PA1实验报告

一、实验进度

1. 阅读了NEMU的框架代码，对整体框架有了初步认识。
2. 完成基础设施内容。
3. 完成表达式求值的内容，包括负数、寄存器取值和指针解引用。
4. 实现监视点（当监视点的值变化时会暂停程序）。

二、必答题

1. 程序是个状态机

(0, x, x) -> (1, 0, x) -> (2, 0, 0) -> (3, 0, 1) -> (4, 1, 1) -> (2, 1, 1) -> (3, 1, 2) -> (4, 3, 2) -> ... -> (3, 4950, 100) -> (4, 5050, 100) -> (5, 5050, 100) -> (5, 5050, 100) -> (5, 5050, 100) -> ...

2. 理解基础设施

$30 * 20 * 500 * 90\% \div 60 = 4500(min) = 75h$30 * 20 * 500 * 90% ÷ 60 = 4500(min) = 75h

节省的时间: $20 * 20 * 500 * 90\% \div 60 = 3000(min) = 50h$

20 * 20 * 500 * 90% ÷ 60 = 3000(min) = 50h

3. RTFM

riscv32有哪几种指令格式? => riscv-spec-1 page: 16

LUI指令的行为是什么? => riscv-spec-1 page: 19

mstatus寄存器的结构是怎么样的? => riscv-privileged page: 20-28

4. shell命令

分别使用命令：

find -name "*.[c|h]" | *xargs wc -l 统计总的行数*

find -name ".[c|h]" | xargs grep -v ^$| wc -l 统计总的行数（去除空行）

PA1: 23892 total; PA1除去空行：20729

PA0 :23469 total; PA0除去空行：20339

PA1写了390行代码。

5. RTFM

-Wall 该选项能发现程序中一系列的常见错误警告
-Werror 该选项要求GCC将所有的警告当成错误进行处理
使用这两个选项可以把我们写的代码中不合理的地方输出，把warning当成error处理导致编译出错，可以避免潜在的风险并且使我们编写的代码更加规范。

三、实验心得与体会

1.

从来没有阅读过这么大的工程文件，导致我在一开始RTFSC的时候完全没有头绪，只知道从main函数开始看，但是由于函数过多，而且调用太复杂（很多都是跨文件的），里面还掺杂着很多宏定义，读代码的时候感觉眼花缭乱，不知所云（这怎么和我大一写的c代码完全不一样），这是我只能选择相信STFW和RTFM的力量，遇到不懂的build-in函数、不了解的宏定义就去网上搜索，并且配合讲义的引导，我终于大概知道整个nemu的框架大概是什么样的，我也从中收获了一些阅读工程文件的经验：不能急躁，必须一步一步来，弄清楚每个函数、模块的功能和它们之间的联系，从而理解整个程序是怎么控制的。

2.

实现打印寄存器信息的时候遇到一个问题， 就是调用vaddr_read函数的时候，如果第二个参数传递的是4, 那么返回的内存信息是反的， 比如连续的四个字节存储的是0xb7020080， 这时候会输出0x800002b7，经过查询，才发现这其实和ISA有关，RISIC-V是小端存储的，所以vaddr_read函数会按照小端存储的方式来读取数据。

3.

实现表达式求值时，因为算法框架讲义已经给出，主要难点在于如何确定实现是否正确，一开始我写了一个简单的测试函数，输入一个表达式的时候，顺序输出识别的token，然后输出值，但是这种纯手工验证效率太低，于是尝试实现讲义中讲的随即测试，但是我踩了一个坑，因为每生成一个表达式的时候，我没有清空buf数组，导致后续生成的表达式都是错的，经过一个晚上的调试才发现这个低级错误。晕！！！

4.

实现监视点主要是一些链表操作，回去复习了一些大一学的知识很快就写好了，但是在头文件引用的时候出现了困难，因为实现的监视点的接口函数需要给别的c文件使用，所以必须在要使用这些函

数的源文件中include watchpoint.h，一开始我把watchpoint.h和watchpoint.c放在一个目录下，我使用#include “...” 一直报错找不到文件，最后我的舍友给我指出了问题，可能和Makefile文件有关，我需要把watchpoint.h放到include文件夹下，然后通过#include <...> 来使用。感谢我的舍友！！！