

华中科技大学

2023

系统能力培养 课程实验报告

题 目:	指令模拟器
专 业:	计算机科学与技术
班 级:	CS2010 班
学 号:	U202015604
姓 名:	刘功华
电 话:	15116898515
邮 件:	rollroll520@qq.com
完成日期:	2024-01-17



目 录

1	课程实验概述	1
1.1	课设目的	1
1.2	实验内容	1
1.3	实验平台与工具	2
2	实验方案设计	3
2.1	PA1 最简单的计算机	3
2.2	PA2 冯诺依曼计算机系统	8
2.3	PA3 批处理系统	14
3	实验总结	20
	参考文献	21

1 课程实验概述

1.1 课设目的

在代码框架中实现一个简化的 riscv32 模拟器 (NJU EMUlator)。

- 1) 可解释执行 riscv32 执行代码
- 2) 支持输入输出设备
- 3) 支持异常流处理
- 4) 支持精简操作系统---支持文件系统
- 5) 支持虚存管理
- 6) 支持进程分时调度

最终在模拟器上运行“仙剑奇侠传”,让学生探究“程序在计算机上运行”的机理,掌握计算机软硬协同的机制,进一步加深对计算机分层系统栈的理解,梳理大学 3 年所学的全部理论知识,提升学生计算机系统能力。

1.2 实验内容

- 1) 世界诞生前夜---开发环境:

安装虚拟机或者 docker, 熟悉相关工具和平台, 安装 sourceinsight 阅读代码框架。

- 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

- 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

- 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

5) 虚实交错的魔法: 分时多任务

PA4.1: 实现分页机制

PA4.2: 实现进程上下文切换

PA4.3: 时钟中断驱动的上下文切换

1.3 实验平台与工具

CPU 架构: x64

操作系统: GNU/Linux

编译器: GCC

编程语言: C 语言

其他工具: GDB, vim, Git, Vscode

课程组提供虚拟机镜像

2 实验方案设计

2.1 PA1 最简单的计算机

2.1.1 PA1-1 简单调试器

该阶段我们需要在 `monitor` 中实现一个简易调试器，需要补充的功能有：单步执行、打印程序状态、表达式求值、扫描内存、设置监视点、删除监视点，其中单步执行、打印程序状态（寄存器状态）、扫描内存存在该阶段完成，其余功能在后两个阶段中实现。

单步执行：如图 2.1.1 所示，在 `monitor/debug/ui.c` 中定义单步执行 `cmd_si_n` 函数，为了实现单步执行 `n` 条指令的功能，需要参数 `n` 来决定循环执行次数。将参数 `args` 通过 `atoi` 函数转化为整数从而获取到参数 `n`。然后直接调用 `void cpu_exec(uint64_t n)` 这个函数即可。

```
//实现单步执行si[n]
static int cmd_si_n(char *args){
    int n=1;//缺省为1
    if(args!=NULL){
        n=atoi(args);//将字符串参数转化为整数
    }
    cpu_exec(n);//循环执行n次
    return 0;
}
```

图 2.1.1 单步执行

打印寄存器状态：如图 2.1.2 与图 2.1.3 所示，在 `reg.c` 中实现 `isa_reg_display` 函数，实现循环打印寄存器值的功能。再在 `ui.c` 中定义 `cmd_info` 函数调用 `isa_reg_display` 函数即可。`cmd_info` 函数打印的程序状态包括寄存器状态与监视点信息，监视点信息在完成监视点功能时完善。`cmd_info` 函数需要根据参数为 `r` 或 `w` 来决定打印寄存器状态或监视点信息，这里通过 `switch` 来实现。

```
const char *regsl[] = {
    "$0", "ra", "sp", "gp", "tp", "t0", "t1", "t2",
    "s0", "s1", "a0", "a1", "a2", "a3", "a4", "a5",
    "a6", "a7", "s2", "s3", "s4", "s5", "s6", "s7",
    "s8", "s9", "s10", "s11", "t3", "t4", "t5", "t6"};

void isa_reg_display()
{
    for (int i = 0; i < 32; i++)
    {
        printf("%s = 0x%08x\n", reg_name(i, 3), reg_l(i));
    }
}
```

图 2.1.2 `isa_reg_display` 函数

```

//打印程序状态info SUBCMD
static int cmd_info(char *args){
    if(args==NULL) return 0;
    switch (args[0])
    {
        case 'r':|
            isa_reg_display();//打印寄存器状态
            break;
        case 'w':
            print_wp();//打印监视点信息
            break;
        default:
            break;
    }
    return 0;
}

```

图 2.1.2 打印寄存器状态 cmd_info 函数

扫描内存：在 ui.c 中定义 cmd_x 函数，此时尚未实现表达式求值功能，因此直接对输入参数进行解析读出待扫描内存的起始地址，并且通过循环调用 vaddr_read 函数将内存中的连续 n 个 4 字节的值打印出来。

2.1.2 PA1-2 表达式求值

该阶段主要在 expr.c 中完成。在词法分析部分，我们需要为表达式中的各种 token 类型添加规则以及对应的正则表达式中。在成功识别出 token 后，将 token 的信息依次记录到 tokens 数组中。

然后实现括号匹配函数 check_parentheses，以及运算符优先级函数 op_priority。这些函数根据 C 语言中运算符的优先级来实现，以确保表达式求值过程中的正确性。随后实现递归求值函数 eval。eval 函数会对传入的参数 p 和 q 进行处理。如果 p 大于 q，则函数直接返回 0，并将 success 标识符设为 false。如果 p 等于 q，则函数会判断当前 token 的类型是否为整数、十六进制整数或寄存器。如果是，则返回相应的值；如果不是，则将 success 设为 false，并返回 0。接下来，函数会进行括号匹配检查。如果成功匹配，则返回 eval(p+1,q-1) 的结果。否则，函数将进入寻找主操作符的行为。如果能找到主操作符，则根据主操作符返回相应的值。如果找不到，则将 success 设为 false，并返回 0。

最后，如图 2.1.4 与图 2.1.5 所示，在 expr 函数中调用 eval 函数和 make_token 函数，并在 ui.c 中定义 cmd_p 函数，在函数中调用来对参数进行表达式求值，即可完成调试器的表达式求值功能。

```

//表达式求值p EXPR
static int cmd_p(char *args){
    bool success=true;
    uint32_t result=expr(args,&success);
    if(!success){//表达式求值错误
        printf("Expression evaluation error!\n");
        return -1;
    }
    else
        printf("%d\n",result);//打印求值结果
    return 0;
}

```

图 2.1.4 表达式求值 cmd_p 函数

```

uint32_t expr(char *e, bool *success) {
    if (!make_token(e)) {
        *success = false;
        return 0;
    }

    /* Codes to evaluate the expression are added. */

    for(int i=0;i<nr_token;i++){
        if(tokens[i].type=='*&(i==0||tokens[i-1].type=='+'||tokens[i-1].type==TK_DEREFERENCE;
        tokens[i].type=TK_DEREFERENCE;
    }
    }
    for(int i=0;i<nr_token;i++){
        if(tokens[i].type=='- '&(i==0||tokens[i-1].type=='+'||tokens[i-1].type==TK_NEGATIVE;
        tokens[i].type=TK_NEGATIVE;
    }
    }
    return eval(0,nr_token-1,success);
}

```

图 2.1.5 expr 函数

2.1.3 PA1-3 监视点与断点

该阶段需要实现监视点功能并实现监视点相关的调试器功能。框架代码在 watchpoint.h 已经定义了 watchpoint 的结构,使用链表将监视点的信息组织起来。但考虑到需要追踪表达式的值,我们应分配一个字符串字段来保存表达式文本,并且为了追踪变化,应该设置一个变量用于保存前一次的值。因此我们根据以上需求完善 watchpoint 的结构,如图 2.1.6 所示。

```

typedef struct watchpoint {
    int NO;
    struct watchpoint *next;

    /* TODO: Add more members if necessary */

    char expr[32]; //记录表达式
    int value; //记录表达式的值
} WP;

#endif

```

图 2.1.6 watchpoint 结构

接下来,在 watchpoint.c 文件中,我们需要实现 new_wp、free_wp 和 print_wp 三个函数。new_wp 函数负责创建新的监控点,它将 free_链表中的节点移动到 head 链表中; free_wp 函数则负责删除监控点,通过监视点的编号在 head 链表中查找并移除相应的节点,然后将其加入到 free_链表中。print_wp 函数用于展示监控点信息。监视点功能主要通过链表操作实现。

在 ui.c 中,我们要利用这些新函数来实现 cmd_x 和 cmd_d 命令,并对 cmd_info 函数进行完善。此外,在 cpu.exec 中,我们需要实现一种逻辑,当监控点的状态变更时,能够使程序暂停执行。程序在每执行一步指令后,会对所有的监控点重新计算表达式的值,并与之前存储的值进行对比,如果发现不一致,就将 NEMU 的状态设置为 NEMU_STOP。

2.1.4 PA1 测试结果

```

7 [src/monitor/monitor.c,36,load_img] No image is given. Use the default build-in
8 image.
9 [src/memory/memory.c,16,register_pmem] Add 'pmem' at [0x80000000, 0x87ffffff]
10 [src/monitor/monitor.c,20,welcome] Debug: ON
11 [src/monitor/monitor.c,21,welcome] If debug mode is on, A log file will be gener
12 ated to record every instruction NEMU executes. This may lead to a large log fil
13 e. If it is not necessary, you can turn it off in include/common.h.
14 [src/monitor/monitor.c,28,welcome] Build time: 21:26:10, Jan 15 2024
15 Welcome to riscv32-NEMU!
16 For help, type "help"
17 (nemu) help
18 help - Display informations about all supported commands
19 c - Continue the execution of the program
20 q - Exit NEMU
21 si - Single Step Execute
22 info - Print details of register || watchpoint
23 p - Expression Evaluation
24 x - Scan memory
25 w - Set a New Watchpoint
26 d - Delete Watchpoint
27 (nemu)

```

图 2.1.7 pa1 简易调试器


```

(nemu) p 1+2
3
(nemu) p 1*2
2
(nemu) p 1+2*4
9
(nemu) p 3/0
Expression evaluation error!

```

图 2.1.8 pa1 表达式求值

```

hust@hust-desktop: ~/workPlace/ics2019/nemu
elp - Display informations about all supported commands
- Continue the execution of the program
- Exit NEMU
i - Single Step Execute
nfo - Print details of register || watchpoint
- Expression Evaluation
- Scan memory
- Set a New Watchpoint
- Delete Watchpoint
nemu) w $t1
ucceed to set Watchpoint!
nemu) info w
0      EXPR      VALUE
      $t1      0
nemu) w $t0
ucceed to set Watchpoint!
nemu) w $t2
ucceed to set Watchpoint!
nemu) info w
0      EXPR      VALUE
      $t2      0
      $t0      0
      $t1      0
nemu)

```

图 2.1.9 pa1 单步执行、打印监视点信息

2.1.5 PA1 必答题

1) 我选择的 ISA 是:

答: riscv-32

2) 我们通过一些简单的计算来体会简易调试器的作用. 首先作以下假设:

- 假设你需要编译500次NEMU才能完成PA.
- 假设这500次编译当中, 有90%的次数是用于调试.
- 假设你没有实现简易调试器, 只能通过GDB对运行在NEMU上的客户程序进行调试. 在每一次调试中, 由于GDB不能直接观测客户程序, 你需要花费30秒的时间来从GDB中获取并分析一个信息.
- 假设你需要获取并分析20个信息才能排除一个bug.

那么这个学期下来, 你将会在调试上花费多少时间?

由于简易调试器可以直接观测客户程序, 假设通过简易调试器只需要花费10秒的时间从中获取并分析相同的信息. 那么这个学期下来, 简易调试器可以帮助你节省多少调试的时间?

图 2.1.9 pa1 题 2 图

答：调试上花费的时间是 $500 \times 90\% \times 30 \times 20 = 270000s = 4500min$

节约的时间为 $500 \times 90\% \times 20 \times 20 = 180000s = 3000min$

3) riscv32 有哪几种指令格式?

答：有 R、I、S、B、U、J 6 种指令格式

4) LUI 指令的行为是什么?

答：将 20 位常量加载到寄存器的高 20 位

5) mstatus 寄存器的结构是怎么样的?

答：如图 2.1.11 所示：

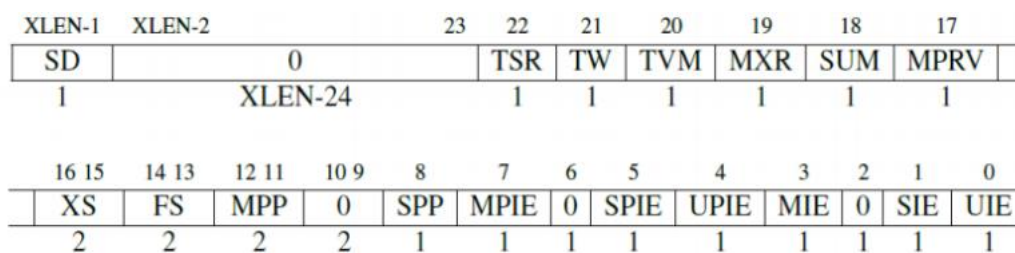


图 2.1.11 mstatus 寄存器结构

2.2 PA2 冯诺依曼计算机系统

2.2.1 PA2-1 运行第一个 C 程序

为了运行程序，我们需要经过取指-译码-执行-更新 PC 四个阶段。取指令通过 `instr_fetch` 进行一次内存访问完成。译码与执行都需要相应的辅助函数，需要在该阶段实现。

首先，通过运行 `make` 命令来执行 `dummy` 程序，显示程序终止运行和错误信息。紧接着，在 `build` 文件夹中会产生相应的反汇编文件。通过分析这些文件，如图 2.2.1 所示，可以识别出程序中尚未实现的部分，然后在相关手册中查询这些未实现指令的具体含义。此外，还需要识别出所有的伪指令。通过这些步骤，我们可以确定 `dummy` 程序需要实现的新指令，包括 `auipc`、`addi`、`jal` 和 `jalr`。

```

1
2 /home/hust/workPlace/ics2019/nexus-an/tests/cputest/build/dummy-riscv32-nemu.elf:
elf32-littleriscv
3
4
5 Disassembly of section .text:
6
7 80100000 <_start>:
8 80100000: 00000413          li      s0,0
9 80100004: 00009117          auipc   sp,0x9
10 80100008: ffc10113          addi    sp,sp,-4 # 80109000 <_end>
11 8010000c: 00c000ef          jal     ra,80100018 <_trm_init>
12
13 Disassembly of section .text.startup:
14
15 80100010 <main>:
16 80100010: 00000513          li      a0,0
17 80100014: 00008067          ret
18
19 Disassembly of section .text._trm_init:
20
21 80100018 <_trm_init>:
22 80100018: 80000537          lui     a0,0x80000
23 8010001c: ff010113          addi    sp,sp,-16
24 80100020: 00050513          mv      a0,a0
25 80100024: 00112623          sw      ra,12(sp)
26 80100028: fe9ff0ef          jal     ra,80100010 <_etext>
27 8010002c: 00050513          mv      a0,a0
28 80100030: 0000006b          0x6b
29 80100034: 0000006f          j       80100034 <_trm_init+0x1c>

```

图 2.2.1 dummy 反汇编代码

接下来, 在 `all-instr.h` 文件中定义这些新指令的执行函数。然后在 `exec.c` 文件中的 `opcode_table` 数据结构中添加这些新指令。此外, 在 `decode.c` 文件中实现相应的指令译码辅助函数。在 `rtl.h` 文件中, 对 `rtl` 指令进行必要的修改和完善, 以确保其正确性。

最后, 在 `compute.c`、`control.c` 等相关文件中, 使用这些 `rtl` 指令来实现正确的执行辅助函数即可。

2.2.2 PA2-2 丰富指令集, 测试所有程序

该阶段需要我们实现更多的指令, 指令繁多, 更易犯错。我们需要先通过反汇编代码分析需要实现的指令, 再在 `control.c` 中编写相应的执行辅助函数, 计算指令在 `compute.c` 中实现。之后, 查阅手册以根据指令的行为编写相应的执行辅助函数。如果函数尚未声明, 则需要在 `decode.h` 中声明译码辅助函数, 并在 `all-instr.h` 中声明执行辅助函数, 如图 2.2.2 所示。

```

#include "cpu/exec.h"

make_EHelper(inv);
make_EHelper(nemu_trap);
make_EHelper(ld);
make_EHelper(st);
make_EHelper(lh);
make_EHelper(lb);
make_EHelper(lui);
make_EHelper(auipc);
make_EHelper(jal);
make_EHelper(jalr);
make_EHelper(branch);
make_EHelper(imm);
make_EHelper(reg);
make_EHelper(sys);

```

图 2.2.2 `all-instr.h` 中定义的执行辅助函数

接下来，在 `exec.c` 的 `opcode_table` 中填写正确的译码辅助函数，执行辅助函数以及操作数宽度。完成这些步骤后，运行 `make` 命令以执行测试文件。若出现错误，则检查相关的译码和执行函数。如果测试通过，继续编写下一个指令，重复此过程直至一键回归测试通过。

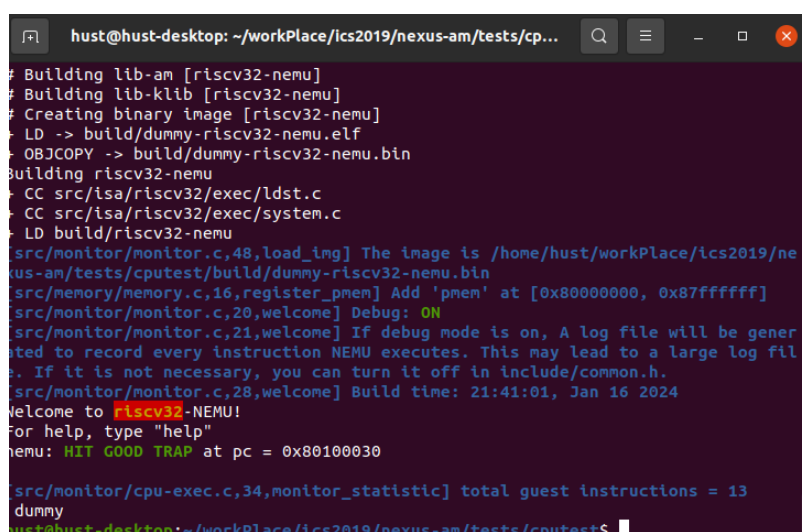
除补充指令外，我们还需要实现 `string` 和 `hello-str` 中使用的 `sprintf`、`strcmp`、`strcat`、`strcpy` 和 `memset`。在 `stdio.c` 以及 `string.c` 中补充即可。

2.2.3 PA2-3 实现 I/O 指令，测试打字游戏

该阶段开始接触设备 `device` 部分。主要完成串口（`trm.c` 中）、时钟（`nemu-timer.c` 中）、键盘（`nemu-input.c` 中）、VGA（`nemu-video.c` 中）四个输入输出设备程序的编写。

访问 `RTC_ADDR` 地址获取启动时间、访问 `RTC_ADDR` 地址来获取当前时间，将 `uptime->hi` 设置为 0，并将 `uptime->lo` 设置为当前时间与启动时间之间的差值，从而实现时钟功能。通过 `KBD_ADDR` 地址来获取键盘按键信息，并将其存储在 `kbd->keycode` 变量中，将按键信息与 `KEYDOWN_MASK` 进行按位与操作，1 表示按键被按下；0 表示按键被释放，从而实现键盘功能。将 `pixels` 数组中的像素信息写入到 VGA 对应的地址空间中从而实现 VGA 相关功能。

2.2.4 PA2 测试结果



```
hust@hust-desktop: ~/workPlace/ics2019/nexus-am/tests/cp...
# Building lib-am [riscv32-nemu]
# Building lib-klib [riscv32-nemu]
# Creating binary image [riscv32-nemu]
LD -> build/dummy-riscv32-nemu.elf
OBJCOPY -> build/dummy-riscv32-nemu.bin
Building riscv32-nemu
CC src/isa/riscv32/exec/ldst.c
CC src/isa/riscv32/exec/system.c
LD build/riscv32-nemu
[src/monitor/monitor.c,48,load_img] The image is /home/hust/workPlace/ics2019/ne
xus-am/tests/cputest/build/dummy-riscv32-nemu.bin
[src/memory/memory.c,16,register_pmem] Add 'pmem' at [0x80000000, 0x87ffffff]
[src/monitor/monitor.c,20,welcome] Debug: ON
[src/monitor/monitor.c,21,welcome] If debug mode is on, A log file will be gener
ated to record every instruction NEMU executes. This may lead to a large log fil
e. If it is not necessary, you can turn it off in include/common.h.
[src/monitor/monitor.c,28,welcome] Build time: 21:41:01, Jan 16 2024
Welcome to riscv32-NEMU!
For help, type "help"
nemu: HIT GOOD TRAP at pc = 0x80100030

[src/monitor/cpu-exec.c,34,monitor_statistic] total guest instructions = 13
dummy
hust@hust-desktop: ~/workPlace/ics2019/nexus-am/tests/cputestS
```

图 2.2.3 pa2-1dummy 通过

```
hust@hust-desktop: ~/workPlace/ics2019/nemu
make: Nothing to be done for 'app'.
EMU 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:~/workPlace/ics2019/nemu$
```

图 2.2.4 pa2-2 一键回归测试通过

```
hust@hust-desktop: ~/workPlace/ics2019/nexus-am/tests/a...
src/device/io/port-io.c,16,add_pio_map] Add port-io map 'screen' at [0x00000100, 0x00000107]
src/device/io/mmio.c,14,add_mmio_map] Add mmio map 'screen' at [0xa1000100, 0xa1000107]
src/device/io/mmio.c,14,add_mmio_map] Add mmio map 'vmem' at [0xa0000000, 0xa0000000]
src/device/io/port-io.c,16,add_pio_map] Add port-io map 'keyboard' at [0x00000000, 0x00000063]
src/device/io/mmio.c,14,add_mmio_map] Add mmio map 'keyboard' at [0xa1000060, 0xa1000063]
src/monitor/monitor.c,25,welcome] Debug: OFF
src/monitor/monitor.c,28,welcome] Build time: 03:04:16, Jan 17 2024
welcome to riscv32-NEMU!
for help, type "help"
try to press any key...
get key: 48 H down
get key: 48 H up
get key: 35 U down
get key: 35 U up
get key: 44 S down
get key: 44 S up
get key: 33 T down
get key: 33 T up
```

图 2.2.5 pa2-3 read_key_test 通过

```
hust@hust-desktop: ~/workPlace/ics2019/nexus-am/tests/a...
src/device/io/mmio.c,14,add_mmio_map] Add mmio map 'serial' at [0xa10003f8, 0xa10003f8]
src/device/io/port-io.c,16,add_pio_map] Add port-io map 'rtc' at [0x00000048, 0x0000004b]
src/device/io/mmio.c,14,add_mmio_map] Add mmio map 'rtc' at [0xa1000048, 0xa100004b]
src/device/io/port-io.c,16,add_pio_map] Add port-io map 'screen' at [0x00000100, 0x00000107]
src/device/io/mmio.c,14,add_mmio_map] Add mmio map 'screen' at [0xa1000100, 0xa1000107]
src/device/io/mmio.c,14,add_mmio_map] Add mmio map 'vmem' at [0xa0000000, 0xa0000000]
src/device/io/port-io.c,16,add_pio_map] Add port-io map 'keyboard' at [0x00000000, 0x00000063]
src/device/io/mmio.c,14,add_mmio_map] Add mmio map 'keyboard' at [0xa1000060, 0xa1000063]
src/monitor/monitor.c,25,welcome] Debug: OFF
src/monitor/monitor.c,28,welcome] Build time: 03:04:16, Jan 17 2024
welcome to riscv32-NEMU!
for help, type "help"
000-0-0 2d:2d:2d GMT (1 second).
000-0-0 2d:2d:2d GMT (2 seconds).
000-0-0 2d:2d:2d GMT (3 seconds).
```

图 2.2.6 pa2-3rtc_test 通过

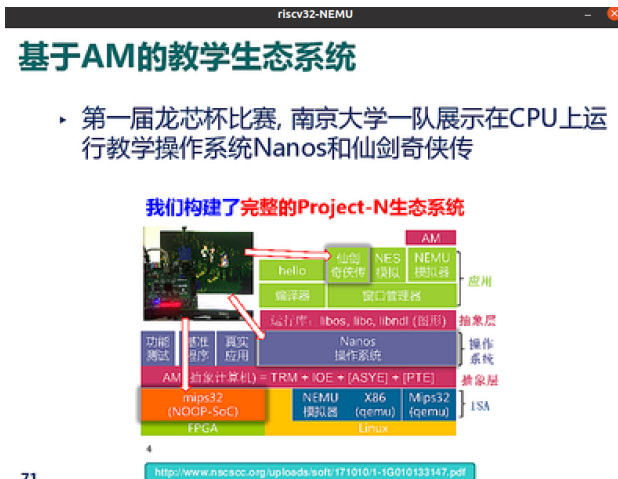


图 2.2.7 pa2-3 slider 通过



图 2.2.8 pa2-3 text 打字游戏通过

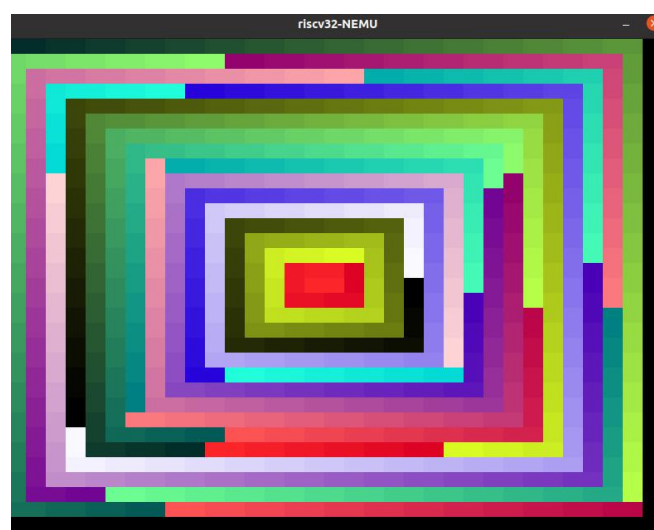


图 2.2.9 pa2-3 display 通过

2.2.5 PA2 必答题

1) 请整理一条指令在 NEMU 中的执行过程:

答: 首先, 通过调用 `instr_fetch` 函数并传递 PC (程序计数器) 的值来获取指令。从获取到的指令中提取操作码 (opcode), 然后在 `opcode_table` 中查找该操作码, 以确定相应的译码辅助函数和执行辅助函数。接下来, 使用译码辅助函数对指令进行译码, 并将译码后的信息存储在 `decinfo` 结构中。然后, 调用执行辅助函数来执行指令, 该函数会使用 RTL 指令对译码信息进行必要的操作, 如计算、读取和存储等。最后, 调用 `update_pc` 函数来更新 PC 的值, 以准备执行下一条指令。

2) 在 `nemu/include/common.h` 中添加一行 `volatile static int dummy`; 然后重新编译 NEMU. 请问重新编译后的 NEMU 含有多少个 `dummy` 变量的实体? 你是如何得到这个结果的?

答: 81。

3) 添加上题中的代码后, 再在 `nemu/include/debug.h` 中添加一行 `volatile static int dummy`; 然后重新编译 NEMU. 请问此时的 NEMU 含有多少个 `dummy` 变量的实体? 与上题中 `dummy` 变量实体数目进行比较, 并解释本题的结果.

答: 82。

4) 修改添加的代码, 为两处 `dummy` 变量进行初始化: `volatile static int dummy = 0`; 然后重新编译 NEMU. 你发现了什么问题? 为什么之前没有出现过这样的问题?

答: 初始化后, 会产生两个强符号, 导致错误。

5) 描述你在 `nemu/` 目录下敲入 `make` 后, `make` 程序如何组织 `.c` 和 `.h` 文件, 最终生成可执行文件 `nemu/build/$ISA-nemu`.

答: 输入 `make` 命令时, 它会将 `makefile` 中列出的第一个目标文件视为最终要构建的目标。如果这个目标文件不存在, 或者它的依赖文件 (通常是 `.o` 文件) 的修改时间比目标文件新, `make` 将会触发重新编译。如果依赖的 `.o` 文件也不存在, `make` 会按照 `makefile` 中定义的规则来创建这些文件, 然后再使用它们来构建上一层的目标文件。

2.3 PA3 批处理系统

2.3.1 PA3-1 实现系统调用

该阶段需要实现 `_yield` 自陷操作及其过程。为了执行自陷操作，必须首先实现支持自陷的指令集。所需实现的指令包括 `'csrrs'`、`'csrrw'`、`'ecall'` 和 `'sret'`。通过 `'ecall'` 指令触发自陷操作，如图 2.3.1 所示，`'csrrs'` 和 `'csrrw'` 指令用于修改控制状态寄存器，而 `'sret'` 指令则在自陷处理后负责恢复执行。

```
11  make_EHelper(sys){
12      switch(decinfo.isa.instr.funct3){
13  >      case 0:{ //ecall&&sret ...
24  >      case 1:{ //csrrw ...
50  >      case 2:{ //csrrs ...
76      }
77  }
```

图 2.3.1 实现自陷相关指令

接下来，在 `'intr.c'` 文件中实现 `'raise_intr'` 函数，该函数负责模拟自陷过程的触发：将当前的 PC 值存储到 `'sepc'` 寄存器中，在 `'scause'` 寄存器中记录异常码，从 `'stvec'` 寄存器获取异常处理程序的入口地址，并跳转到该地址执行。在自陷操作被触发后，需要保存当前的执行上下文。根据 `'trap.S'` 汇编文件中栈操作的顺序，重新构建 `'_Context'` 结构体的成员如图 2.3.2 所示。随后，为了正确地分发事件，需要在 `'__am_irq_handle'` 函数中通过异常码识别出自陷异常，并在 `'do_event'` 函数中识别出特定的自陷事件 `'_EVENT_YIELD'`。最后，使用 `'sret'` 指令返回到正常的执行流程并恢复之前的上下文。

```
1  #ifndef __ARCH_H__
2  #define __ARCH_H__
3
4  struct Context {
5      uintptr_t gpr[32], cause, status, epc;
6      struct _AddressSpace *as;
7  };
8
9  #define GPR1 gpr[17]
10 #define GPR2 gpr[10]
11 #define GPR3 gpr[11]
12 #define GPR4 gpr[12]
13 #define GPRx gpr[10]
14
15 #endif
16
```

图 2.3.2 `_Context` 结构体

2.3.2 PA3-2 实现文件系统

该阶段需要实现用户程序的加载和系统调用，以支撑 TRM 程序的运行。加载用户程序是由 loader 模块负责的，文件系统需要在 pa3-3 中实现，此时的内存中只有一个文件，因此只需直接通过 `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。先在 `do_event` 函数中添加对 `do_syscall` 的调用，再根据 `nanos.c` 中的 `ARGS_ARRAY` 在 `riscv32-nemu` 中正确实现通用寄存器（GPR）的宏定义。随后，实现 `SYS_yield`、`SYS_read`、`SYS_write` 和 `SYS_brk` 等系统调用。最后，完成 `_sbrk` 函数，实现堆区管理即可。

2.3.3 PA3-3 运行仙剑奇侠传

该阶段我们需要实现文件系统。首先完成文件系统相关的函数及其系统调用。程序的数量增加之后，我们就要知道哪个程序在 `ramdisk` 的什么位置因此需要将 loader 函数中的 `ramdisk_read` 替换掉，并修改其逻辑，以便能够通过文件名在加载器中指定要加载的程序。

为了实现虚拟文件系统（VFS），需要将输入/输出设备（IOE）抽象为文件。首先，在 VFS 中增加对多种特殊文件的支持。然后实现 `serial_write` 函数，以完成串口的写入操作。接着实现 `events_read` 函数，以支持读取操作。最后，需要实现 `init_fs`、`fb_write`、`fbsync_write`、`init_device` 和 `dispinfo_read` 等函数，以实现支持对 VGA 设备的支持。

```
nanos-lite > src > C fs.c > File_table
32     return 0;
33 }
34
35 /* This is the information about all files in disk. */
36 static Finfo file_table[] __attribute__((used)) = {
37     {"stdin", 0, 0, invalid_read, invalid_write},
38     {"stdout", 0, 0, invalid_read, serial_write},
39     {"stderr", 0, 0, invalid_read, serial_write},
40     {"/dev/fb", 0, 0, 0, invalid_read, fb_write},
41     {"/dev/events", 0, 0, 0, events_read, invalid_write},
42     {"/dev/fbsync", 0, 0, 0, invalid_read, fbsync_write},
43     {"/proc/dispinfo", 128, 0, 0, dispinfo_read, invalid_write},
44     {"/dev/tty", 0, 0, 0, invalid_read, serial_write},
45 #include "files.h"
46 };
47
48 #define NR_FILES (sizeof(file_table) / sizeof(file_table[0]))
49
50 > int fs_open(const char *pathname, int flags, int mode){--
60 > size_t fs_read(int fd, void *buf, size_t len){--
72 > int fs_close(int fd){--
76 > size_t fs_write(int fd, const void *buf, size_t len){--
95
96 > size_t fs_lseek(int fd, size_t offset, int whence){--
110
111 > void init_fs() {--
```

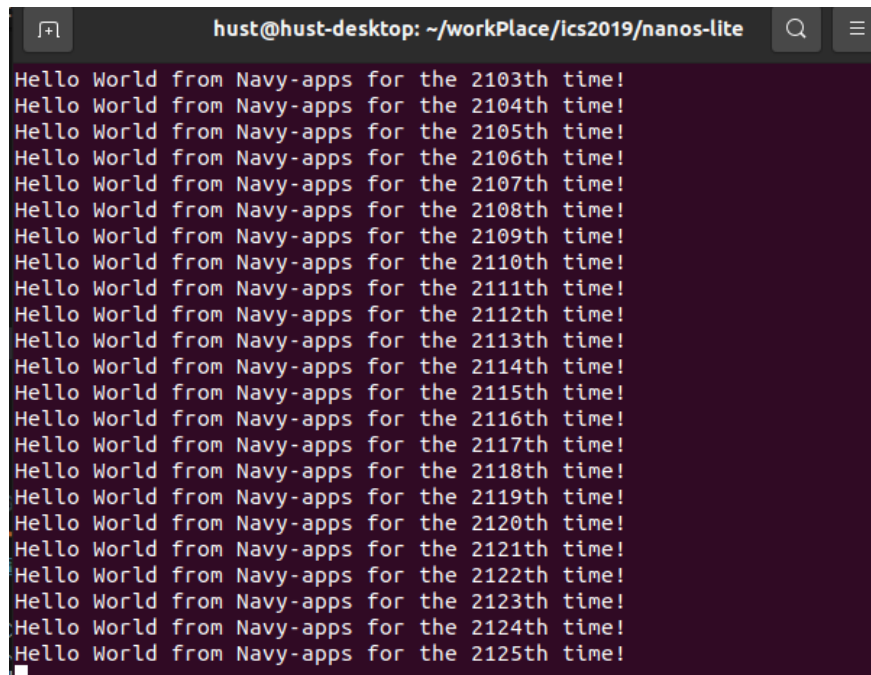
图 2.3.3 VFS 相关函数

2.3.4 PA3 测试结果

```
on
[/home/hust/workPlace/ics2019/nanos-lite/src/irq.c,9,do_event] self trap
[/home/hust/workPlace/ics2019/nanos-lite/src/main.c,39,main] system panic: Should not reach here
nemu: HIT BAD TRAP at pc = 0x80100bb0

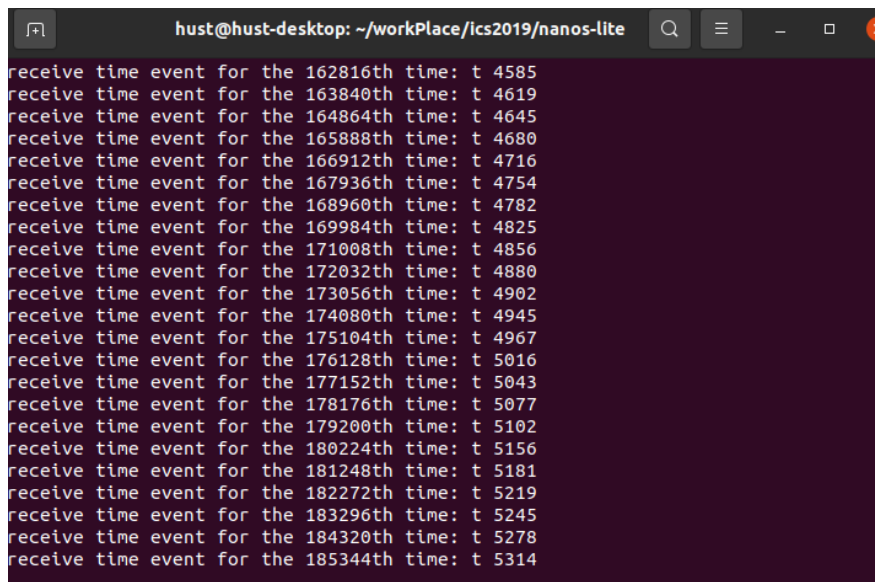
[src/monitor/cpu-exec.c,34,monitor_statistic] total guest instructions = 374263
make[1]: Leaving directory '/home/hust/workPlace/ics2019/nemu'
hust@hust-desktop:~/workPlace/ics2019/nanos-lite$
```

图 2.3.4 pa3-1 运行结果



```
hust@hust-desktop: ~/workPlace/ics2019/nanos-lite
Hello World from Navy-apps for the 2103th time!
Hello World from Navy-apps for the 2104th time!
Hello World from Navy-apps for the 2105th time!
Hello World from Navy-apps for the 2106th time!
Hello World from Navy-apps for the 2107th time!
Hello World from Navy-apps for the 2108th time!
Hello World from Navy-apps for the 2109th time!
Hello World from Navy-apps for the 2110th time!
Hello World from Navy-apps for the 2111th time!
Hello World from Navy-apps for the 2112th time!
Hello World from Navy-apps for the 2113th time!
Hello World from Navy-apps for the 2114th time!
Hello World from Navy-apps for the 2115th time!
Hello World from Navy-apps for the 2116th time!
Hello World from Navy-apps for the 2117th time!
Hello World from Navy-apps for the 2118th time!
Hello World from Navy-apps for the 2119th time!
Hello World from Navy-apps for the 2120th time!
Hello World from Navy-apps for the 2121th time!
Hello World from Navy-apps for the 2122th time!
Hello World from Navy-apps for the 2123th time!
Hello World from Navy-apps for the 2124th time!
Hello World from Navy-apps for the 2125th time!
```

图 2.3.5 pa3-2 dummy 通过



```
hust@hust-desktop: ~/workPlace/ics2019/nanos-lite
receive time event for the 162816th time: t 4585
receive time event for the 163840th time: t 4619
receive time event for the 164864th time: t 4645
receive time event for the 165888th time: t 4680
receive time event for the 166912th time: t 4716
receive time event for the 167936th time: t 4754
receive time event for the 168960th time: t 4782
receive time event for the 169984th time: t 4825
receive time event for the 171008th time: t 4856
receive time event for the 172032th time: t 4880
receive time event for the 173056th time: t 4902
receive time event for the 174080th time: t 4945
receive time event for the 175104th time: t 4967
receive time event for the 176128th time: t 5016
receive time event for the 177152th time: t 5043
receive time event for the 178176th time: t 5077
receive time event for the 179200th time: t 5102
receive time event for the 180224th time: t 5156
receive time event for the 181248th time: t 5181
receive time event for the 182272th time: t 5219
receive time event for the 183296th time: t 5245
receive time event for the 184320th time: t 5278
receive time event for the 185344th time: t 5314
```

图 2.3.6 pa3-2 events 测试通过

```
hust@hust-desktop: ~/workPlace/ics2019/nanos-lite
ello World from Navy-apps for the 2326th time!
ello World from Navy-apps for the 2327th time!
ello World from Navy-apps for the 2328th time!
ello World from Navy-apps for the 2329th time!
ello World from Navy-apps for the 2330th time!
ello World from Navy-apps for the 2331th time!
ello World from Navy-apps for the 2332th time!
ello World from Navy-apps for the 2333th time!
ello World from Navy-apps for the 2334th time!
ello World from Navy-apps for the 2335th time!
ello World from Navy-apps for the 2336th time!
ello World from Navy-apps for the 2337th time!
ello World from Navy-apps for the 2338th time!
ello World from Navy-apps for the 2339th time!
ello World from Navy-apps for the 2340th time!
ello World from Navy-apps for the 2341th time!
ello World from Navy-apps for the 2342th time!
ello World from Navy-apps for the 2343th time!
ello World from Navy-apps for the 2344th time!
ello World from Navy-apps for the 2345th time!
ello World from Navy-apps for the 2346th time!
ello World from Navy-apps for the 2347th time!
ello World from Navy-apps for the 2348th time!
ello World from Navy-apps for the 2349th time!
```

图 2.3.7 pa3-2 HELLO 测试通过

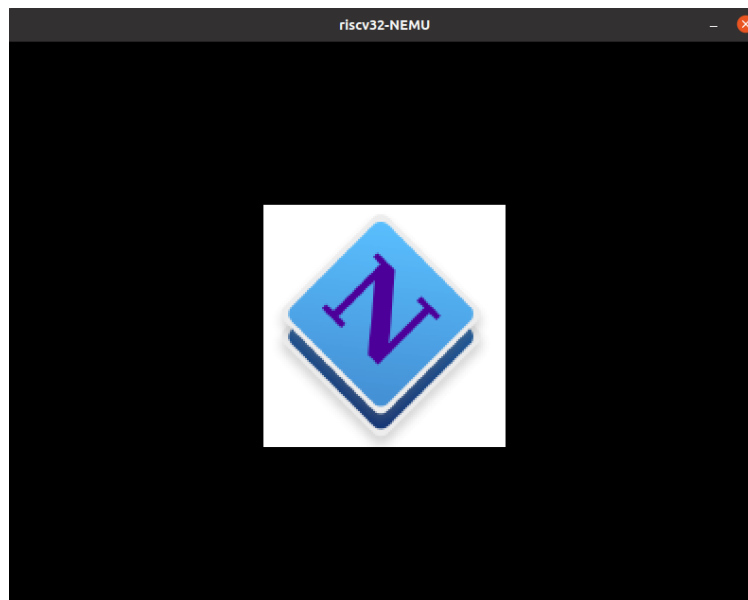


图 2.3.8 pa3-2 bmptest 测试通过

```
[/home/hust/workPlace/ics2019/nanos-lite/src/main.c,14,main] 'Hello World!' from
Nanos-lite
[/home/hust/workPlace/ics2019/nanos-lite/src/main.c,15,main] Build time: 01:02:4
8, Jan 18 2024
[/home/hust/workPlace/ics2019/nanos-lite/src/ramdisk.c,30,init_ramdisk] ramdisk
info: start = , end = , size = -2146422964 bytes
[/home/hust/workPlace/ics2019/nanos-lite/src/device.c,56,init_device] Initializi
ng devices...
[/home/hust/workPlace/ics2019/nanos-lite/src/irq.c,22,init_irq] Initializing int
errupt/exception handler...
[/home/hust/workPlace/ics2019/nanos-lite/src/proc.c,25,init_proc] Initializing p
rocesses...
[/home/hust/workPlace/ics2019/nanos-lite/src/loader.c,40,naive_uoload] Jump to en
try = -0x7cfff0c
PASS!!!
nemu: HIT GOOD TRAP at pc = 0x80100da8

[src/monitor/cpu-exec.c,34,monitor_statistic] total guest instructions = 1548602
make[1]: Leaving directory '/home/hust/workPlace/ics2019/nemu'
hust@hust-desktop:~/workPlace/ics2019/nanos-lite$
```

图 2.3.9 pa3-2 bmptest 测试通过

```
const char *name, *bin, *arg1;
} items[] = {
{"Litenes (Super Mario Bros)", "/bin/litenes", "/share/games/nes/mario.nes"},
{"Litenes (Yie Ar Kung Fu)", "/bin/litenes", "/share/games/nes/kungfu.nes"},
{"PAL - Xian Jian Qi Xia Zhuan", "/bin/pal", NULL},
{"bmptest", "/bin/bmptest", NULL},
{"dummy", "/bin/dummy", NULL},
{"events", "/bin/events", NULL},
{"hello", "/bin/hello", NULL},
{"text", "/bin/text", NULL},
};
```

图 2.3.10 pa3-3 init 开机界面菜单



图 2.3.11 pa3-3 仙剑奇侠传运行!

2.3.5 PA3 必答题

- 1) 理解上下文结构体的前世今生。

答：上下文通过*c 引用。在自陷操作中，上下文由 trap.S 中的汇编代码负责填充的。trap.S 文件中的汇编代码会对上下文结构体进行初始化，确保自陷操作的顺利进行。

2) 理解穿越时空的旅程。

答：Nanos-lite 调用中断，通过 AM 发起自陷指令的汇编代码，在自陷指令执行之前，会保存当前的执行上下文。接着，控制流转移到 CPU 自陷指令指定的内存地址空间，完成自陷处理。自陷指令执行结束后，恢复之前保存的上下文，并返回到触发自陷前的运行状态。

3) Hello 程序是什么，它从而何来，要到哪里去

答：在被编译成 ELF 格式文件之后，`hello.c` 程序被存储在 ramdisk 中。通过 `naive_uoload` 函数，该 ELF 文件被加载到内存中的适当位置，并交由操作系统进行调用和执行。程序在运行过程中，使用 `SYS_write` 系统调用来输出文本信息。当程序执行完毕后，操作系统负责回收该程序所占据的内存空间。

3 实验总结

作为大学四年的最后一次课程设计实验，这次 pa 实验的难度和复杂度不言而喻。我们在给出的项目框架的基础上一步一步实现一个完整的计算机系统。

本以为自己在备考 408 后对计算机系统结构的了解已经足够深入了，但真正着手编写代码起来却发现自己学的知识只是一些系统框架方面的内容，仍然需要面对更多的问题、查阅更多的手册、获取更多的知识才能够支撑起这样完整的 NEMU 设计的工作。比如，我清楚取指-译码-执行-更新 PC 的四个阶段，但我却无法完整说出这些过程中所使用的所有汇编指令，只能查看反汇编文件，同样我记得 riscv 的指令格式，却无法具体实现各个指令，必须查阅手册才得以完成。除此之外，还有很多问题让我无法通过手册互联网解决，经常在一个小错误停留大量时间。

尽管在完成这个课程设计的过程中遇到了各种问题，但我也收获了很多。在大学四年经历的所有课程设计实验中，没有哪个实验能比 PA 实验更能培养我们的计算机系统能力，就连调试 bug 的难度也是之前所有实验不可比拟的。在这次实验中，我们面对更加接近完整的计算机系统结构项目，更“近距离”地接触如何从“零”实现一个计算机系统。

这次课程设计实验完成时间较短，有很多方案书中给出的思考问题，以及很多值得学习的内容都没有来得及去接触，实验也只完成到 PA3 部分。即使课程结束，这次实验还有许许多多未被我们发掘的价值，值得反复回味。

总之，这次课程设计实验也是我们了解计算机系统如何工作的宝贵经验，它让我认识到我们现在所学的只是茫茫计算机世界的冰山一角，未来还有更多更丰富的知识等着我们。

参考文献

- [1] 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.
- [2] 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.
- [3] 张晨曦, 王志英. 计算机系统结构. 高等教育出版社. 2008 年.