

华中科技大学

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
1.1	课设介绍	1
1.2	课程目标	1
2	实验方案设计	4
2.1	PA0	4
2.2	PA1	4
2.3	PA2	7
2.4	PA3	11
3	实验结果与结果分析.....	17
3.1	PA1 实验结果	17
3.2	PA2 实验结果	18
3.3	PA3 实现结果	22
4	实验回顾与归纳总结.....	26
4.1	问题与解决	26
4.2	心得与体会	27
	参考文献	30

1 课程实验概述

1.1 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。

1.2 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。

2 实验方案设计

2.1 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。

2.2 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。

（2）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。

（3）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。

（4）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。

（5）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。

（6）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。

（7）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。

2.3 PA2

PA2 总共有三个阶段任务。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

2.3.1 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。

```
div-riscv32-nemu.txt
~/hust/ics2019/nexus-am/tests/cputest/build

73 801000ec: 01010113      addi    sp,sp,16
74 801000f0: 00008067      ret
75
76 Disassembly of section .text._halt:
77
78 801000f4 <_halt>:
79 801000f4: 00050513      mv      a0,a0
80 801000f8: 0000006b      0x6b
81 801000fc: 0000006f      j       801000fc < halt+0x8>

e. If it is not necessary, you can turn it off in include/common.h.
[src/monitor/monitor.c,28,welcome] Build time: 12:46:58, Jan 6 2022
Welcome to riscv32-NEMU!
For help, type "help"
nemu: HIT BAD TRAP at pc = 0x801000f8
```

图 2-1 未开 DIFF-TEST

```
div-riscv32-nemu.txt
~/hust/ics2019/nexus-am/tests/cputest/build

40 80100068: 00b00693      li      a3,11
41 8010006c: 00062703      lw      a4,0(a2)
42 80100070: 00100793      li      a5,1
43 80100074: 02f70733      mul     a4,a4,a5
44 80100078: 00178793      addi    a5,a5,1

DIFF: QUEM-a4=0x00000002 ; NEMU-a4=0x00000001 ; pc at 0x80100078
$0 =0x00000000
ra =0x80100114
sp =0x80108fe0
gp =0x00000000
tp =0x00000000
t0 =0x00000000
t1 =0x00000000
t2 =0x00000000
s0 =0x00000000
```

图 2-2 开启 DIFF-TEST

2.3.2 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`。

2.3.3 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。

2.3.4 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）。

（2）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。

（3）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 $\oplus 0x8000$ ，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：

若`key`为通码，则`keycode = key`, `keydown = 0`

若`key`为断码，则`keycode = key^0x8000`, `keydown = 1`

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。

(4) 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。

2.4 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

2.4.1 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：

```
8 typedef struct {
9     struct {
10         rtlreg_t _32;
11     } gpr[32];
12
13     vaddr_t pc;
14     rtlreg_t sepc;
15     rtlreg_t sstatus;
16     rtlreg_t scause;
17     rtlreg_t stvec;
18 } CPU_state;
```

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 `_Context` 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`。

2.4.2 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；

(2) 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；

(3) 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`；

(4) “正确的内存位置”在哪里？

答：以 `VirtAddr` 为首地址的虚拟内存位置。

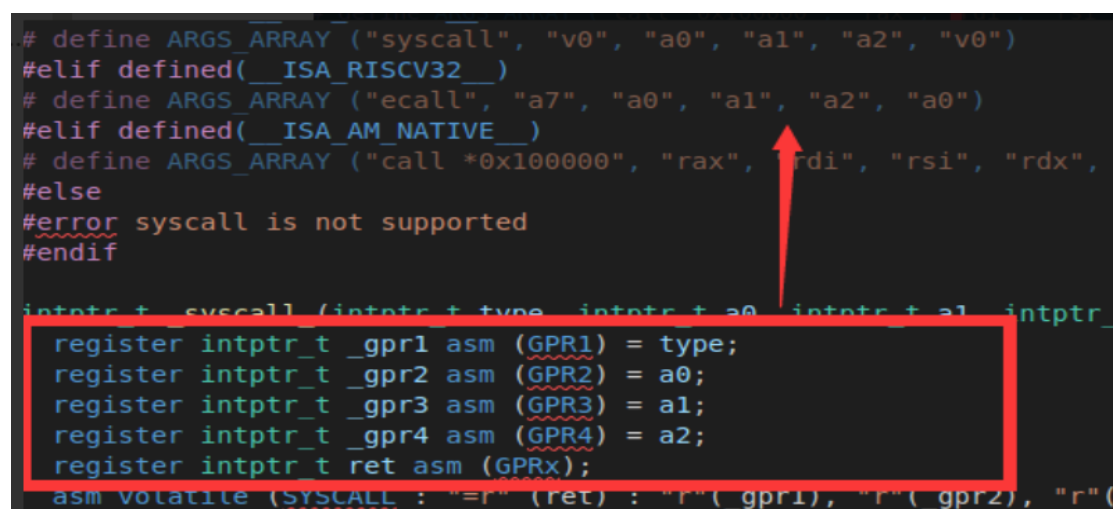
理解了 `loader` 加载用户程序的步骤后需要理解可执行文件头 `Ehdr` 和 `Phdr` 的关系。通过查看结构体的相关注释可知 `Ehdr` 是 ELF 文件 `header` 部分的数据结构，而 `Program Header` 的数据结构是 `Phdr`，`Phdr` 就是之前提到的 `Segment` 的结构，包括我们所需要的相关参数。

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 GRP_x 中的返回值为 0。实现成功后 dummy 会 HIT GOOD TRAP。



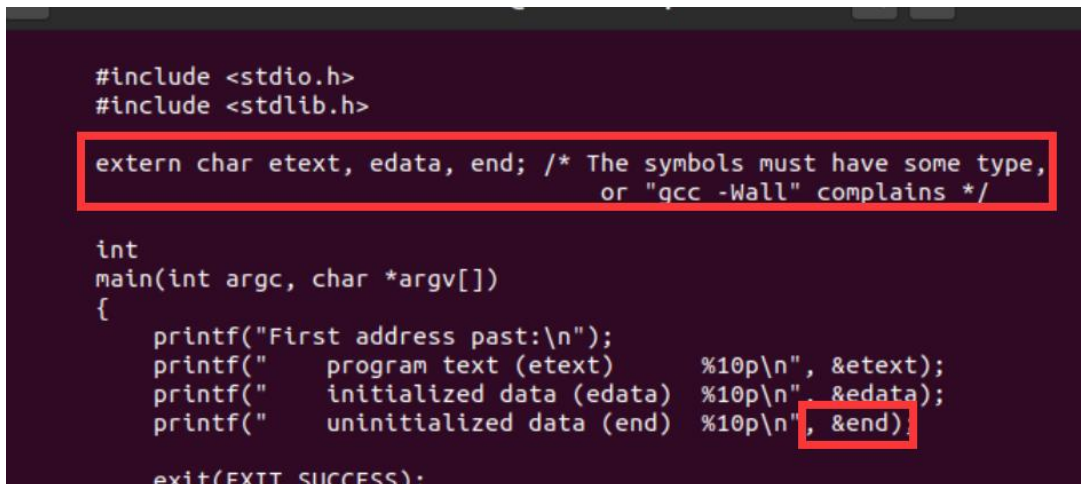
```
# define ARGS_ARRAY ("syscall", "v0", "a0", "a1", "a2", "v0")
#elif defined(__ISA_RISCV32__)
# define ARGS_ARRAY ("ecall", "a7", "a0", "a1", "a2", "a0")
#elif defined(__ISA_AM_NATIVE__)
# define ARGS_ARRAY ("call *0x100000", "rax", "rdi", "rsi", "rdx",
#else
#error syscall is not supported
#endif

intptr_t _syscall (intptr_t type, intptr_t a0, intptr_t a1, intptr_t
register intptr_t _gpr1 asm (GPR1) = type;
register intptr_t _gpr2 asm (GPR2) = a0;
register intptr_t _gpr3 asm (GPR3) = a1;
register intptr_t _gpr4 asm (GPR4) = a2;
register intptr_t ret asm (GRPx);
asm volatile (SYSCALL : "=r" (ret) : "r" (_gpr1), "r" (_gpr2), "r" (
```

图 2-4 RISC-V 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：



```
#include <stdio.h>
#include <stdlib.h>

extern char etext, edata, end; /* The symbols must have some type,
                               or "gcc -Wall" complains */

int
main(int argc, char *argv[])
{
    printf("First address past:\n");
    printf("    program text (etext)    %10p\n", &etext);
    printf("    initialized data (edata) %10p\n", &edata);
    printf("    uninitialized data (end) %10p\n", &end);
    exit(EXIT_SUCCESS);
}
```

图 2-5 `end` 的使用方法

2.4.3 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_upload` 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/fb sync 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并放到对应目录下，在 Nanos-lit 中加载并运行/bin/pal 即可运行仙剑奇侠传。

3 实验结果与结果分析

3.1 PA1 实验结果

以下是 PA1 各指令的运行结果：

```
rd every instruction NEMU executes. This may lead to a large log file. If it
y, you can turn it off in include/common.h.
[src/monitor/monitor.c,28,welcome] Build time: 20:17:16, Nov 23 2021
Welcome to riscv32-NEMU!
For help, type "help"
(nemu) help
help - Display informations about all supported commands
c - Continue the execution of the program
q - Exit NEMU
si - Stop the program after stepping N steps.N defaults to 1. (si [N])
info - Print program status.SUBCMD: r(register)/w(watchpoint). (info SUBCMD)
p - Expression evaluation. (p EXPR)
x - Scan memory. (x N EXPR)
w - Set Watchpoint. (w EXPR)
d - Cancel Watchpoint. (d N)
(nemu)
```

图 3-1 help 指令

```
Welcome to riscv32-NEMU!
For help, type "help"
(nemu) si
80100000: b7 02 00 80          lui 0x80000,t0
(nemu) si asd
Incorrect format, please re-enter the instructions like "si [N]"
(nemu) si 8id1
Incorrect format, please re-enter the instructions like "si [N]"
(nemu) si 10
80100008: 03 a5 02 00          lw 0(t0),a0
8010000c: 6b 00 00 00          nemu trap
nemu: HIT GOOD TRAP at pc = 0x8010000c
[src/monitor/cpu-exec.c,29,monitor_statistic] total guest instructions = 4
(nemu)
```

图 3-2 si 指令

```
(nemu) p $a0 +9
9
(nemu) p 3*(10+$ra)
30
(nemu) p 2&&3 &&(10- 10)
0
(nemu) p 5 ||0
1
(nemu) p 28/0
ERROR: Divisor cannot be 0.
0
```

图 3-3 p 指令

```

e. If it is not necessary, you can turn it off in include/common.h.
[src/monitor/monitor.c,28,welcome] Build time: 22:27:15, Jan 13 2022
Welcome to riscv32-NEMU!
For help, type "help"
(nemu) x 6 3*(2+8)
0x8000001e: 0x00000000
0x80000022: 0x00000000
0x80000026: 0x00000000
0x8000002a: 0x00000000
0x8000002e: 0x00000000
0x80000032: 0x00000000
(nemu)

```

图 3-4 x 指令

```

(nemu) w 100+2+31
Watchpoint set succeed.
(nemu) info w

```

NO	EXPR	VALUE
2	100+2+31	133
1	4+55	59
0	2+4	6

```

(nemu) d 1
Delete watchpoint.
(nemu) info w

```

NO	EXPR	VALUE
2	100+2+31	133
0	2+4	6

```

(nemu) w 10+10
Watchpoint set succeed.
(nemu) info w

```

NO	EXPR	VALUE
3	10+10	20
2	100+2+31	133
0	2+4	6

```

(nemu) w 10+10+10
Watchpoint set succeed.
(nemu) info w

```

NO	EXPR	VALUE
1	10+10+10	30
3	10+10	20
2	100+2+31	133
0	2+4	6

图 3-5 监视点相关指令

3.2 PA2 实验结果

以下是 PA2 的实验运行结果：

```

[src/device/io/port-io.c,16,add_pio_map] Add port-io map 'keyboard' at [0x00000060, 0x00000063]
[src/device/io/mmio.c,14,add_mmio_map] Add mmio map 'keyboard' at [0xa1000060, 0xa1000063]
[src/monitor/monitor.c,20,welcome] Debug: ON
[src/monitor/monitor.c,21,welcome] If debug mode is on, A log file will be generated to record every instruction NEMU executes. This may lead to a large log file. If it is not necessary, you can turn it off in include/common.h.
[src/monitor/monitor.c,28,welcome] Build time: 15:55:56, Dec 23 2021
Welcome to riscv32-NEMU!
For help, type "help"
nemu: HIT GOOD TRAP at pc = 0x80100030

```

图 3-6 dummy 指令运行成功

```

hust@hust-desktop: ~/hust/ics2019/nemu
make: 对“app”无需做任何事。
NEMU compile OK
compiling testcases...
testcases compile OK
[ add-longlong] PASS!
[ add] PASS!
[ bit] PASS!
[ bubble-sort] PASS!
[ div] PASS!
[ dummy] PASS!
[ fact] PASS!
[ fib] PASS!
[ goldbach] PASS!
[ hello-str] PASS!
[ if-else] PASS!
[ leap-year] PASS!
[ load-store] PASS!
[ matrix-mul] PASS!
[ max] PASS!
[ min3] PASS!
[ mov-c] PASS!
[ movsx] PASS!
[ mul-longlong] PASS!
[ pascal] PASS!
[ prime] PASS!
[ quick-sort] PASS!
[ recursion] PASS!
[ select-sort] PASS!
[ shift] PASS!
[ shuixianhua] PASS!
[ string] PASS!
[ sub-longlong] PASS!
[ sum] PASS!
[ switch] PASS!
[ to-lower-case] PASS!
[ unalign] PASS!
[ wanshu] PASS!
hust@hust-desktop: ~/hust/ics2019/nemu$

```

图 3-7 cputest 运行成功

```

these qemu log files will help you a lot for debugging, but also significantly reduce the performance. If it is not necessary, you can turn it off in include/mon.h.
Connect to QEMU successfully
[src/monitor/monitor.c,20,welcome] Debug: ON
[src/monitor/monitor.c,21,welcome] If debug mode is on, A log file will be generated to record every instruction NEMU executes. This may lead to a large log file. If it is not necessary, you can turn it off in include/common.h.
[src/monitor/monitor.c,28,welcome] Build time: 16:21:57, Jan 7 2022
Welcome to riscv32-NEMU!
For help, type "help"
2000-0-0 2d:2d:2d GMT (1 second).
2000-0-0 2d:2d:2d GMT (2 seconds).
2000-0-0 2d:2d:2d GMT (3 seconds).
2000-0-0 2d:2d:2d GMT (4 seconds).
2000-0-0 2d:2d:2d GMT (5 seconds).
2[src/monitor/cpu-exec.c,29,monitor_statistic] total guest instructions = 516
qemu-system-riscv32: terminating on signal 15 from pid 14655 ()
hust@hust-desktop: ~/hust/ics2019/nexus-am/tests/amtest$

```

图 3-8 时钟实现

```

hust@hust-desktop: ~/hust/ics2019/nexus-am/tests/amtest
Get key: 45 D down
Get key: 45 D down
Get key: 45 D down
Get key: 45 D down
Get key: 45 D down
Get key: 45 D down
Get key: 45 D down
Get key: 45 D down
Get key: 45 D down
Get key: 45 D down
Get key: 45 D down
Get key: 45 D down
Get key: 45 D down
Get key: 45 D down
Get key: 45 D up
Get key: 70 SPACE down
Get key: 61 N down
Get key: 61 N up
Get key: 48 H down
Get key: 48 H up
Get key: 70 SPACE up
[src/monitor/cpu-exec.c,29,monitor_statistic] total guest instructions = 22215
qemu-system-riscv32: terminating on signal 15 from pid 14655 ()

```

图 3-9 键盘实现

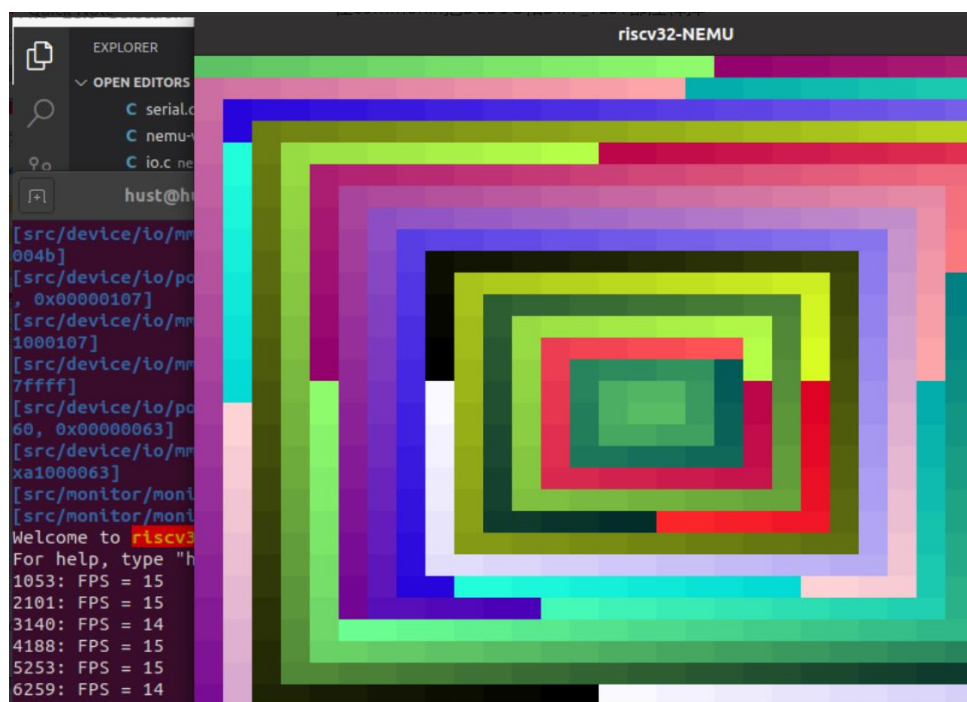


图 3-10 VGA 实现



图 3-11 打字游戏实现

下面是跑分程序，结果是 368 分。

```
hust@hust-desktop: ~/hust/ics2019/nexus-am/apps/microbench
[fib] Fibonacci number: * Passed.
    min time: 15647 ms [180]
[sieve] Eratosthenes sieve: * Passed.
    min time: 11204 ms [351]
[15pz] A* 15-puzzle search: * Passed.
    min time: 1931 ms [232]
[dinic] Dinic's maxflow algorithm: * Passed.
    min time: 2637 ms [412]
[lzip] Lzip compression: * Passed.
    min time: 2809 ms [270]
[ssort] Suffix sort: * Passed.
    min time: 857 ms [525]
[md5] MD5 digest: * Passed.
    min time: 9439 ms [182]
=====
MicroBench PASS          368 Marks
                        vs. 100000 Marks (i7-7700K @ 4.20GHz)
Total time: 59285 ms
nemu: HIT GOOD TRAP at pc = 0x801041e0

[src/monitor/cpu-exec.c,29,monitor_statistic] total guest instructions = 1864968
742
make[1]: 离开目录"/home/hust/hust/ics2019/nemu"
hust@hust-desktop:~/hust/ics2019/nexus-am/apps/microbench$
```

图 3-12 跑分结果

3.3 PA3 实现结果

以下是 PA3 的实验运行结果：

```
[src/monitor/monitor.c,28,welcome] Build time: 17:29:53, Jan 10 2022
Welcome to riscv32-NEMU!
For help, type "help"
[/home/hust/hust/ics2019/nanos-lite/src/main.c,14,main] 'Hello World!' from Nano
s-lite
[/home/hust/hust/ics2019/nanos-lite/src/main.c,15,main] Build time: 19:11:51, Ja
n 10 2022
[/home/hust/hust/ics2019/nanos-lite/src/ramdisk.c,28,init_ramdisk] ramdisk info:
start = , end = , size = -2146429635 bytes
[/home/hust/hust/ics2019/nanos-lite/src/device.c,35,init_device] Initializing de
vices...
[/home/hust/hust/ics2019/nanos-lite/src/irq.c,32,init_irq] Initializing interrup
t/exception handler...
[/home/hust/hust/ics2019/nanos-lite/src/proc.c,25,init_proc] Initializing proces
ses...
[/home/hust/hust/ics2019/nanos-lite/src/main.c,33,main] Finish initialization
Error event[5]: yield.
[/home/hust/hust/ics2019/nanos-lite/src/main.c,39,main] system panic: Should not
reach here
nemu: HIT BAD TRAP at pc = 0x80100628
```

图 3-13 触发 yield 异常触发 main 的 panic


```
hust@hust-desktop: ~/hust/ics2019/nanos-lite
[src/monitor/monitor.c,25,welcome] Debug: OFF
[src/monitor/monitor.c,28,welcome] Build time: 17:29:53, Jan 10 2022
Welcome to ftscv32-NEMU!
For help, type "help"
[/home/hust/hust/ics2019/nanos-lite/src/main.c,14,main] 'Hello World!' from Nano
s-lite
[/home/hust/hust/ics2019/nanos-lite/src/main.c,15,main] Build time: 17:18:03, Ja
n 11 2022
[/home/hust/hust/ics2019/nanos-lite/src/ramdisk.c,28,init_ramdisk] ramdisk info:
start = 801019C5, end = 8010868D, size = 27848 bytes
[/home/hust/hust/ics2019/nanos-lite/src/device.c,35,init_device] Initializing de
vices...
[/home/hust/hust/ics2019/nanos-lite/src/irq.c,34,init_irq] Initializing interrup
t/exception handler...
[/home/hust/hust/ics2019/nanos-lite/src/proc.c,27,init_proc] Initializing proces
ses...
[/home/hust/hust/ics2019/nanos-lite/src/loader.c,30,naive_uoload] Jump to entry =
830000C8
Error event[5]: yield.
nemu: HIT GOOD TRAP at pc = 0x80100828

[src/monitor/cpu-exec.c,29,monitor_statistic] total guest instructions = 477804
make[1]: 离开目录"/home/hust/hust/ics2019/nemu"
```

图 3-14 dummy 成功运行

```
Hello World from Navy-apps for the 614th time!
Hello World from Navy-apps for the 615th time!
Hello World from Navy-apps for the 616th time!
Hello World from Navy-apps for the 617th time!
Hello World from Navy-apps for the 618th time!
Hello World from Navy-apps for the 619th time!
Hello World from Navy-apps for the 620th time!
Hello World from Navy-apps for the 621th time!
Hello World from Navy-apps for the 622th time!
Hello World from Navy-apps for the 623th time!
Hello World from Navy-apps for the 624th time!
Hello World from Navy-apps for the 625th time!
Hello World from Navy-apps for the 626th time!
Hello World from Navy-apps for the 627th time!
[src/monitor/cpu-exec.c,29,monitor_statistic] total guest instructions = 33940
4
make[1]: 离开目录"/home/hust/hust/ics2019/nemu"
```

图 3-15 hello 成功运行

```
t/exception handler...
[/home/hust/hust/ics2019/nanos-lite/src/proc.c,27,init_proc] Initializing proces
ses...
[/home/hust/hust/ics2019/nanos-lite/src/loader.c,53,naive_uoload] Jump to entry =
830002F4
_end=0x8300a9f0
new_pb=0x8300abb0
new_pb=0x8300b000
PASS!!!
nemu: HIT GOOD TRAP at pc = 0x80100e24

[src/monitor/cpu-exec.c,29,monitor_statistic] total guest instructions = 1586710
make[1]: 离开目录"/home/hust/hust/ics2019/nemu"
```

图 3-16 text 成功运行

```
receive time event for the 126976th time: t 3787
receive time event for the 128000th time: t 3810
receive time event for the 129024th time: t 3836
receive event: ku N
receive event: kd F
receive time event for the 130048th time: t 3873
receive event: ku F
receive time event for the 131072th time: t 3899
receive event: kd N
receive time event for the 132096th time: t 3924
receive time event for the 133120th time: t 3951
receive event: ku N
receive time event for the 134144th time: t 3979
receive time event for the 135168th time: t 4002
receive time event for the 136192th time: t 4027
receive time event for the 137216th time: t 4052
[src/monitor/cpu-exec.c,29,monitor_statistic] total guest instructions = 1047902
18
make[1]: 离开目录“/home/hust/hust/ics2019/nemu”
```

图 3-17 events 成功运行

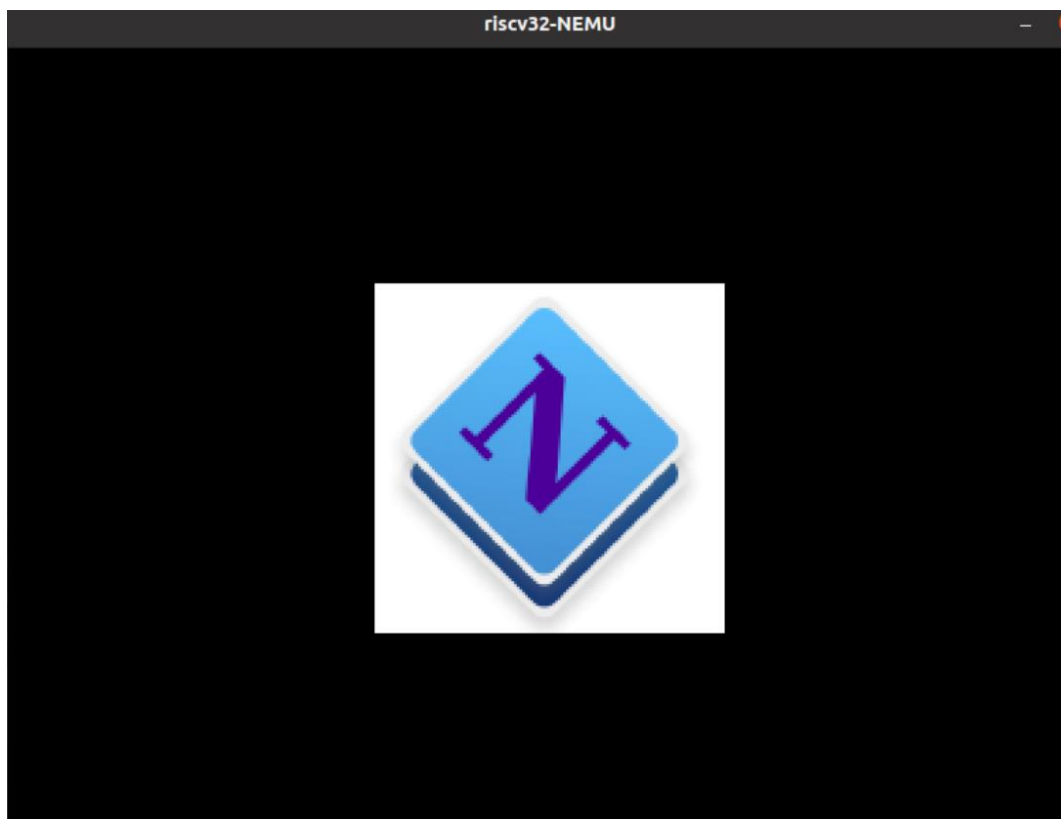


图 3-18 bmptest 成功运行

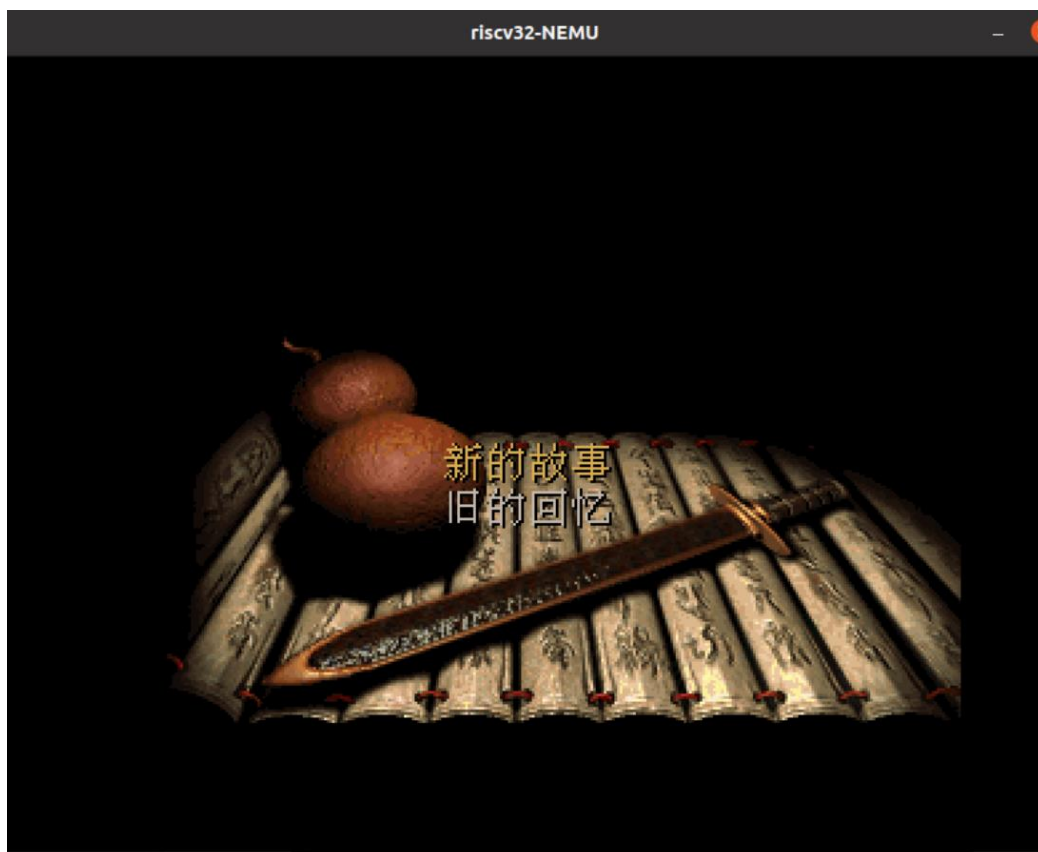


图 3-19 仙剑奇侠传成功运行



图 3-20 仙剑奇侠传游戏界面

4 实验回顾与归纳总结

4.1 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。

4.2 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

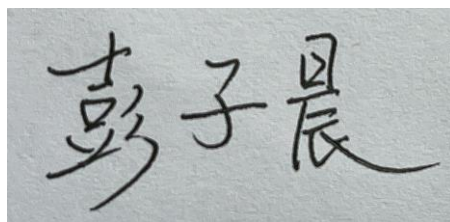
解释和任务需求,需要自己深入阅读和理解文档,通过 STFW 和 RTFC 去解决这个大工程。

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其原因,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:

A rectangular image showing a handwritten signature in black ink on a light gray background. The signature consists of three Chinese characters: '彭子晨' (Peng Zichen). The characters are written in a cursive, flowing style. The first character '彭' is on the left, followed by '子' in the middle, and '晨' on the right. The strokes are connected, with a long horizontal stroke extending from the bottom of the '晨' character.

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.