***2021***



**计算机系统能力综合训练课程实验报告**

|  |  |
| --- | --- |
| 题 目： | RISC-V指令集模拟器的实现 |
| 专 业： | 计算机科学与技术 |
| 班 级： | CS1804班 |
| 学 号： | U201814755 |
| 姓 名： | 彭子晨 |
| 电 话： | 15258242073 |
| 邮 件： | 1289513580@qq.com |
| 完成日期： | 2022-1-13 |

**目 录**

[1 课程实验概述 1](#_Toc93062012)

[1.1 课设介绍 1](#_Toc93062013)

[1.2 课程目标 1](#_Toc93062014)

[2 实验方案设计 4](#_Toc93062015)

[2.1 PA0 4](#_Toc93062016)

[2.2 PA1 4](#_Toc93062017)

[2.3 PA2 7](#_Toc93062018)

[2.4 PA3 11](#_Toc93062019)

[3 实验结果与结果分析 17](#_Toc93062020)

[3.1 PA1实验结果 17](#_Toc93062021)

[3.2 PA2实验结果 18](#_Toc93062022)

[3.3 PA3实现结果 22](#_Toc93062023)

[4 实验回顾与归纳总结 26](#_Toc93062024)

[4.1 问题与解决 26](#_Toc93062025)

[4.2 心得与体会 27](#_Toc93062026)

[参考文献 30](#_Toc93062027)

# 课程实验概述

## 课设介绍

计算机系统能力是指能自觉运用系统观，理解计算机系统的整体性、关联性、层次性、动态性和开放性，并用系统化方法，掌握计算机软硬件协同工作及相互作用机制的能力。系统能力包括系统分析能力、系统设计能力和系统验证及应用能力三个方面，三个方面相辅相成，共同构成计算机专业本科毕业生的基本能力和专业素养。

《系统能力培养综合实践之指令模拟器》就是为了培养计算机专业学生的系统能力而设置的。课程以软件实践为主，内容循序渐进、由浅入深，使学生能够快速入门，且帮助学生通过实践对理论、技术和方法进行巩固和理解，但是又具有高阶性、趣味性和挑战度，不断激发学生的兴趣和创造性；课程强化系统观，能够强化并检验学生的系统化综合能力；课程结合工程应用，能帮助学生理解计算机系统从底层硬件到高层软件的全套技术，使得学生对计算机系统各层次的技术有更加深刻的认知。

## 课程目标

《系统能力培养综合实践之模拟器》的具体目标包括：

**目标 1**：**安装与配置开发工具链。**基于主流版本的虚拟机平台与开源操作系统Linux，安装基本的代码编辑、编译、调试、版本管理软件与重要插件，如，vim、gcc、gdb、git，并掌握这些软件的基本使用方法，为下一步的系统级开发奠定坚实的基础。

**目标 2**：**构建基础编译环境。**通过git工具下载模拟器软件，并根据文档完成基础开发环境的配置。通过此过程充分理解Linux系统中环境变量的功能与重要性，及其在Makefile中的常见使用方式。

**目标 3**：**打造基础调试工具。**开发一些必要的调试基础设施，如，表达式求值功能、监视点、（寄存器）比较器，为解决开发过程中可能遇到的复杂软件缺陷做好准备。

**目标 4：理解模拟器的基本原理。**掌握使用一段程序模拟一条（类）机器指令的基本方法，并能够在系统给出的基本框架下，结合指令集相关文档，实现X86、MIPS或RISC-V等众多主流指令集中的一个。

**目标 5：构造基本运行时环境。**理解运行时环境的基本概念及其在系统中的重要作用，并通过阅读文档，在系统给出的基本框架下，完成主要库函数的功能开发。

**目标 6：构建基本输入输出环境。**理解计算机系统与输入输出设备交互的基本原理与方式方法，并依据文档要求，在系统中添加对于（虚拟）键盘、时钟、图形控制器等常见输入输出设备的支持。在完成上述目标后，系统即能具备较好的可交互性与可展示性。

**目标 7：理解系统调用的基本原理与实现流程。**了解计算机系统权限分级的原因与现状，理解自陷指令的功能与基本流程，理解系统调用的作用，并依据文档要求实现基本的系统调用。

**目标 8：理解文件系统的基本工作原理。**了解文件系统的主要功能，依据文档要求，实现一个简化的文件系统，支撑对复杂应用所包含的独立数据文件的访问，从而使得模拟器能够支持复杂应用。

**目标 9：理解多任务系统的基本工作原理。**了解多进程/任务并行的基本原理，了解虚存的基本原理及其对于多任务并行的重要性，了解分时功能的必要性，按文档要求，实现一个支持上述功能的简易多任务系统。

# 实验方案设计

## PA0

PA0的主要内容是配置环境。包括了虚拟机环境配置、开发环境配置、常用工具安装以及个人信息的设置。

首先去官网下载VitualBox，通过课程wiki上老师给的镜像文件创建好虚拟机。操作系统是Ubuntu20.04(64-bit)，可以防止在之后的开发中因为环境问题多走了弯路。在安装完虚拟机之后就可以克隆网站上的ICS2019框架代码。设置完学号、邮箱、姓名、账号密码等信息后环境配置就差不多结束了。

接下来是开发环境的配置以及常用开发工具安装。我在这里选择了VScode，使用IDE的主要目的是方便文件的切换和管理，而且有高亮提示和一些快捷键能够使后面的开发更加容易。

## PA1

PA1的主要内容是实现一个简易调试器，包括单步执行、打印寄存器状态、扫描内存、实现算术表达式求值等功能。

首先观察文件架构和文档的要求，可知nemu/src/monitor/debug目录下的几个文件是和该功能相关的代码。需要在ui.c里面集合各个指令的功能函数，在cmd\_table中已经有几个框架内置的函数做参考。

1. 帮助

help指令实现起来是最简单的，只要把每个指令对应的指令符号、指令描述、功能函数按统一结构写到cmd\_table中。最后在cmd\_help函数中读表然后将每一项打印出来即可。

1. 单步执行

si [N]是单步执行功能，因为这条指令有两个参数，因此在cmd\_si的实现中使用了strtok来分别获得两个参数，当后面参数为空时则默认为1。参考框架的cmd\_c函数使用cpu\_exec(N)来完成运行N条指令的功能。对于一些特殊情况，比如第二个参数非整型，就可以抛出一个Error信息。

1. 表达式求值

expr方法涉及到了一点编译原理相关知识。根据文档给的eval函数的框架，通过将表达式分割成两个子表达式和一个主运算符的方式进行递归求值，可以得到表达式的结果。在实现的过程中要查阅文档资料获得各个符号的优先级等级，通过正则表达式表示整型数字以及对应的寄存器。在获取主运算符时也需要正确排除左右括号，注意几个条件语句的先后情况。对于除数为零等特殊情况也需要抛出ERROR信息。

1. 扫描内存

scan Memory指令注意用到了上面实现的表达式求值函数expr。获取参数值获得expr的值后得到地址addr。由于riscv32的物理地址是从0x80000000开始的，因此在访问地址前要加上物理内存的起始地址。最后打印出N条地址及其对应的内容，内容可通过vaddr\_read获取。

1. 设置监视点

所有监视点以一个数组的方式建立，互相之间的联系类似链表的形式存在，每个监视点都有如下属性：序号（NO），指向下个监视点的指针（next），表达式（expr），表达式的结果值（val），是否使用中（enable）。在初始化wp\_pool时会生成一个head指针和free\_指令，功能分别是指向使用中的wp和空闲的wp。新建一个监视点的逻辑如下：判断free\_是否为null，若不是则说明有空闲的wp，为这个新的wp的属性赋值并设置enable为true，并让wp->next指向head，而wp成为新的head。最后要遍历数组找到第一个disable为false的wp作为free\_的新值。

1. 删除监视点

删除监视点的逻辑如下：如果需要删除的wp就是head，则直接把head置为wp->next，然后把wp的enable设为false即可；否则通过遍历整条监视点链表判断当前的next是否为对应的wp，若是则直接更改next为wp->next即可。每次删除监视点都要重新设置free\_的值为刚刚删除的监视点。

1. 打印程序状态

info r需要能够打印每个寄存器的名字及状态。寄存器相关信息通过reg.h找到reg\_name函数以及可以对寄存器取值的已经定义好的reg\_l函数。reg相关功能也写在reg.c文件下，isa\_reg\_display()函数完成了打印和输出的功能。

info w需要能够打印正在使用中的监视点信息，从head开始遍历整个监视点链并按照一定格式输出对应监视点的序号、表达式、表达式值即可。

## PA2

PA2总共有三个阶段性任务。这一阶段的核心内容是理解CPU执行一条指令的过程，按照取指、译码、执行的过程实现多条指令。

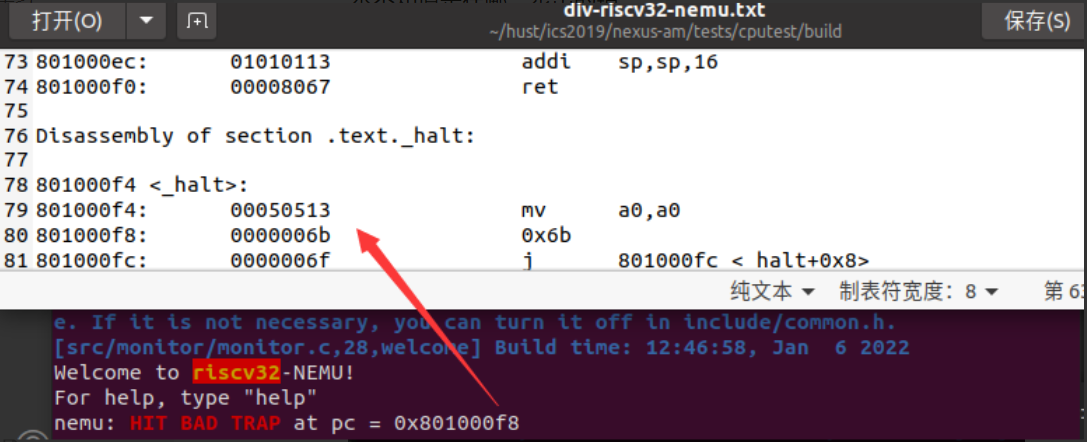
* Task PA2.1: 在nemu中运行第一个C程序dummy
* Task PA2.2: 实现更多指令，在nemu中运行所有cputest
* Task PA2.3: 运行打字小游戏

### Diff-test

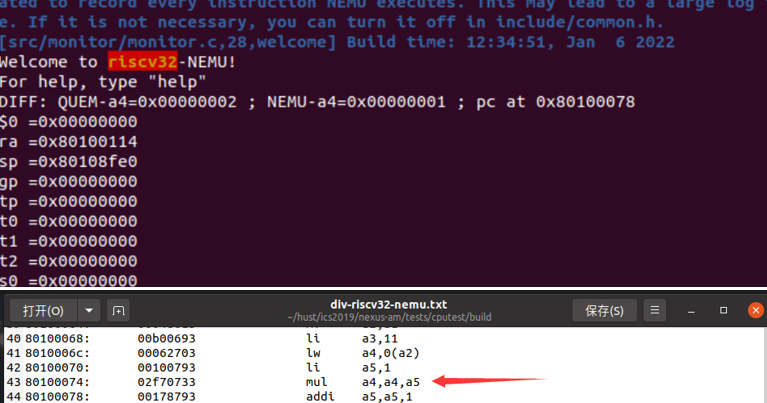
在实现这么多指令的过程中debug是一件很麻烦的事情，在指令发生错误的时候只能看到Hit Bad Trap，错误抛出的pc值也是在0x6b的nemu\_trap指令上，对发现bug没有帮助。因此，通过和QEMU每一步对比实现的Diff-Test可以及时捕捉到Error。

Diff-Test只要实现isa\_difftest\_checkregs()函数即可，把通用寄存器和PC从QEMU中读出的寄存器的值进行比较，若结果一致则返回true，若不一样则打印错误信息并返回false，框架代码会自动停止客户程序的运行。

下图 2‑1和图 2‑2分别是不开Diff-Test和开启Diff-Test的对比，可以明显看出在不开difftest的时候只能看到PC停在nemu\_trap，而开启后可以明显看到a4寄存器出了问题，PC位置也定位了错误未知，查看txt文件PC为0x80100074的指令就能发现是一个mul指令。



**图 2‑1 未开DIFF-TEST**



**图 2‑2 开启DIFF-TEST**

### PA2.1

为了能成功运行dummy，通过查看dummy-riscv32-nemu.txt可以发现未实现的指令有li、auipc、addi、jal、li、ret、lui，通过查阅Risc-V手册可知li、ret为伪指令，分别扩展为lui和jalr，因此能够确定我们需要实现的指令。通过STFW和RTFC，可以总结出实现一条指令的完整流程所需要的步骤：

1. 在exec.c的opcode\_table中添加新指令，根据opcode6\_2确定指令的位置，用IDEX进行译码和执行；
2. 在exec的all-instr.h中声明执行辅助函数；
3. 在include/isa/decode.h中声明译码辅助函数；
4. 在decode.c中实现具体的译码辅助函数；
5. 在exec目录下的对应c文件内实现具体的译码执行函数。
6. 在rtl.h中适当完善所需的rtl指令。

接下来的主要任务就是根据Risc-V手册的指令编写每条指令对应的译码辅助函数以及执行辅助函数。指令实现正确后运行dummy会得到HIT GOOD TRAP。

### PA2.2

相比PA2.1，PA2.2需要实现能够成功运行整个cputest的指令。在熟悉了实现一条指令的流程之后就只是重复的流水线工作，在实现指令的过程中应该充分理解Instr结构，能在实现各类指令的时候更好地进行区分和判断。在实现的过程中可以通过一键回归测试快速检查所有文件的情况，开启DIFF-TEST并检查没有PASS文件的txt可以快速找到还未正确实现的指令。

PA2.2需要实现的指令主要分成5类：

1. B型指令。需要实现的B型指令都是跳转指令，所以可以统一用rtl\_jrelop方法进行处理；
2. R型指令。需要实现的R型指令的数量超过了7个，因此需要通过funct7和funct3一起区分对应的指令；
3. I型指令，立即数操作的一类指令。
4. Load指令，主要需要补充lb和lh指令；
5. Store指令，由于框架已经写好所以只需要补充store\_table即可。

在实现完以上指令后大部分文件都已经PASS了，根据文档最后需要实现的文件是string.c和hello-str.c。string.c要求实现strcpy、strcat等常用库函数，而hello-str.c要求实现sprintf以及vsprintf（支持%d与%x），只需要注意一下指针的使用和一些特殊情况即可。

### PA2.3

PA2.3的任务是能够运行打字小游戏。因此需要实现串口、键盘、时钟、VGA这些IO设备的功能。

在AM中可以通过使用native来检查正确的运行结果。通过给mainargs=H可以看到菜单以及所需实现的测试指令。

1. 串口

在PA2.2已经实现了vsprintf和sprintf，因此在这里只需要实现printf即可，整体实现思路和sprintf基本一致。根据main.c可知H指令还需要实现printf中的“%c”（这个功能在后面的PA3代码中实现）。

1. 时钟

时钟的实现主要是初始化部分以及时间更新部分。由于文档不要求实时时间，所以不需要更改\_DEVREG\_TIMER\_DATE部分。初始化函数是\_\_am\_timer\_init()，可以在nemu.h找到RTC寄存器地址的定义，访问到当前时间，在初始化时赋值给start\_time。在\_DEVREG\_TIMER\_UPTIME中访问到当前时间cur\_time，将cur\_time-start\_time赋值给lo来表示系统启动的毫秒数。

1. 键盘

当按下一个键的时候，键盘会发送该键的通码；当释放一个键的时候，键盘会发送该键的断码。可以通过KBD\_ADDR获取键盘的信号。由于通码的值为断码|0x8000，可以理解为最高位取反，因此通过key&0x8000可以知道是断码还是通码。因为keycode为按键的断码，keydown=1表示按下，=0表示释放，所以可以推断：

接下来就只要在\_DEVREG\_INPUT\_KBD用三目表达式写入相应的逻辑即可。

1. VGA

VGA是用于显示颜色像素的输出设备。由于代码只模拟了400x300x32的图形模式，因此需要在\_DEVREG\_VIDEO\_INFO中设置宽高为400x300。由于pixels的内容是按照行优先排列的，因此要向（x,y）坐标绘制w\*h的矩形图像，需要计算每个像素对应到fb中对应的位置。

## PA3

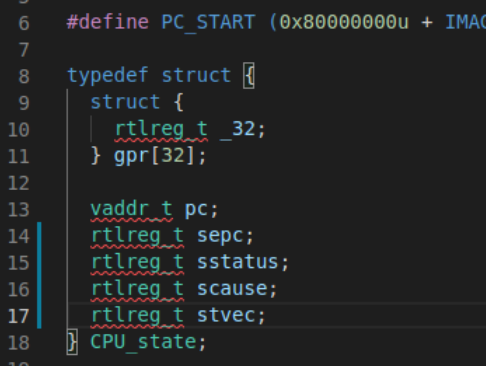
PA3的最终目标是运行仙剑奇侠传。主要内容分为三个阶段：

* Task PA3.1: 实现自陷操作\_yield()及其过程
* Task PA3.2: 实现用户程序的加载和系统调用
* Task PA3.3: 运行仙剑奇侠传并展示批处理系统

### PA3.1

PA3.1的任务是实现自陷操作\_yield()及其过程。需要理解触发异常后riscv32处理异常事件的执行过程。

首先要实现新指令ecall，在ecall中调用raise\_intr()函数。raise\_intr()函数的功能就是riscv32触发异常后硬件的响应过程：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。而保存程序状态以及跳转到异常入口地址的工作, 都是由硬件自动完成的,不需编写指令。需要的寄存器类型需要自行在cpu的结构体内定义，如下图 2‑3所示：



**图 2‑3 寄存器定义**

第二个任务需要实现新指令并重新组织\_Contetx结构体。通过观察trap.S文件可以发现还未实现的指令包括csrr（扩展为csrrs）、csrw（扩展为csrrw）和sret，根据RiscV文档一一实现即可。而根据trap。S最开始压栈的顺序是先push regs，再是scause、sstatus、sepc，根据这个顺序再去修改\_Context结构体为正确顺序。

第三个任务需要实现正确的事件分发。根据\_yield()中的汇编代码“li a7, -1; ecall”可以判断出a7寄存器中的值为-1就表示event\_yield事件。因此可以在\_\_am\_irq\_handle中通过-1异常号识别出自陷异常，并在do\_event中根据不同的EVENT打印不同的内容。

最后的任务是恢复上下文，也就是sret指令，注意通过之前保存sepc回复上下文时需要对pc+4。最后的输出结果为打印了yield时间后返回并触发了main函数结尾的panic。

### PA3.2

PA3.2的主要任务是实现用户程序的加载和系统调用。

第一个任务就是实现loader，loader函数通过把用户程序和数据加载到正确的内存位置，然后直接执行用户程序的入口地址entry起到加载用户程序的作用。通过文档整理出四个问题的答案：

1. 可执行文件在哪？

答：在ramdisk偏移为0处；

1. 代码和数据在可执行文件的哪个位置？

答：通过Header可以知道每个对应Segment的位置，根据Segment判断从ELF文件的第Offset字节开始；

1. 代码和数据有多少？

答：在内存中占用大小为MemSiz，内容大小为FileSiz;

1. “正确的内存位置”在哪里？

答：以VirtAddr为首地址的虚拟内存位置。

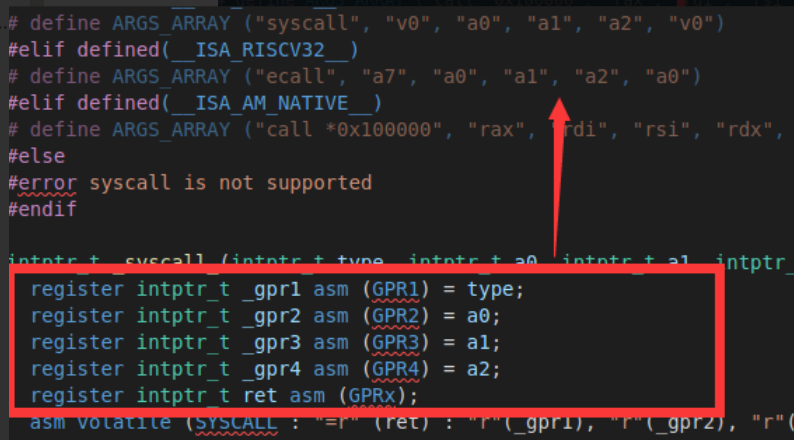
理解了loader加载用户程序的步骤后需要理解可执行文件头Ehdr和Phdr的关系。通过查看结构体的相关注释可知Ehdr是ELF文件header部分的数据结构，而Program Header的数据结构是Phdr，Phdr就是之前提到的Segment的结构，包括我们所需要的相关参数。

Loader函数的实现思路是先读取header，然后获得header属性中的e\_phoff和e\_phnum得知Segment的offset和数量。循环判断每个Segment的type属性是否为PT\_LOAD,如果不是则不需加载。然后将数据和程序通过ramdisk\_read函数拷贝到虚拟内存处，最后通过memset将[filesz, memsz)的位置清零。返回值为程序入口地址。

在这一步的ramdisk初始化Log会通过%p和%x打印地址，因此需要在vsprintf补充这两种情况的实现。

第二个任务是识别系统调用。通过观察native的cte.c发现将YIELD以外的事件都处理成了SYSCALL事件，然后在do\_event事件中的EVENT\_SYSCALL内调用do\_syscall()函数。

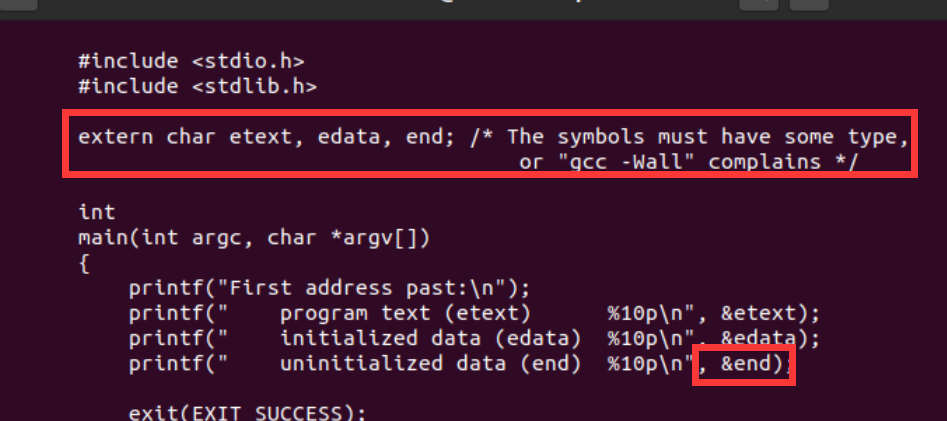
第三个任务是实现系统调用，根据navy-apps中宏定义的\_\_ISA\_RISCV32\_\_寄存器以及\_syscall\_函数的实现，可以写出正确的GPR宏，如下图 2‑4所示。在do\_syscall中添加SYS\_yield系统调用，并设置GRPx中的返回值为0。实现成功后dummy会HIT GOOD TRAP。



**图 2‑4 RISCV的GPR寄存器**

第四个任务是运行hello，需要实现SYS\_write，判断参数fd如果为1或2则使用putc进行一个字符一个字符的输出，若输出成功则返回值为len，若失败则返回-1。实现成功后hello会循环输出结果。

最后的任务是实现堆区管理，就按照wiki文档介绍的\_sbrk()的工作方式写即可。因为要保存旧的program break，因此需要在声明的时候加上static关键词。下图 2‑5为man中end的使用方法：



**图 2‑5 end的使用方法**

### PA3.3

PA3.3的任务是实现文件系统，并将串口、时钟、键盘、VGA等IO设备抽象成文件的形式实现。

第一个任务是实现文件系统，主要是实现fs\_open, fs\_read, fs\_close(),fs\_write()和fs\_lseek()。STFW后实现即可，注意对文件读写后open\_offset的变化。为了支持用文件名调用用户程序的功能，loader函数也需要重新改写，通过使用fs\_lseek()以及fs\_read()实现用户程序和数据的加载功能。然后在nano.c中添加open、write、close、lseek、read的系统调用函数，再在syscall.c补上对应的系统调用处理。在修改完naïve\_uload的filename参数后,若是实现正确，“/bin/text”会输出“PASS！！！”。

第二个任务是把串口抽象成文件，主要内容就是把之前fd=1或2的情况写到serial\_write内，然后将file\_table中stdout以及stderr的

write设置为serial\_write。

第三个任务是把设备输入（键盘、时钟）抽象成文件。需要在events\_read中使用read\_key()和uptime()两个IOE的API获取键盘与时钟的输入，然后通过条件判断语句保证键盘事件的优先级。

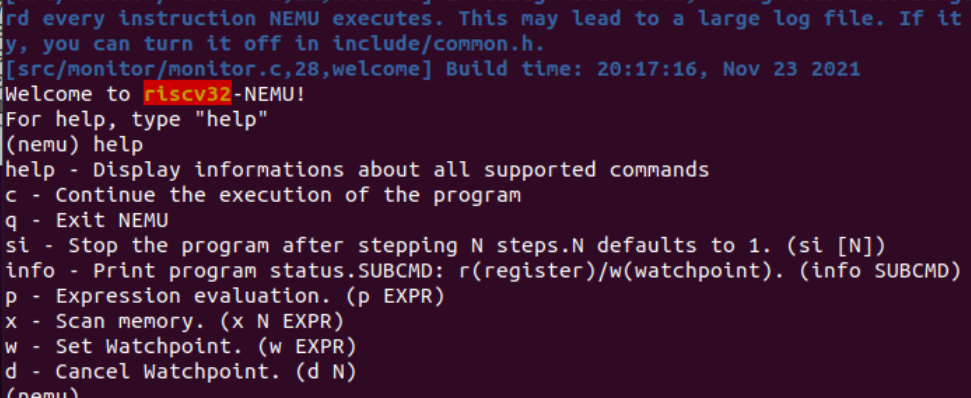
然后根据文档需要在file\_table中添加对/dev/fb，/dev/fbsync和/proc/dispinfo三个文件的支持并实现相关读写函数。完成之后就可以显示Project-N的logo图片了。

最后一步就是下载仙剑奇侠传安装包饼放到对应目录下，在Nanos-lit中加载并运行/bin/pal即可运行仙剑奇侠传。

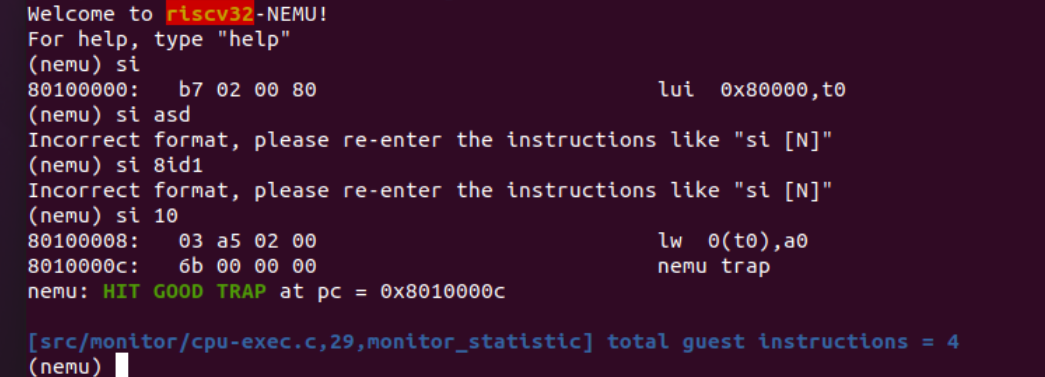
# 实验结果与结果分析

## PA1实验结果

以下是PA1各指令的运行结果：



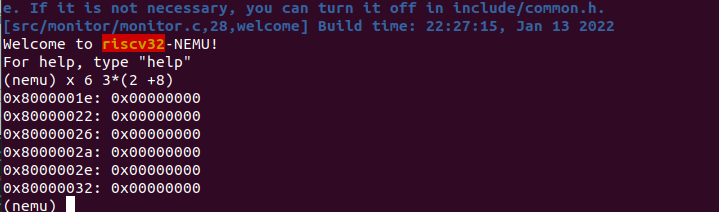
**图 3‑1 help指令**



**图 3‑2 si指令**



**图 3‑3 p指令**



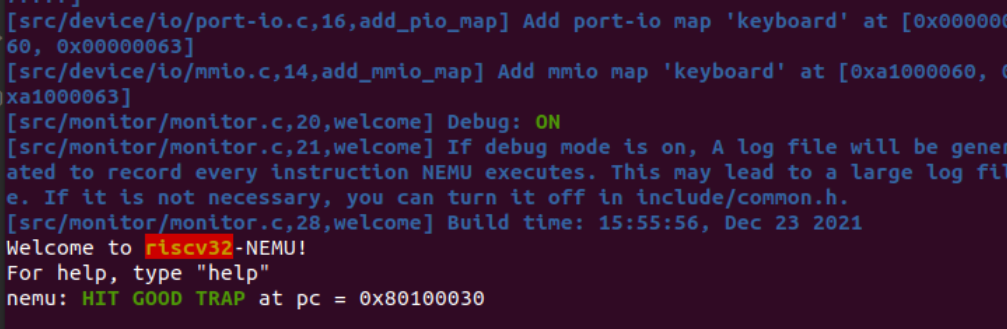
**图 3‑4 x指令**



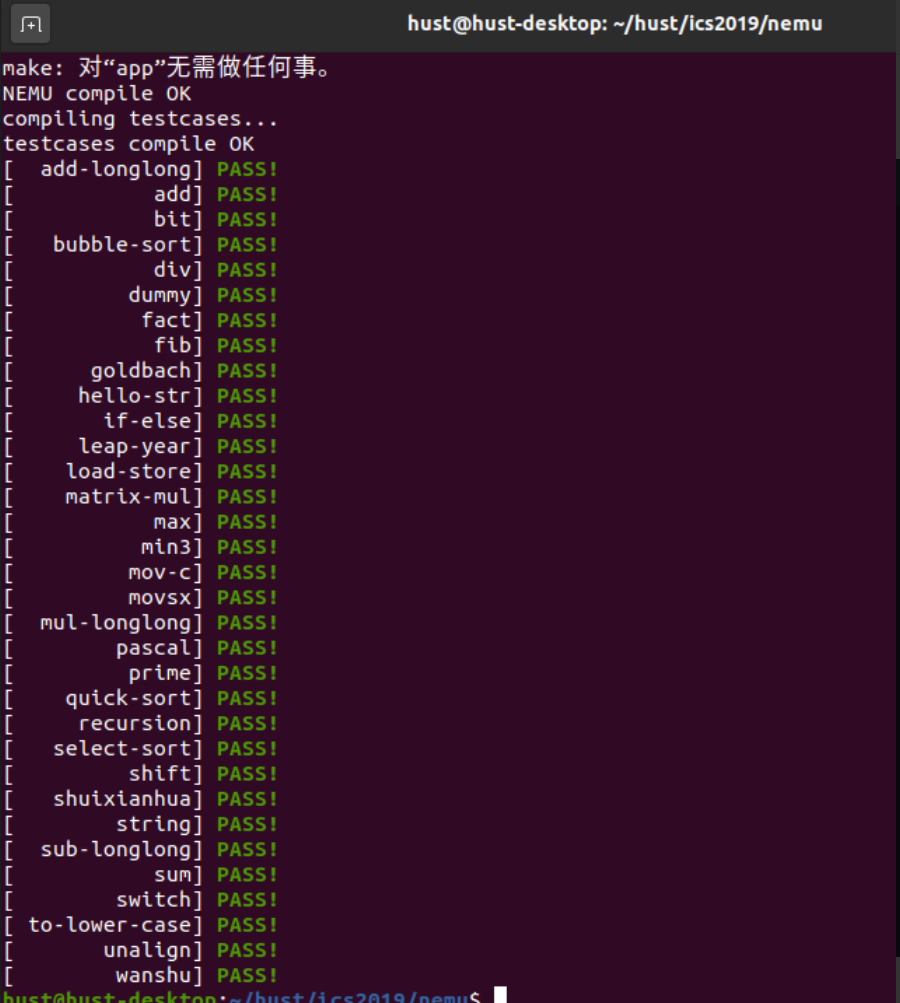
**图 3‑5 监视点相关指令**

## PA2实验结果

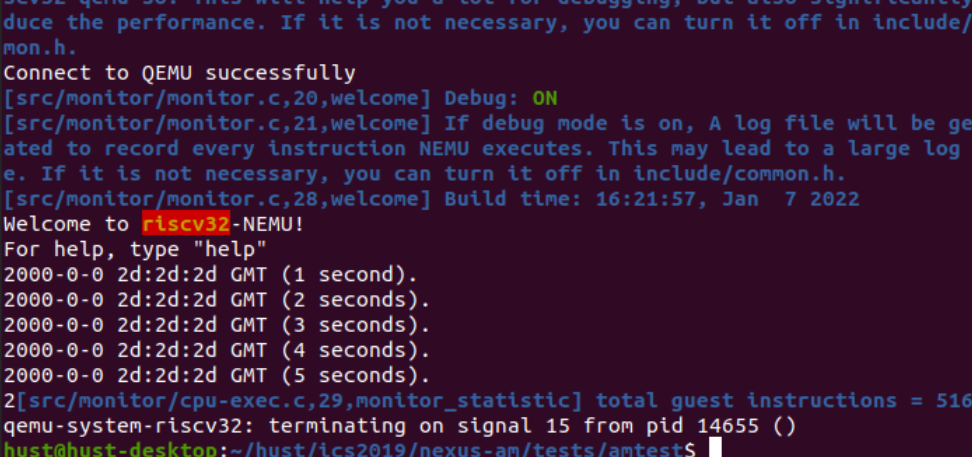
以下是PA2的实验运行结果：



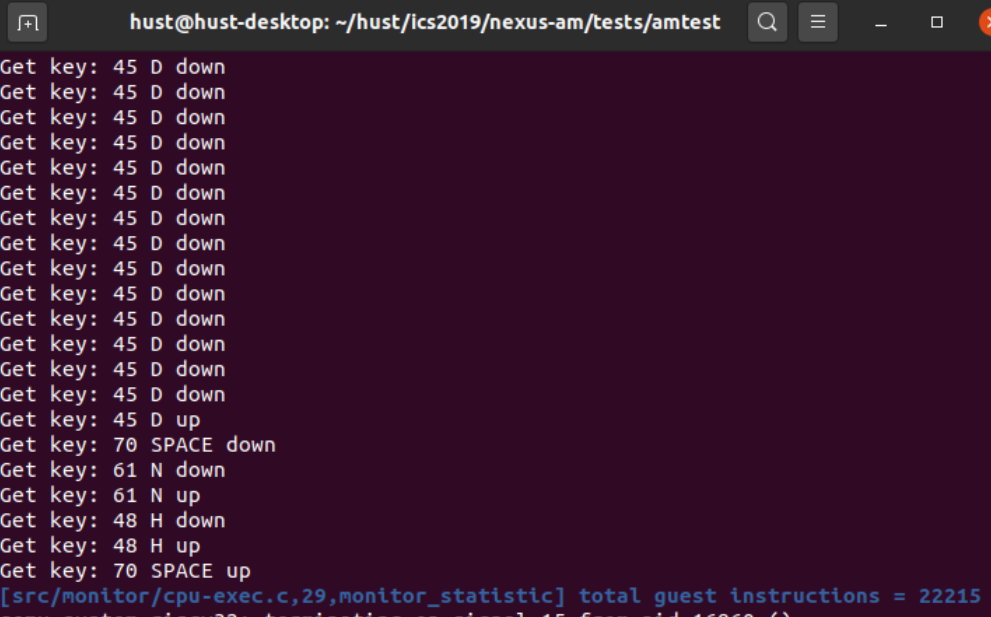
**图 3‑6 dummy指令运行成功**



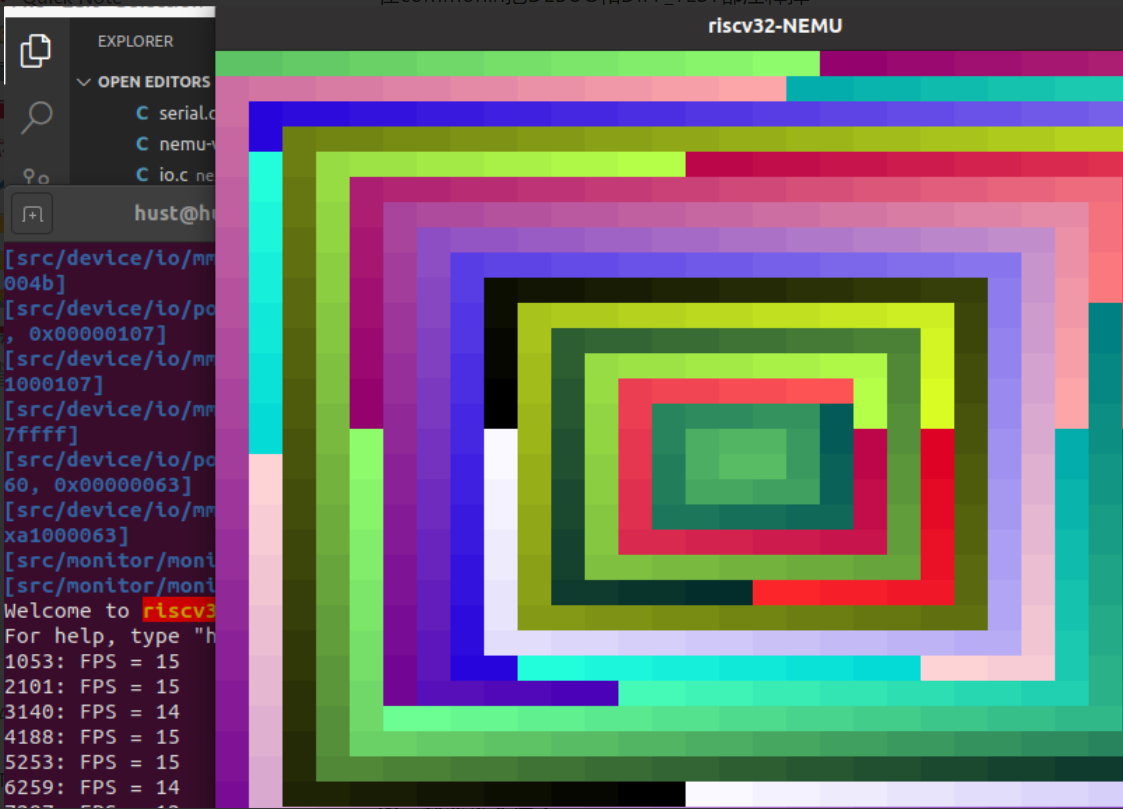
**图 3‑7 cputest运行成功**



**图 3‑8时钟实现**



**图 3‑9 键盘实现**

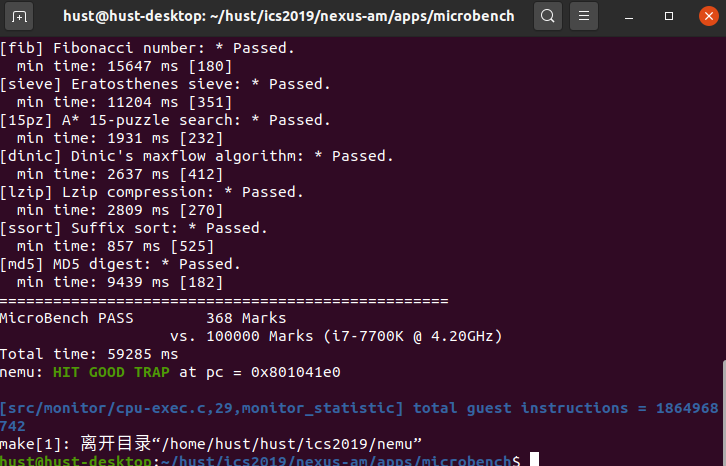


**图 3‑10 VGA实现**



**图 3‑11 打字游戏实现**

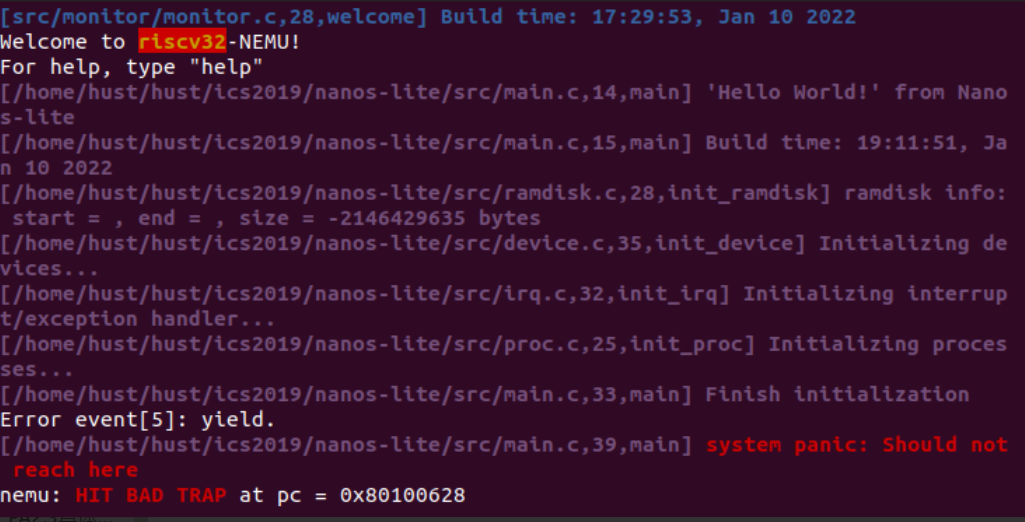
下面是跑分程序，结果是368分。



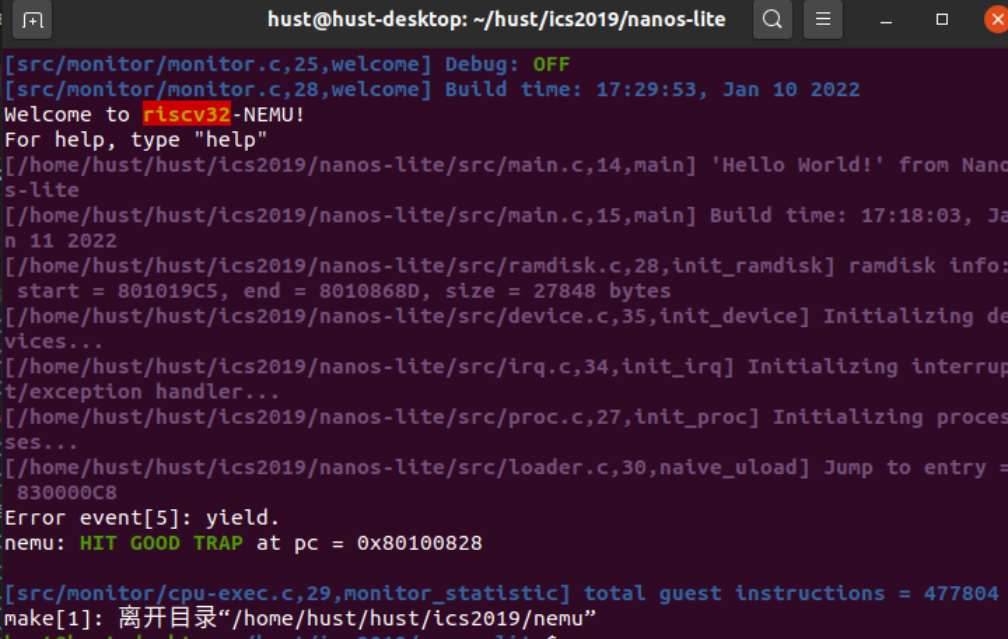
**图 3‑12跑分结果**

## PA3实现结果

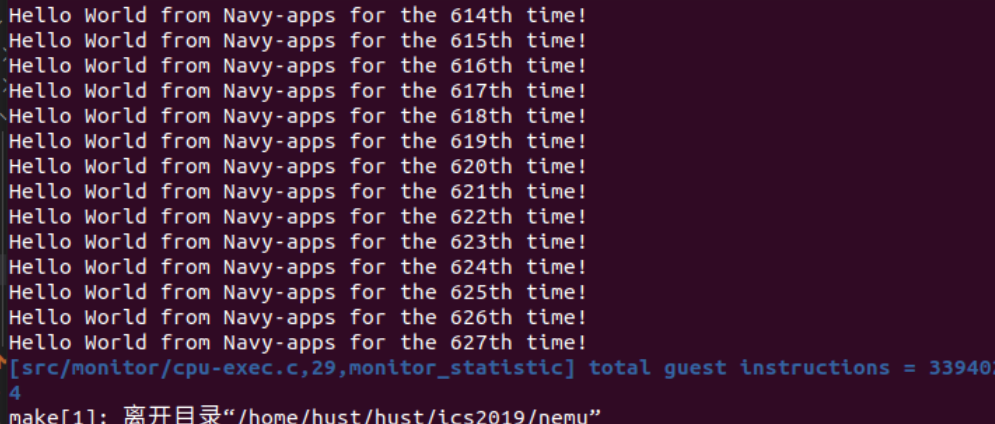
以下是PA3的实验运行结果：



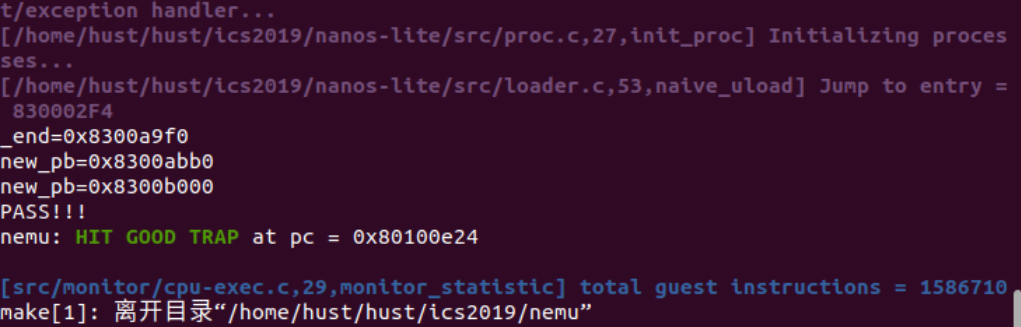
**图 3‑13 触发yield异常触发main的panic**



**图 3‑14 dummy成功运行**



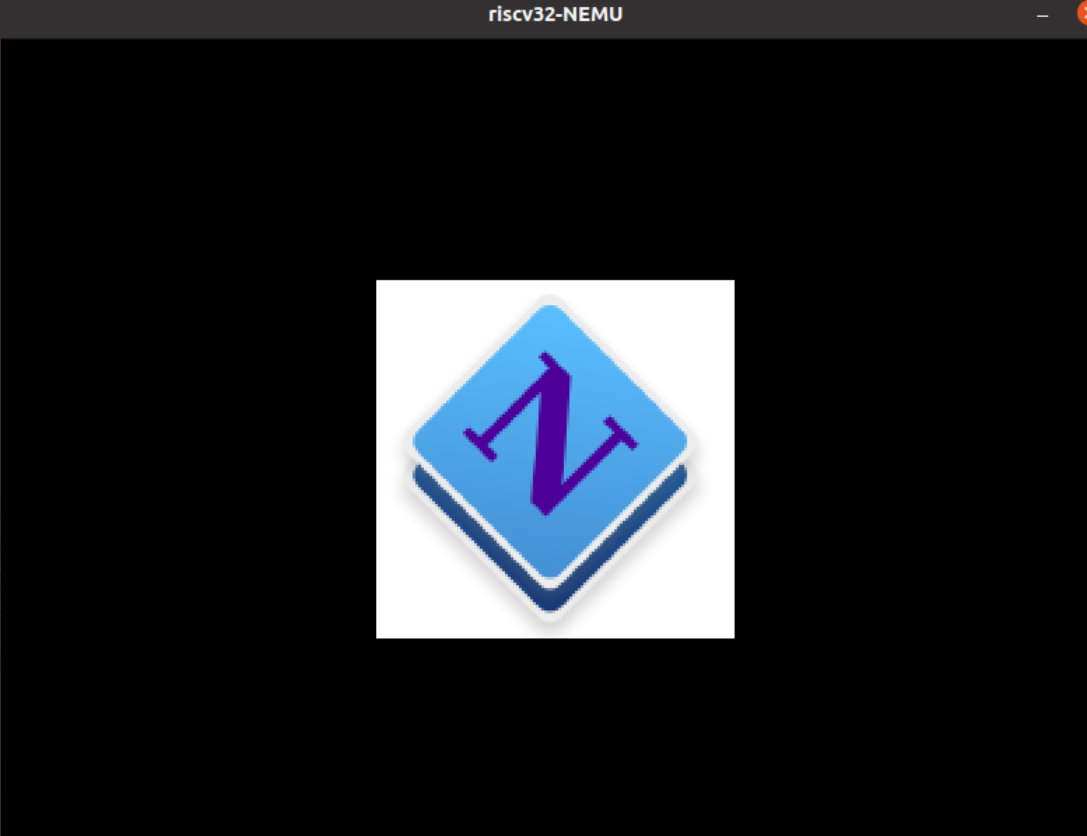
**图 3‑15 hello成功运行**



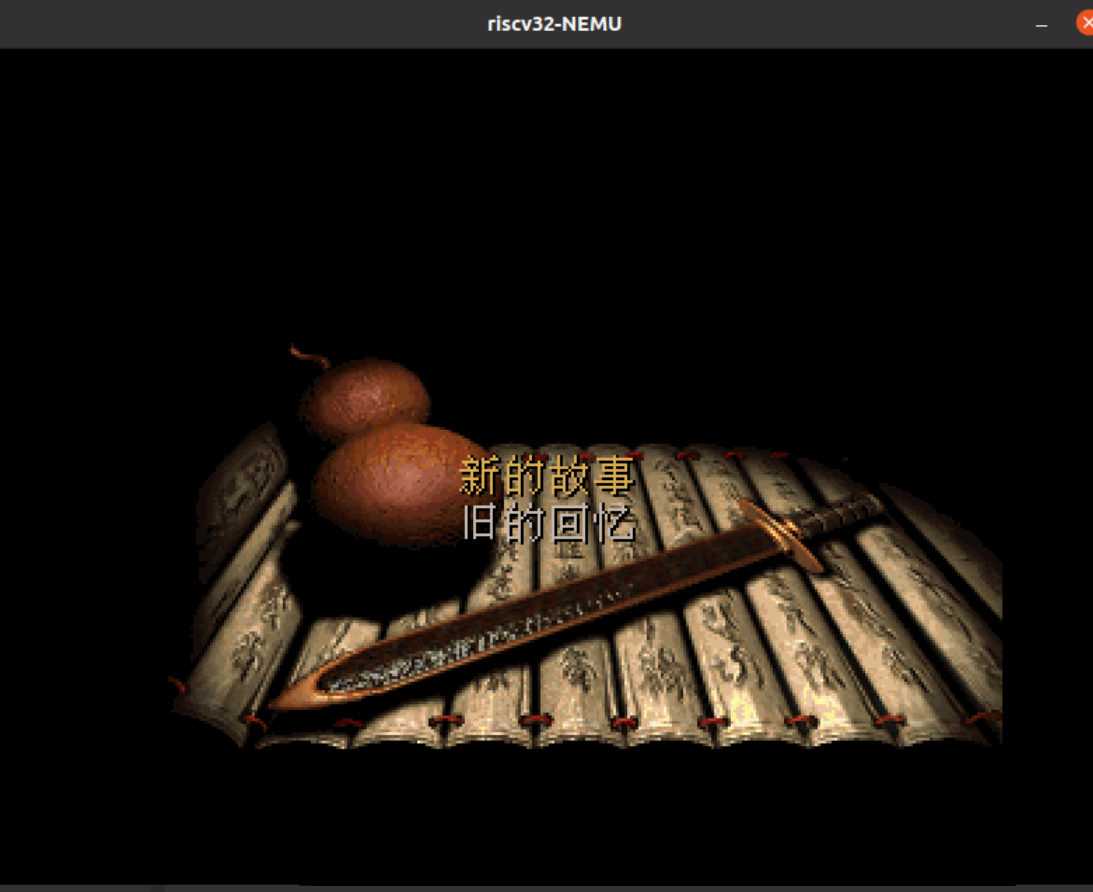
**图 3‑16 text成功运行**



**图 3‑17 events成功运行**



**图 3‑18 bmptest成功运行**



**图 3‑19 仙剑奇侠传成功运行**



**图 3‑20 仙剑奇侠传游戏界面**

# 实验回顾与归纳总结

## 问题与解决

在做PA实验的过程中，遇到了各种各样的问题和困难。总结下来分三类：一类是看文档和读源码就能解决的，一类是需要理解工作原理的，还有一类是单纯的C语言语法问题。接下来是我在实验过程中遇到的一些相对重点的问题记录以及最后的解决方案。

在PA1遇到的最大问题就是在写表达式求值相关的eval()函数的时候一直计算错误，为了debug在每个计算步骤都打印了中间结果，然后发现是主运算符查找错误的原因。主运算符的计算要考虑两个条件：1.要跳过并匹配左右括号；2.必须是加减乘除与或运算符。最后通过定位错误的方式发现两个条件判断语句的先后顺序有误，导致在发现不满足条件2的时候直接进入了下一步，而没有把括号计算进去。除了此问题以外就是没有认真阅读文档，导致一开始不知道在查询内存的时候没有加上物理内存的起始地址导致地址越界。

在PA2实现指令期间的最多时间就是在查询和理解Risc-V指令文档。在实现PA2.1之前由于没有写Diff-Test所以导致指令报错时debug要耗费很多无用时间去寻找错误的指令，在写完Diff-Test之后查找错误指令的时间大大缩短，方便精准定位错误。而在实现PA2.3的时候也因为没有设置屏幕大小、像素位置计算错误而出现了各种各样的问题。整体来说实现了Diff-Test的PA2虽然代码工作量比较大（要实现很多指令），涉及很多相关文件，但是debug时定位问题也更加轻松了。

我在调试PA3的过程中，发现Diff-Test不再那么好用，缓慢的速度经常导致我误认为实现出了错，而由于PA3的错误基本发生在异常调用以及文件读写函数中，所以就关闭了DIFF-TEST，取而代之用Log打印相关变量以及地址来定位错误。

在做PA的过程中我在PA3.2的hello显示部分被一个bug卡了两天，具体问题是在hello中的printf语句一旦用“%d”输出大于10的值就会报错“address（0x7fffffff）is out of bound”，而使用“%x”就能正常输出。因为在堆栈管理\_sbrk()中使用sprintf也有相同的问题，一开始我认为是自己实现的vsprintf有问题，所以多次修改，但是结果不变，最后选择把vsprintf全部删除，发现问题还在，甚至把整个PA3重写了一遍也依旧没有解决。最后通过询问老师才知道这里的sprintf是标准实现，很有可能是PA2的指令出现了问题。最后一个个排查才发现是少实现了一个“divu”指令，由于之前的测试集都没有涉及到才成了漏网之鱼。通过结果逆推过程其实可以判断，标准实现的“%d”在大于10的时候会通过除法（应该就是divu指令）获得每个位数，而“%x”之所以没有问题估计是使用的左移shl、sal指令。这个问题让我意识到前后的关联性，以及自己的不足，在debug工具不好用的情况下无法精准定位到问题。

## 心得与体会

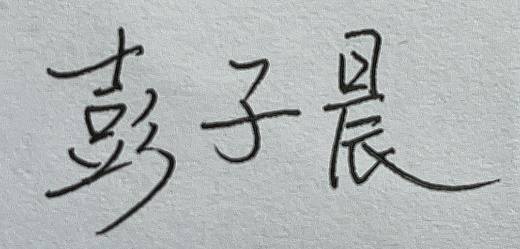
这次PA实验是大学过程中综合程度最高，也最有启发性的一个实验。官方文档不会一个个步骤列出来交你怎么做，而是给了你原理解释和任务需求，需要自己深入阅读和理解文档，通过STFW和RTFC去解决这个大工程。

在做PA的过程中常常被一个问题、一个报错卡住一个下午或一个晚上，最后的体会反而是文档内的每一句话都很有用，即使第一遍看下来只理解了50%，再看一遍配合STFW就能到达70%、80%，再根据看懂的内容去实验文件内写代码调试，根据BUG信息反馈调试后自己的理解也能上升到100%。整个做实验的过程就是一个不断理解、不断试错、不断进化的过程。

另外我很深的一个感触就是前面的实现会影响后面的结果。在快到截止日期的时候，我在PA3.2的hello显示部分被一个bug卡了两天两夜，具体问题在上面的问题与解决部分。最后回顾自己做PA的过程中发现早在PA2结束的打字小游戏中就有了征兆。打字小游戏左上角的FPS并没有稳定在几十这样的正常数值，反而是一直在往上升，从几百到几千再到几万。由于没有影响打字游戏的功能所以我就没有深究原因，没有想到这个得过且过的心态直接影响了最后PA3的运行，导致自己反而找不到问题的根源所在。因此在实现PA2的指令集时，不应该只抱着能通过所有测试集的心态，而是应该尽量抱着一个系统设计和实现者的心态去完成这个指令集。

这次PA实验让我收获了很多，包括阅读和理解文档的能力，定位错误和debug的能力，以及STFW和RTFC的能力。感谢老师在我遇到无法解决问题时的提点，感谢这门实验的创造者和文档撰写者，感谢这次宝贵的经验，让我的开发者意识得到了锻炼。

电子签名:



# 参考文献

1. 张晨曦，王志英. 计算机系统结构. 高等教育出版社，2008年.