

华中科技大学

2022

系统能力培养 课程实验报告

题 目: riscv32 指令模拟器设计

专 业: 计算机科学与技术

班 级: CS1902 班

学 号: I201920024

姓 名: 木林

电 话: +85577449298

邮 件: 792967028@qq.com

完成日期: 2022-10-05



目 录

1	课程实验概述.....	1
1.1	课设目的	1
1.2	课程任务	1
1.3	实验环境	2
2	实验方案设计与结果分析	4
2.1	PA1 - 开天辟地的篇章: 最简单的计算机	4
2.1.1	PA1.1 实现单步执行, 打印寄存器状态, 扫描内存	4
2.1.2	PA1.2 实现算术表达式求值	5
2.1.3	PA1.3 实现所有要求, 提交完整的实验报告	7
2.2	PA2 - 简单复杂的机器: 冯诺依曼计算机系统	11
2.2.1	PA2.1 在 NEMU 中运行第一个 C 程序 dummy	11
2.2.2	PA2.2 实现更多的指令, 在 NEMU 中运行所有 cputest	12
2.2.3	PA2.3 运行打字小游戏, 提交完整的实验报告	14
2.3	PA3 - 穿越时空的旅程: 批处理系统	21
2.3.1	PA3.1 实现自陷操作_yield()及其过程	21
2.3.2	PA3.2 实现用户程序的加载和系统调用, 支撑 TRM 程序的运行 22	
2.3.3	PA3.3 运行仙剑奇侠传并展示批处理系统, 提交完整的实验报告 24	
3	总结与心得	32
4	电子签名	34

1 课程实验概述

1.1 课设目的

基于 riscv32 架构实现一个经过简化但是功能完备得模拟器 NEMU (NJU EMUlator)，最终在 NEMU 上运行游戏“仙剑奇侠传”，通过实验探索程序在计算机上运行的原理。

主要实验内容如下：

- (1) 图灵机与简易调试器；
- (2) 冯诺依曼计算机系统；
- (3) 批处理系统；
- (4) 分时多任务；
- (5) 程序性能优化；

通过实验：

- (1) 提升学生的计算机系统层面的认知与设计能力，能从计算机系统的高度考虑和解决问题；
- (2) 培养学生具有系统观的，能够进行软、硬件协同设计的思维认知；
- (3) 培养学生对系统有深刻的理解，能够站在系统的高度考虑和解决应用问题的。

1.2 课程任务

- (1) PA0 - 世界诞生的前夜: 开发环境配置
 - 安装合适的虚拟机；
 - git clone 框架代码并阅读框架代码；
- (2) PA1 - 开天辟地的篇章: 最简单的计算机
 - PA1.1: 实现单步执行，打印寄存器状态，扫描内存
 - PA1.2: 实现算术表达式求值
 - PA1.3: 实现所有要求，提交完整的实验报告
- (3) PA2 - 简单复杂的机器: 冯诺依曼计算机系统
 - PA2.1: 在 NEMU 中运行第一个 C 程序 dummy
 - PA2.2: 实现更多的指令，在 NEMU 中运行所有 cputest
 - PA2.3: 运行打字小游戏，提交完整的实验报告
- (4) PA3 - 穿越时空的旅程: 批处理系统
 - PA3.1: 实现自陷操作_yield() 及其过程
 - PA3.2: 实现用户程序的加载和系统调用，支撑 TRM 程序的运行
 - PA3.3: 运行仙剑奇侠传并展示批处理系统，提交完整的实验报告
- (5) PA4 - 虚实交错的魔法: 分时多任务
 - PA4.1: 实现基本的多道程序系统
 - PA4.2: 实现支持虚存管理的多道程序系统
 - PA4.3: 实现抢占式分时多任务系统，并提交完整的实验报告

1.3 实验环境

- (1) CPU 架构: x64
- (2) 平台: windows+ VirtualBox+Ubuntu64
- (3) 操作系统: gnu/linux
- (4) 编译器: gcc、riscv-none-embed-gcc
- (5) 编程语言: c 语言

```
gcc version 9.3.0 (Ubuntu 9.3.0-10ubuntu2)
```

2 验方案设计与结果分析

2.1 PA1 - 开天辟地的篇章: 最简单的计算机

PA1 的主要目的是创建一个监视器, 即一个基本的 gdb, 具有以下功能:

重新启动暂停的程序。info SUBCMD: info r 发布寄存器的状态; info w 打印监控点信息。p EXPR: 确定表达式EXPR的值。有关 EXPR 提供的操作的更多信息, 请参阅调试的表达式评估部分。x N EXPR: 确定表达式 EXPR 的值, 将结果作为起始内存位置, 并以十六进制形式打印连续的 N 4 个字节。d N: 从列表中删除序号为N的监控点

2.1.1 PA1.1 实现单步执行, 打印寄存器状态, 扫描内存

修改的文件

nemu/src/moniter/debug/ui.c

nemu/src/isa/riscv32/reg.c

实验过程

指定单步执行的函数在ui.c中, 使用atoi函数将参数args转换为int类型值n, 然后如果传入的args参数为整数或NULL, 则使用函数cpu_exec执行n步。

定义扫描内存的函数在ui.c中, static int cmd_x(char *args)传入两个参数, 一个是扫描内存的个数N, 由atoi函数解析, 另一个是初始地址 扫描到的内存, 由expr函数解析, 但是表达式expr函数在PA1.2部分设计, 所以expr总是返回0。调用vaddr_read宏循环读取起始地址后的N个4字节值。

结果

```
$0 = 0x00000000
ra = 0x00000000
sp = 0x00000000
gp = 0x00000000
tp = 0x00000000
t0 = 0x80000000
t1 = 0x00000000
t2 = 0x00000000
s0 = 0x00000000
s1 = 0x00000000
a0 = 0x00000000
a1 = 0x00000000
a2 = 0x00000000
a3 = 0x00000000
a4 = 0x00000000
a5 = 0x00000000
a6 = 0x00000000
a7 = 0x00000000
s2 = 0x00000000
s3 = 0x00000000
s4 = 0x00000000
s5 = 0x00000000
s6 = 0x00000000
s7 = 0x00000000
s8 = 0x00000000
s9 = 0x00000000
s10 = 0x00000000
s11 = 0x00000000
t3 = 0x00000000
t4 = 0x00000000
t5 = 0x00000000
t6 = 0x00000000
```

```
Welcome to riscv32-NEMU!  
For help, type "help"  
(nemu) si 2  
80100000: b7 02 00 80          lui  0x80000,t0  
80100004: 23 a0 02 00          sw   0(t0),$0
```

2.1.2 PA1.2 实现算术表达式求值

修改的文件

ics2019/nemu/src/monitor/debug/expr.c
nemu/src/moniter/debug/ui.c

实验过程

该规则规定必须将字符串解析为标记。Token结构中有一个整数成员类型记录了token类型，还有一个字符数组成员str记录了token ID。例如， $[0-9]^+$ 是令牌 TK INT 的正则表达式，而其余部分没有单独说明。

定义 `int operator priority(int op)` 函数返回运算符的优先级，优先级 17，其中 1 为最高优先级。

定义 `bool check parentheses(int p, int q)` 函数，该函数检测边缘括号匹配问题。

完全支持 TK DEREFERENCE 和 TK NEGATIVE 类型。

解析令牌，然后使用 `eval` 函数计算表达式的值。

结果

```
Welcome to riscv32-NEMU!  
For help, type "help"  
(nemu) x 4 $pc  
80100000  
0x80100000 : 0x800002b7  
0x80100004 : 0x0002a023  
0x80100008 : 0x0002a503  
0x8010000c : 0x0000006b
```

```
Welcome to riscv32-NEMU!  
For help, type "help"  
(nemu) p $pc+123  
decimal: -2146434949  
hex    : 0x8010007b  
(nemu) p $pc  
decimal: -2146435072  
hex    : 0x80100000  
(nemu) p -1+2  
decimal: 1  
hex    : 0x1  
(nemu) p (6-2)/2+(3+1)*5  
decimal: 22  
hex    : 0x16  
(nemu) p ((1-1)  
expression error
```

2.1.3 PA1.3 实现所有要求, 提交完整的实验报告

修改的文件:

```
nemu/src/moniter/debug/ui.c
nemu/include/monitor/watchpoint.h
nemu/src/monitor/debug/watchpoint.c
nemu/src/monitor/cpu-exec.c
```

实验过程

在 watchpoint.h 中, 将一个 char expr[32] 成员和一个 uint32_t 值成员添加到 WP 结构中, 这将用于存储表达式的名称及其值。

在 watchpoint.c 中, 定义函数 WP* free wp(int NO), 其中 NO 是观察点的序列号。NO 的范围是否为 0-31 且 head 不等于 NULL, 返回 NULL; 否则, 扫描头部链表, 查看节点序号是否为 NO; 如果是, 则将其从头链表中删除, 并将节点插入链表中的空闲; 否则, 返回 NULL。

2.2 PA2 - 简单复杂的机器: 冯诺依曼计算机系统

2.2.1 PA2.1 在 NEMU 中运行第一个 C 程序 dummy

修改的文件

```
nemu/src/isa/riscv32/exec/all-instr.h
nemu/src/isa/riscv32/exec/compute.c
nemu/src/isa/riscv32/exec/control.c
nemu/src/isa/riscv32/exec/exec.c
nemu/src/isa/riscv32/decode.c
nemu/src/isa/riscv32/include/isa/decode.h
```

实验过程

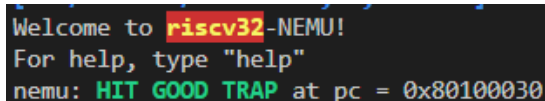
在 nexus-am/tests/cptest 目录下, 执行 make ARCH=riscv32-nemu ALL=dummy run 命令, 然后在 nexus-am/tests/cptest/build/ 下查找 dummy-riscv32-nemu.txt 文件 目录。当你反汇编代码时, 你会注意到有一个

不执行子指令。未实现的指令包括 auipc (U 型指令)、add (I 型指令)、jal (J 型指令)、jalr (即 ret, I 型指令); 要正确执行 dummy.c, 必须执行这些指令。

如下声明执行助手函数: 包含必须包含在 all-instr.h 文件中的执行助手函数。

在 exec.c 中的操作码表中填写您指定的指令的解码辅助函数、执行辅助函数和操作码。

结果



```
Welcome to riscv32-NEMU!
For help, type "help"
nemu: HIT GOOD TRAP at pc = 0x80100030
```

2.2.2 PA2.2 实现更多的指令, 在 NEMU 中运行所有 cptest

修改的文件


```
nemu/src/isa/riscv32/exec/all-instr.h
nemu/src/isa/riscv32/exec/compute.c
nemu/src/isa/riscv32/exec/control.c
nemu/src/isa/riscv32/exec/exec.c
nemu/src/isa/riscv32/decode.c
nemu/src/isa/riscv32/include/isa/decode.h
nexus-am/Makefile.check
nexus-am/libs/klib/src/stdio.c
nexus-am/libs/klib/src/string.c
nemu/include/common.h
nemu/src/isa/riscv32/diff-test.c
```

实验过程

在nexus-am/tests/cptest目录下执行make ARCH=riscv32-nemu ALL=dummy run命令，然后找到并检查nexus-am/tests/cptest/build/下的dummy-riscv32-nemu.txt文件 目录。当你反汇编代码时，你会看到有一个

不遵循子说明。未实现的指令有auipc（U型指令）、add（I型指令）、jal（J型指令）、jalr（即ret，I型指令）；要正确执行 dummy.c，必须执行这些指令。

声明执行助手函数：添加必须写在all-instr.h文件中的执行助手函数。

改进操作码表：将自己定义的解码辅助函数、执行辅助函数、指令操作码填入exec.c中的操作码表。

2.2.3 PA2.3 运行打字小游戏, 提交完整的实验报告

修改的文件

```
nemu/src/device/vga.c  
nemu/include/common.h  
nexus-am/am/src/nemu-common/nemu-input.c  
nexus-am/am/src/nemu-common/nemu-timer.c  
nexus-am/am/src/nemu-common/nemu-video.c
```

实验过程

本节的实验目的是完成串口、时钟、键盘和VGA输入输出设备的编程。要激活设备, 请在common.h中定义宏#define HAS_IOE。运行mainargs=h测试hello.c程序; 终端将显示Hello, AM World @ riscv32 10行。时钟设备程序: 在nemu-timer.c文件中定义静态变量start time来接收键调用nemu-input.c文件中的inl函数读取KBD_ADDR获取键盘活动信息, 将其返回值保存在kbd->keycode中, 保存kbd->的真实值 keycode & KEYDOWN_MASK in Enter kbd->keydown, 其中KEYDOWN_MASK = 0x8000, 也就是键盘掩码。

结果

```
Welcome to riscv32-NEMU!  
For help, type "help"  
Hello, AM World @ riscv32  
Hello, AM World @ riscv32  
Hello, AM World @ riscv32  
Hello, AM World @ riscv32  
Hello, AM World @ riscv32  
Hello, AM World @ riscv32  
Hello, AM World @ riscv32  
Hello, AM World @ riscv32  
Hello, AM World @ riscv32  
Hello, AM World @ riscv32  
nemu: HIT GOOD TRAP at pc = 0x80100e60
```



2.3 PA3 - 穿越时空的旅程: 批处理系统

PA3 的主要内容是实现系统调用和文件系统。

2.3.1 PA3.1 实现自陷操作_yield()及其过程

修改的文件

```
nemu/src/isa/riscv32/exec/all-instr.h
nemu/src/isa/riscv32/exec/system.c
nemu/src/isa/riscv32/include/isa/reg.h
nemu/src/isa/riscv32/exec/exec.c
nemu/src/isa/riscv32/intr.c
nexus-am/am/include/arch/riscv32-nemu.h
nexus-am/am/src/riscv32/nemu/cte.c
nanos-lite/include/common.h
nanos-lite/src/irq.c
```

实验过程

在 all-instr.h 中声明 make EHelper(system) 并在 common.h 中声明 #define HAS CTE 以创建主函数进行额外的设置工作。

查看nexus-am/am/src/\$ISA/nemu/trap.S的汇编指令, 将_Context结构中元素的位置更新为 uintptr t gpr[32], cause, status, epc。

添加静态 _Context * do event (_Event e, _Event e, _Event e, _Event e function _Context*c) 根据事件编号处理不同的事件, 将事件分为三类: _EVENT YIELD、_EVENT SYSCALL、_EVENT ERROR。要实现陷阱事件, 请处理 _EVENT YIELD 并使用 LOG 函数输出必要的信息。

2.3.2 PA3.2 实现用户程序的加载和系统调用, 支撑 TRM 程序的运行

修改的文件

```
nanos-lite/src/loader.c
nanos-lite/src/syscall.c
nanos-lite/src/irq.c
nanos-lite/src/proc.c
nexus-am/am/src/riscv32/nemu/cte.c
navy-apps/libs/libos/src/nanos.c
nexus-am/am/include/arch/riscv32-nemu.h
```

实验过程

在syscall.c文件中使用Do syscall(Context *c)来处理系统调用, 增强功能。首先, 您必须将 riscv32-nemu.h 文件包含在宏#define GPR2 gpr[10]、#define GPR3 gpr[11]、#define GPR4 gpr[12]、#define GPRx gpr[10], 返回do syscall函数, 使数组元素指向GPR 寄存器, a(1) = c->GPR2, a(2) = c->GPR3, a(3) = c->GPR4。

结果:

```
Hello World!  
Hello World from Navy-apps for the 2th time!  
Hello World from Navy-apps for the 3th time!  
Hello World from Navy-apps for the 4th time!  
Hello World from Navy-apps for the 5th time!  
Hello World from Navy-apps for the 6th time!  
Hello World from Navy-apps for the 7th time!  
Hello World from Navy-apps for the 8th time!  
Hello World from Navy-apps for the 9th time!  
Hello World from Navy-apps for the 10th time!  
Hello World from Navy-apps for the 11th time!  
Hello World from Navy-apps for the 12th time!  
Hello World from Navy-apps for the 13th time!  
Hello World from Navy-apps for the 14th time!  
Hello World from Navy-apps for the 15th time!
```

2.3.3 PA3.3 运行仙剑奇侠传并展示批处理系统, 提交完整的实验报告

修改的文件

```
nanos-lite/src/loader.c  
nanos-lite/src/syscall.c  
nanos-lite/src/proc.c  
nexus-am/am/src/riscv32/nemu/cte.c  
navy-apps/libs/libos/src/nanos.c  
nanos-lite/Makefile  
nanos-lite/src/fs.c  
nanos-lite/src/device.c  
navy-apps/libs/libos/src/nanos.c  
nexus-am/am/src/riscv32/nemu/cte.c  
nanos-lite/src/syscall.c
```

实验过程

update-ramdisk-single src/syscall.h 已更改为 update-ramdisk-fsimg src/syscall.h。
然后 make clean 删除之前运行的 ramdisk.img 并重新创建一个新的 ramdisk.img。
要在打开文件后记录读取和写入指针, 请将打开偏移字段添加到 fs.c 文件的 Finfo 结构中。

执行完 fs xxx 例程后, 编辑 naive_upload(NULL, "/bin/text"), 再次运行, 观察终端输出信息。
在 device.c 中, 实现 serial_write(const void *buf, size_t offset, size_t len) 函数并利用 _putc() 方法输出每个字符的 buf 数据字符

3 结与心得

PA 是一门令人愉快和困难的课程。它包含大量的信息和知识点，可以拓宽学生的视野，增强学生的系统开发能力，帮助学生有条不紊地应对挑战。我觉得自己很幸运选择了这门课程作为系统综合能力培养课程的项目。通过这个项目，我有以下经验和知识。PA1 的主要目标是提供一个基本的调试器，它是 NEMU 中基础设施的关键部分。实现一个基本的调试器可以提高调试效率，同时也让你熟悉框架代码，为未来的困难铺平道路。

另一个需要注意的模块是观察点函数。要实现观察点功能，必须熟悉链表结构。因为观察点在数组中，所以必须考虑数组空间边距。PA2 的主要目的是实现 riscv 指令和 klib 库函数。整个 PA2 的开发过程都离不开指令的设计，所以在开始实验之前一定要对指令的结构有深入的了解，所以在开始实验之前，建议先阅读相关书籍，然后对以下 nemu 目录中的框架，一般来说，PA2 是一个更具挑战性的部分，需要花费更多时间阅读相关文献和检查框架代码。

。

