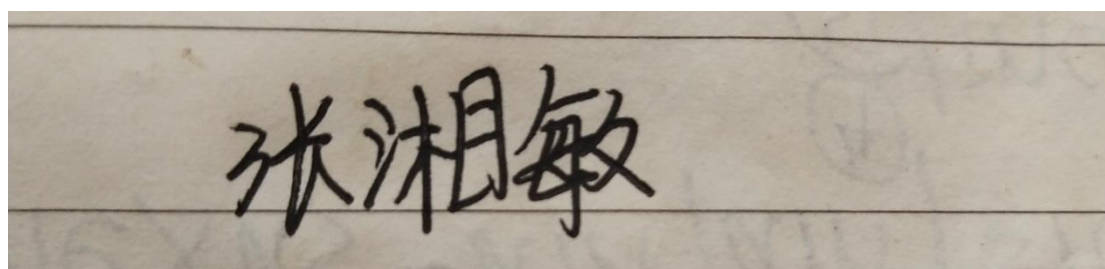


学号: U201814729

班级: CS1808

姓名: 张湘敏



一、必答题

PA1

1. 我选择的是 ISA 是 riscv32

2. 理解基础设施

$500 \times 90\% \times 30 \times 20 = 270000s = 75h$, 所以一学期要在调试上花 75 个小时。

$500 \times 90\% \times 20 \times 20 = 180000s = 50h$, 所以一学期可节省 50 个小时的调试时间。

3. 查阅 riscv32 手册

(1) riscv32 有哪几种指令格式?

6 种

(2) LUI 指令的行为是什么?

根据 `lui rd, immediate`, 可知该指令用于高位立即数加载。

(3) mstatus 寄存器的结构是什么样的?

- mstatus (Machine Status) 它保存全局中断使能, 以及许多其他的状态, 如图 10.4 所示。

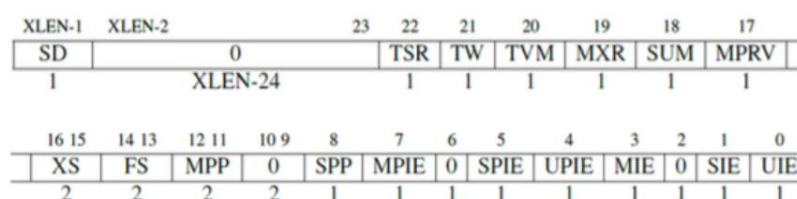


图 10.4: mstatus 控制状态寄存器。在仅有机模式且没有 F 和 V 扩展的简单处理中, 有效的域只有全局中断使能、MIE 和 MPIE (它在异常发生后保存 MIE 的旧值)。RV32 的 XLEN 是 32, RV64 是 40。

(4) 完成 PA1 的内容之后, nemu/目录下所有.c 和.h 文件总共有多少行代码? 你是使用什么命令得到这个结果的? 和框架代码相比, 你在 PA1 中编写了多少行代码? (Hint: 目前 pa1 分支中记录的正好是做 pa1 之前的状态, 思考一下应该如何回到“过去”?) 你可以把这条命令写入 Makefile 中, 随着实验进度的推进, 你可以很方便地统计工程的代码行数, 例如敲入 `make count` 就会自动运行统计代码行数的命令。再来个难一点的, 除去空行之外, nemu/目录下的所以.c 和.h 文件总共有多少行代码?

通过 `git checkout pa1` 回到刚完成 pa1 的状态。在 nemu/目录下输入

```
find -name "*. [hc]" | xargs wc -l
```

即可输出 nemu 目录下所有 .c 和 .h 文件的总行数。如图 1.1 所示。

```
hust@hust-desktop:~/ics2019/nemu$ find -name "*. [hc]" | xargs wc -l
 22 ./include/monitor/diff-test.h
 8 ./include/monitor/expr.h
 22 ./include/monitor/watchpoint.h
 27 ./include/monitor/log.h
 16 ./include/monitor/monitor.h
 40 ./include/device/map.h
 35 ./include/common.h
 24 ./src/isa/riscv32/exec/ldst.c
 1 ./src/isa/riscv32/exec/system.c
29 ./src/isa/riscv32/exec/special.c
 9 ./src/isa/riscv32/exec/all-instr.h
33 ./src/isa/riscv32/exec/exec.c
 7 ./src/isa/riscv32/exec/compute.c
12 ./src/cpu/inv.c
20 ./src/cpu/relop.c
22 ./src/cpu/cpu.c
41 ./src/memory/memory.c
12 ./src/main.c
5486 total
hust@hust-desktop:~/ics2019/nemu$
```

图 1.1 查看文件总行数

对于空行，可以用 `find -name "*. [hc]" | xargs cat | sed '/^\s*$/d' | wc -l` 指令来去除。结果如图 1.2 所示。

```
hust@hust-desktop:~/ics2019/nemu$ find -name "*. [hc]" | xargs cat | sed '/^\s*$/d' | wc -l
4503
```

图 1.2 去除空行查看文件总行数

5、使用 `man` 打开工程目录下的 `Makefile` 文件，你会在 `CFLAGS` 变量中看到 `gcc` 的一些编译选项。请解释 `gcc` 中的 `-Wall` 和 `-Werror` 有什么作用？为什么要使用 `-Wall` 和 `-Werror`？

`-wall` 打开 `gcc` 所有警告；

`-werror` 将所有警告当成错误进行处理；

使用 `-wall` 以及 `-werror` 有利于发现代码中不严谨的地方，如类型的隐式转换。

PA2

1. 请整理一条指令在 `nemu` 中的执行过程。

以 `lw` 指令为例

isa_exec 函数被执行

```
|  
|==> 通过 instr_fetch 函数获得当前 pc 值对应的内存位置中的指令  
      调用idex, 即译码-执行函数  
|  
|==> 调用decode_ld 函数, 获得操作数, 将其保存在全局变量 id_src  
以及 id_dest 中  
      调用exec_load 函数,  
|  
|==> 根据 funct3 字段, 索引 load_table, 找到对应的执行函数  
exec_ld, 同时设定位宽 width  
      调用函数 exec_ld  
|  
|==>根据 lw 指令的逻辑, 调用rtl 函数  
      调用rtl_lm 将 id_src.addr 所指向的地址取出  
decinfo.width 宽度的数据, 保存在临时寄存器 s0 中  
      调用rtl_sr 将 s0 保存在 id_dest.reg 指向的寄存器中  
      根据宽度调用print_asm_template2 打印具体的汇编指令
```

2. 在 nemu/include/rtl/rtl.h 中, 你会看到由 static inline 开头定义的各种 RTL 指令函数。选择其中一个函数, 分别尝试去掉 static, 去掉 inline 或去掉两者, 然后重新进行编译, 你可能会看到发生错误。请分别解释为什么这些错误会发生/不发生? 你有办法证明你的想法吗?

去掉 static 不会发生错误;

去掉 inline, 编译时不会发生错误, 链接时会发生错误。LD 在链接不同的 object 时, 会发现同一个符号在不同的 object 中被多次定义, 并且我们没有告诉 LD 这种情况下应该怎么进行链接, 因此报错。

3. 编译与链接

(1) 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?

一个。

(2) 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。

两个。

`static` 关键字表示变量只有在当前文件可以被识别, `volatile` 表示这处代码不会被编译器优化, 而 `debug.h` 通过 `include` 包含了 `common.h`, 所以重新声明的变量不会覆盖之前的 `dummy`。

(3) 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这样的问题? (回答完本题后可以删除添加的代码。)

出现重定义。因为给变量赋值之后, 声明就变成了定义, 所以会发生重定义问题。

4. 请描述你在 `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`。(这个问题包括两个方面: `Makefile` 的工作方式和编译链接的过程。)

通过阅读文档中给出的 C 语言基础中的 `makefile` 基础一节以及 `man` 文档, 对 `makefile` 有一个基本的了解。 `Makefile` 基本的工作方式如下:

1. 首先依次读取变量“`MAKEFILES`”定义的 `makefile` 文件列表
2. 读取工作目录下的 `makefile` 文件
3. 一次读取工作目录下的 `makefile` 文件指定的 `include` 文件
4. 查找重建所有已读的 `makefile` 文件的规则
5. 初始化变量值并展开那些需要立即展开的变量和函数并根据预设条件确定执行分支
6. 建立依赖关系表
7. 执行 `rules`

具体到 `/nemu/Makefile` 这个文件本身, 具体的编译链接过程如下:

1. `Makefile` 默认的 `ISA` 为 `x86`, 如果定义了具体的 `ISA`, 则需要保证其有效
2. 根据 `ISA` 确定需要 `include` 的文件列表
3. 确定编译目标文件夹 (默认是 `build/`) 确定编译器以及链接器 (`gcc`)
4. 设置编译选项 `CFLAGS`
5. 读取所有需要编译的 `.c` 的文件并将其编译为 `.o` 文件
6. 进行链接
7. 执行 `git commit`

PA3

1. 理解上下文结构体的前世今生（见PA3.1 阶段）

你会在 `__am_irq_handle()` 中看到有一个上下文结构指针 `c`, `c` 指向的上下文结构究竟在哪里? 这个上下文结构又是怎么来的? 具体地, 这个上下文结构有很多成员, 每一个成员究竟在哪里赋值的? `$ISA-nemu.h`, `trap.S`, 上述讲义文字, 以及你刚刚在 `NEMU` 中实现的新指令, 这四部分内容又有什么联系?

`__am_irq_handle` 函数在整个项目中, 唯一出现的调用文件就是 `trap.S`。

其中通过 `jal` 指令直接跳转到对应的函数体, 而函数的参数是保存在栈当中的, 可以看到在 `jal` 指令之前有诸多压栈操作, 按照顺序是

1. 32 个寄存器, 通过 `MAP(REGS, PUSH)`
2. 成员 `cause`, 通过 `sw t0 OFFSET_CAUSE(sp)`
3. 成员 `status`, 通过 `sw t1 OFFSET_STATUS(sp)`
4. 成员 `epc`, 通过 `sw t2 OFFSET_EPC(sp)`

于是不同的成员被赋值, 可以在 `__am_irq_handle` 函数中使用。

2. 理解穿越时空的旅程（见PA3.1 阶段）

从 `Nanos-lite` 调用 `_yield()` 开始, 到从 `_yield()` 返回的期间, 这一趟旅程具体经历了什么? 软 (`AM`, `Nanos-lite`) 硬 (`NEMU`) 件是如何相互协助来完成这趟旅程的? 你需要解释这一过程中的每一处细节, 包括涉及的每一行汇编代码/C 代码的行为, 尤其是一些比较关键的指令/变量. 事实上, 上文的必答题“理解上下文结构体的前世今生”已经涵盖了这趟旅程中的一部分, 你可以把它的回答包含进来。

首先 `_yield` 函数, 通过内联汇编代码将 `a7` 设置为 -1, 表示当前的 `ecall` 类型是 `_yield`, 接着执行了 `ecall` 指令。

汇编 `ecall` 指令将会由 `ecall` 对应的 `EHelper` 来执行相关的函数, 函数中会调用 `raise_intr` 函数, 参数 `N0` 即为 `a7` 寄存器的值, 表示中断号。

在 `raise_intr` 函数中会保存 `epc` 到 `sepc` 寄存器，将中断号保存到 `scause` 寄存器，并从 `stvec` 获得中断入口地址并进行跳转。也就是 `__am_asm_trap` 函数的入口地址，也就是汇编代码 `trap.S` 中的起始位置。开始执行。

汇编代码会执行到上述的 `__am_irq_handle` 函数。

`__am_irq_handle` 函数根据 `c->cause` 来分别进行处理，如果是 -1 就表示 `yield` 事件，如果是 0 到 19 (支持的系统调用的个数) 就说明是系统调用。此处是 `yield`，于是填充 `ev.event` 成员为 `_EVENT_YIELD` 并调用用户定义的回调函数 `do_event`。

同样是根据 `event` 的类型来分别处理，如果是 `_EVENT_YIELD` 就打印出信息到终端，如果是 `_EVENT_SYSCALL` 的话就调用 `do_syscall`，打印信息到终端。

函数结束之后将会回到 `trap.S` 汇编代码。恢复上下文并调用 `sret` 指令。

`sret` 指令将会调用 `nemu` 中针对 `sret` 指令的执行函数，从 `sepc` 寄存器中读出之前保存的 `pc`，加 4，表示中断发生时的下一跳指令的地址，并进行跳转。

至此 `yield` 函数执行完毕。

3. `hello` 程序是什么，它从而何来，要到哪里去（见 PA3.2 阶段）

我们知道 `navy-apps/tests/hello/hello.c` 只是一个 C 源文件，它会被编译链接成一个 ELF 文件。那么，`hello` 程序一开始在哪里？它是怎么出现内存中的？为什么会出现在目前的内存位置？它的第一条指令在哪里？究竟是怎么执行到它的第一条指令的？`hello` 程序在不断地打印字符串，每一个字符又是经历了什么才会最终出现在终端上？

`hello` 程序对应的 elf 文件会在整个项目编译的时候，将其在 `ramdisk` 的偏移位置保存在 `files.h` 的记录表中 (`disk_offset` 成员)。

接着通过 `load` 函数解析对应的 elf 文件，得到具体的入口地址，保存在 `e_entry` 中。

通过 `((void(*)())entry)()` 跳转到对应位置进行执行。

在 main 函数中通过 printf 进行输出，printf 函数首先会尝试进行 _brk，如果失败则一个字符一个字符的通过 write 输出到终端，如果成功则将字符串作为整体调用 write 进行输出。

write 函数会调用 _write 系统调用，在对应的处理函数中，发现输出的对象是 stdout，则直接通过 serial_write 进行输出。

4. 运行仙剑奇侠传时会播放启动动画，动画中仙鹤在群山中飞过。这一动画是通过 navyapps/apps/pal/src/main.c 中的 PAL_SplashScreen() 函数播放的。阅读这一函数，可以得知仙鹤的像素信息存放在数据文件 mgo.mkf 中。请回答以下问题：库函数，libos, Nanos-lite, AM, NEMU 是如何相互协助，来帮助仙剑奇侠传的代码从 mgo.mkf 文件中读出仙鹤的像素信息，并且更新到屏幕上？

换一种 PA 的经典问法：这个过程究竟经历了些什么？

在 navy-apps/apps/palc/hal/hal.c 中的 redraw 函数中通过 NDL_DrawRect 和 NDL_Render 更新屏幕。

NDL 通过之前的初始化操作维护了一块画布 canvas，并将其绘制操作限定在该画布上。

NDL_DrawRect 会将传入的 pixels 保存到会把传入的像素逐个存入画布的对应位置。

首先调用了 libndl 库，在该库中会打开设备文件 /dev/fb 和 /dev/fbsync，在接收到该函数调用后会向 /dev/fb 设备文件中写入，在该函数中，判断出要写的文件是 /dev/fb 设备文件之后，会调用 fb_write 帮助函数，之后会调用 draw_rect 函数，该函数位于 nexus-ambsibc/io.c 中，在函数内会调用 _io_write 函数。

_io_write 会转发给 __am_video_write 函数，该函数中会执行 out 汇编指令，将数据传送给 vga 设备中。

vga 设备在接收到数据后会保存在定义的显存中，当之后 NDL 库向 /dev/fbsyn 设备文件中写入时，vga 设备最终会调用 SDL 库来更新画面。

NDL_Render 函数会对画布的每一行先调用 fseek 把偏移量定位到该行起点在屏幕中对应的位置，然后调用 fwrite 输出画布的一行，并调用 fflush() 刷新缓冲，最后调用putc 向 fbsyncdev 输出 0 进行同步，并调用 fflush 刷新缓冲。

二、测试结果及说明

PA1

不同指令的测试

1. 帮助指令 help

```
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 - Evaluate given expression.(p EXPR)  
x - Scan memory.(x N EXPR)  
w - Set watchpoint.(w EXPR)  
d - Cancel Watchpoint.(d N)  
(nemu) █
```

2. 单步执行 si

```
(nemu) si 2  
80100000: b7 02 00 80          lui 0x80000,t0  
80100004: 23 a0 02 00          sw 0(t0),$0  
(nemu)
```

3. 打印寄存器信息 info r

```
(nemu) info r  
[src/monitor/debug/ui.c,122,cmd_info] 打印参数点  
$0 = 0x00000000 ra = 0x00000000 sp = 0x00000000 gp = 0x00000000  
tp = 0x00000000 t0 = 0x00000000 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  
[Show Applications]  
(nemu) █
```

4. 表达式求值

```
(nemu) p $t6+10+7*8-8  
[src/monitor/debug/expr.c,369,expr] 共有9个符号  
结果为58  
(nemu) █
```

5. 监视点相关


```
PA [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
Activities Terminal 2月11 19:58
hust@hust-desktop: ~/ics2019/nemu

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

(nemu) info w
[src/monitor/debug/ui.c,126,cmd_info] 打印监视点
(nemu) w 100+9*10-10
[src/monitor/debug/watchpoint.c,28,new_wp] 新建的监视点的value为100+9*10-10
[src/monitor/debug/expr.c,369,expr] 共有7个符号

(nemu) info w
[src/monitor/debug/ui.c,126,cmd_info] 打印监视点
WP NO.0 "100+9*10-10" value=180
(nemu) w 1+$t0
[src/monitor/debug/watchpoint.c,28,new_wp] 新建的监视点的value为1+$t0
[src/monitor/debug/expr.c,369,expr] 共有3个符号

(nemu) info w
[src/monitor/debug/ui.c,126,cmd_info] 打印监视点
WP NO.1 "1+$t0" value=1
WP NO.0 "100+9*10-10" value=180
(nemu) w 233
[src/monitor/debug/watchpoint.c,28,new_wp] 新建的监视点的value为233
[src/monitor/debug/expr.c,369,expr] 共有1个符号

(nemu) info w
[src/monitor/debug/ui.c,126,cmd_info] 打印监视点
WP NO.2 "233" value=233
WP NO.1 "1+$t0" value=1
WP NO.0 "100+9*10-10" value=180
(nemu) d 1
(nemu) info w
[src/monitor/debug/ui.c,126,cmd_info] 打印监视点
WP NO.2 "233" value=233
WP NO.0 "100+9*10-10" value=180
(nemu) w 111-2
[src/monitor/debug/watchpoint.c,28,new_wp] 新建的监视点的value为111-2
[src/monitor/debug/expr.c,369,expr] 共有3个符号

(nemu) info w
[src/monitor/debug/ui.c,126,cmd_info] 打印监视点
WP NO.1 "111-2" value=109
WP NO.2 "233" value=233
WP NO.0 "100+9*10-10" value=180
(nemu)
```

PA1 成功完成。

PA2

1. 运行 dummy

```
[src/device/isa/mio.c,14,add_mio_map] Add mio map 'keyboard' at [0x00000000, 0x00000003]
[src/monitor/monitor.c,25,welcome] Debug: OFF
[src/monitor/monitor.c,25,welcome] Build time: 13:02:36, Feb 14 2022
Welcome to riscv32-NEMU!
For help, type 'help'
nemu: HIT GOOD TRAP at pc = 0x00100030

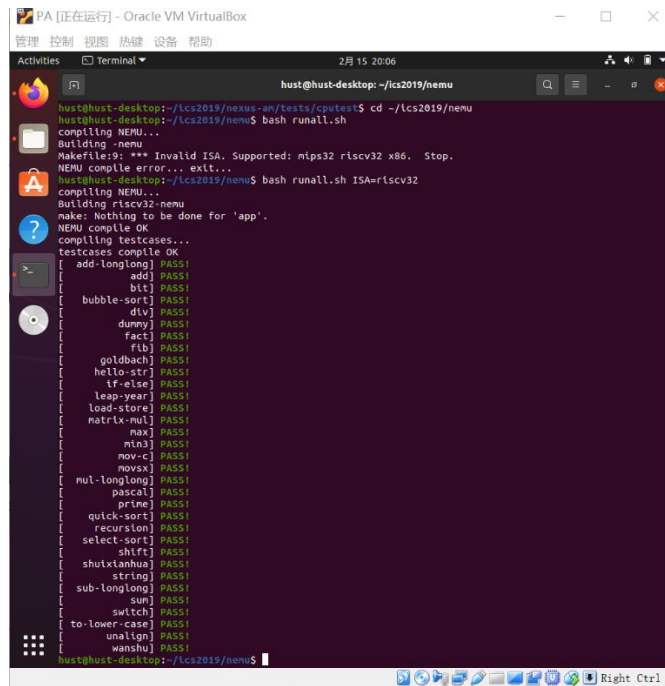
[src/monitor/cpu-exec.c,30,monitor_statistic] total guest instructions = 13
dummy
hust@hust-desktop: ~/ics2019/nexus-un/tests/cputests
```

2. 实现 string

```
hust@hust-desktop: ~/ics2019/nexus-un/tests/cputests$ make ARCH=riscv32 nemu ALL=string run
hakeft@17: warning: overriding recipe for target 'image'
/home/hust/ics2019/nexus-un/arch/platform/nemu.mk:28: warning: ignoring old recipe for target 'image'
hakeft@18: warning: overriding recipe for target 'run'
/home/hust/ics2019/nexus-un/arch/platform/nemu.mk:27: warning: ignoring old recipe for target 'run'
# Building string [riscv32-nemu] with ARCH=NONE (/home/hust/ics2019/nexus-un)
# Building lib-a (riscv32-nemu)
# Building lib-klib (riscv32-nemu)
# Creating binary image [riscv32-nemu]
# LD -> Build/string-riscv32-nemu.elf
# OBJCOPY -> Build/string-riscv32-nemu.bin
Building riscv32-nemu
[src/monitor/monitor.c,48,load_img] The image is /home/hust/ics2019/nexus-un/tests/cputests/build/string-riscv32-nemu.bin
[src/memory/memory.c,18,register_gmem] Add 'gmem' at [0x00000000, 0x07ffffff]
[src/device/isa/part-io.c,14,add_ptio_map] Add port-io map 'serial' at [0x00000000, 0x00000003]
[src/device/isa/mio.c,14,add_mio_map] Add mio map 'serial' at [0x00000000, 0x00000003]
[src/device/isa/part-io.c,14,add_ptio_map] Add port-io map 'rtc' at [0x00000000, 0x00000003]
[src/device/isa/mio.c,14,add_mio_map] Add mio map 'rtc' at [0x00000000, 0x00000003]
[src/device/isa/part-io.c,14,add_ptio_map] Add port-io map 'screen' at [0x00000000, 0x00000003]
[src/device/isa/mio.c,14,add_mio_map] Add mio map 'screen' at [0x00000000, 0x00000003]
[src/device/isa/part-io.c,14,add_ptio_map] Add port-io map 'vram' at [0x00000000, 0x00000003]
[src/device/isa/mio.c,14,add_mio_map] Add mio map 'vram' at [0x00000000, 0x00000003]
[src/device/isa/part-io.c,14,add_ptio_map] Add port-io map 'keyboard' at [0x00000000, 0x00000003]
[src/device/isa/mio.c,14,add_mio_map] Add mio map 'keyboard' at [0x00000000, 0x00000003]
[src/monitor/monitor.c,25,welcome] Debug: OFF
[src/monitor/monitor.c,25,welcome] Build time: 16:37:14, Feb 15 2022
Welcome to riscv32-NEMU!
For help, type 'help'
nemu: HIT GOOD TRAP at pc = 0x00100158

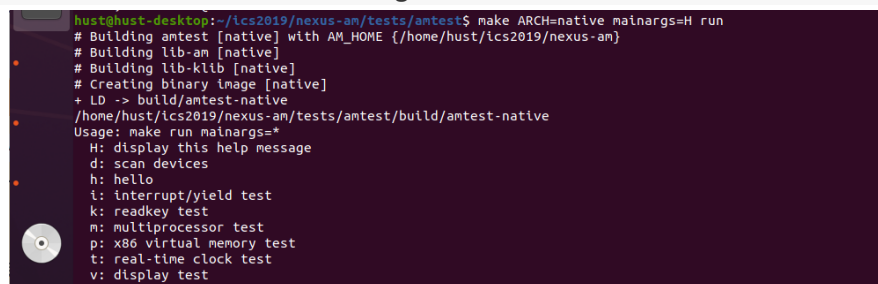
[src/monitor/cpu-exec.c,30,monitor_statistic] total guest instructions = 1484
string
hust@hust-desktop: ~/ics2019/nexus-un/tests/cputests
```

3. 回归测试

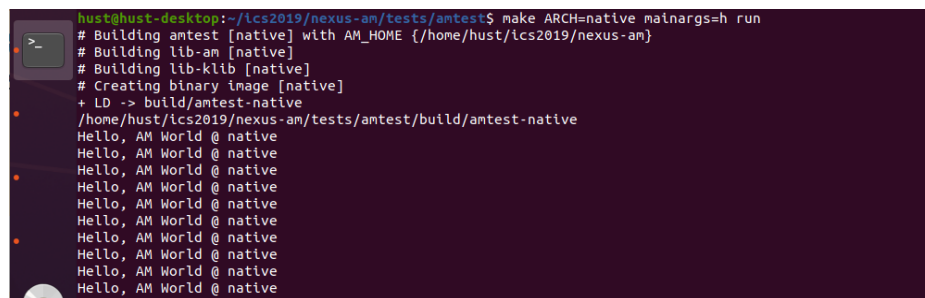


4. 输入输出

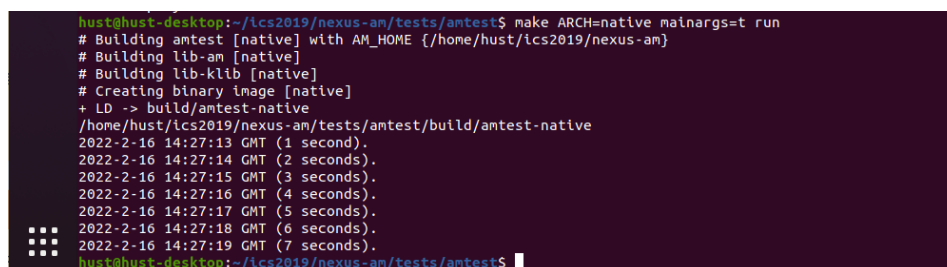
(1) 输入 make ARCH=native mainargs=H run



(2) 串口



(3) 时钟



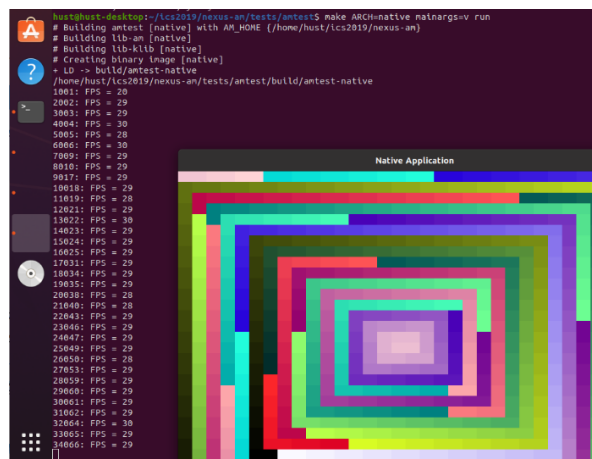
(4) 跑分

```
Min time: 21 ms [21447]
[md5] MD5 digest: * Passed.
min time: 40 ms [43097]
=====
MicroBench PASS      24216 Marks
vs. 100000 Marks (i7-7700K @ 4.20GHz)
Total time: 1070 ms
hust@hust-desktop:~/ics2019/nexus-am/apps/microbench$
```

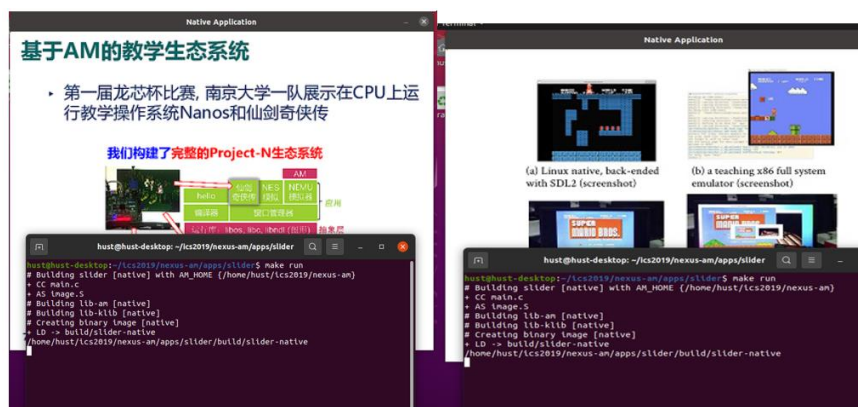
(5) 键盘

```
Get key: 43 A down
Get key: 43 A up
Get key: 45 D down
Get key: 45 D up
Get key: 17 3 down
Get key: 17 3 up
Get key: 33 T down
Get key: 33 T up
Get key: 53 APOSTROPHE down
Get key: 53 APOSTROPHE up
Get key: 54 RETURN down
Get key: 54 RETURN up
Get key: 39 LEFTBRACKET down
Get key: 39 LEFTBRACKET up
Get key: 24 0 down
Get key: 24 0 up
Get key: 49 J down
Get key: 49 J up
hust@hust-desktop:~/ics2019/nexus-am/tests/antest$ make mainargs=k run
```

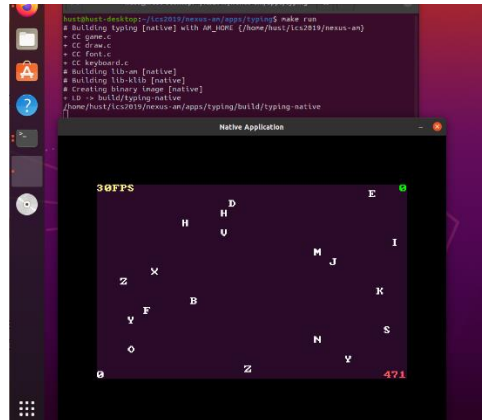
(6) VGA



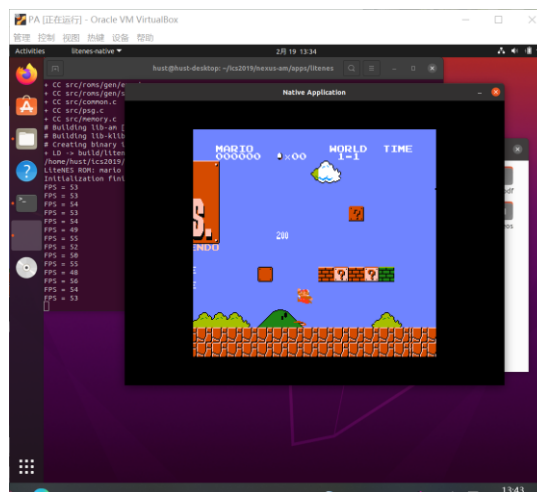
(7) 幻灯片



(8) 打字小游戏



(9) LiteNES: 输入 make mainargs=mario run.



PA3

1. 实现自陷操作_yield()及其过程

```
[/home/hust/ics2019/nanos-lite/src/main.c:14,main] 'Hello World!' from Nanos-lite
[/home/hust/ics2019/nanos-lite/src/main.c:15,main] Build time: 16:23:56, Feb 17 2022
[/home/hust/ics2019/nanos-lite/src/randisk.c:29,init_randisk] randisk info: start = 0x80102e88, end = 0x80109b00, size = 27896 bytes
[/home/hust/ics2019/nanos-lite/src/device.c:61,init_device] Initializing devices...
[/home/hust/ics2019/nanos-lite/src/irq.c:27,init_irq] Initializing interrupt/exception handler...
[/home/hust/ics2019/nanos-lite/src/proc.c:27,init_proc] Initializing processes...
[/home/hust/ics2019/nanos-lite/src/main.c:33,main] Finish initialization
self int
[/home/hust/ics2019/nanos-lite/src/main.c:49,main] system panic: Should not reach here
nemu: HIT BAD TRAP at pc = 0x80100e00
[src/monitor/cpu-exec.c:30,monitor_statistic] total guest instructions = 381992
make[1]: Leaving directory '/home/hust/ics2019/nemu'
hust@hust-desktop:~/ics2019/nanos-lite$
```

2.运行 dummy

```
[/home/hust/ics2019/nanos-lite/src/main.c:14,main] 'Hello World!' from Nanos-lite
[/home/hust/ics2019/nanos-lite/src/main.c:15,main] Build time: 16:09:06, Feb 18 2022
[/home/hust/ics2019/nanos-lite/src/randisk.c:29,init_randisk] randisk info: start = 0x80102f4c, end = 0x80109c44
size = 27896 bytes
[/home/hust/ics2019/nanos-lite/src/device.c:61,init_device] Initializing devices...
[/home/hust/ics2019/nanos-lite/src/irq.c:15,init_irq] Initializing interrupt/exception handler...
[/home/hust/ics2019/nanos-lite/src/fs.c:75,fs_open] open file : /dev/fb
[/home/hust/ics2019/nanos-lite/src/proc.c:27,init_proc] Initializing processes...
[/home/hust/ics2019/nanos-lite/src/loader.c:52,naive_upload] hello
[/home/hust/ics2019/nanos-lite/src/fs.c:75,fs_open] open file : /bin/events
[/home/hust/ics2019/nanos-lite/src/loader.c:54,naive_upload] Jump to entry = 830000c8
self int
nemu: HIT GOOD TRAP at pc = 0x80100e70
[src/monitor/cpu-exec.c:30,monitor_statistic] total guest instructions = 498629
make[1]: Leaving directory '/home/hust/ics2019/nemu'
hust@hust-desktop:~/ics2019/nanos-lite$ make ARCH=rv32cvc32-nemu run
```

3.运行 hello-world

```
hust@hust-desktop:~/ics2019/nanos-lite$
[~/home/hust/ics2019/nanos-lite/src/device.c:61,init_device] Initializing devices...
[~/home/hust/ics2019/nanos-lite/src/irq.c:15,init_irq] Initializing interrupt/exception handler...
[~/home/hust/ics2019/nanos-lite/src/fs.c:75,fs_open] open file : /dev/rtc
[~/home/hust/ics2019/nanos-lite/src/proc.c:27,init_proc] Initializing processes...
[~/home/hust/ics2019/nanos-lite/src/loader.c:39,naive_load] hello
[~/home/hust/ics2019/nanos-lite/src/fs.c:75,fs_open] open file : /bin/events
[~/home/hust/ics2019/nanos-lite/src/loader.c:54,naive_load] Jump to entry = 83000120
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!
Hello world from Navy-apps for the 16th time!
Hello world from Navy-apps for the 17th time!
Hello world from Navy-apps for the 18th time!
Hello world from Navy-apps for the 19th time!
Hello world from Navy-apps for the 20th time!
Hello world from Navy-apps for the 21th time!
Hello world from Navy-apps for the 22th time!
Hello world from Navy-apps for the 23th time!
Hello world from Navy-apps for the 24th time!
Hello world from Navy-apps for the 25th time!
Hello world from Navy-apps for the 26th time!
```

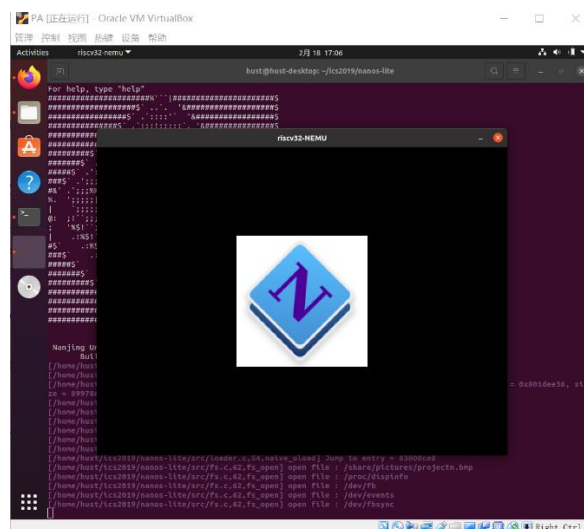
3.运行测试程序/bin/text

```
[~/home/hust/ics2019/nanos-lite/src/loader.c:54,naive_load] Jump to entry = 830002f4
[~/home/hust/ics2019/nanos-lite/src/fs.c:62,fs_open] open file : /share/texts/nan
PASS!!!
nemu: HIT GOOD TRAP at pc = 0x80100e74
[~/src/monitor/cpu-exec.c:30,monitor_statistic] total guest instructions = 1559649
make[1]: Leaving directory '/home/hust/ics2019/nemu'
hust@hust-desktop:~/ics2019/nanos-lite$
```

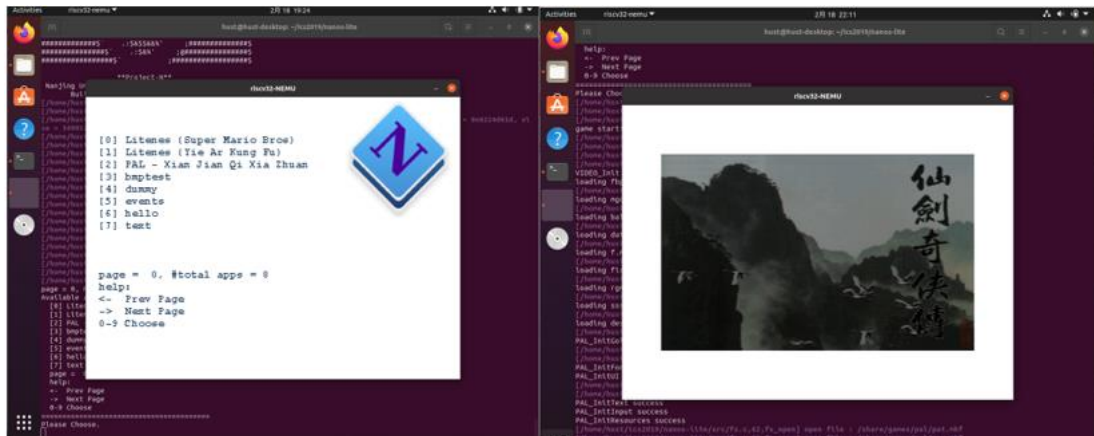
4.加载/bin/events

```
receive time event for the 62464th time: t 6222
receive event: ku L
receive event: ku J
receive time event for the 63488th time: t 6333
receive event: kd L
receive event: kd Q
receive event: kd I
receive time event for the 64512th time: t 6439
receive event: ku Q
receive event: ku L
receive event: ku I
receive time event for the 65536th time: t 6568
receive time event for the 66560th time: t 6652
receive time event for the 67584th time: t 6738
receive time event for the 68608th time: t 6824
receive time event for the 69632th time: t 6911
receive time event for the 70656th time: t 7081
receive time event for the 71680th time: t 7109
receive time event for the 72704th time: t 7203
receive time event for the 73728th time: t 7294
receive time event for the 74752th time: t 7387
receive time event for the 75776th time: t 7478
receive time event for the 76800th time: t 7572
```

5.加载/bin/bmptest



6.仙剑奇侠传运行（这个没有成功实现，只能加载出仙鹤，存档之类未实现）



三、心得体会

这次任务可以说是我目前遇到的难度最高的任务了，因为涉及很多知识，从基础的 c 语言到数据结构算法、具体的指令等等。PA 一共有五个阶段，由于我的能力有限，我只做到了 PA3。

首先需要配置好环境，在手册中提到关机使用 poweroff, 我最初就是直接关机导致出错，所以要谨慎。

pa1 重点是表达式求值和实现监视点。表达式求值主要是细节处理，如区分解引用和乘法。pa2 主要就是实现 RV32I 以及 RV32M 指令集。一开始看代码的时候确实感觉很复杂。理清了整个指令执行的过程之后，发现其实基本上就是体力活，只需要根据指令的具体说明，在函数体中调用rtl 相关的函数即可。Pa3 这个部分理解起来还比较困难，主要是因为涉及到了汇编以及 c 的联合编译，以及 elf 等具体的格式。但是在整个调用的流程完全弄清楚之后又非常的开心，尤其是汇编和 c 之间的完美合作，感觉非常的神奇。

总的来说，pa 项目收获良多。比如对于 makefile 的理解。我之前对于 makefile 一直是一知半解，对于具体的规则都不是很了解，看到 pa 中那么复杂的 makefile 也不想多看，但是在后来的必答题中发现需要回答相关的问题，就耐着性子看完 makefile 的一些基础知识，这时候再回过头看项目中涉及到的 makfile 就理解了大半。从硬件到软件，从指令集的实现到上层简单操作系统的实现，都让我学到了很多。