华中科技大学

2021

计算机系统结构　课程实验报告

题　　目：	系统能力综合训练报告
专　　业：	计算机科学与技术
班　　级：	CS1807 班
学　　号：	U201814683
姓　　名：	荣烁
电　　话：	15153089759
邮　　件：	905869875@qq.com
完成日期：	2022-1-11

计算机科学与技术学院

目 录

1　课程实验概述

1.1 课设目的

理解程序是如何在计算机上运行的。在提供的代码框架中，实现一个简化的基于 riscv32 架构的模拟器 NEMU，最终能在 NEMU 上运行游戏“仙剑奇侠传”，来让学生探究“程序在计算机上运行”的基本原理。

实验内容包括五个部分：

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

2　实验方案设计

2.1 PA1

2.1.1 课程实验概述

PA1 要求实现实验中用到的基础设施：简易调试器，要求在原先已经实现的帮助、继续运行和退出三个命令的基础上，继续实现其他命令。包括单步执行、打印程序状态、表达式求值、扫描内存、设置监视点以及删除监视点等供能。

PA1 分为三个阶段。第一个阶段要求实现单步执行，打印寄存器状态，扫描内存。第二个阶段要求实现表达式求值的功能。第三个阶段要求实现监视点相关的功能。

2.1.2 实验方案设计

PA1.1 方案设计：

需要在已经提供的 ui.c 文件的基础上增添更多的功能。首先实现单步执行，打印寄存器状态和扫描内存三个函数。三者分别定义为 cmd_si, cmd_info 和 cmd_x 函数。

cmd_si 通过调用已经实现的 cpu_exec 来执行输入参数指定的单步执行次数。

cmd_info 通过调用提供的 isa_reg_display 函数实现寄存器状态的打印。

cmd_x 根据输入参数，使用 vaddr_read 函数读取相应地址的数据并且打印。

PA1.2 方案设计：

需要在 ui.c 文件中实现表达式求值的功能。功能的具体实现则在 expr.c 文件中。

首先需要完成 make_token 函数功能的拓展，能够识别包括加减乘除、左右括号、十六进制数字、十进制数字、寄存器、逻辑运算符号等 token。

实现 eval 函数。eval 函数的整体逻辑设计已经给出。如果输入是单个 token 则对其进行求值处理，将原本是字符串的数据转换为数字来作为结果返回。如果不是单个 token 判断其是否是被一对括号包括的，如果是则对去除这层括号后的 tokens 递归调用 eval 函数。如果不是被一对括号包括的，则找到这些 tokens 中的主运算符，对 tokens 分裂处理，将两个结果根据主运算符计算后的结果作为返回值。

完成 eval 函数后，将 make_token 与 eval 两个函数整合到 expr 函数中就完成了表达式求值的功能。只需要在 ui.c 文件中的 cmd_p 函数中调用 expr 函数即可。

PA1.3 方案设计：

要求实现监视点相关的功能。包括设置监视点、删除监视点和监视监视点的功能。

首先修改了 watchpoint 结构的定义，增添了两个变量，分别用来保存监视的表达式和监测值的旧值。

随后在 watchpoint.c 文件中实现了 new_wp, free_wp, print_wp, free_wp_byNO

等函数。new_wp 是从 watchpoint 池中获得一个空闲的 watchpoint。free_wp 是将原本在使用中的 watchpoint 放回 watchpoint 池中。print_wp 是用来打印当前设置了的 watchpoint 信息。free_wp_byNO 是根据输入参数 watchpoint 的序号来从 watchpoint 池中删除对应的 watchpoint 的。

完成了上述的函数后，只需要在 ui.c 文件中添加 cmd_w 与 cmd_d 函数和在 cpu_exec.c 文件中实现在每执行一条指令后都要根据监视点的状态是否改变来决定是否暂停执行的逻辑。这样就分别实现了设置监视点、删除监视点和监视监视点的功能。

2.1.3 问题解答

1.riscv32
2.270000s 180000s
3.阅读 RISC-V 手册
Page24

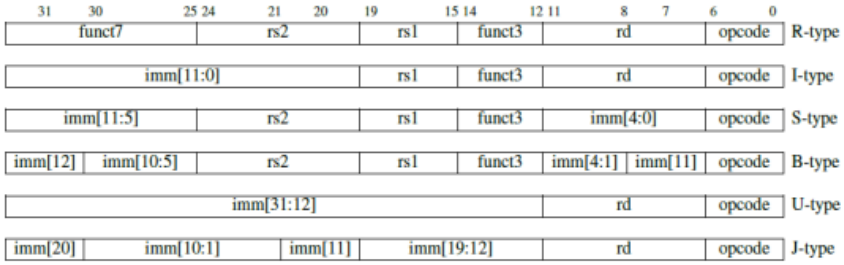
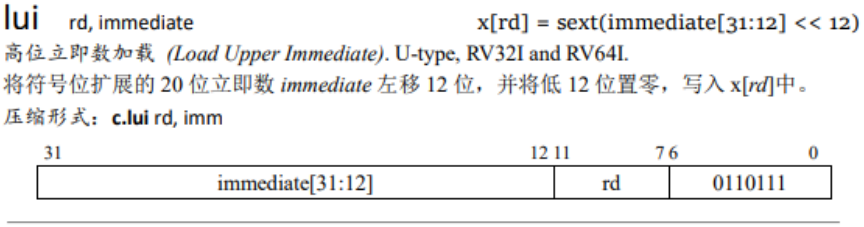


图 2.2：RISC-V 指令格式。我们用生成的立即数值中的位置（而不是通常的指令立即数域中的位置）(imm[*x*])标记每个立即数子域。第十章解释了控制状态寄存器指令使用 I 型格式的稍微不同的做法。（本图基于 Waterman 和 Asanovi'c 2017 的图 2.2）。

Page 151



Page101

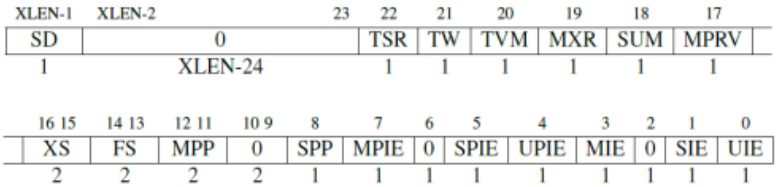


图 10.4：mstatus 控制状态寄存器。在仅有机器模式且没有 F 和 V 扩展的简单处理中，有效的域只有全局中断使能、MIE 和 MPIE（它在异常发生后保存 MIE 的旧值）。RV32 的 XLEN 时 32，RV64 是 40。（来自[Waterman and Asanovic 2017]中的表 3.6；有关其他域的说明请参见该文档的第 3.1 节。）

4.

find . -name ”*[.h|.c]“ |xargs cat|wc -l 5476 行
PA1 之前共 5009 行
5.
-Wall 的作用是打开 gcc 所有警告
-Werror 的作用是要求所有警告当成错误处理

2.1.4 心得体会

主要难点就是在于对解引用和乘法的区分上。如何区分出来解引用主要根据*这个符号前后的 token 来判别。因为乘法要求前后两个 token 都是表达式，而解引用则可以在前面是是加减乘除或者什么都没有，也可以是另一个解引用。只要在调用 eval 前，将两者根据上述规则区分为两种 token 就可以了。

2.2 PA2

2.2.1 课程实验概述

PA2.1 要求在 NEMU 中运行第一个 C 程序 dummy，主要是通过查看汇编代码，找出未实现的指令，编写对应的译码和执行函数。

PA2.2 要求实现更多的指令，在 NEMU 中运行所有的 cputest。只要查看所有的测试文件的汇编代码，找到其中没有实现的指令，并且编写译码和执行函数即可。

PA2.3 运行打字小游戏和其他测试。需要根据文档编写设备功能，并且完成相关的输入输出函数。

2.2.2 实验方案设计

PA2.1 方案设计：

要生成可以在 nemu 上运行的可执行程序，首先要配置相应的编译环境。根据文档中的教程下载相应的交叉编译环境，然后对提供的 dummy.c 程序进行编译，然后尝试在 nemu 上运行。因为还没有在 nemu 中实现执行程序中用到的指令，所以会提示 abort。这时候找到 build 文件夹中生成的反汇编文件，在 nemu 中实现所有的未实现的指令即可。

实现新的指令，需要在首先在 decode.c 文件中，添加相应的译码辅助函数。这里因为的译码辅助函数包括对 R、B、U、J、ld、st 的译码辅助函数。all-instr.h 头文件中添加执行辅助函数的声明，随后在 exec.c 中的 opcode_table 中根据 riscv 手册中指令的 opcode 的 6 到 2 位为索引，在对应位置添加译码辅助函数对执行辅助函数的映射。因为是根据 opcode 为索引，而有一组指令会有相同的 opcode，所以不能建立译码辅助函数到单个指令的映射。这里针对在 opcode_table 中不能实现译码辅助函数到单个指令的映射的一组指令的执行辅助函数，在其中添加了根据 funct3 字段映射到对应单个指令的执行辅助函数的映射。在实现执行辅助函数时，用到了 rtl 库中提供的 rtl 指令。一些 rtl 指令需要自己实现，大部分已经实现好了。完成上述步骤后，重新编译运行即可。

PA2.2 方案设计：

大致步骤同 PA2 的第一阶段。只要每个测试都能通过，就完成了第二阶段的设计。在通过 hello-str 于 string 两个测试程序时，需要使用到库函数，例如 strcmp、strcat 等函数。这些函数没有实现，需要自己实现。实现这些函数后，就可以通过 hello-str 与 string 两个测试。最后使用一键回归测试可以测试所有的程序。

PA2.3 方案设计：

根据文档要求，需要实现串口、时钟、键盘、VGA 四个输入输出设备程序。串口部分程序已经实现，只要完善 printf 函数即可。完成后需要通过 hello 测试程序。对于时钟，只要通过设定 uptime 的两个成员变量 hi 与 lo 即可。完成后需要通过 tct 程序测试。对于键盘部分的实现，需要通过得到当前键盘发送到寄存器的码，判断这是某键的断码还是通码，并且根据这个设定 keydown 与 keycode 两个值。keydown 表示当前按键是按下还是释放，keycode 则应该是按键的断码。因为约定在 AM 中，通码是断码按位或 0x8000，也就是断码的最高位一定是 1。根据这个我们只要通过按位与，判断最高位是 0 还是 1 就可以判断是当前的码是按键的通码还是断码。根据这个就可以设定 keydown 与 keycode。完成后需要通过 readkey 测试。最后是 VGA 的实现，需要首先在 nemu 实现对同步屏幕的支持，将 vga_io_handler 函数补充，逻辑是在收到写入信号时，更新屏幕。之后完善 am 中的 video_read 和 video_write 函数即可。写入的逻辑就是将传入参数的像素数据写入 VGA 对应的地址空间中。

2.2.3 问题解答

1.请整理一条指令在 NEMU 中的执行过程

在 nemu 中执行一条指令时，isa_exec()做的第一件事就是取指令。通过 instr_fetch()，实现上是进行一次内存访问，就能将执行的指令取出。之后就要进行指令的译码、执行的阶段。这里通过指令的 opcode 去 opcode_table 上找到指令对应的译码辅助函数和执行辅助函数并进行指令的译码和执行。执行结束后最后再更新 PC 值。这样就完成一条指令的执行。

2.去掉 static 不会发生错误。去掉 inline 后，会在链接时发生错误。LD 在链接不用的对象文件时，会因为同一个符号在不同的对象文件中被多次定义而不能进行链接而出错。加上 inline 后因为函数会在预编译时就展开而不会发生错误。

3.

(1)添加后使用 am -a riscv32-nemu | grep dummy 指令查看，发现共有 34 个。

(2)发现共有 34 个。头文件展开后 dummy 声明了两次被覆盖了

(3)发生报错，因为进行赋值之后声明就变成了定义，就产生了重定义的问题。

4.Makefile 的工作方式基本是这样的：首先默认的 ISA 是 x86，如果定义了具体的 ISA，则会使用具体的 ISA。之后会根据 ISA 会确定不同的 include 的目录。这里是./include 和./src/isa/riscv32/include 两个目录。指定的编译目标文件夹是 build。使用的编译器与链接器都为 gcc。后续会在 src/和 src/isa/riscv32/两个目录下读取所有需要编译的.c 后缀的文件都编译为.o 目标文件，之后进行链接。最

后会执行 git commit 作为结束。

2.2.4 心得体会

PA2 的主要问题就是在对于指令的实现过程中，如果出现了问题，很难将问题定位到具体的某个指令上。文档先是让你实现指令，并且通过每个测试文件。在这之后才提及 diff-test。导致在实现指令的时候，寻找错误很多时候都是在用 PA1 中实现的调试工具单步调试，并且比照反汇编代码，一步步的寻找错误。如果能先实现 diff-test 就能更方便的定位到出现问题的指令。

在 native 上测试传入参数时，键入 make ARCH=native mainargs=H run 后，显示程序 core dumped。最后通过 gdb 单步调试定位到 printf 在实现时没有实现针对%c 的功能。后续实验中也遇到了类似的问题。因为在前面的实验中文档没有提及这部分的实现，所以只实现了%d 与%s 的功能。希望文档中可以提及这部分的实现，要求事先实现这部分的功能。

在实现 VGA 部分的功能后，运行 display test 测试的时候，因为没有关掉 debug 和 diff-test 导致屏幕一致都是显示黑色。在从头到尾检查代码之后，问题依然没有得到解决。后来因为没有办法就尝试性的关掉了这两个选项后终于能正确显示。我认为文档应该在测试前提醒这个问题，减少时间上不必要的浪费。

2.3 PA3

2.3.1 课程实验概述

PA3.1 要求实现自陷操作_yield()及其过程
PA3.2 要求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

2.3.2 实验方案设计

PA3.1 要求实现自陷操作。从全局来看，首先要在原来的 32 个寄存器的基础上加入 sepc、scause、sstatus、stvec 四个寄存器。之后在此基础上仿照 PA2 的过程，实现四个指令 csrrs、csrrw、ecall 以及 sret。编写 intr.c 文件中的 raise_intr 函数，逻辑是从 stvec 寄存器中取出异常入口地址，跳转到异常入口地址，将 PC 与异常号放到 sepc 与 status 寄存器中保存。之后根据实验文档顺序，实现保存上下文。重构_Context 的结构，之后在_am_irq_handle 中根据异常号打包自陷事件，交给 do_event 处理。在 do_event 中加入对_EVENT_YIELD 事件的处理。这里是设计成输出一句话，"yield!"。

PA3.2 首先要求实现用户程序的加载。为了实现 loader 函数，这里用到了实验提供的 API 来对 ramdisk 进行读操作。根据将每一个加载的 segment 的 Offset, VirtAddr, FileSiz 和 MemSiz 这些参数读出，之后从文件的 Offset 开始，读入到内存的[VirtAdder, VirtAddr + MemSiz]这一连续区间，然后将[VirtAddr + FileSiz, VirtAddr + MemSiz)对应的物理空间清零。这样就能实现现阶段的用户程序的加

载。之后要实现用户程序的系统调用，之后只要根据文档实现 yield、wirite、brk 三个系统调用即可。这里的 write 是使用_putc()将数据输出到串口的。

PA3.3 首先要实现一个简易文件系统。这个文件系统的文件数目、大小是固定的，而且没有目录结构，而且不能创建新文件。首先实现 fs_open()、fs_read()、fs_close()三个功能，使用这三个 API 重新实现上一阶段的 loader 函数。之后实现 fs_write 和 fs_lseek()，并将将其作为系统调用添加到程序中，就可以运行测试程序/bin/text。在实现文件系统的基础上，实验已经实现了实现虚拟文件系统 VFS。在 VFS 的基础上，根据实验文档实现对/dev/events、/dev/fb、/dev/fbsync、/proc/dispinfo 几个虚拟文件使用的过程细节即可。运行仙剑奇侠传进行测试。

2.3.3 问题解答

1._am_irq_handle 函数调用了 trap.S 文件。可以看到在 jal 之前有多个压栈操作。按照顺序分别是 32 个寄存器，scause、sstatus 以及 sepc。按照这个顺序去重新组织 Context 结构体即可。

2.yield 函数使用了内联汇编，来将寄存器 a7 设置为-1，之后执行了 ecall 指令。ecall 指令中会调用 raise_intr 函数，即保存 sepc 寄存器，将中断号保存到 scause 寄存器，从 stvec 获得中断入口地址进行跳转。跳转后会执行到__am_asm_trap 的部分，其根据 c->cause 来进行事件的分发并 do_event 函数来处理事件。结束后会通过 sret 函数进行状态恢复，返回到触发自陷的下一个指令。

3.hello 程序一开始会保存在 ramdisk 之中，经过 loader 函数，将程序从 ramdisk 中读出。打印字符串时，因为 hello 调用了 printf，其会先使用系统调用 brk 申请缓存，失败了就一个字符一个字符的输出。成功了就将字符串整个输出。输出则是调用了 write 系统调用。

2.3.4 心得体会

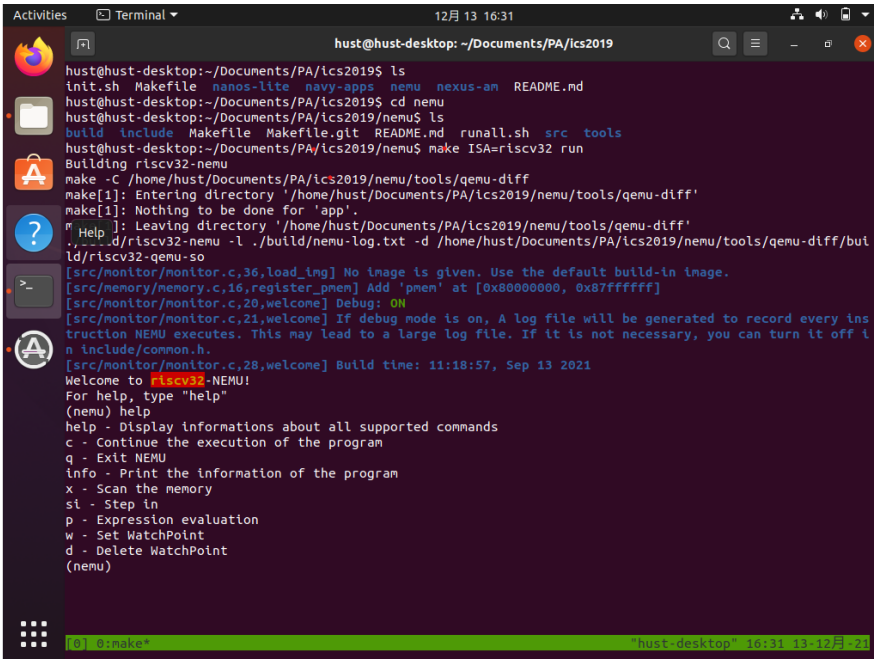
在第一阶段的实验中，因为要实现 csrw 指令而去查 riscv 中文手册。手册中写到其等同于 csrrs。实现之后发现程序报错 csrw 指令没有被实现。在找了大概两个小时的问题后发现是中文手册有问题。这里的 csrw 应该是等同于 csrrw 指令，而 csrrw 指令没有实现所以出现了问题。

在指定目录使用 make 指令后发现出现有 ISA C99 的宏定义错误。经过反复试错修改后发现是 make 的 ARCH 参数输成了 ISA 参数。

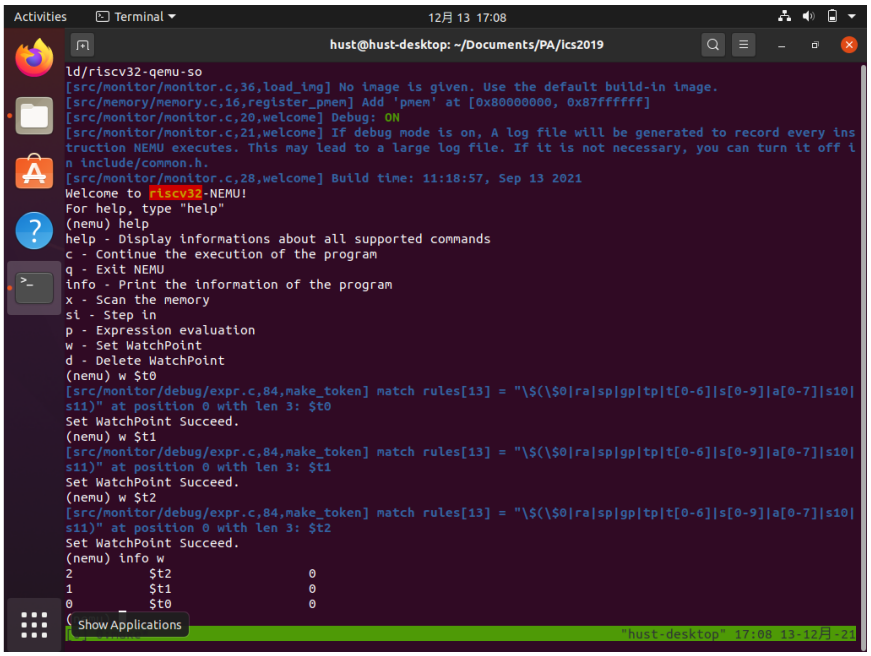
3　实验结果与结果分析

3.1 PA1

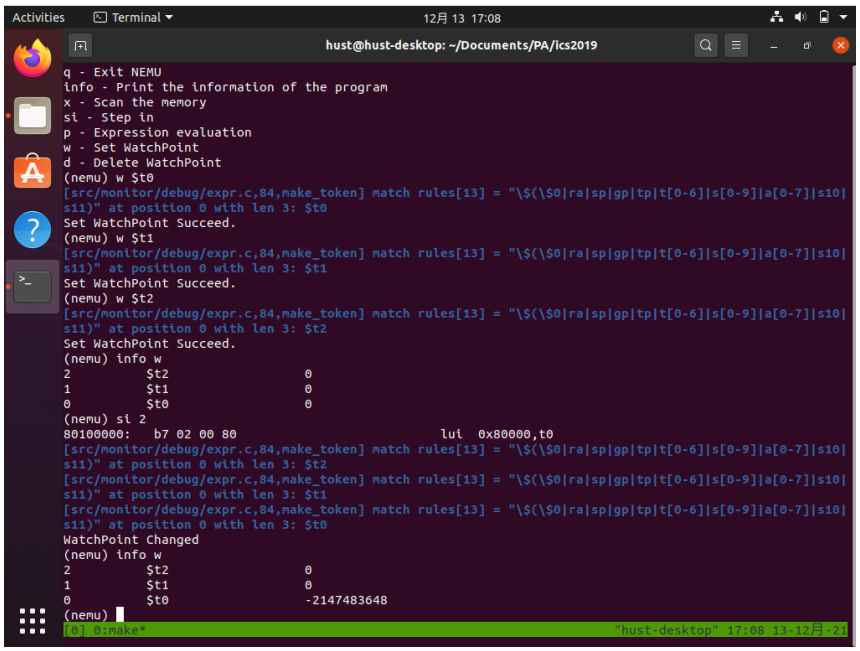
进入简易调试器后，输入 help 指令显示各个指令对应的信息。



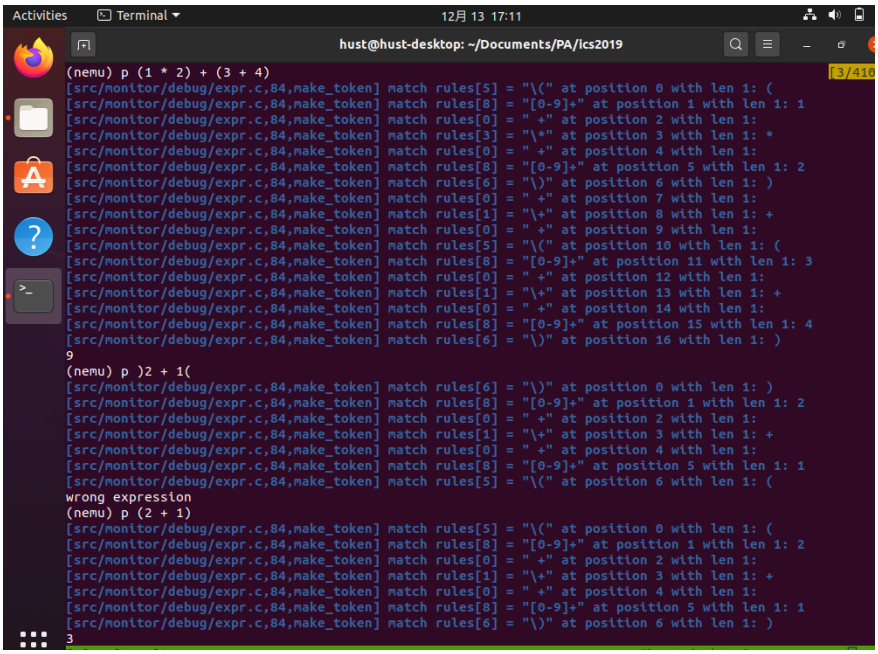
设置三个监视点后，使用 info 指令打印监视点信息。



使用 si 指令执行两步，之后程序因为监视点$t0 发生改变，程序暂停，使用 info 指令打印显示监视点的值，发现 t0 寄存器的值发生了变化。



对 p 指令进行测试。使用三个表达式进行测试。



3.2 PA2

一键回归测试结果。

```
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!
```

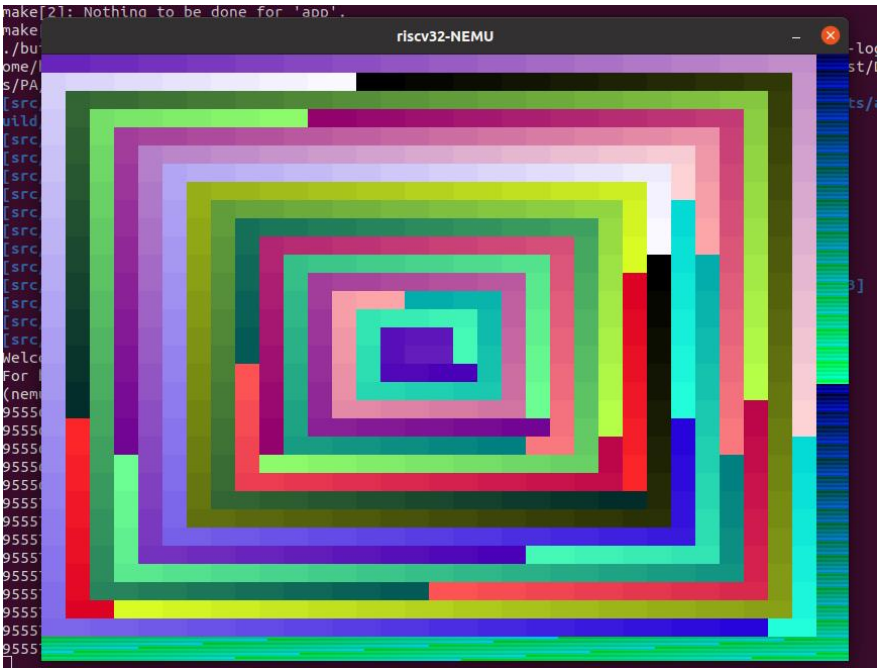
microbench 测试结果

```
Welcome to riscv32-NEMU!
For help, type "help"
(nemu) c
Empty mainargs. Use "ref" by default
======= Running MicroBench [input *ref*] =======
[qsort] Quick sort: * Passed.
  min time: 830 ms [616]
[queen] Queen placement: * Passed.
  min time: 1226 ms [383]
[bf] Brainf**k interpreter: * Passed.
  min time: 6704 ms [353]
^[[B^[[B^[[B^[[B^[[B^[[B^[[B^[[B^[[B^[[B^[[B^[[B^[[A^[[A^[[A^[[A^[[A^[[A^[[A^[[A^[[A^[[A^[[A^[[A^[[A^[[A
^[[A^[[A[fib] Fibonacci number: * Passed.
  min time: 13362 ms [211]
[sieve] Eratosthenes sieve: * Passed.
  min time: 13986 ms [281]
[15pz] A* 15-puzzle search: * Passed.
  min time: 1836 ms [244]
[dinic] Dinic's maxflow algorithm: * Passed.
  min time: 2366 ms [459]
[lzip] Lzip compression: * Passed.
  min time: 1997 ms [380]
[ssort] Suffix sort: * Passed.
  min time: 845 ms [533]
[md5] MD5 digest: * Passed.
  min time: 12337 ms [139]
==================================================
MicroBench PASS        359 Marks
                   vs. 100000 Marks (i7-7700K @ 4.20GHz)
Total time: 64001 ms
nemu: HIT GOOD TRAP at pc = 0x801041f4
```

readkey test 测试结果:

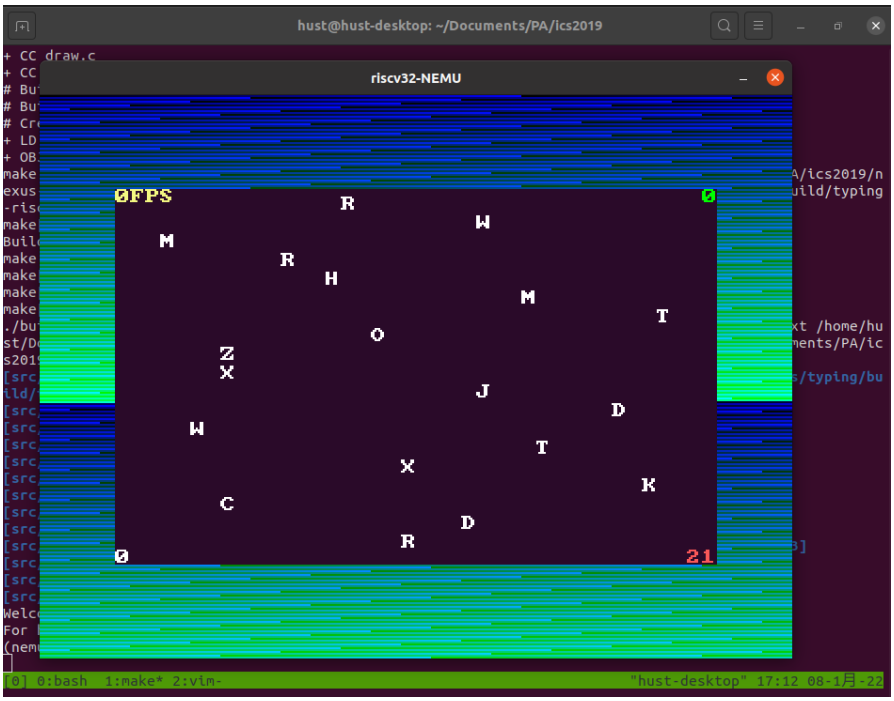
```
Welcome to riscv32-NEMU!
For help, type "help"
(nemu) c
Try to press any key...
Get key: 44 S down
Get key: 45 D down
Get key: 44 S up
Get key: 46 F down
Get key: 45 D up
Get key: 46 F up
Get key: 30 W down
Get key: 30 W up
Get key: 31 E down
Get key: 32 R down
Get key: 31 E up
Get key: 32 R up
Get key: 43 A down
Get key: 45 D down
Get key: 43 A up
Get key: 46 F down
Get key: 45 D up
Get key: 43 A down
Get key: 45 D down
Get key: 46 F up
Get key: 43 A up
Get key: 45 D up
```

display test 测试结果：



slider 测试结果：

typing 测试结果：



3.3 PA3

运行仙剑奇侠传测试结果。


李逍遥：
一大早就有客人上门啦？


醉道士：
哎呀！没酒喝我一步也走不动
你就行行好吧．．．

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.