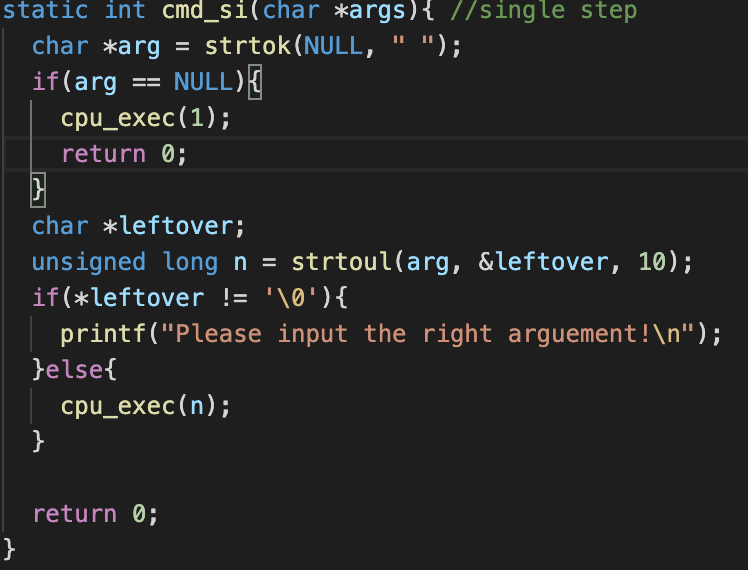
### 详细设计

1. 寄存器



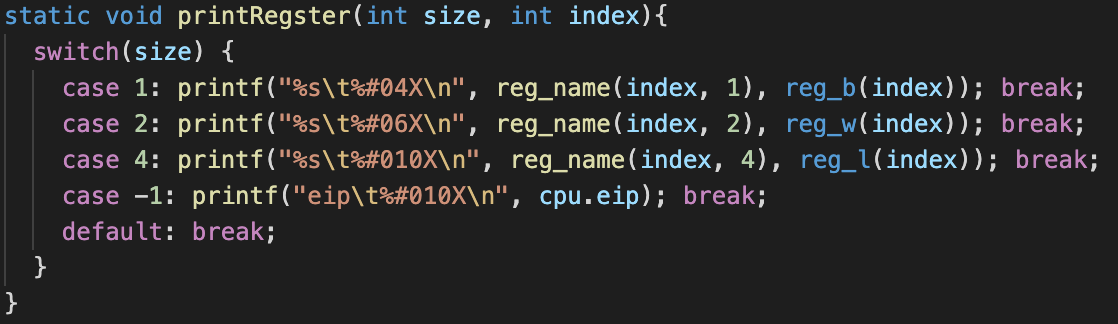
使用联合来公用32位寄存器的低16位和16位寄存器和两个8位寄存器。

1. 单步执行



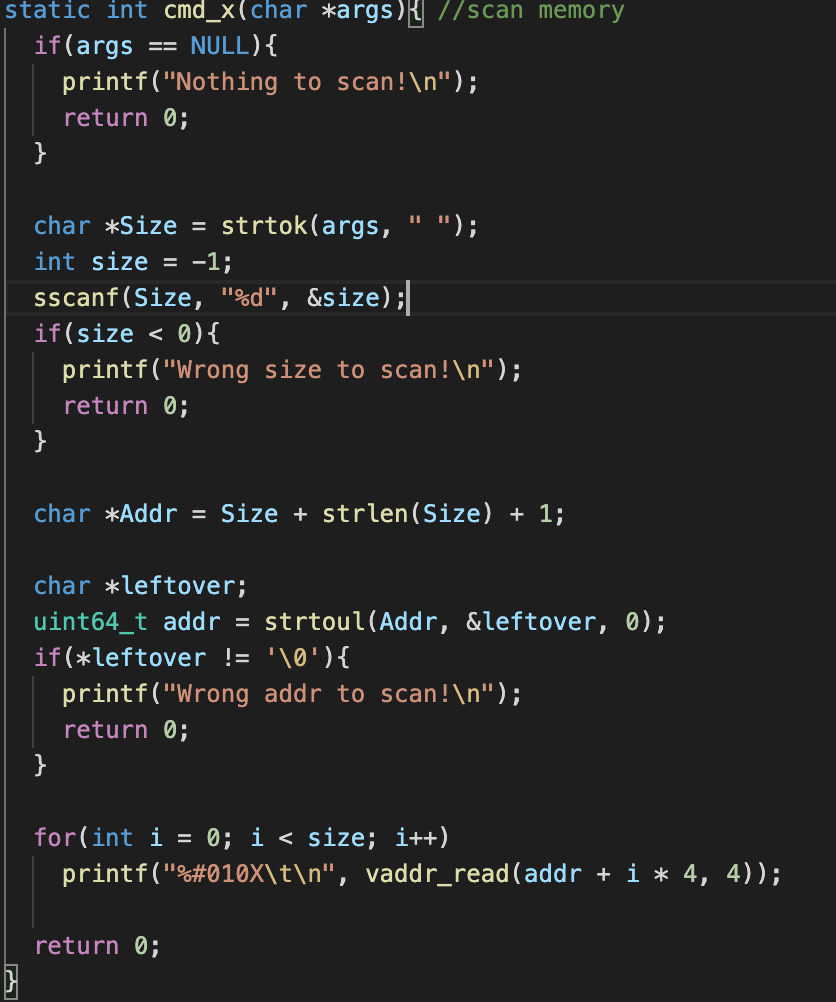
首先判断输入参数是否为空，空则单步执行1，不为空则判断参数是否为数字，为数字则执行n条指令，不为数字则输出报错信息。

1. 打印寄存器状态



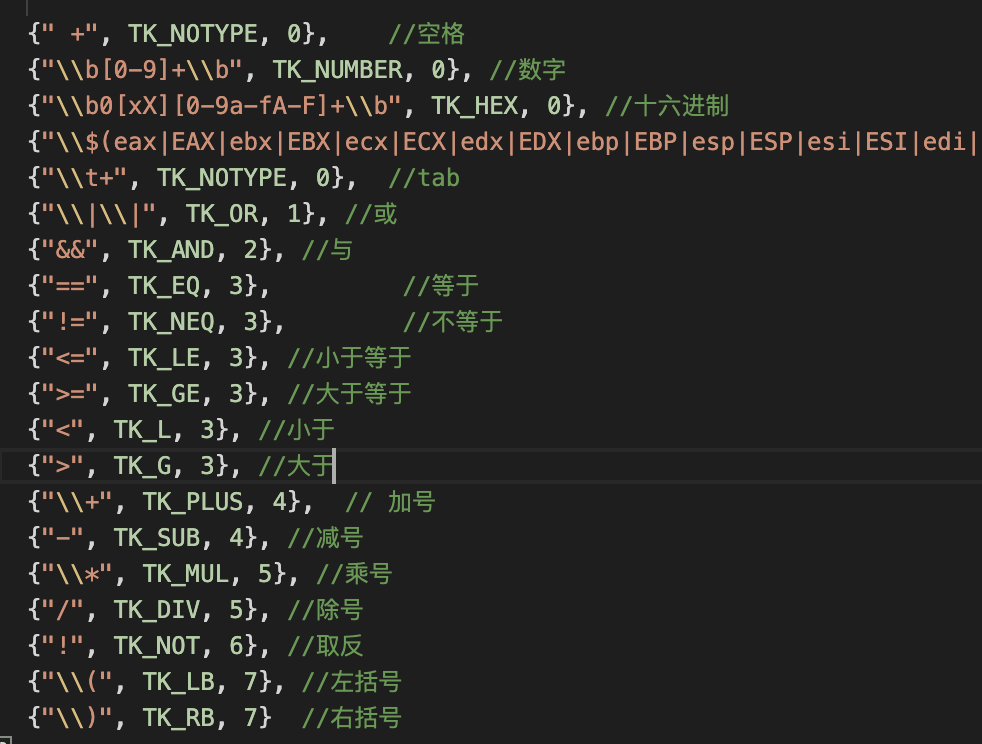
完成打印单个寄存器信息的函数，之后遍历调用即可。

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。

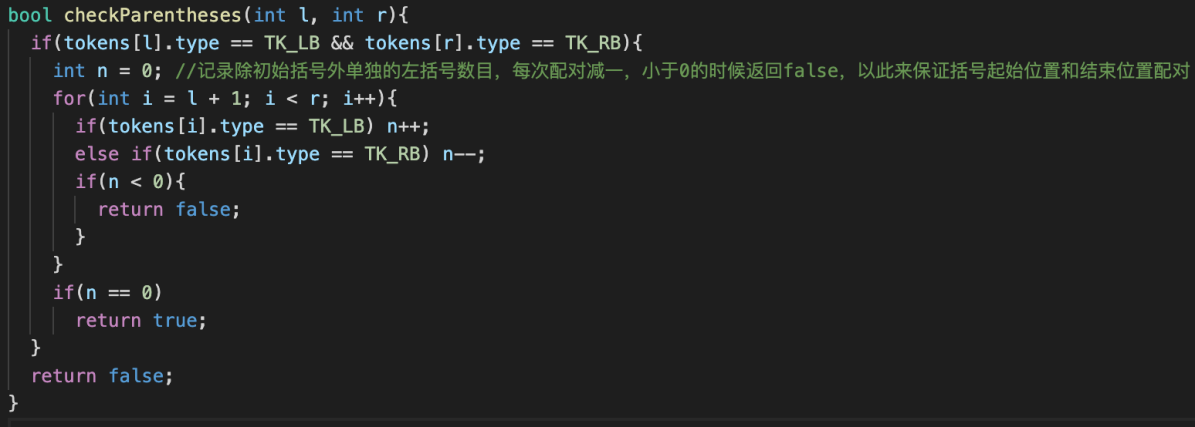
1. 算数表达式求值

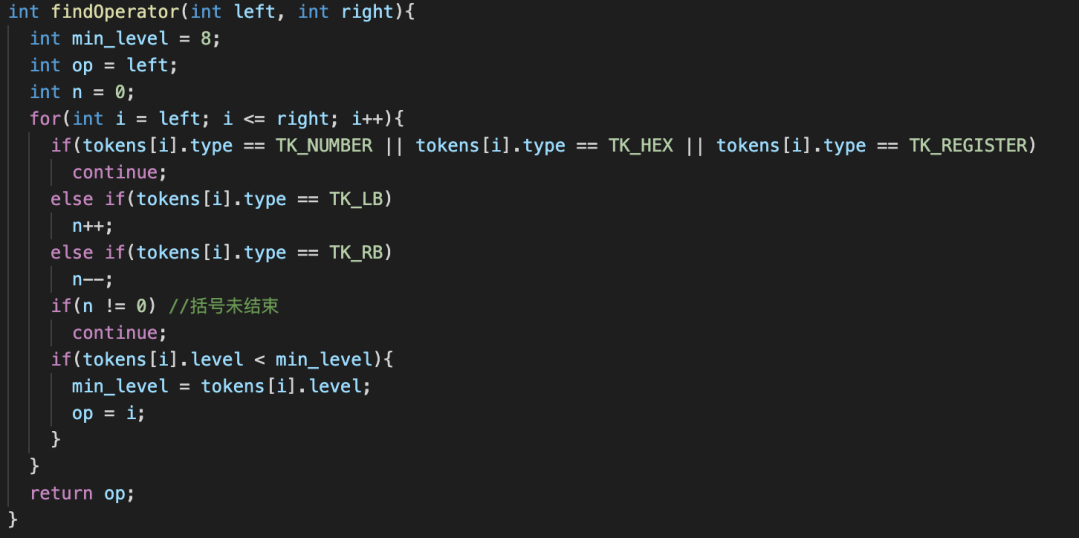


表达式求值过程主要难点在于正则表达式匹配，填写正则表达式中所需要匹配的字符，规则以及算数优先级。

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。



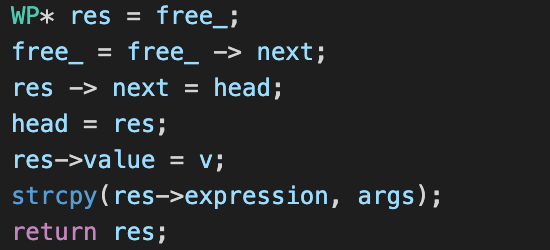


整个计算过程需要注意的是，在判断\*和-时，注意判断寄存器和括号，另外在正则匹配>和>=号时，需要优先匹配>=号，否则会被忽略掉。

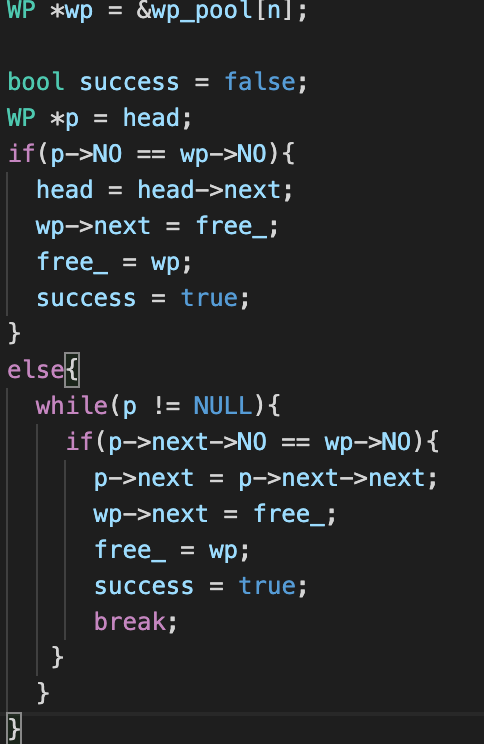
1. 监视点设置和删除

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：



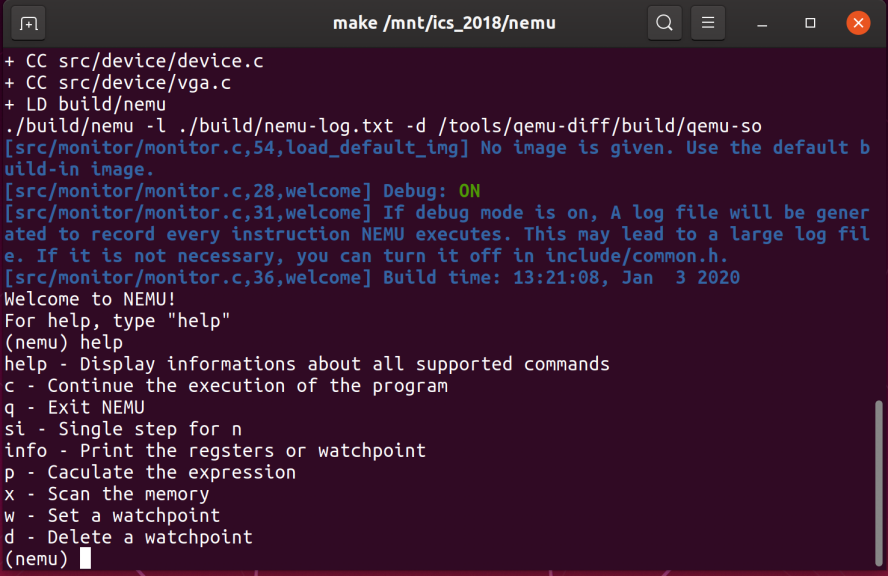
删除部分如下：



到此为止，PA1的主体功能实现完毕。

### 运行结果

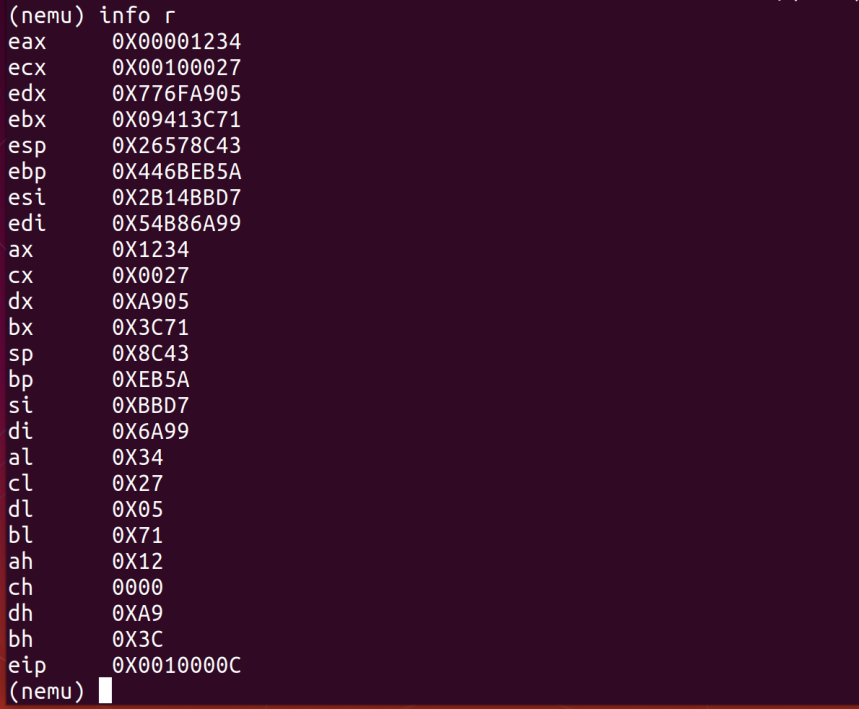
1）初始化运行并执行help



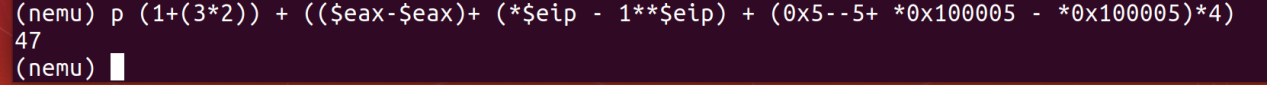
2）单步执行



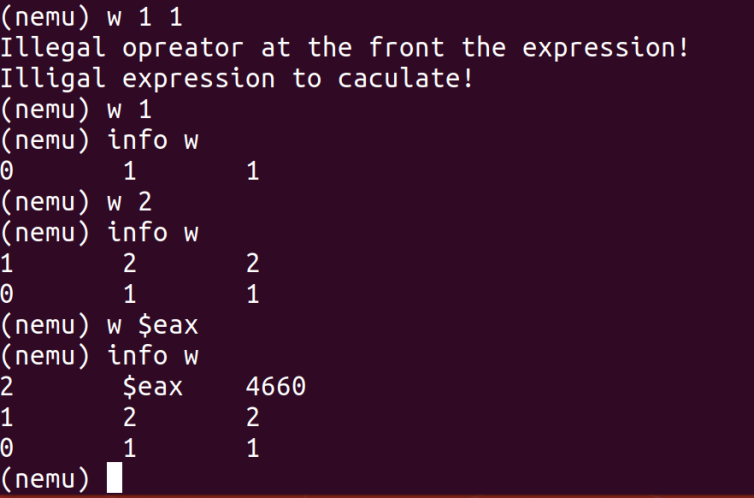
3）打印寄存器信息

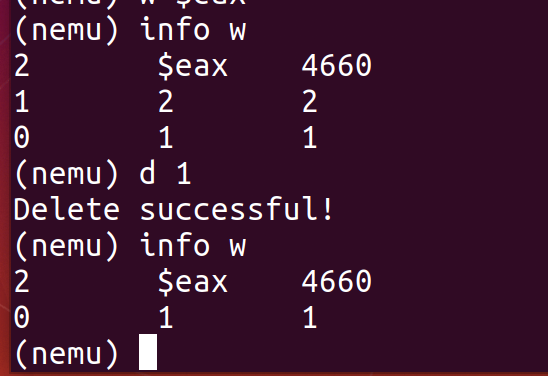


4）表达式求值



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

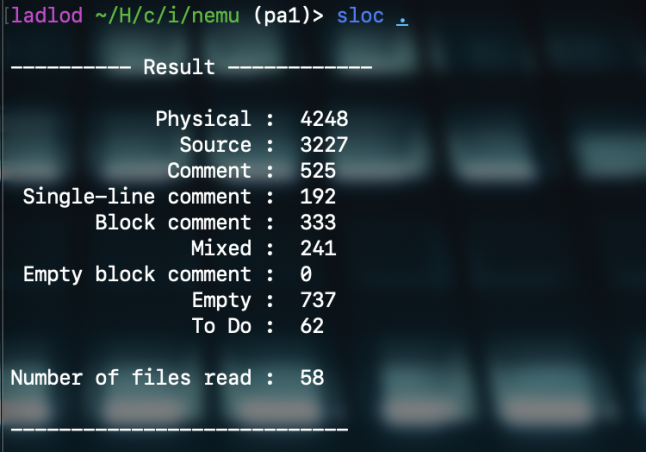
mov ecx，al

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

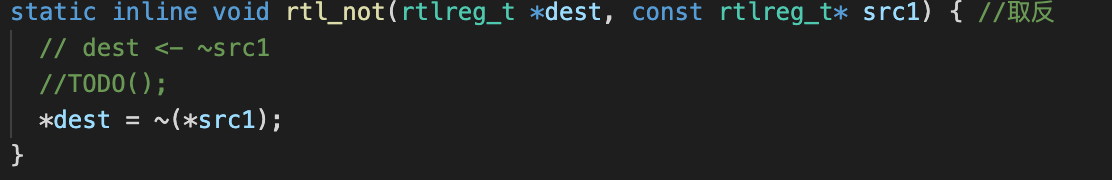
通过内存映射完成io功能，完成以下任务：

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



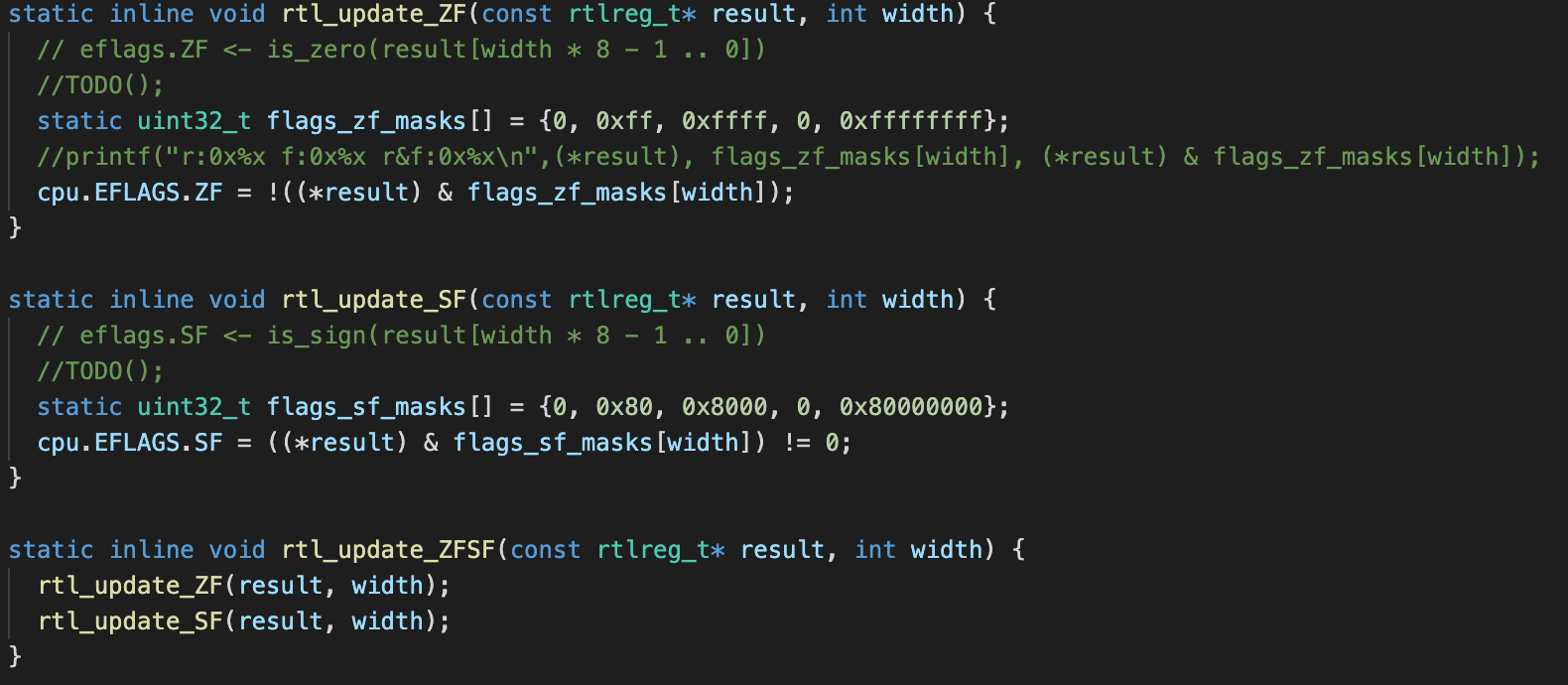
入栈



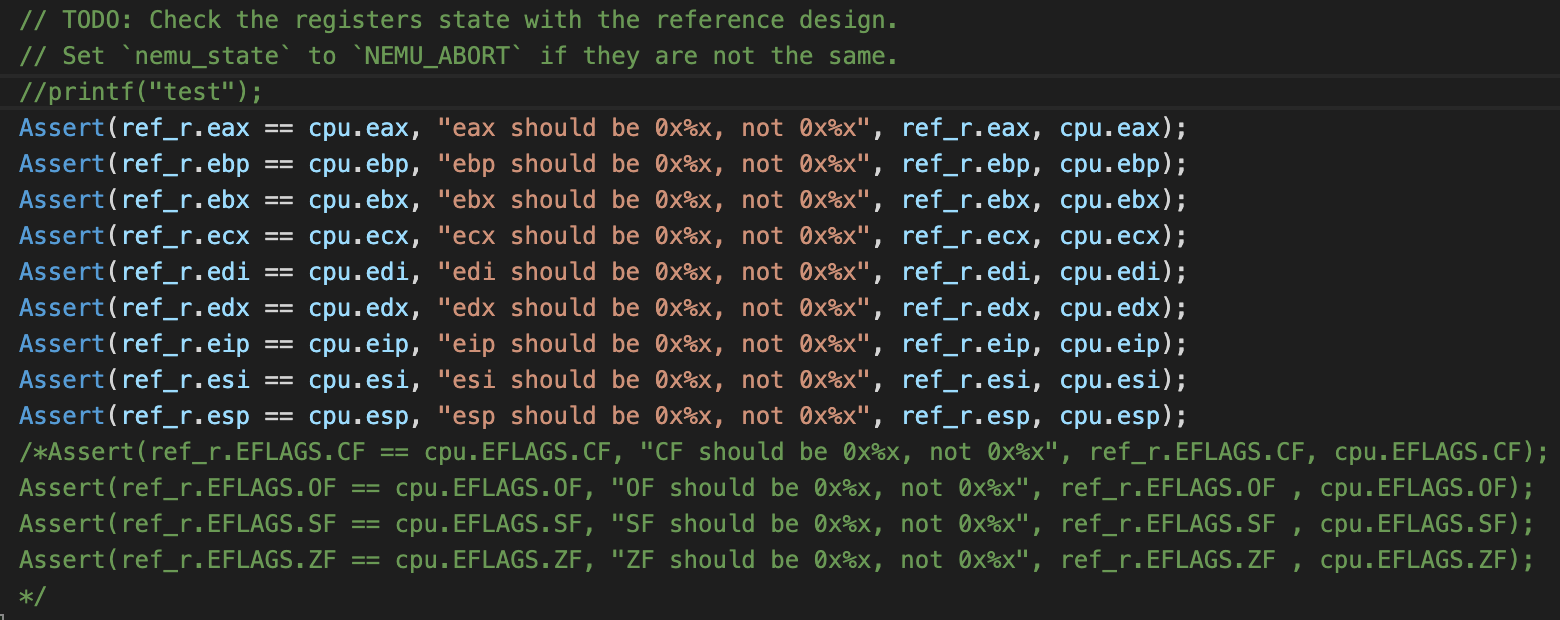
出栈



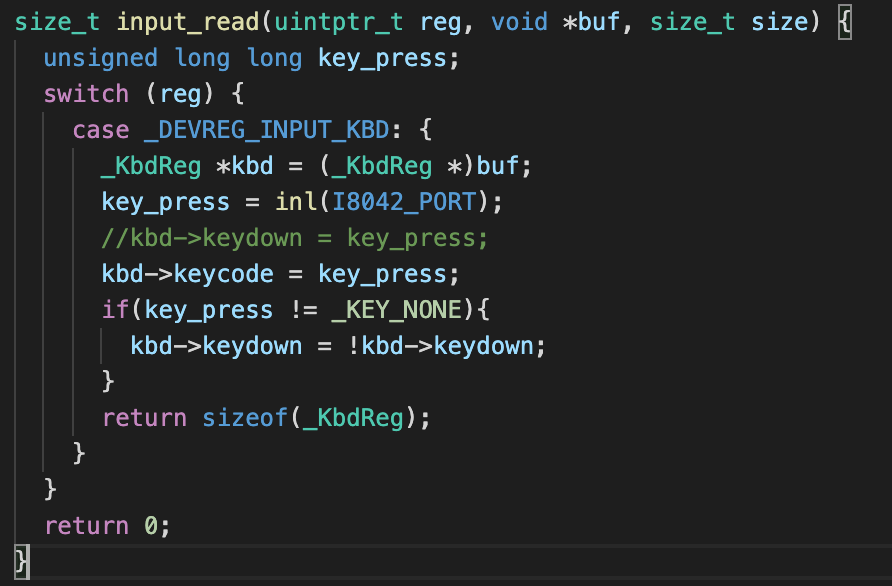
更新标志位

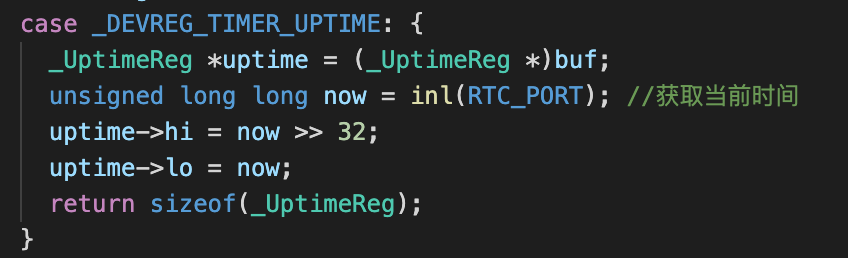


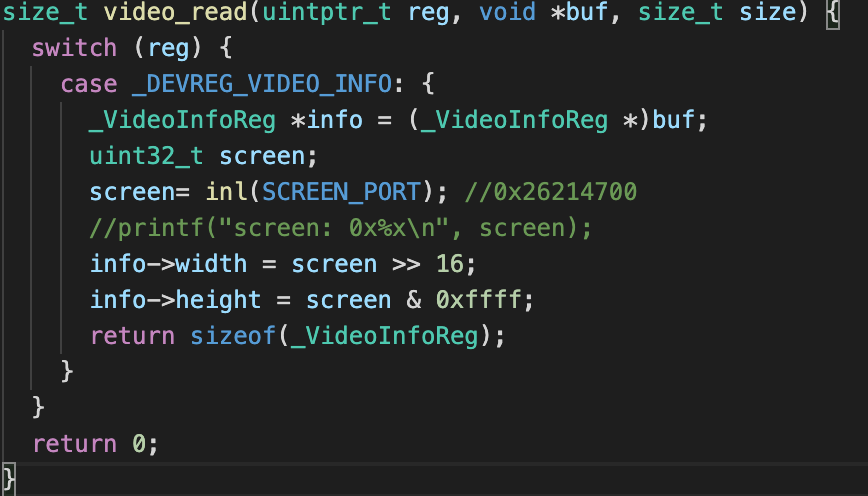
1. 参照i386手册，在opcode table中填写需要用到的指令并在all-instr.h头中添加需要实现的执行函数，分别在exec目录下的各个文件中实现这些执行函数。
2. 完成klib中的printf函数和string.c中的函数
3. 在上述过程中，极易出现译码填写或执行函数编写错误的情况，由于一开始没有完成difftest模块，每个bug调试起来都需要几个小时的时间，所以完成difftest模块，在common.h中定义DIFFTEST，在diff-test.c中实现difftest，代码如下：



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

### 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. 了解Makefile：

## PA3

### 总体设计

### 详细设计

### 运行结果

### 问题解答

# 设计总结与心得

## 课设总结

## 课设心得

# 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.