体系实习LAB-2-1报告

盛朱恒 1600012830

一、实验目标

了解RISCV64I指令集的基本指令，为其制作一个简单的单周期模拟器。

二、实验过程

（1）riscv工具链的安装

首先在GitHub上下载riscv-gnu-toolchain，参照手册与GitHub进行安装。需要注意的是，手册上的：

./configure --prefix=/opt/riscv

make

存在错误。

如果这样make，在后续的步骤中，无法编译出仅有rv64i的elf文件，而是会要求加上D拓展集。所以用以下的设置make:

./configure –with-arch=rv64i –with-abi=lp64 --prefix=/opt/riscv

make

这样就能编译出只有rv64i的elf文件了。

（2）编译代码：

riscv64-unknown-elf-gcc -Wa,-march=rv64i -o xxx xxx.c

（3）模拟器编写与编译。

以参考代码为蓝本进行补充和改造。

编译命令： g++ Simulation.cpp -c -o Sim.o ../myrv64isim/

g++ Read\_Elf.cpp -c -o Read.o ../myrv64isim/

g++ Sim.o Read.o -o myrvsim

三、模拟器介绍

（1）运行。

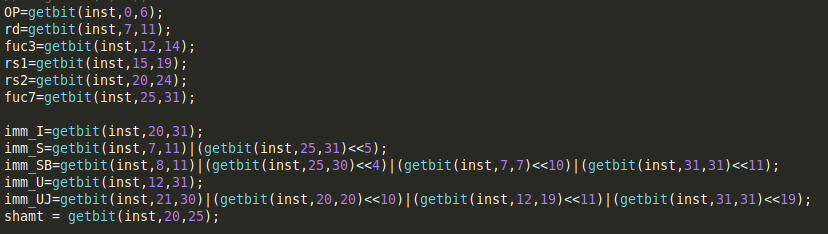
./myrvsim 进行运行，然后按要求输入elf文件名，选择是否需要单步模式，是否需要显示执行的每条指令，是否需要译码的信息。在单步模式下，每一步后可以进行寄存器和内存的查看。寄存器查看直接输出32个寄存器的值，内存查看需要输入地址、查看的大小单位、查看的块数。比如输入 100，4，10，指查看地址100开始，以4Byte为一个单位进行输出，输出10个单位。

（2）编写。

编写分为两个部分，readelf和simulation。

Readelf部分主要需要根据elf的格式对可执行文件进行解析，获取各个段的信息，以便将其load到memory中。这一部分在网络上有一些开源的代码可以进行参考。

Simulation部分。根据之前的解析结果，将数据load到memory中。并从main的入口开始进行模拟，直到main结束。这里带来一个问题，由于只模拟main函数，所以初始化部分没有模拟，因此在c代码的书写时，需要把变量设为全局，不需要初始化的类型。并且由于没有模拟库函数，所以系统调用需要自己设计，以完成输出。

指令模拟过程：首先是translate，这时还不知道指令类型，所以按照所有可能的类型将各个部分的寄存器、op code、立即数等都先取下来，这样在执行的时候可以直接使用。如图： 执行过程，根据opcode、func3、func7确定指令，然后按照green\_card进行模拟。其中SCALL（系统调用部分是自己设计的，在后续部分说明）。

遇到的BUG点：

1. 对于需要以PC作为输入的函数，由于在translate后就进行了“PC”+4，所以在这里需要的PC其实是目前的“PC”这一数据的-4后的值。
2. 对于branch指令，需要比较大小，默认情况下reg[]的类型是unsigned long long int但是会有负数参与比较大小，所以在比较时应该是先转成long long int在进行比较。
3. 立即数的有符号拓展和无符号拓展需要分清楚

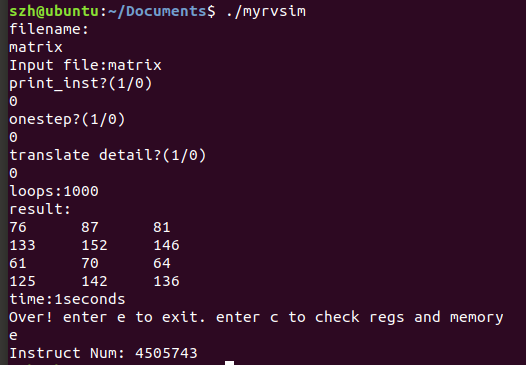
（3）系统调用设计。

由于只执行main函数，所以不能进行输入输出、获得时间等功能。因此自行设计了上面的系统调用。自己的“库函数”在myhead.h中，完成了输出字符串、字符、整数，输入整数，获得时间，退出函数的功能。lab1.1中的代码需要进行相对的改动后才能运行。

主要过程是用x17作为系统调用号，用x10传参数。然后在模拟的时候，根据系统调用号知道调用是什么类型，然后进行对应的操作。

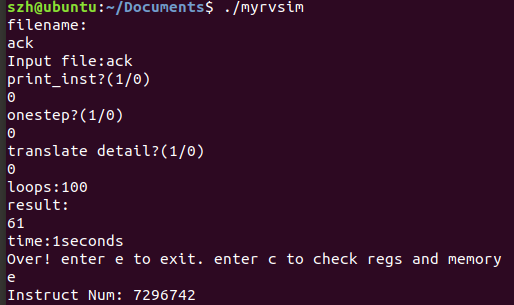
（4）模拟器运行示意。

矩阵乘法：



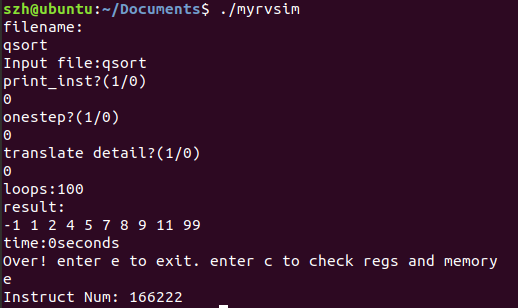
Loops到time的部分就是程序的输出。输出了答案和用时。

Ackerman数：



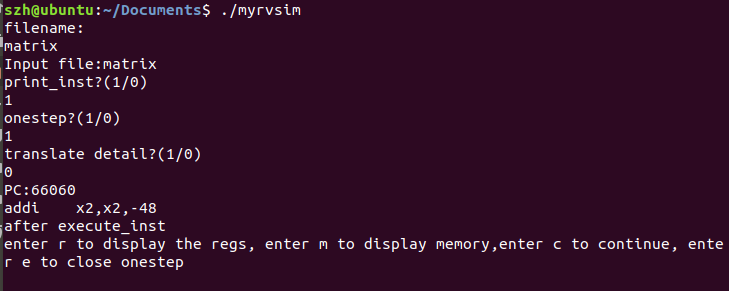
Loops到time的部分就是程序的输出。输出了答案和用时

Qsort:

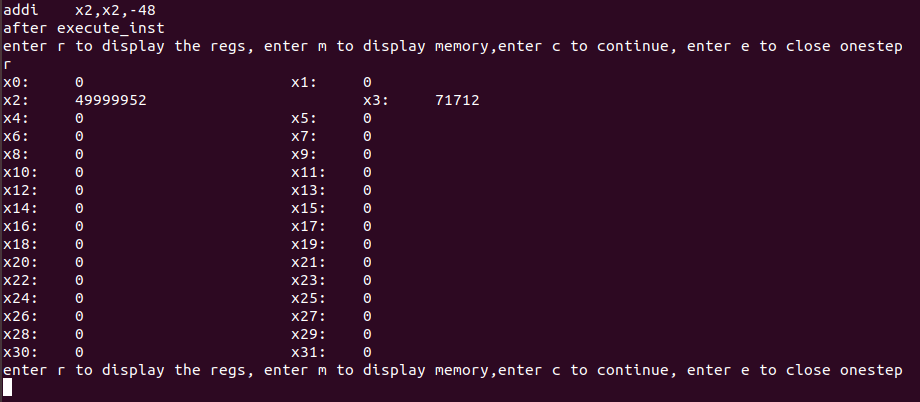


Loops到time的部分就是程序的输出。输出了答案和用时

单步执行：



查看寄存器：



查看内存：

