S

***2022***



**系统能力培养 课程实验报告**

|  |  |
| --- | --- |
| 题 目： | riscv指令模拟器 |
| 专 业： | 计算机科学与技术 |
| 班 级： | CSXJ1901班 |
| 学 号： | U201911122 |
| 姓 名： | 闫浩 |
| 电 话： | 14794520002 |
| 邮 件： | U201911122@hust.edu |
| 完成日期： | 2022-02-05 |

目录

[1 课程实验概述 1](#_Toc11400)

[1.1 课设目的 1](#_Toc18564)

[1.2 实验环境 1](#_Toc23131)

[2. 实验方案设计 2](#_Toc8416)

2.1世界诞生前夜---开发环境：

框架。

2.2开天辟地---图灵机

PA1.1 简易调试器

PA1.2 表达式求值

PA1.3 监视点与断点

2.3简单复杂计算机--冯诺依曼计算机

PA2.1 运行第一个 C 程序

PA2.2 丰富指令集，测试所有程序

PA2.3 实现 I/O 指令，测试打字游戏

2.4穿越时空之旅：异常控制流

PA3.1 实现系统调用

PA3.2 实现文件系统

PA3.3 运行仙剑奇侠传

2.5 问题解答

[3. 实验过程与结果分析 6](#_Toc3448)

**4.实验心得与体会**

# 课程实验概述

## 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目 (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 "https://nju-ics.gitbooks.io/ics2018-programming-assignment/content/_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: Linux(VirtualBox)
* 编译器: GCC
* 编程语言: C语言

# 实验方案设计

* 1. PA0

本实验要求完成五个PA（Programming Assignment),

PA包括一个准备实验以及连贯的实验内容:

– PA0: 开发环境配置(准备实验)

– PA1: 简易调试器

– PA2: 冯诺依曼计算机系统

– PA3: 批处理系统

– PA4: 分时多任务

本小节完成pa0。

PA指导学生实现一个功能完备的模拟器NEMU(NJU EMUlator)

► 支持x86(教学版子集)/mips32/riscv32

– 最终在NEMU上运行真实游戏“仙剑奇侠传”

– 揭示“程序在计算机上运行”的基本原理

PA组成-NEMU全系统模拟器

► 简易调试器(Monitor) - 基础设施

– 单步执行, 打印寄存器/内存, 表达式求值, 监视点

► CPU

– 完整的指令周期: 取指, 译码, 执行

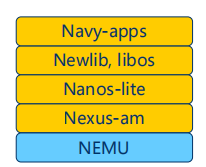
– 异常处理模块

– 支持分页的MMU

► 内存

► 设备

– 时钟, 键盘, VGA, 串口(功能经过简化)



PA组成-Nexus-AM ISA抽象层

► 把机器功能抽象成C语言API, 向程序屏蔽ISA细节

► TRM 计算抽象

– \_putc() 输出一个字符

– \_halt() 终止程序运行

► IOE 输入输出抽象

– uptime() 返回当前时间

– read\_key() 返回按键

– draw\_rect() 在屏幕上绘制矩形像素

► CTE 上下文管理抽象

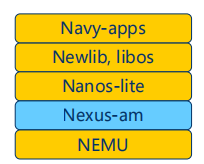
– 上下文保存/恢复

– 事件处理回调函数

► VME 虚存抽象

– \_protect() 创建虚拟地址空间

– \_map() 添加va到pa的映射关系



PA组成-Nanos-lite简易多任务OS

► Nanos(Nanjing U OS)简化版, 能支撑真实程序

– ELF加载器

– 中断/异常事件分发

– 6+2个系统调用

► open, read, write, lseek, close, brk

► exit, exec

– 简易VFS

► 文件数量, 大小皆固定, 没有目录

► 文件系统基于ramdisk, 无需持久化

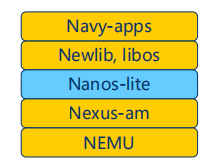
► 3个设备文件(映射到设备抽象层)

– /dev/fb, /dev/events, /proc/dispinfo

– 设备抽象层

– 简易分页存储管理 (顺序分配, 不释放)

– 两个进程的简易轮转调度



库和应用程序Navy-apps

► Newlib – C库, 支持POSIX标准

► libos – 系统调用接口

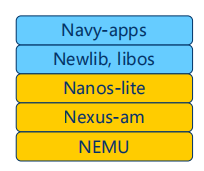
► Navy-apps – 应用程序集

– hello, 仙剑奇侠传, LiteNES (PA用)

– Lua解释器, busybox套件(shell, vi)

– NWM窗口管理器, Nterm终端

– NCC编译器



* 1. PA1

PA1 最终要求实现一个简易调试器，相当于一个简化版的 gdb，要求实现单步执行、打印寄存器状态、扫描内存、表达式求值、监视点等功能。

* + 1. PA1.1

需要实现实现单步执行，打印寄存器状态，扫描内存，需要在 ui.c中声明并实现相关的函数。三个功能需要实现：单步执行，打印寄存器信息，扫描内存。在ui.c中，这三个功能分别用cmd\_si、cmd\_info和cmd\_x表示。cmd\_si通过调用cpu\_exec完成指定步数的执行，cmd\_info调用isa\_reg\_display打印寄存器名称和值，isa\_reg\_display使用reg\_name和reg\_l打印寄存器的信息，cmd\_x对输入的参数进行分析，通过vaddr\_read函数读取对应地址的数值并输出。

* + 1. PA1.2

进行表达式求值，首先要在 expr.c 中补充规则和相应的正则表达式，例如 十六进制数的正则表达式与 token 类型为{"0[Xx][0-9a-fA-F]+", TK\_HEX}。

随后完成函数 make\_token，用于为算术表达式中的各种 token 类型添加规 则，并在成功识别出 token 后,将 token 的信息依次记录到 tokens 数组中。 接着完成括号匹配函数 check\_parentheses，完成运算符优先级函数 op\_priority，根据 C 语言中的优先级来实现对应的操作符的优先级。

* + 1. PA1.3

完善 watchpoint 结构，随后在 watchpoint.c 中实现函数 new\_wp，

free\_wp，print\_wp。在 ui.c 中使用上述函数完成命令 cmd\_x 与 cmd\_d，

完善函数 cmd\_info，同时要在 cpu.exec 中实现当监视点状态改变使程序暂 停执行的逻辑，该逻辑较为简单，程序每执行一步就对所有的监视点进行表 达式求 值，并与旧值进行比较，如果两值不同，则置 nemu 的状态为 NEMU\_STOP。

* 1. PA2
     1. PA2.1

实现对于部分指令的支持，需要先运行dummy.c并在反汇编代码中找出未实 现的指令，随后通过查询手册确定需要实现的新指令，并在all-instr.h中定义 相关的执行函数，修改decode.c、rtl.h、compute.c、control.c等文件实现相应 的译码和执行辅助函数，最后重新编译运行即可。

* + 1. PA2.2

要求在 NEMU 中实现更多的指令，这需要先找出反汇编代码中的未实现的 指令，再编写译码和执行函数。先完成 diff-test 会方便测试。实现指令的过 程类似于 PA2.1，以 sum.c 为例，需要实现的指令包括 beq、bne、add 和 sltiu。 译码函数在对应的辅助函数中编写，执行函数在 control.c 和 compute.c 中 编写，要根据指令的相关行为编写对应的执行辅助函数

* + 1. PA2.3

在相关设备文件中完成四个输入输出设备程序的编写，包括串口，时钟，键 盘和 VGA。串口已经在trm.c中实现，时钟需要在nemu-timer.c中完善，键 盘需要在nemu-input.c中完善，VGA需要在nemu-video.c中完善。

* 1. PA3

主要任务是实现系统调用和文件系统，使得 nemu 最终能够运行仙剑

奇侠传

* + 1. PA3.1

现自陷操作\_yield 及其过程，要实现自陷操作，首先要在 cpu

结构中添加控制状态寄存器，然后实现自陷需要的指令，通过异常号识别出 自陷异常，完成自陷事件。实现自陷指令，需要实现的指令有 csrrs、csrrw、 ecall 和 sret，通过 ecall 指令进入自陷操作，csrrs 和 csrrw指令对控 制状态寄存器进行修改，sret 指令用于自陷后返回，然后需要在 intr.c 文件中进行 raise\_intr 函数的编写，raise\_intr 函数用于模拟相应过将 当 前 PC值保存到 sepc 寄存器，在 scause 寄存器中设置异常号，从 stvec 寄存器中取出异常入口地址，跳转到异常入口地址；触发自陷操作后， 需要保存上下文，根 据 trap.S汇编代码的压栈的顺序重构\_Context 成 员体结构，接着要实现正确的事件 分发，需要在\_\_am\_irq\_handle 函数中 通过异常号识别出自陷异常，在 do\_event 函数中识别出自陷事件 \_EVENT\_YIELD，最后通过 sret 指令返回并恢复上下文。

* + 1. PA3.2

需要实现用户程序的加载和系统调用, 支撑 TRM 程序的运行，需要通过实 现 loder 函数，增加系统调用，完善堆区管理函数。实现用户程序的加载和 系统调用, 支撑 TRM 程序的运行，为了加载用户程 序首先要实现 loader 函数，因为目前还没有实现文件系统，所以直接在 loader 函数中 通过 ramdisk\_read 函数把可执行文件中的代码和数据放置 在正确的内存 位置，然后跳转到程序入口，接着需要完成系统调用的实现，和 3.1 中识 别出自陷事件类似，让 nemu 能够识别出系统调用，然后在 do\_event 中 添 加do\_syscall 的调用，根据 nanos.c 中的 ARGS\_ARRAY 在 riscv32-nemu 中实现正确的 GPR 宏，随后添加 SYS\_yield、SYS\_read、 SYS\_write 和 SYS\_brk 系统调用，完成函数\_sbrk 实现堆区管理。

* + 1. PA3.3

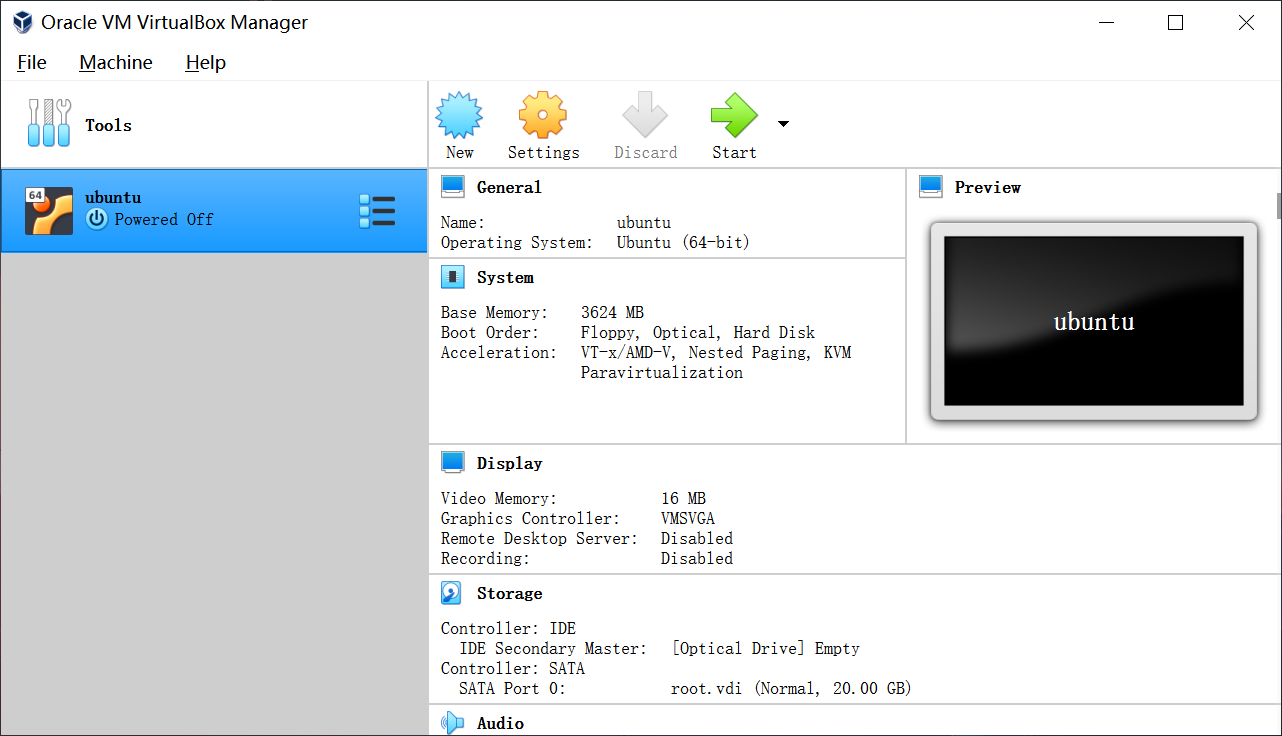
需要实现文件系统，完成 fs\_open,fs\_read 等函数，完成设备函数的编

写，使用 fs 函数修改 loader，最终可运行仙剑奇侠传。需要实现文件系统， 添加设备的支持，最终能够运行仙剑奇侠传。首先需要实现函数 fs\_open, fs\_read 和 fs\_close，因为 ramdisk 中的文件数 量增加之后，就不适合 直接在 loader 函数中直接使用 ramdisk\_read，在完成了这几 个 fs 函数 后，替换掉 loader 函数中的 ramdisk\_read 函数，修改其逻辑这样就可以 在14loader 中使用文件名来指定加载的程序了，随后完成 fs\_write 和 f s\_lseek 函数，使其能够进行输出，完成这些函数后，要补充相关的系统调 用；要实现虚拟文件系统 VFS，把 IOE 抽象成文件，首先需要在 VFS 中 补充多种特殊文件的支持，接着实现函数 serial\_write,完成串口的写入，然 后完成实现 events\_read 函数，支持读操作，最后需要完成 init\_fs、fb\_write、 fbsync\_write、init\_device、dispinfo\_read函数，实现对 VGA 设备的支持。 要注意的是因为在 Finfo 结构中添加了读函数指针和写函数指针，所以要修 改修改 fs\_write 和 fs\_read 的逻辑，完成这些以后，如果没有错误，就能 够运行仙剑奇侠传了。

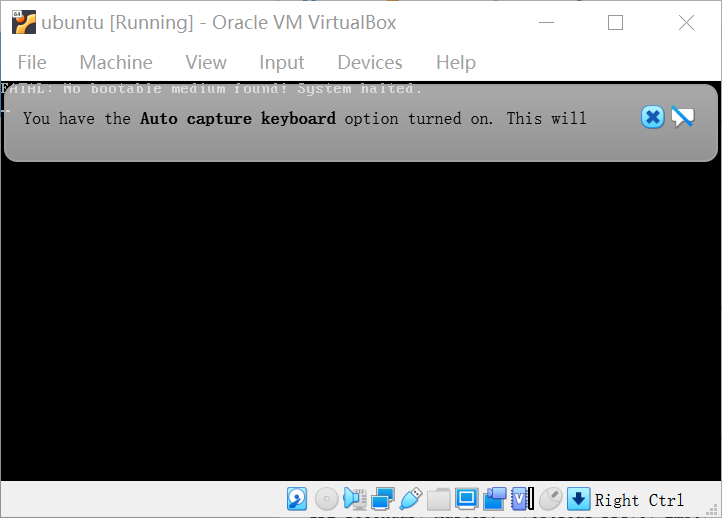
# 实验过程与结果分析

* 1. PA0:世界诞生的前夜: 开发环境配置

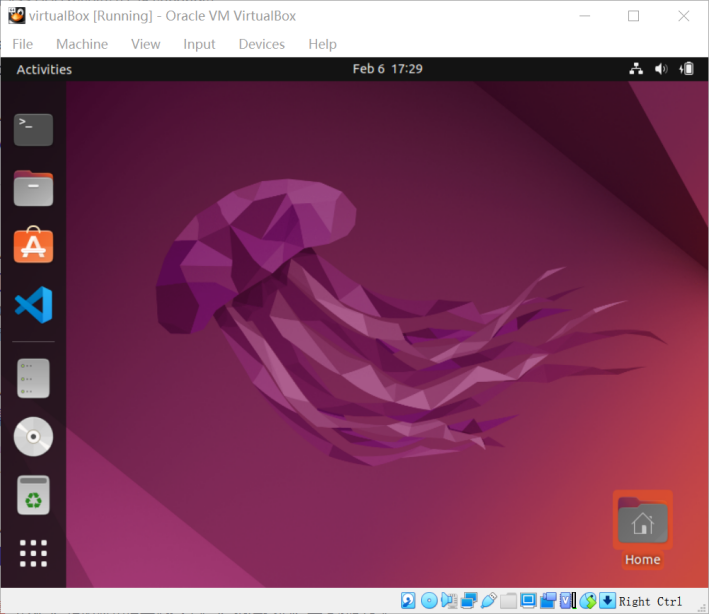
首先是进行开发环境的配置，本次实验使用virtualbox虚拟机搭建ubuntu系统进行实验，下面是virtualbox的环境配置。



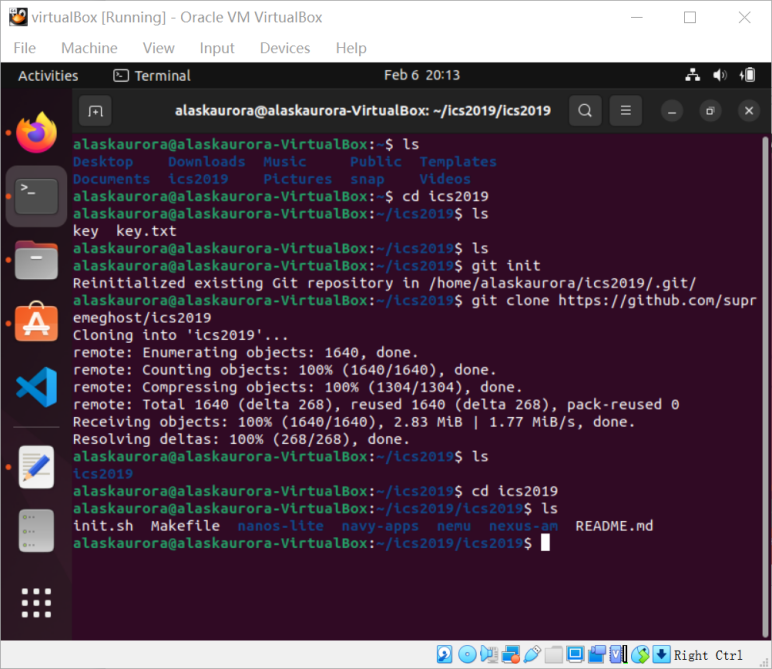
加载ubuntu镜像vdi文件后，出现以下问题。



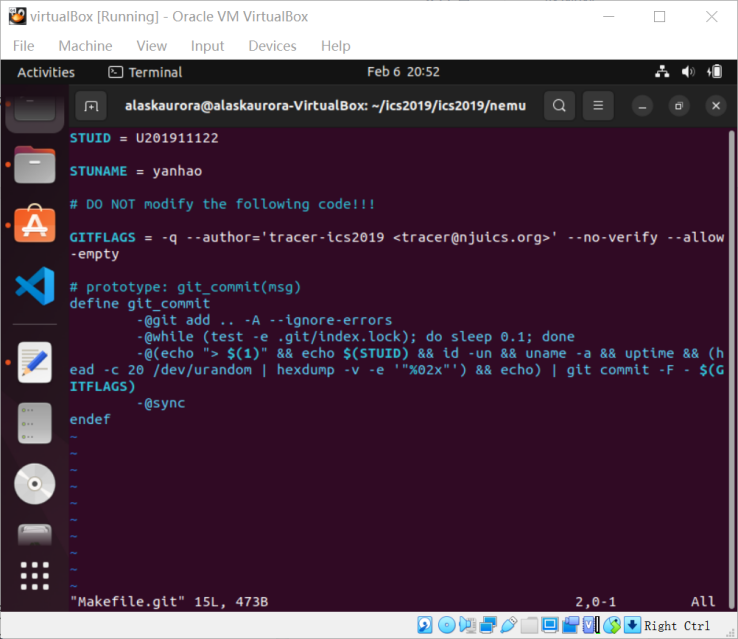
经考察，是因为未加载ubuntu镜像。

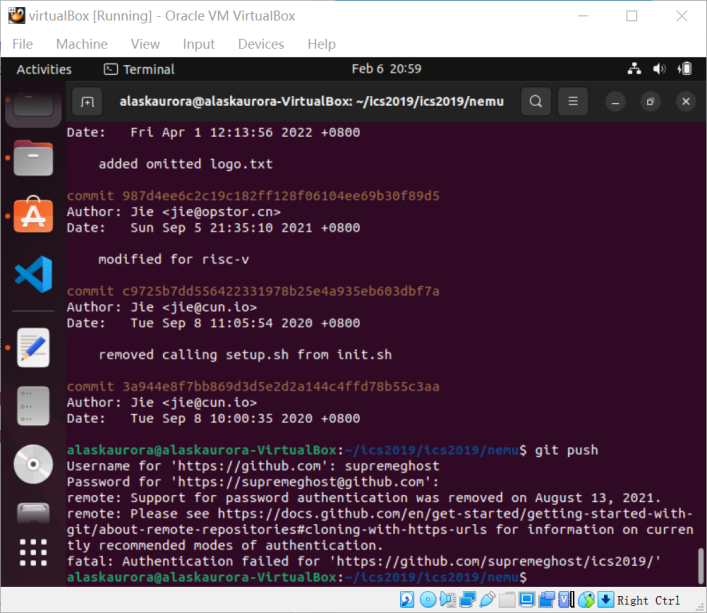


安装好linux后，安装git，vim等程序

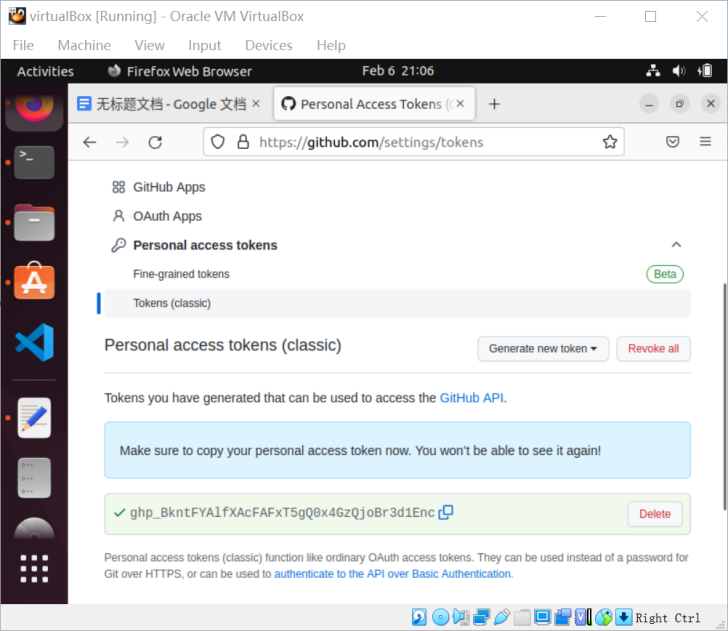


将nemu/Makefile.git 改为自己的学号，发现问题。





上网搜索。原来是github在去年停止使用用户密码检查，改为token

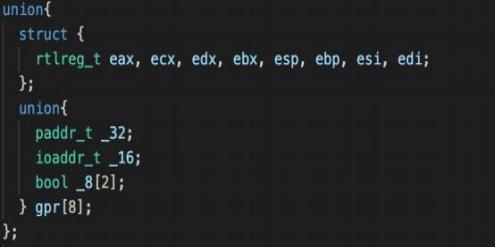


Github更改token，之后顺利更改git中学号姓名文件

至此，pa0开发环境配置结束。

3.2PA1:

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。

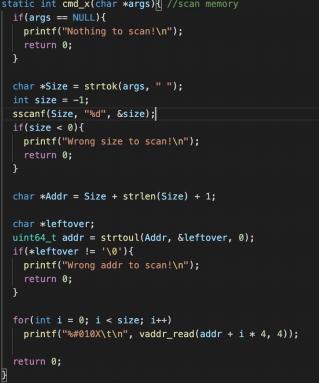
2 ) 单步执行

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。

3 ) 打印寄存器状态

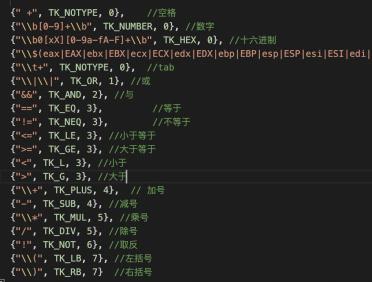
完成打印单个寄存器信息的函数， 之后遍历调用即可。

4 ) 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。

5 ) 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。

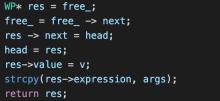




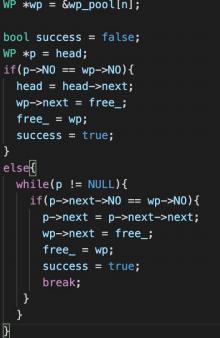
整个计算过程需要注意的是， 在判断\*和-时， 注意判断寄存器和括号， 另外在正则匹配> 和>=号时， 需要优先匹配>=号， 否则会被忽略掉。

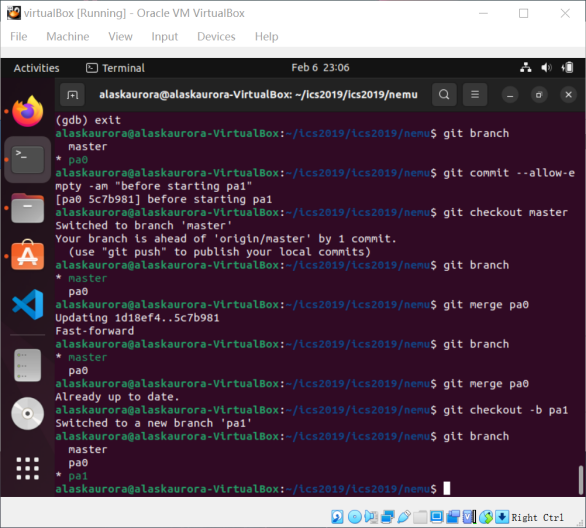
6 ) 监视点设置和删除

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：

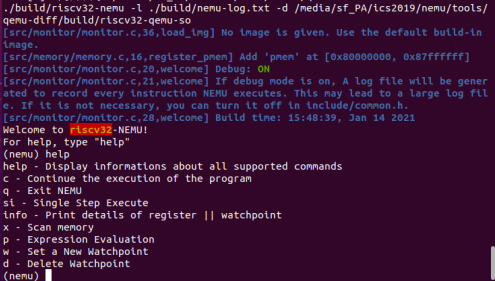


删除部分如下：





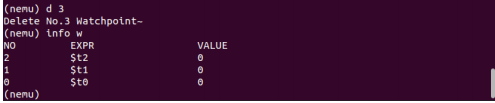
创建git分支，将各个pa分离开来.



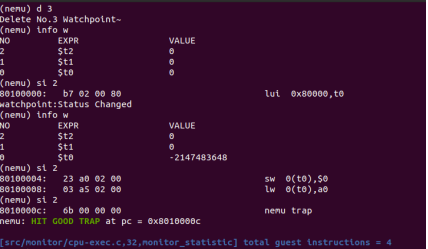
进入简易调试器后，输入 help 显示各个指令对应的信息



设置四个监视点后，使用 info 打印监视点信息，



删除 3 号监视点后打印监视点信息。



使用命令 si 执行两步，由于寄存器 t0 的值发生改变，程序暂停，使用 info

命令显示监视点的值，发现 t0 寄存器的值变为一个负数，实际上是十六进制

0x80000000，继续使用 si 单步执行，程序最终会 HIT GOOD TRAP



使用 p 命令进行表达式求值，对于正确的表达式会求出其值，对于错误的表

达式会给出相应的提示

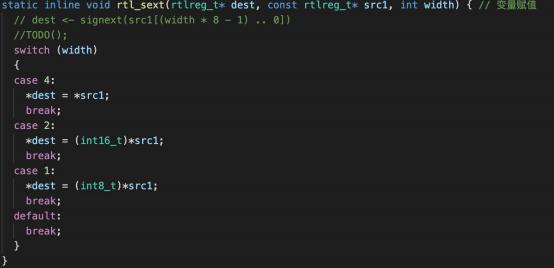
3.3PA2:

1 ) 首先实现未实现的 rtl：

取反



变量赋值



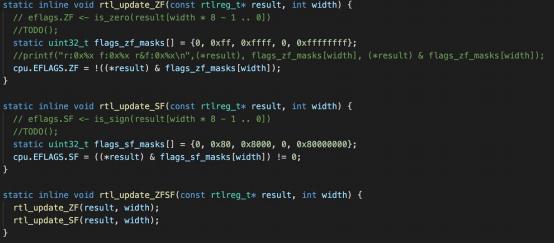
入栈



出栈



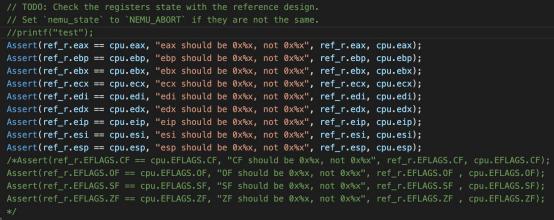
更新标志位



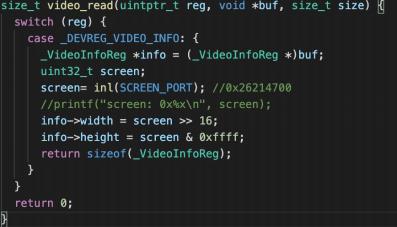
2 ) 参照 i386 手册， 在 opcode table 中填写需要用到的指令并在 all-instr.h 头 中添加需要实现的执行函数， 分别在 exec 目录下的各个文件中实现这些执行函 数。

3 ) 完成 klib 中的 printf 函数和 string.c 中的函数

4 ) 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：

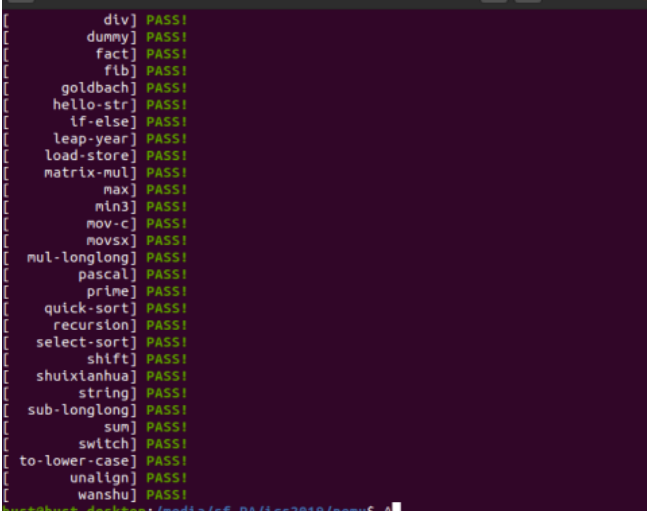


5 ) 完成 src/device 目录下的 input.c,timer.c,video.c 中的功能函数， 将设备抽 象为 IOE

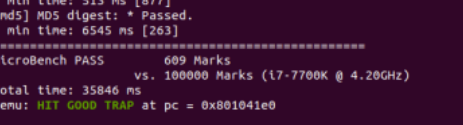


完成以上功能之后， 即可调试运行 microbanch 以及打字游戏等测试程序

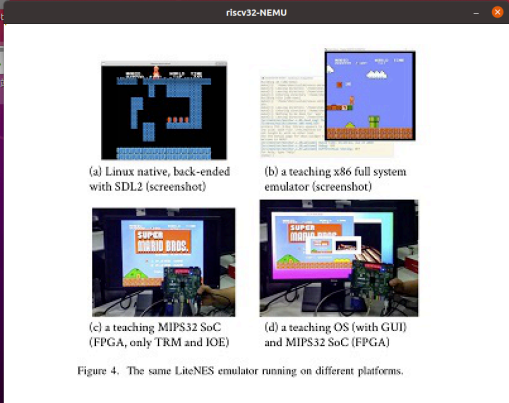
到此为止， PA2 的主体功能实现完毕。



microbench 测试的运行结果



游戏界面



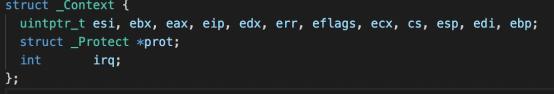
3.4PA3

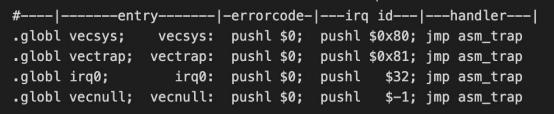
要实现自陷操作，要完成自陷操作， 首先要实现自陷指令，需要实现 的指令有 csrrs、 csrrw、 ecall 和 sret， 通过 ecall 指令进入自陷操作， csrrs 和 csrrw 指令对控制状态寄存器进行修改， sret 指令用于自陷后返回， 然后需要在 intr.c 文件中进行raise\_intr 函数的编写， raise\_intr 函数用于模拟相应过程： 将当前 PC 值保存到 sepc 寄存器， 在 scause 寄存器中设置异常号， 从 stvec 寄存器中取出异 常入口地址，跳转到异常入口地址；触发自陷操作后，需要保存上下文，根据 trap.S 汇编代码的压栈的顺序重构\_Context 成员体结构， 接着要实现正确的事件分发， 需要在\_\_am\_irq\_handle 函数中通过异常号识别出自陷异常，在 do\_event 函数中 识别出自陷事件\_EVENT\_YIELD， 最后通过 sret 指令返回并恢复上下文。

PA3.2 需要实现用户程序的加载和系统调用, 支撑 TRM 程序的运行， 为了加 载用户程序首先要实现loader 函数， 因为目前还没有实现文件系统， 所以直接在 loader 函数中通过 ramdisk\_read 函数把可执行文件中的代码和数据放置在正确的 内存位置， 然后跳转到程序入口， 接着需要完成系统调用的实现， 和 3.1 中识别 出 自陷事件类似， 让 nemu 能够识别出系统调用， 然后在 do\_event 中添加 do\_syscall 的调用，根据 nanos.c 中的 ARGS\_ARRAY 在 riscv32-nemu 中实现正确 的 GPR 宏， 随后添加 SYS\_yield、 SYS\_read、 SYS\_write 和 SYS\_brk 系统调用， 完成函数\_sbrk 实现堆区管理。

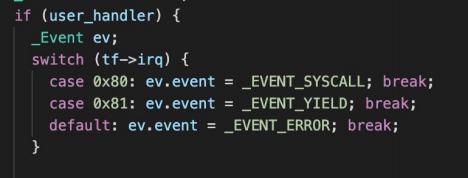
PA3.3 需要实现文件系统， 添加设备的支持，最终能够运行仙剑奇侠传。 首 先需要实现函数 fs\_open, fs\_read 和 fs\_close， 因为 ramdisk 中的文件数量增加之 后， 就不适合直接在 loader 函数中直接使用ramdisk\_read， 在完成了这几个 fs 函 数后， 替换掉 loader 函数中的 ramdisk\_read 函数， 修改其逻辑这样就可以 在loader 中使用文件名来指定加载的程序了， 随后完成 fs\_write 和 fs\_lseek 函数， 使其能够进行输出， 完成这些函数后， 要补充相关的系统调用； 要实现虚拟文件 系统 VFS， 把 IOE 抽象成文件， 首先需要在 VFS 中补充多种特殊文件的支持， 接着实现函数 serial\_write,完成串口的写入， 然后完成实现 events\_read 函数，支 持读操作， 最后需要完成 init\_fs、fb\_write、fbsync\_write、init\_device、dispinfo\_read 函数，实现对 VGA 设备的支持。要注意的是因为在 Finfo 结构中添加了读函数 指针和写函数指针，所以要修改修改fs\_write 和fs\_read 的逻辑，完成这些以后 ， 如果没有错误，就能够运行仙剑奇侠传了。

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

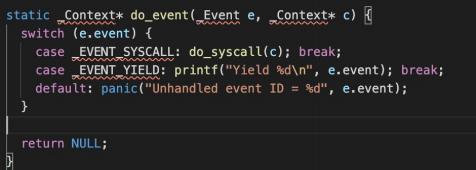




在 cte.c 中完成 irq.c 实现对0x8系统调用和 0x81 自陷的事件分发

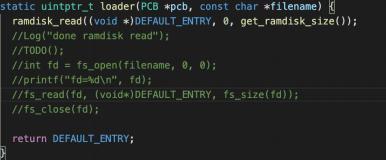


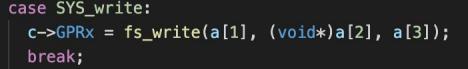
在 irq.c 中完成对自陷的处理

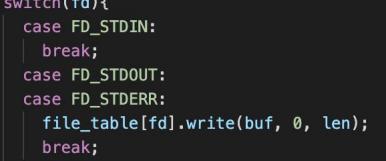


到此完成 PA3. 1

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







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

PA3.2 到此结束。

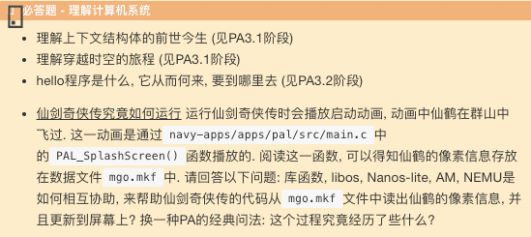
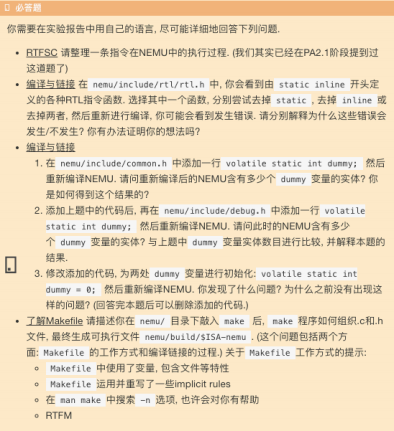
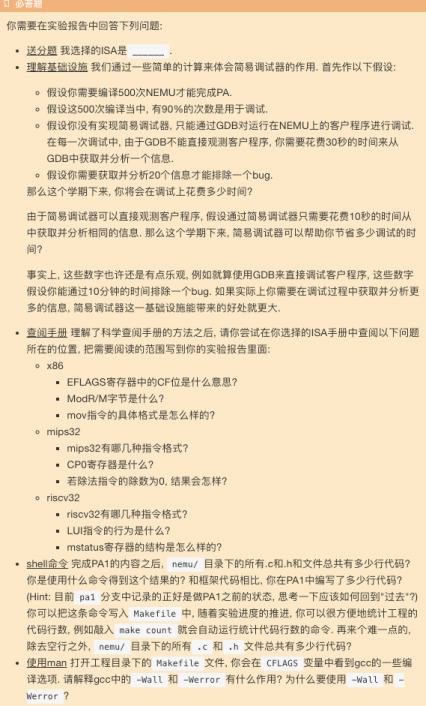
实现 fs\_open， fs\_read， fs\_write 等函数， 修改 loader 读取文件。

修改 makefile， 从 ramdisk 中读取 text 程序进行调试。





**3.5 问题解答**



ISA： riscv32。

调试需要花费的时间是 500\*90%\*30\*20=270000s=4500min

简易调试器可以节约的时间为 500\*90%\*20\*20=180000s=3000min riscv32 有 R、 I、 S、 B、 U、 J 6 种指令格式。

LUI 指令是将 20 位常量加载到寄存器的高 20 位。

mstatus 寄存器的结构如图 2.6 所示。

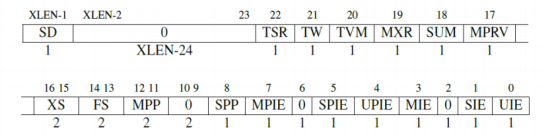


图 2.6 mstatus 寄存器结构

使用命令 find . -name "\*[.h|.c]" |xargs cat|wc –l 得出来的行数是 5877。

使用命令 find . -name "\*[.h|.c]" |xargs cat|grep -v ^$|wc –l 过滤空行后， 得出的 行数是 4822。

-Wall 的作用是打开 gcc 所有警告。

-Werror 的作用是要求 gcc 将所有警告当成错误处理。

1) 一条指令在 nemu 中执行的过程：

首先根据 PC 值通过 instr\_fetch 函数取指令，从选出的指令中选取其opcode ， 在 opcode\_table 中进行索引， 找到该指令对应的译码辅助函数和执行辅助函数， 随后通过译码辅助函数进行译码， 将译码得到的相关信息保存在 decinfo 中， 接 着通过执行辅助函数执行指令，执行辅助函数通过 rtl 指令对译码得到的信息进 行相关操作，计算、读取、 保存等等， 最后通过update\_pc 函数更新 PC 值。

2) 以 rtl\_li 函数为例单独去掉 static 和单独去掉 inline 进行重新编译，都不会 报错， 但将两者同时去掉时， 就会报错。 原因是都去掉时， 在另一个文件中也有 对 rtl\_li 的定义， 会出现重复定义的错误， 而具有 static 关键字时， 函数会被限制 在本文件内， 不会出现重复定义的错误， 具有 inline 关键字时， 函数在预编译时 就会展开， 不会出现重复定义的错误， 但如果将两者同时去掉， 就会出现重复定 义的错误。

3) 添加后， 使用grep 命令查看， 共有 81 个 dummy 实体；继续添加后， 重 新编译运行， 使用 grep 命令， 共有 82 个 dummy 实体； 修改代码后， 重新编译 报错， 原因是两个都初始化后， 会产生两个强符号， 导致错误。

4)敲入 make 后，会将 makefile 文件中第一个目标文件作为最终的目标文件 ， 如果文件不存在， 或是文件所依赖的后面的.o 文件的修改时间比这个文件晚， 就 会重新编译；如果目标文件依赖的.o 文件也不存在，就根据这个.o 文件的生成 规则生成， 然后生成上一层.o 文件，中间某一步出错就会直接报错。

1) c 指向的上下文结构\_Context 是在执行自陷操作时， 通过 trap.S 的汇编程 序运行进行赋值的。 riscv-nemu.h 定义了相关的结构， trap.S 对上下文结构体进 行赋值， 讲义讲清了流程，实现的指令使自陷操作顺利执行。

2) Nanos-lite 调用中断， 操纵 AM 发起自陷指令的汇编代码， 随后保存上下 文， 转入 CPU 自陷指令的内存区域，执行完毕后， 恢复上下文， 返回运行时环 境。

3) hello.c 被编译成 ELF 文件后， 位于 ramdisk 中， 通过 naive\_uload 函数读 入内存并放在正确的位置， 交给操作系统进行调用执行， 它通过 SYS\_write 系统 调用来输出字符，程序执行完毕后操作系统会回收其内存空间。

4) 操作系统通过库函数读出画面的像素信息， 画面通过 VGA 输出， VGA 被抽象成设备文件， fs\_wrtie 函数在一步步执行中调用了 draw\_rect 函数 ， draw\_rect 函数把像素信息写入到 VGA 对应的地址空间中， 最后通过 update\_screen 函数将画面显示在屏幕上。

**4.设计心得与体会**

本次课设我在做的过程中,遇到了很多困难的文体。不过克服困难，我觉得是一件非常有意义的事情。

PA1 的整个核心思想就是实现一个编译器，最难的点可能就是实现四则运算， 遇到了各种各样的玄学 bug。

PA2 就变得十分类似于组原实验了， 对于 riscv32 的指令的不熟悉让我非常 尴尬， 也翻阅了大量资料去解决每一个 bug。

PA3 开始就变得十分精彩了， 因为有游戏可以玩了， 实现了自陷指令、系统 调用、文件系统

这次PA串联了我们在大学前三年里学习的很多内容，从PA1中基本的程序设计和算法，到PA2中涉及到的汇编语言、组成原理等，再到PA3和PA4中的操作系统。通过完成PA的内容，我回顾了很多以前学习过的知识，对其中的一些内容也有了不少新的认识，但我认为这门课程还是存在很多需要改进的地方。

，我希望这门课程设计可以增加一些自动测试和评分的系统，就像组成原理实验与课设中使用的educoder自动测试那样。对于PA1的内容，主要是程序设计和算法的内容，使用大量的样例进行测试即可；对于PA2的内容，由于指令执行的结果具有确定性，因此比较每条指令执行后的处理器状态就可以验证对错。这两部分，虽然我们在实验中也实现了qemu-diff这样的功能，但是倘若老师能够提供一套评测系统，我们在实验中也可以更加准确的对实现结果正确与否进行认定。至于PA3和PA4这些与操作系统相关的内容，参考课程6.828的设计，同样可以提供一套测试评分功能。我认为，对于PA这种以功能复现为目的的实验，自动测评对我们的实验可以起到很大的帮助，它能够让我们准确的判断和掌握实验进度，不至于到了实验后期才发现前面的bug，导致大量的更改。