系统能力培养

院系：计算机学院

专业班级： CS2002

姓名：冯就康

学号：I201920029

指导教师：

2023 年 09 月 11 日

PA1：最简单的计算机

一、实验进度

PA1 的内容全部已完成。

二、任务描述

P1.1 --- 实现单步执行，打印寄存器状态，扫描内存

只需要在 ui.c 中声明并实现相关的函数就行。添加了三个函数分别为：cmd_si, cmd_info, cmd_x。其中

- cmd_si: Single step execution
- cmd_info: Print information of registers
- cmd_x: Scan memory

P1.2 --- 实现表达式求值功能

需要实现表达式求值功能，这是整个 PA1 的核心，表达式求值的实现 最终需要在 expr.c 文件中实现，在这期间，需要完成补充正则表达式、括号匹配 函数、递归求值 eval 函数，并最终整合到一起实现函数 expr。

P1.3 --- 实现监视点的功能

需要实现监视点的功能，需要补充 watchpoint 结构，完成监视点的相关操作函数，然后在 ui.c 中完成对监视点相关命令函数的编写，最后需要实现监 视点状态改变的暂停逻辑。

三、测试结果

1. 帮助 help 指令

```
(nemu) help
help - Display informations about all supported commands
c    - Continue the execution of the program
q    - Exit NEMU
si   - Single step execution
info - Print information of registers or watchpoints
p    - Evaluate expression
x    - Scan memory
w    - set watchpoint
d    - delete watchpoint
```

2.单步执行 si 指令

```
(nemu) si
80100000:   b7 02 00 80                                 lui  0x80000,t0
(nemu) si 2
80100004:   23 a0 02 00                                 sw  0(t0),$0
80100008:   03 a5 02 00                                 lw  0(t0),a0
```

3.打印寄存器 info 指令和监视点 w 指令

```
(nemu) w $t0
watchpoint $t0 set success
(nemu) w $t1
watchpoint $t1 set success
(nemu) w $t2
watchpoint $t2 set success
(nemu) w $t3
watchpoint $t3 set success
(nemu) info w
NO   EXPR                                DECIMAL      HEX
3    $t3                                 0            0x0
2    $t2                                 0            0x0
1    $t1                                 0            0x0
0    $t0                                 0            0x0
```

4.表达式求值 p 指令

```
(nemu) p (1+2)*(2*3)
decimal: 18
hex    : 0x12
(nemu) p --9
decimal: 9
hex    : 0x9
(nemu) p (2*5)+)
expression error
```

四、必答题

1：

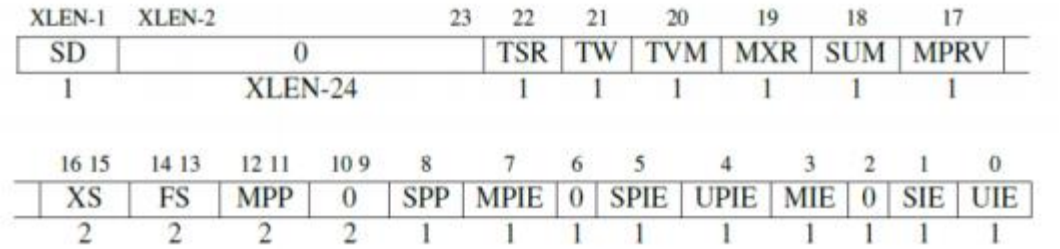
我选择的 ISA 是 RISC-V32

2：

调试需要花费的时间是 500*90%*30*20=270000s=4500min 简易调试器可以节约的时间为 500*90%*20*20=180000s=3000min

3：

- riscv32 有 R、I、S、B、U、J 6 种指令格式。
- LUI 指令是将 20 位常量加载到寄存器的高 20 位。
- Mstatus 结构如下图：



4：

- 使用命令 find . -name "*[.h|.c]" |xargs cat|wc -l 得出来的行数是 5467。
- 使用命令 find -name "*[.h|.c]" | xargs cat|grep -v ^$|wc -l 得到 4497

5:

- Wall 的作用是打开 gcc 所有警告。
- Werror 的作用是要求 gcc 将所有警告当成错误处理。

PA2：简单复杂的机器

一、实验进度

PA2 的内容全部已完成。

二、任务描述

P2.1 --- 在 NEMU 中运行第一个 C 程序

Step 1：在 all-instr.h 中定义相关指令函数
Step 2：在 exec.c 的 opcode_table 中添加新指令
Step 3：在 decode.c 中实现相应的译码辅助函数
Step 4: 在 rtl.h 中修改完善 rtl 指令
Step 5: 在 compute.c 和 control.c 等文件中使用 rtl 指令实现正确的执行辅助函数
完成上述步骤后，编译运行可以完成 P2.1。

P2.2 --- 实现更多的指令

Step 1: 过程类似于 PA2.1，只不过要实现更多的指令。更多的指令意味着更容易出错，因此首先完成 diff-test。
Step 2: 完成 diff-test 后，一边使用 make 命令测试一边查阅手册了添加新指令。过程像 PA2.1。只不过添加了 diff-test。
Step 3： 测试测试文件 hello-str 和 string 中要使用库函数 sprintf、strcmp、strcat、strcpy 和 memset。这些函数要在 nexus-am/libs/klib/src/stdio.c 以及 string.c 中编写。
Step 4: 在 nemu/目录下：
执行一键回归测试： bash runall.sh ISA=riscv32
可以测试所有程序。

P2.3 --- 运行打字小游戏

要完成串口、时钟、键盘、VGA 四个输入输出设备程序的编写。
Step 1：串口 在 trm.c 中已经实现，我们还需要编写 printf 函数以便程序运行。
Step 2: 时钟的功能需要在 nemu-timer.c 中完成，在启动 init 时通过 RTC_ADDR 获取启动时间，运行时钟功能时通过地址 RTC_ADDR 获 取当前时间，将 uptime->hi 设为 0，uptime->lo 设为当前时间与启动时间的差值， 即可完成时钟功能。
Step 3: 键盘的功能需要在 nemu-input.c 中完善，通过地址 KBD_ADDR 获取键盘按键信息，放入 kbd->keycode 中，将按键信息与 KEYDOWN_MASK 相与，如果值为 1 就说明是按下，值为 0 说明是松开。

Step 4: 需要实现的是 VGA 设备，在 nemu-video.c 中完善相关的功能，VGA 设备需要将 pixels 中的像素信息写入 VGA 对应的地址空间中，就能正确显示图像

三、测试结果

1. 一键回归测试

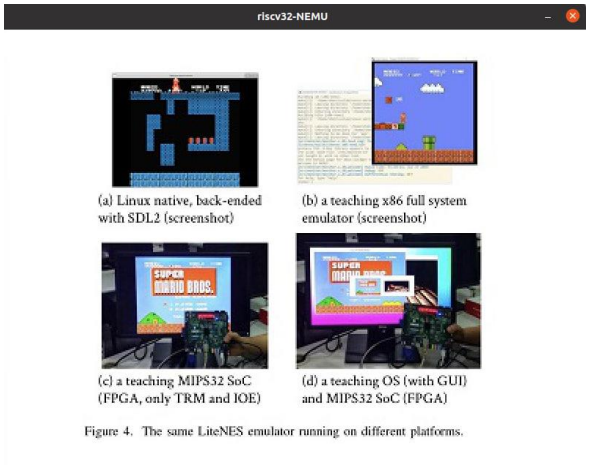
```
[  add-longlong] PASS!
[           add] PASS!
[           bit] PASS!
[   bubble-sort] PASS!
[           div] PASS!
[         dummy] PASS!
[          fact] PASS!
[           fib] PASS!
[      goldbach] PASS!
[     hello-str] PASS!
[       if-else] PASS!
[     leap-year] PASS!
[    load-store] PASS!
[    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!
```

2. microbench 测试:

```
======= Running MicroBench [input *ref*] =======
[qsort] Quick sort: * Passed.
  min time: 1181 ms [433]
[queen] Queen placement: * Passed.
  min time: 1643 ms [286]
[bf] Brainf**k interpreter: * Passed.
  min time: 9380 ms [252]
[fib] Fibonacci number: * Passed.
  min time: 19165 ms [147]
[sieve] Eratosthenes sieve: * Passed.
  min time: 19648 ms [200]
[15pz] A* 15-puzzle search: * Passed.
  min time: 2534 ms [177]
[dinic] Dinic's maxflow algorithm: * Passed.
  min time: 3301 ms [329]
[lzip] Lzip compression: * Passed.
  min time: 2829 ms [268]
[ssort] Suffix sort: * Passed.
  min time: 1159 ms [388]
[md5] MD5 digest: * Passed.
  min time: 16817 ms [102]
================================================
MicroBench PASS        258 Marks
                 vs. 100000 Marks (i7-7700K @ 4.20GHz)
Total time: 88926 ms
nemu: HIT GOOD TRAP at pc = 0x801041e0
```

3. 打字小游戏测试：

4. Slider 测试



5. Litenes 测试

四、必答题

1：

一条指令在 nemu 中执行的过程： 首先根据 PC 值通过 instr_fetch 函数取指令，从选出的指令中选取其 opcode， 在 opcode_table 中进行索引，找到该指令对应的译码辅助函数和执行辅助函数，随后通过译码辅助函数进行译码,将译码得到的相关信息保存在 decinfo 中,接 着通过执行辅助函数执行指令,执行辅助函数通过 rtl 指令对译码得到的信息进 行相关操作，计算、读取、保存等等，最后通过 update_pc 函数更新 PC 值。

2：

以 rtl_li 函数为例单独去掉 static 和单独去掉 inline 进行重新编译，都不会 报错，但将两者同时去掉时，就会报错。原因是都去掉时，在另一个文件中也有 对 rtl_li 的定义，会出现重复定义的错误，而具有 static 关键字时，函数会被限制 在本文件内，不会出现重复定义的错误，具有 inline 关键字时，函数在预编译时 就会展开，不会出现重复定义的错误，但如果将两者同时去掉，就会出现重复定 义的错误。

3：

添加后，使用 grep 命令查看，共有 81 个 dummy 实体；继续添加后，重 新编译运行，使用 grep 命令，共有 82 个 dummy 实体；修改代码后，重新编译 报错，原因是两个都初始化后，会产生两个强符号，导致错误。

4：

敲入 make 后，会将 makefile 文件中第一个目标文件作为最终的目标文件， 如果文件不存在，或是文件所依赖的后面的.o 文件的修改时间比这个文件晚，就 会重新编译；如果目标文件依赖的.o 文件也不存在，就根据这个.o 文件的生成 规则生成，然后生成上一层.o 文件，中间某一步出错就会直接报错

PA3：穿越时空的旅程

一、实验进度

PA3 的内容全部已完成。

二、任务描述

P3.1 --- 实现自陷操作_yield()

Step 1：首先，实现 csrrs，csrrw，ecall 和 sret 指令

Step 2：在 intr.c 文件中编写 raise_intr()

Step 3：根据 trap.S 重构 _Context 成员结构体

Step 4：实现事件分发的__am_irq_handle（）函数

Step 5：在 do_event()函数中识别自陷事件 EVENT_YIELD

P3.2 --- 实现用户程序的加载和系统调用

Step 1：实现 loader() 函数，使用 ramdisk_read() 函数实现

Step 2：完成系统调用的实现，然后在 do_event()中添加 do_syscall 的调用

Step 3：根据 nanos.c 中的 ARGS_ARRAY，在 riscv-nemu.h 中实现 GPR 的宏

Step 4：添加 SYS_yield, SYS_read, SYS_write, SYS_brk 系统调用

Step 5：完成_sbrk() 实现堆区管理

P3.3 --- 运行仙剑奇侠

Step 1：要实现 fs_open，fs_read，fs_close，fs_Iseek 函数

Step 2：使用 fs 函数替换 loader（）函数中的 ramdisk_read()

Step 3：实现虚拟文件系统 VFS

Step 5：实现 serial_write()，init_fs()，fb_write()，fbsync_write()，init_deivce，dispinfo_read() 等函数

三、测试结果

1. 运行 dummy 测试：

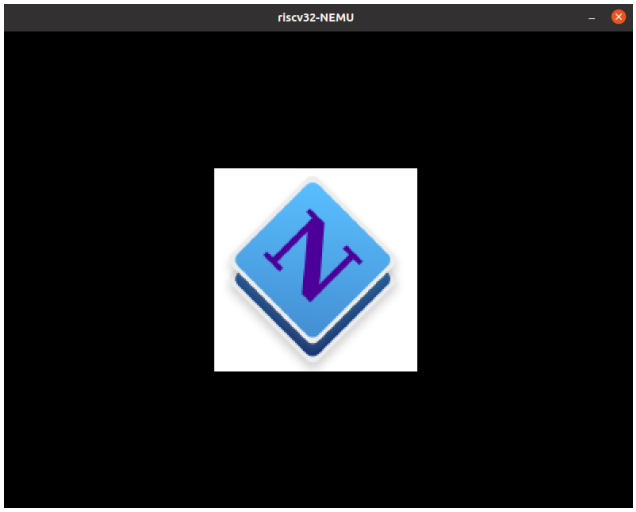
2. 运行 Hello 测试:

```
Hello World from Navy-apps for the 20845th time!
Hello World from Navy-apps for the 20846th time!
Hello World from Navy-apps for the 20847th time!
Hello World from Navy-apps for the 20848th time!
Hello World from Navy-apps for the 20849th time!
Hello World from Navy-apps for the 20850th time!
Hello World from Navy-apps for the 20851th time!
Hello World from Navy-apps for the 20852th time!
Hello World from Navy-apps for the 20853th time!
Hello World from Navy-apps for the 20854th time!
Hello World from Navy-apps for the 20855th time!
Hello World from Navy-apps for the 20856th time!
Hello World from Navy-apps for the 20857th time!
Hello World from Navy-apps for the 20858th time!
Hello World from Navy-apps for the 20859th time!
Hello World from Navy-apps for the 20860th time!
```

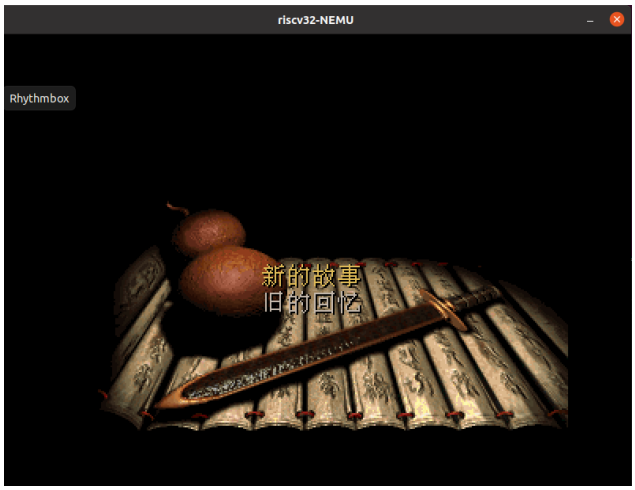
3. 运行 text 测试：

```
Welcome to riscv32-NEMU!
For help, type "help"
[/home/hust/ics2019/nanos-lite/src/main.c,14,main] 'Hello World!' from Nanos-lit
e
[/home/hust/ics2019/nanos-lite/src/main.c,15,main] Build time: 13:06:07, Oct  4
2023
[/home/hust/ics2019/nanos-lite/src/ramdisk.c,28,init_ramdisk] ramdisk info: star
t = , end = , size = -2146426564 bytes
[/home/hust/ics2019/nanos-lite/src/device.c,35,init_device] Initializing devices
...
[/home/hust/ics2019/nanos-lite/src/irq.c,20,init_irq] Initializing interrupt/exc
eption handler...
[/home/hust/ics2019/nanos-lite/src/proc.c,27,init_proc] Initializing processes..
.
[/home/hust/ics2019/nanos-lite/src/loader.c,35,naive_uload] Jump to entry = -0x7
cffd1ac
PASS!!!
nemu: HIT GOOD TRAP at pc = 0x80100bdc
```

4. 运行 bmptest 测试：



5. 运行仙剑奇侠传


riscv32-NEMU
Rhythmbox
新的故事
旧的回忆


riscv32-NEMU
李大娘：
李逍遥！你皮痒啊？
敢说老娘是什么鬼婆！


进度一 4
进度二 4
进度三 4
进度四 4
进度五 4
新的故事
旧的回忆

四、必答题

1：

函数__am_irq_handle 中，上下文结构体 c 是函数的参数，该 函数是 trap.S 中通过 call 指令调用的。c 指向的上下文结构_Context 包含 32 个通用寄存器，scause，sstatus，sepc，在 trap.S 中他们会被先后压栈，成员赋值是在执行自陷操作时，通过 trap.S 的汇编程 序运行进行赋值的。riscv-nemu.h 定义了结构体_Context，同时确定了成员的顺序，trap.S 通过汇编程序 对上下文结构体进 行赋值，讲义则说明了流程，方便我们理解，实现的指令 csrrw，csrrs，ecall，sret 实现了异常调用，异常返回等，使得自陷操作能够正常执行。

2：

Nanos-lite 调用中断，操纵 AM 发起自陷指令的汇编代码，随后保存上下 文，转入 CPU 自陷指令的内存区域，执行完毕后，恢复上下文，返回运行时环境。

3：

hello 程序在磁盘上，hello.c 被编译成 ELF 文件后，位于 ramdisk 中。当用户运行该程序时，通过 naive_uload 函数读入指定的内存并放在正确的位置。加载完成后，操作系统从其 ELF 信息中获取到程序入口地址，通过上下文切换从入口地址处继续执行，hello 程序便获取到 CPU 的控制权开始执行指令。

对于字符串在终端的显示，首先调用 printf 等库函数，然后通过 SYS_write 系统调用来输出字符，系统调用通过调用外设的驱动程序最终将内容在外设中表现出来，程序执行完毕后操作系统会回收其内存空间。

4：

操作系统通过库函数读出画面的像素信息，画面通过 VGA 输出，VGA 被抽象成设备文件，fs_wrtie 函数在一步步执行中调用了 draw_rect 函数， draw_rect 函数把像素信息写入到 VGA 对应的地址空间中，最后通过 update_screen 函数将画面显示在屏幕上。