***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： | CS1610 |
| 学 号： | U201614782 |
| 姓 名： | 郑晓铧 |
| 电 话： | 18370799783 |
| 邮 件： | 1951433825@qq.com |
| 完成日期： | 2020-01-04 |

目 录

[1 课程设计概述 1](#_Toc29239639)

[1.1 课设目的 1](#_Toc29239640)

[1.2 课设任务 1](#_Toc29239641)

[1.3 实验环境 1](#_Toc29239642)

[2 实验过程 3](#_Toc29239643)

[2.1 PA0 3](#_Toc29239644)

[2.2 PA1 3](#_Toc29239645)

[2.2.1 总体设计 3](#_Toc29239646)

[2.2.2 详细设计 3](#_Toc29239647)

[2.2.3 运行结果 7](#_Toc29239648)

[2.2.4 问题解答 9](#_Toc29239649)

[2.3 PA2 9](#_Toc29239650)

[2.3.1 总体设计 9](#_Toc29239651)

[2.3.2 详细设计 10](#_Toc29239652)

[2.3.3 运行结果 13](#_Toc29239653)

[2.3.4 问题解答 16](#_Toc29239654)

[3 设计总结与心得 17](#_Toc29239655)

[3.1 课设总结 17](#_Toc29239656)

[3.2 课设心得 17](#_Toc29239657)

[参考文献 19](#_Toc29239658)

# 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/)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

本次课设，我总共完成了PA0，PA1和PA2三个任务。

PA0的主要任务是开发环境配置以及基本工具的学习，包括git，man，GDB和vim等。

PA1的任务主要是实现NEMU中的CPU结构，然后理解整体代码框架，完善NEMU的基础设施，实现单步执行、打印程序状态、表达式求职、扫描内存、设置监视点和删除监视点等命令。最后实现算数表达式的递归求值。

PA2主要任务就是实现x86指令集中的指令译码和执行函数，然后完善运行AM，实现NEMU的输入输出，最终实现打字小游戏和马里奥小游戏。具体指令集就是要求能实现cputest中的所有c文件。

## 实验环境

实验环境我选择采用VirtualBox上运行虚拟机的方法，物理机的基本硬件系统如图1.2.1所示。



图1.2.1 开发机器的配置

虚拟机的系统配置如图1.2.2所示。

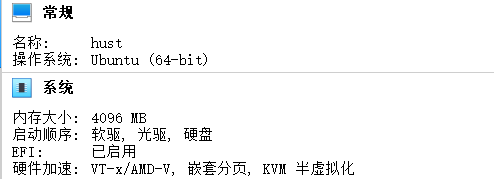


图1.2.2 虚拟机的基本配置

# 实验过程

## PA0

PA0的主要工作就是搭建实验环境和实验工具的基本学习，大体上就跟着pdf资料走就行了，对于一些不能运行的命令就根据报错的信息，用apt-get命令安装对应的库即可。

## PA1

### 总体设计

PA1的总体设计概括来说就是完善NEMU的基础设施，类似于GDB中的调试功能。通过对整体代码的框架理解，我们可以对monitor这个文件下的程序进行完善，从而实现实验要求中给定的指令功能，如图2.2.1所示。

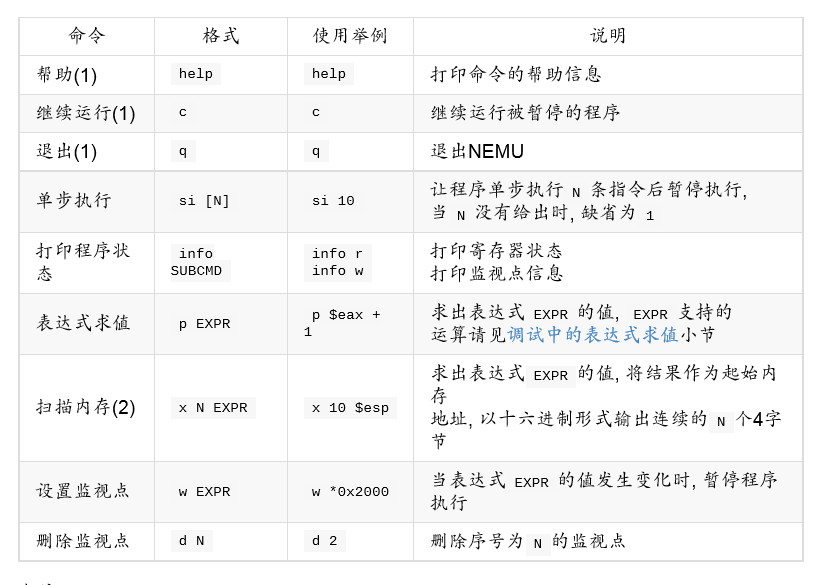


图2.2.1 简易调试器的命令格式和功能

### 详细设计

按照指导书的过程，一步步进行。首先是根据提示，利用匿名联合，很自然的根据指导书中的图2.2.2，写出结构体中模拟CPU中寄存器的存储结构。

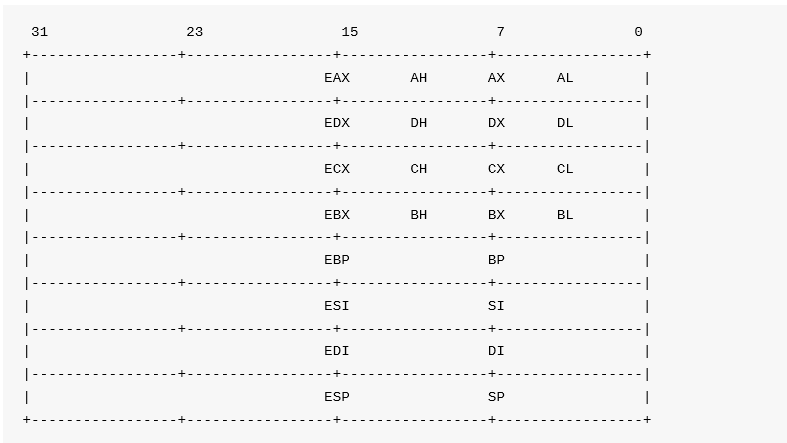


图2.2.2 NEMU中的寄存器存储结构

结构体代码如图2.2.3所示。

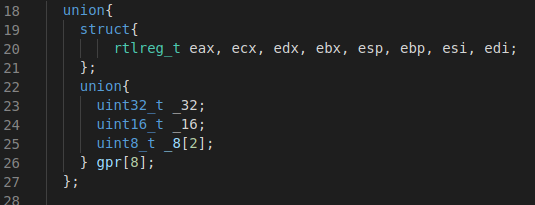


图2.2.3 CPU结构体

然后在简易调试器已经实现的功能中，观察代码框架可以很容易发现，数组cmd\_table就是存放指令名、功能描述、指令对应的处理函数，然后在ui\_mainloop函数中根据输入的命令在数组中查找对应的指令，进行相对应的处理函数。如图2.2.4和图2.2.5所示。

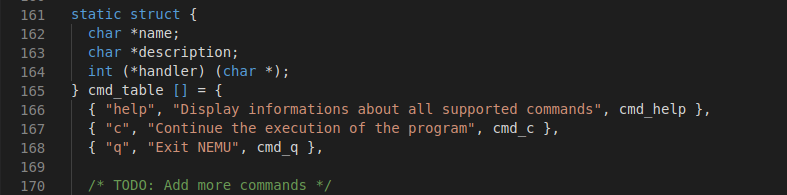


图2.2.4 存储命令的数组

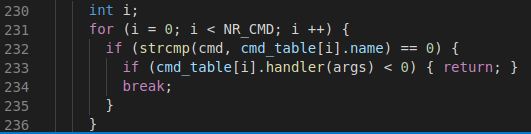
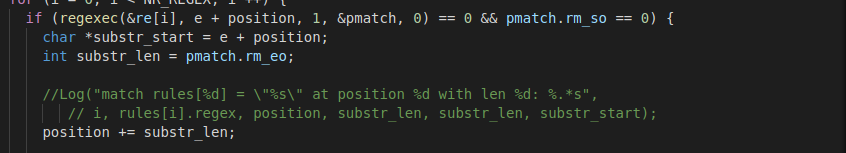


图2.2.5 处理输入的命令行

OK，到现在基本了解了NEMU简易调试器的工作原理，那么后面添加要实现的指令的时候，只要在数组中添加相关信息即可。我首先实现的命令式单步执行si [N]、打印程序状态和扫描内存，这三个指令就对应将参数接收，分别对应传入cpu\_exec参数为n、按照顺序打印寄存器、按照要求打印内存值。

接下来就是表达式求值，对于输入的表达式进行求值。这里指导书中从递归的思想一步一步给出了方法，还给出了代码框架，首先对于加减乘除和整数，这几种最基本的运算符，先用正则表达式实现，如图2.2.6所示。后面会遇到指针和负号以及十六进制数，到时候直接添加即可。



2.2.6 正则表达式匹配Tokens

然后主要我们要实现的就是check\_parentheses函数和求主运算符函数。 check\_parentheses() 函数用于判断表达式是否被一对匹配的括号包围着,同时检查表达式的左右括号是否匹配,如果不匹配, 这个表达式肯定是不符合语法的,也就不需要继续进行求值了。那么很容易想到，只要从左到右扫描表达书，读出其中的括号‘（’、‘）’，若每扫描一个字符检查，一旦发现右括号比左括号多，则不符合；若表达式全部扫描完之后检查，发现左右括号不相等，则不符合；否则，则符合。流程图如图2.2.7所示。

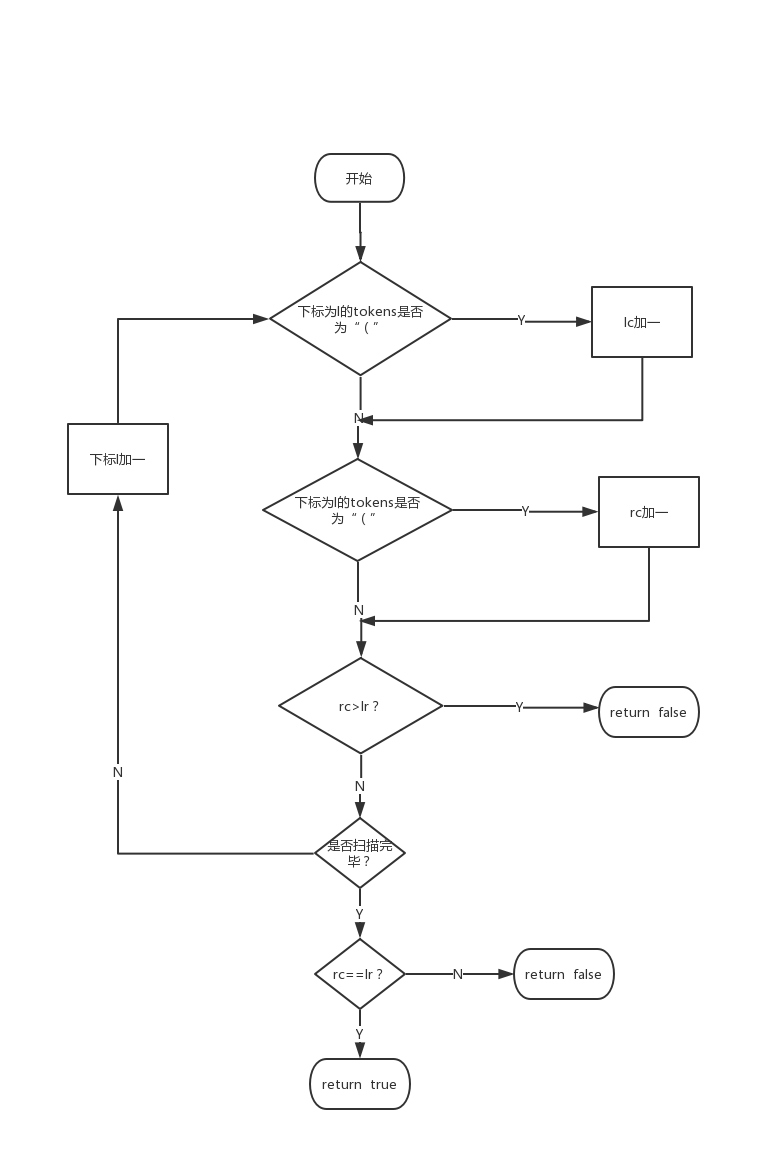


图2.2.7 检查表达式是否括号匹配的流程图

求主运算符的的方法，我主要是利用算符优先级来实现的。在Token中加入优先级这个属性，如图2.2.8所示。

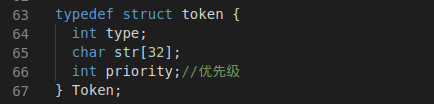


图2.2.8 添加token优先级

有了优先级这个属性，查找主运算符的过程就很简单了根据指导书中给出的要求，显然非运算符的token不是主运算符，优先级是最低的，主运算符的优先级在表达式时最低的，从左到右扫描找出最低优先级的运算符，如果有多个运算符的优先级都是最低的时候，选取表达式最右边的运算符即可。

负号和指针就根据pdf给的方法，在expr函数中对tokens扫描，判断相对应的减号‘-’和乘号‘\*’是否满足是负号和指针引用的条件，如果是，则将该token的type和优先级修改，在后面求val的时候，根据对应的运算符求值即可。如图2.2.9所示。



图2.2.9 识别负号和指针引用

最后要实现的就是设置监视点和删除监视点的工作，这一部分的工作较为简单，因为链表作为数据结构中最为熟悉的线性表之一，对它的操作已经较为熟练。采用头插法，每次建立一个监视点的时候，就在free\_的链表中取下第一个，插入到head的链表的第一个节点中，根据顺序编号。要实现监视点的监视功能，那么只需要存储监视点的表达式和表达式的值，然后根据指导书的要求以及代码中的提示，在cpu\_exec中加入监视点的check函数，调用expr函数对监视点中的表达式都计算一遍，与监视点中存储的旧值对照，有变化的输出即可，记得要将nemu\_state改为NEMU\_STOP。

至此简易调试器的功能已经全部实现，PA1的实验任务基本完成。

### 运行结果

接下来就是对实验任务的完成的测试，根据网站上面的测试样例，得到的测试结果正确。读取内存结果如图2.2.10所示。

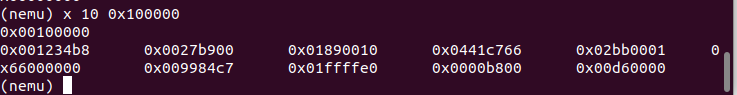


图2.2.10 打印内存

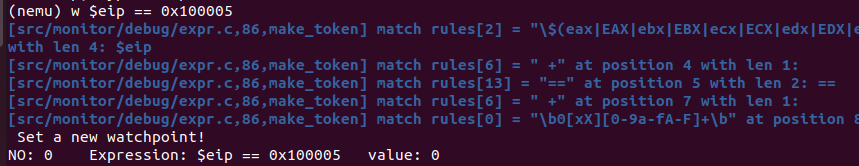


图2.2.11 设置监视点

再设置$eax和$ecx之后打印监视，如图2.2.12所示。

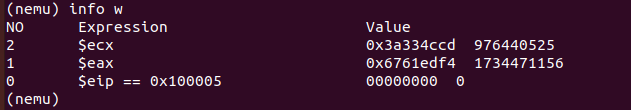


图2.2.12 打印监视点

打印寄存器信息如图2.2.13所示。

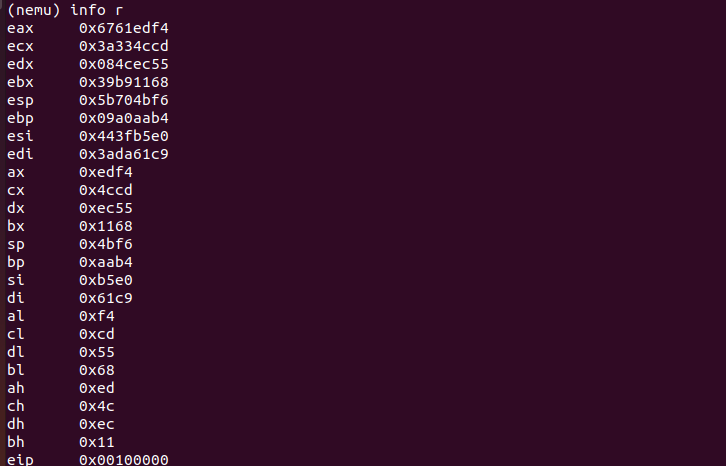


图2.2.13 打印内存

测试计算表达式的样例的结果如图2.2.14所示。



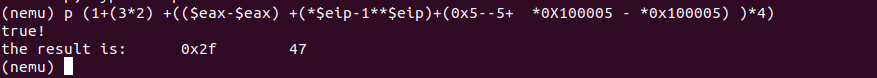


图2.2.14 计算表达式

在实验过程中和结果测试中，遇到的bug基本上是函数实现逻辑有错误或者是val求值的时候没有

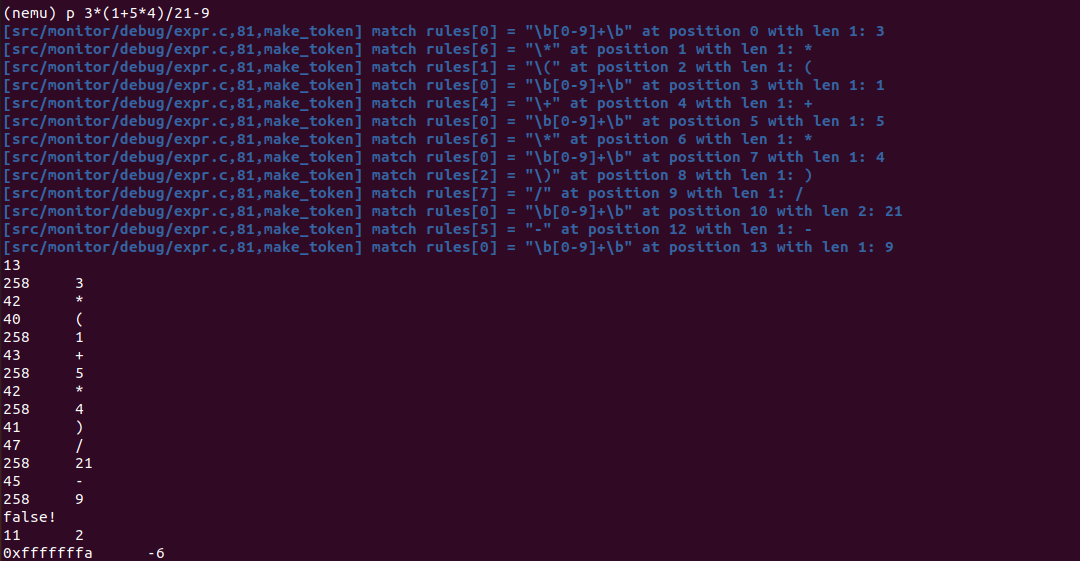


图2.2.15 实验初期边实现边调试

### 问题解答

问题一，理解基础设施。调试花费的时间是500\*90%\*30s\*20=270 000s=75h，如果采用简易调试器，花费的时间是500\*90%\*10s\*20，节省了（500\*90%\*30s\*20-500\*90%\*10s\*20）/（500\*90%\*30s\*20）\*100% = 66.7%的调试时间。

问题二，EFLAGS中CF标志位表示，算术操作产生的结果在最高有效位发生进位或借位，反之清零。

ModR/M是由Mod、Reg/Opcode和R/M三个部分组成的，Mod: 占最高位的6～7共2个bit，提供寻址模式: 11 = register（寄存器）  11 != memory（地址）。 Reg/Opcode: 占中间位的3～5共3个bit，这个地方可以表示为Reg或者Opcode——至于到底什么时候表示Reg，什么时候表示Opcode，这就要由Code来决定了 R/M: 占最低位的0～2共3个bit。R / M 字段可以将寄存器指定为操作数的位 置，或者可以与上述字段结合形成寻址模式编码的一部

可以用 find 命令找到对应文件，然后再用 wc 命令统计行数即可。如图2.2.16所示。有4310行代码。



图2.2.16

问题三，-Wall表示允许gcc提供有用的报警信息都输出，-Werror表示警告信息当做错误信息处理，出现警告信息即停止编译。可以在开发的时候让开发者注意警告，并小心一些可能会造成的bug。

## PA2

### 总体设计

PA2的总体设计概括来说就是实现NEMU中的指令集，运行所有cputest的c程序，然后完善AM，实现机器无关的库函数和输入输出设备，最后运行小程序。

指令集的实现，主要依靠i386手册，遇到测试文件中不能通过的指令说明未实现，则查阅i386手册，对指令的行为用函数描述，如果需要用到相关RTL函数则完善之。机器无关的库函数，就是完善实验要求的klib文件下的那些函数，需要了解相对应函数的功能。将设备访问抽象为IOE就是根据相对应的API，实现串口、时钟、键盘、VGA等。

### 详细设计

首先指导书要求实现最简单的那个dummy程序，根据反汇编结果，里面只需要实现6条指令。根据指导书，先利用i386查阅EFLAGS的结构，如图2.3.1所示。在结构体中实现如图2.3.2所示。

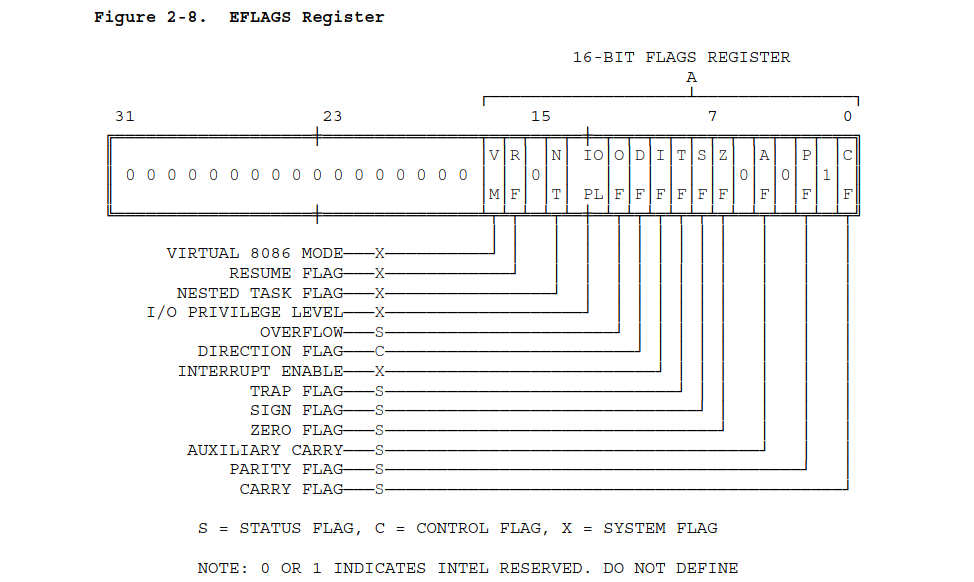


图2.3.1 EFLAGS

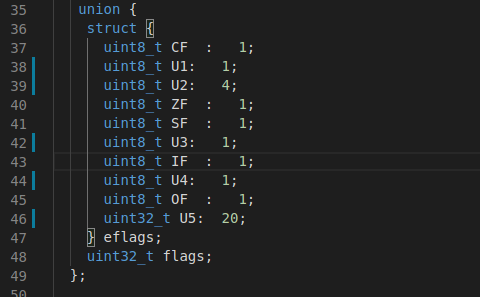


图2.3.2 EFLAGS结构体

以sub为例，首先需要在i386手册中查阅相关信息，如图2.3.3所示。

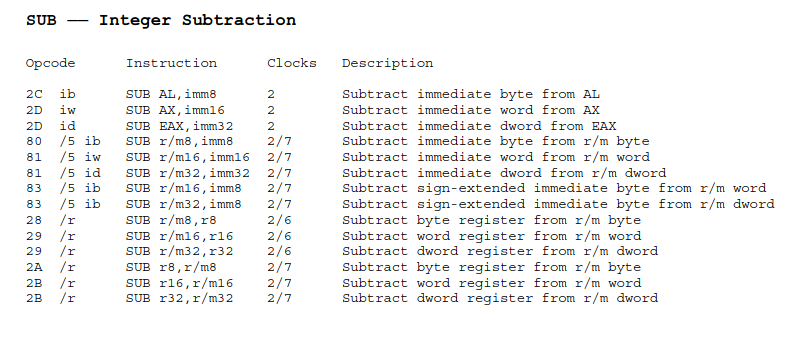


图2.3.3 指令sub

那么第一步就是将opcode填入opcode\_table相对应的位置中，/5表示后面跟着一个ModR/M字节，解释为扩展opcode。通过手册知道opcode为81的sub，该指令将immediate立即数转化为一个r/m的16位或者32位（E型），因此idex的译码信息应该是I2E，加上操作数宽度，这样对应指令的译码和执行信息就编程了sub\_I2E\_l这种结构化的信息，我们只需要实现对应的 make\_EHelper(sub)即可。

第二部就是实现make\_EHelper(sub)，这里需要注意减法对EFLAGS的OF、CF、ZF 、SF都有影响。所以需要判断溢出，是否在最高有效为出现了借位，如果有，则需要设置CF位为1，然后判断是否溢出，若数据高位进位，符号位进位未进位，溢出。若数据位高位未进位，符号位进位，溢出。溢出则需要将OF位置为1。最后ZF和SF位是直接根据结果位来设置的，直接调用底层的rtl\_update\_ZFSF即可。代码如图2.3.4所示。

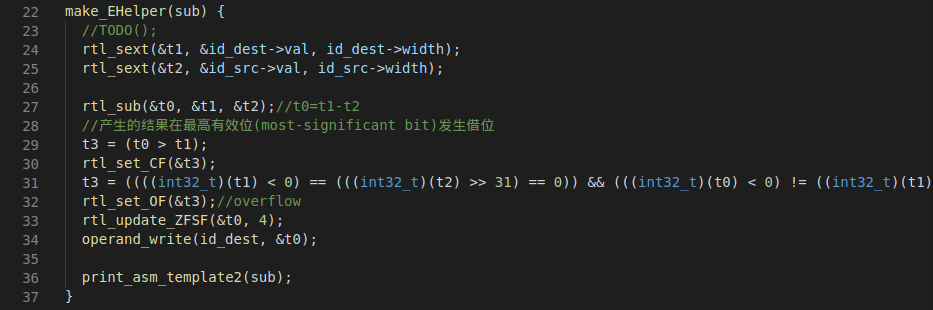


图2.3.4 make\_EHelper(sub)

然后在srith.c中添加make\_EHelper(sub)的声明，就可以运行该指令了。然后其他的指令可以按照一样的流程完成。

接着代码框架中有很强大的的debug工具，完成Differential Testing的代码，让NEMU和QEMU逐条指令地执行同一个客户程序。双方每执行完一条指令,就检查各自的寄存器和内存的状态, 如果发现状态不一致,就马上报告错误,停止客户程序的执行。如图2.3.5所示。

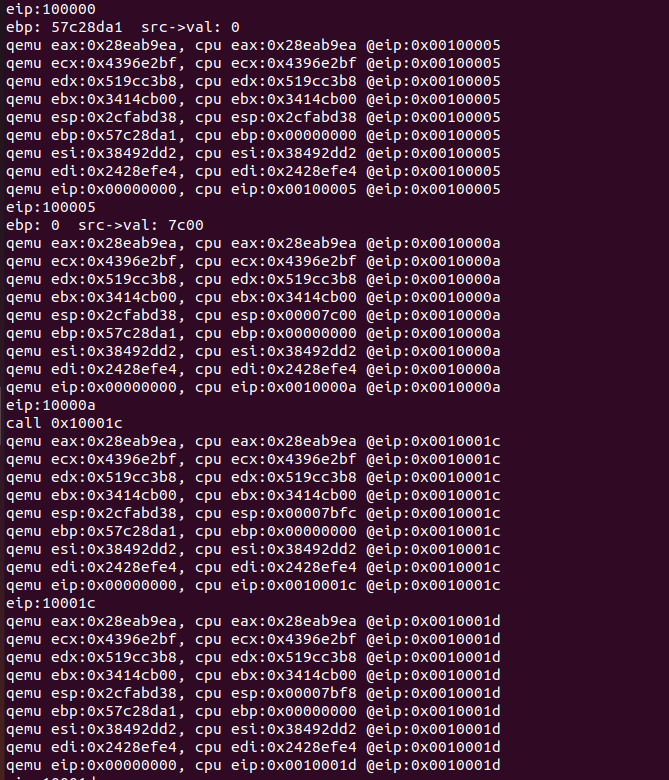


图2.3.5 寄存器状态对比

因为可以看到正确的程序的执行的时候寄存器的状态这样就可以很容易找出指令的错误，在一些错误上面，给了我非常大的帮助。

然后就是实现PA2.3的内容，实现sprintf，就要理解stdarg.h中的数据类型和宏。存取未命名的参数，首先必须在不定参数函数中声明va\_list数据类型的变量。调用va\_start并传入两个参数：第一个参数为va\_list数据类型的变量，第二个参数为函数第一个参数的名称，接着每一调用va\_arg就会返回下一个参数，va\_arg的第一个参数为va\_list，第二个参数为返回的数据类型。最后va\_end必须在函数返回前被va\_list呼叫。然后将对应字符复制到目的字符串中。

最后的IOE就根据指导书的提示按步骤实现即可，要注意端口号和设备寄存器的对应关系。

### 运行结果

指令集的测试和string.c的测试如图2.3.6所示。

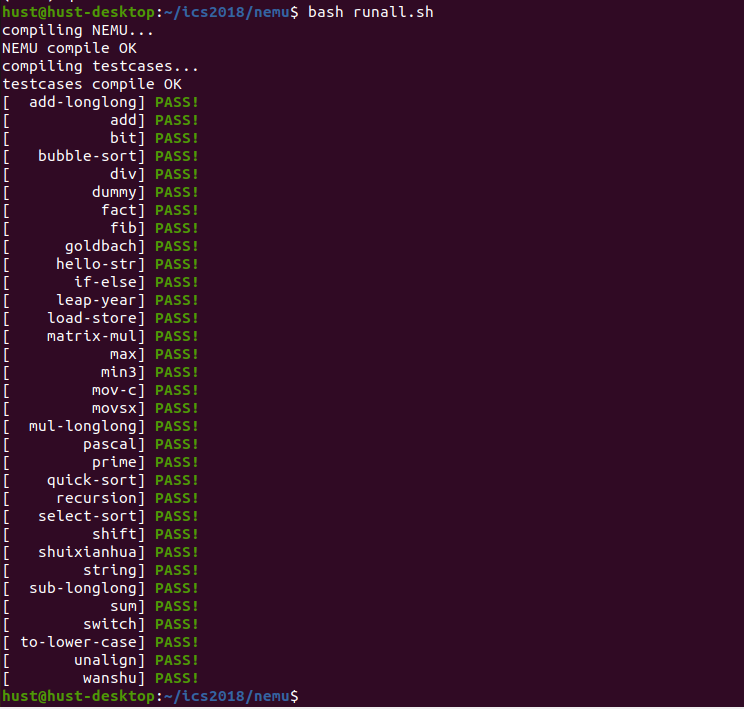


图2.3.6 cputest结果

在测试cputest中的string.c的时候，遇到了一个Difftest都没找到的bug，如图2.3.7所示。

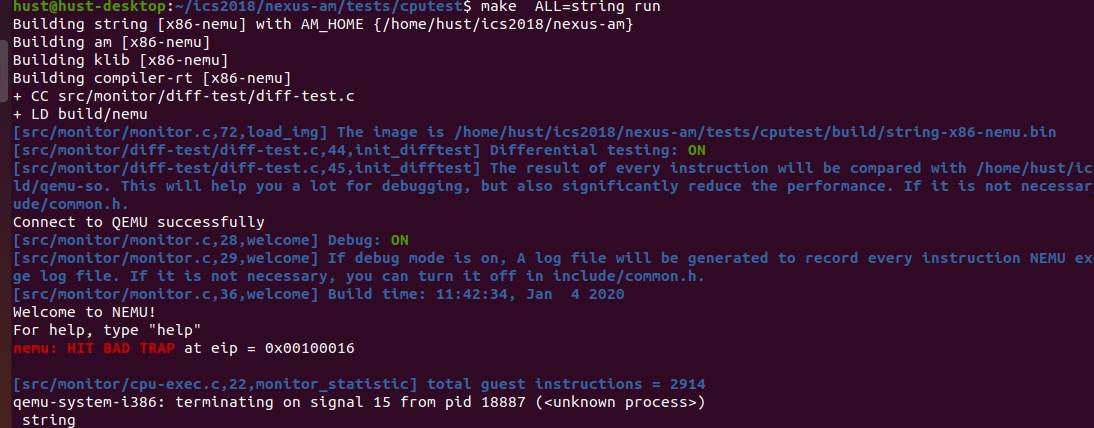


图2.3.7 string.c测试遇到的bug

输出的时候发现，QEMU和NEMU的寄存器状态一致，，但是走到那一步，就会进入nemu\_trap中，所以可以知道，指令没有错误，应该就是stdio.c中实现的memset或者memcmp错误了，我将代码修改如图2.3.8所示。

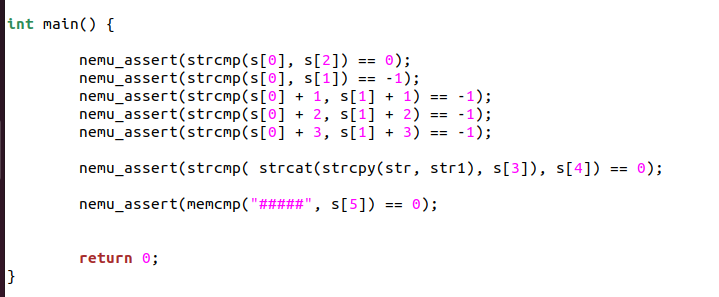


图2.3.8 修改string.c

发现程序可以正常退出，那么肯定就是memset出错了，后来找bug很久才发现，原来是我对void\*指向的地址理解太粗心了！在复制地址中的内容到目的地的时候，我一直理解的是一个字符一个字符的复制，地址递增的时候，它的地址可能不是一个字节一个字节的递增，因为类型不确定，如果是int则是四个字节四个字节的递增，所以这里要强制转换为字符类型，他才能一个字节一个字节递增，然后复制完一个字符下一次地址指向的就是下一个字符的地址。如图2.3.9所示。

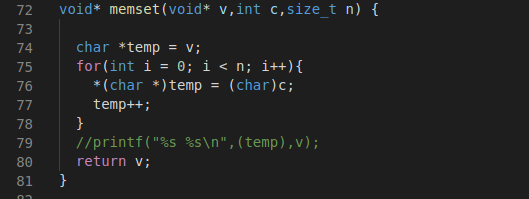


图2.3.9

然后是MicroBench测试的分数大概是80-120波动，如图2.3.10所示。



图2.3.10 MiroBench分数

马里奥小程序运行结果如图2.3.11所示。



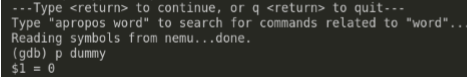
图2.3.11 进入游戏

### 问题解答

问题一，去掉static函数会出现编译错误，static关键字定的全局变量只在定义他的源文件内有效，其他源文件无法访问它，这样的话就保证了在当前文件下命名不重复即可，不用担心和其他文件中的函数名冲突。去掉inline也会出错，内联函数在编译的时候会直接将代码嵌入到被调用的地方，这样减少调用函数的开销。这些错误都是warning，如果在makefile文件中去掉-Werror那句话就不会报错（static去掉之后如果其他源文件定义了相同文件名也会报error）。两者都去掉也会出错，文件一起编译的时候就会，没有static的函数会被编译多次，从而导致链接错误。

问题二，在 nemu/include/common.h 中添加一行 volatile static int dummy 会产生 1 个 dummy 变量实体，static关键字说明这是一个静态全局变量，只在本文件内有效。

再添加一行volatile static int dummy 仍然还是产生 1 个 dummy 变量实体，可以利用gdb打印变量得到，如图2.3.12所示。



2.3.12

两处都初始化为0，会导致重定义错误。之前没有实现是因为之前为初始化的dummy是弱符号，初始化之后是强符号，这个时候多个相同名字的强符号是不被允许的。

问题三，Makefile的规则大致是：

target... : prerequisites ...

command

…

…

target也就是一个目标文件，可以是Object File，也可以是执行文件。prerequisites就是，要生成那个target所需要的文件或是目标。command也就是make需要执行的命令（任意的Shell命令）。Target这一个或多个的目标文件依赖于prerequisites中的文件，其生成规则定义在command中。说白一点就是说，prerequisites中如果有一个以上的文件比target文件要新的话，command所定义的命令就会被执行。/nemu下面的Makefile顶一个了构建规则需要的路径相关环境变量的值，设置编辑器为gcc，C编辑器选项为CFLAGS。SRCS利用find命令找到src目录下的\*.c文件，然后用OBJS让make将找到的.c文件添加到依赖关系中，然后找到每一个target的前置条件，根据command构建target。最后定义了怎么样的链接规则，进行git commit提交。

# 设计总结与心得

## 课设总结

本次课设总体工作总结主要是以下几个方面：配置实验环境，基本掌握几种使用工具的使用。对整个基础设施有了基本了解，然后根据指导书的步骤阅读代码，理解代码运行逻辑，整个代码框架的结构，复习了正则表达式的基本规则和用法，在算术表达式的计算过程中实现了要求的表达式计算。对监视点的进一步实现，使得观察寄存器的值更加方便，最后实现了整个简易调试器，能够根据要求输入输出，并且遇到不合法信息能够识别并且报错。

然后就是对整个指令集的代码框架的阅读和理解，查阅了i386手册，对于指令集和CPU内部寄存器的理解更加深刻了，对一些运算要注意的溢出、标志位改变等等更加熟悉。完善了REM中的字符处理函数，模拟了串口的功能，实现了映射I/O，实现了时钟和相关的IOE，实现键盘的一些基本功能，实现了常用的输出设备VGA的功能，最后能够运行游戏小程序，虽然很卡顿，但是能够正常运行和正确响应对应的键盘事件。

## 课设心得

本次课设总体感觉就是很想之前做的计算机组成原理实验，只不过当初是在logsim当中直接模拟硬件层面中CPU工作时的方式和指令功能如何在CPU上面实现的，可惜时间不够，nemu后面实现多道程序等是在计算机组成原理实验中没有体验过的。总体来说在硬件层面还是工作比软件实现要少一些，因为指令的译码和执行逻辑电路写好了就行了，直接执行即可。但是nemu中指令集需要一个一个的填写，这就相当于考验我们对指令的理解和代码的编写能力，所以从这个层面来说，nemu显然更注重于对代码框架、指令的理解，计算机组成原理实验侧重于对硬件层面、电路逻辑的理解。

在PA中复习到了许多之前的知识这是最有益处的，其实整体就是在于之前计算机组成原理的基础，然后细节的地方，我也复习了之前编译原理课堂学习的正则表达式和语法分析的知识，还有实现键盘等IOE的时候，把设备访问抽象成统一的API，对于设备访问和物理地址划分都是操作系统课堂的内容。所以在实验过程中会遇到这些问题，然后解决这些问题就是运用之前学习的知识，这样就可以即复习了以前的知识，又能体会到以前学习的知识是如何在一个计算机系统中得到体现的，为什么要这么做，这么实现，这些问题都使我们对于TRM对于操作系统理解更加深刻。

还有整个代码框架也是学习到的地方。随着实验一步步深入，就会慢慢理解各个文件的函数之间，调用的逻辑，执行的方式，各有不同。在写完一个PA后，又从头看我开始入手的地方，就会有恍然大悟的感觉，这个地方为什么要这么写。只有经历了正果过程，对整个要实现的东西理解更深一层，才会有一种我来写应该也是按照怎么样的框架来的感觉。

实验中实现的小程序跑起来相当卡顿，回想小的时候，一个“小霸王”游戏机加上一个手柄就能跑上百个诸如此类的程序了，而那么多的程序也只是放在一张的小小的卡片上，运行起来都是十分流畅，让人玩的不亦乐乎。所以我们知道，真正的计算机硬件层加操作系统需要更多的优化和细节，我们又一次回顾了整个CPU到汇编指令执行的过程，对于更多的优化，对于今日体积更小而计算能力更强大的个人PC，我们还有很长的路要走。

# 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.