RISC-V RV64I 模拟器实验报告

姓名： 张煌昭
学号： 1400017707
学院： 元培学院
邮箱： zhang_hz@pku.edu.cn
手机： 17888838127

本次实验全部由本人在所的提供模版的基础上独立完成，未与任何组织或个人分享合作，如有雷同，纯属巧合！

一．总体概述

本次实验（LAB2.1）完成了对 RISC-V 中 RV64I 指令的功能模拟，该模拟器可以加载一个由 RISC-V 编译工具链编译生成的程序，运行程序并得到内存和寄存器结果。此外，还对 RV64C 压缩指令进行了实现，同时也制作了简单的图形界面（GUI）以方便调试和使用。

通过本次试验，对 ELF 文件格式以及程序链接和加载有了比较深入的理解；对于功能模拟器的实现方法也有了较为具体的认识；并且从头到尾独立完成了一个比较完整的模拟器项目。

此外本次实验还有很多不足之处，由于时间比较紧迫，模拟器还存在某些未解决的 bug，这些 bug 将在此后进行完善。

二．基础知识

1. RV64 寄存器堆和内存模型

64 位 RISC-V（RV64）的寄存器堆中的寄存器均为通用寄存器，寄存器堆的大小为 32，每个寄存器的大小为 64 位，从低到高标号为 x0-x31。寄存器堆如下 Figure 1 左图示意，其中规定 x0 为零寄存器，写作 zero；约定 x1 为返回地址寄存器，写作 ra；约定 x2 为栈指针寄存器，写作 sp；约定 x3 为全局数据指针寄存器，写作 gp。以上 zero，sp，gp 均为本词实验的模拟器需要设置的寄存器，其它寄存器均不需要模拟器特殊处理。

RV64 的内存模型为一维线性寻址内存。其中的代码段、数据段，和堆栈等如下 Figure 1 右图示意。模拟器实现时需要从程序文本中加载代码段和数据段，堆栈为程序运实时运行时使用的，不需要特殊操作。

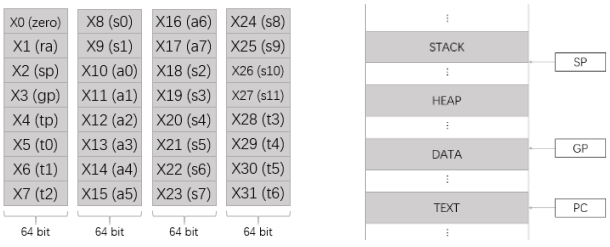
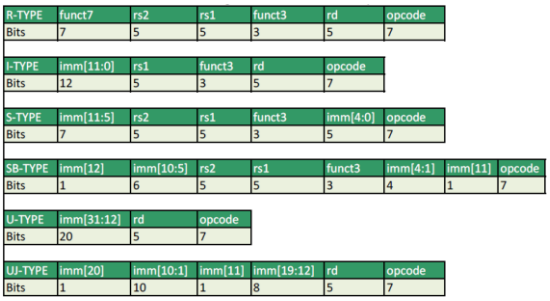


Figure 1. RV64 寄存器堆和内存模型示意图

2. RV64 I 集指令

RV64 I 集指令都为 32 位定长指令，根据低 7 位，将其分作 R 型、I 型、S 型、SB 型、U 型和 UJ 型 6 种。一般的，指令会被切分为 opcode（低 0-6 位），rd（低 7-11 位），rs1（低 15-19 位），rs2（低 20-25 位），funct7（低 27-31 位），funct3（低 12-14 位），imm（不固定）等段域中的几个，不同类型的切分指令的方式如下图 Figure 2 所示[1]。



R-TYPE	funct7	rs2	rs1	funct3	rd	opcode
Bits	7	5	5	3	5	7

I-TYPE	imm[11:0]	rs1	funct3	rd	opcode
Bits	12	5	3	5	7

S-TYPE	imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
Bits	7	5	5	3	5	7

SB-TYPE	imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode
Bits	1	6	5	5	3	4	1	7

U-TYPE	imm[31:12]	rd	opcode
Bits	20	5	7

UJ-TYPE	imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode
Bits	1	10	1	8	5	7

Figure 2. RV64I 不同类型指令的切分方式

具体的指令编码请见参考资料[2]。

3. ELF 文件格式与可执行文件的链接和加载

ELF 文件格式中存储了可执行文件运行时需要的所有信息，以及代码和数据[3]。

首先，64 位 ELF 文件具有 64 字节的 ELF Header，其格式如下图 Figure 3。其中与程序链接和加载相关的部分用红色框出。

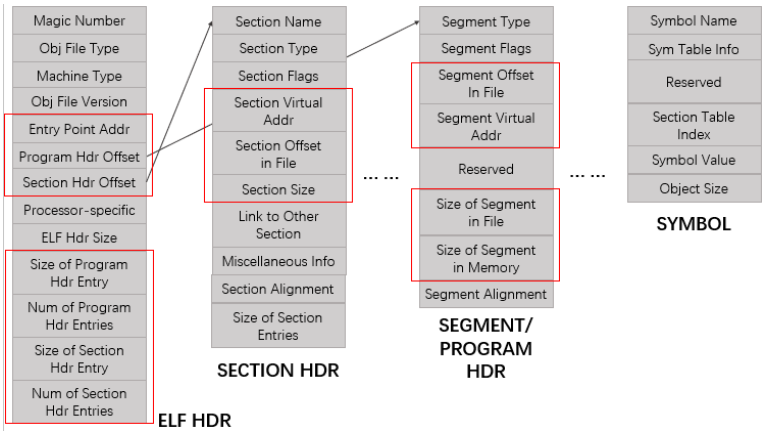


Figure 3. ELF 文件格式

需要注意的是，ELF 文件中所有 Section Name 都存放在一个特殊的 Section 中，称作 shstr（Section Header String），Section Name 的字符串按线性存放在该 Section 内，通过偏移量的方式获取；所有 Symbol Name 都存放在一个特殊的 Section 中，称作 strtab（String Table），Symbol Name 的字符串按线性存放在该 Section 内，通过偏移量的方式获取。

程序加载时直接加载 TEXT 和 DATA 段，Segment 中 Type 为 LOAD，Flags 权限为 RX（可读可执行）的 Segment 为 TEXT 段；Type 为 LOAD，Flags 权限为 RW（可读可写）的 Segment 为 DATA 段。此外还需要注明程序入口地址（PC，一般通过 ELF HDR 获得，本试验中在符号表中寻找 main），栈指针（SP），全局指针（GP，通过读取符号表寻找 __global_pointer&）等。

三．实验步骤与实现方式

1. 解析 ELF 文件

首先需要按照 ELF HDR，ELF SEGMENT HDR，ELF SECTION HDR，ELF SYMTABLE 的格式定义各个数据类型 Elf64_Ehdr，Elf64_Shdr，Elf64_Sym，Elf64_Phdr。

第一步需要解析 ELF HDR，所有其余内容的解析必须从 ELF HDR 入手，因为 Segment 和 Section 的偏移量均在 ELF HDR 中给出。直接打开 ELF 文件，之后读取前 64 字节放入 Elf64_Ehdr 之中，解析读取各信息。

第二步解析 ELF Section HDR。首先需要从 ELF HDR 中读出 shstrndx（shstr 的 Section 下标），读出 shstr 获得所有 Section 的名称字符串；第二步从 ELF HDR 给出的 Section HDR Offset 起读取连续的 Number of Section HDR Entries 个 ELF Section HDR 到 Elf64_Shdr 之中，解析读取个信息，其中尤其要读取 Sym Table Section。

第三步解析 ELF Sym Table。首先寻找并读取名称为".strtab"的 Section，获得所有 Symbol 的名称字符串；第二步将 Sym Table 中各个 Symbol 读取到 Elf64_Sym 之中，解析读取各信息，其中尤其要读取 main，__global_pointer$和 atexit 作为程序入口地址，gp 指针，退出地址。

最后解析 ELF Segment Table。从 ELF HDR 给出的 Program HDR Offset 起读取连续的 Number of Program HDR Entries 个 ELF Segment HDR 到 ELF64_Phdr 之中，解析读取各信息，其中尤其要读取 LOAD+RX 和 LOAD+RW 的 Segment。

2. 模拟器硬件模拟和程序加载

RISC-V 的内存模型为一维线性的，并且最小的对齐单位为 byte，因此使用一个 unsigned char 类型的数组来模拟内存。RISC-V 的寄存器为 32 个 64 位通用寄存器，使用一个大小为 32 的 unsigned long long 类型的数组来模拟寄存器。PC 指针使用一个 unsigned long long 类型的整型进行模拟，指向内存数组中的 16 位对齐的位置（否则将引起异常，导致停机）。

通过第 1 步获得了 ELF 格式的可执行文件的所有信息，下面可以对程序进行加载，加载的过程如下。首先初始化模拟器的 32 个 64 位寄存器，初始化模拟器的内存数组（使用 unsigned ）；之后将 TEXT 段（LOAD+RX）和 DATA 段（LOAD+RW）载入到内存数组相应位置；再设置 PC 指针为 main 函数入口，reg[2]为 sp，reg[3]为 gp。

通过以上步骤完成程序加载，下面说明模拟过程。

3. 指令的功能模拟

由于本次试验只需要进行功能模拟，因此可以获取定长的指令编码，译码后直接进行相应的内存数组、寄存器数组或 PC 的设置即可。整个指令模拟分为 IF（取指）和 ID（译码）两部分。

IF 直接从 PC 位置取出 32 位指令，之后将 inst_num 加 1，并将 PC 加 4，将取出的指令和之前的 PC 保存以便传递给 ID 阶段，之后便完成取指操作。

ID 读取 IF 取出的指令和该指令对应的 PC 后，首先通过 opcode 判断指令类型，之后按照对应类型的切分规则进行切分，得到该指令需要的寄存器，立即数等；最后按照指令手册的要求对内存、寄存器或 PC 进行更新赋值，完成译码和执行操作。

循环 IF 和 ID 便可以逐条指令地进行模拟，最终当 PC=endPC 时（endPC 为程序的退出地址），或者遭遇异常时退出。异常的情况包括：PC 未按 16 位对齐，指令非法（指令的 opcode 或 funct 未知）。

4. 结果展示

由于该模拟器目前无法进行系统调用和库函数调用，因此无法进行打印操作，所以采用如下方法展示运行结果。

在编写测试程序时，将感兴趣的变量全部定义为全局变量并赋初值，保证其会定义在 DATA 段中，这样便可以通过读取 ELF 文件获得其内存位置；在模拟器运行结束后，打印这些内存位置，便打印出所有感兴趣的变量。

四．实验结果

命令行界面的 RV64 I 模拟器请见 code/riscv-sim，make 编译得到模拟器可执行文件 sim，之后使用命令“./sim ./YourProgram ValueOfInterest1 ValueOfIntereste2 …”运行模拟器，其中./YourProgram 为待模拟的 RISCV 程序，后面的若干 ValueOfInterest 为程序中感兴趣的变量的变量名。更多信息请见 code/riscv-sim/README。

运行 code/riscv-sim/qsort，code/riscv-sim/Ackermann 和 code/risc-sim/matmul 程，运行结果如下图 Figure 4。上图为 qsort 运行结果，ori_array 为原始的数组，array 为快速排序后的数组；下图中为 Ackermann 运行结果，res 为 m=1-4，n=1-4 的所有 Ackermann 函数的值，ack_res 为 m=1-4，n=1-4 的 Ackermman 运行结果；下图为 matmul 运行结果（为了支持程序，向模拟的指令集中添加了 addw 和 mulw），res 为正确的输出，C 为矩阵乘法得到的结果。三个结果均符合预期。

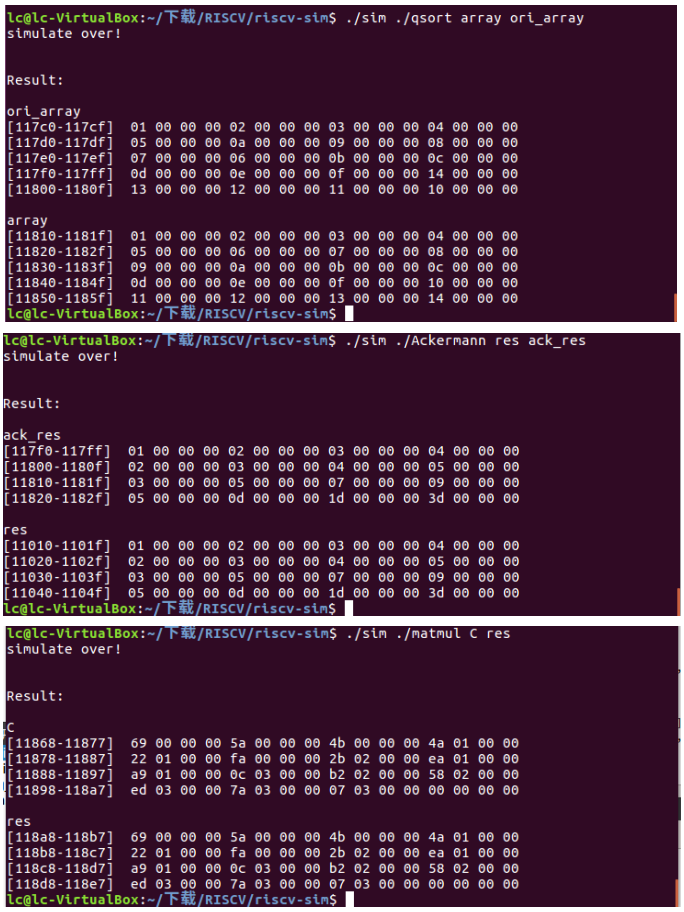


Figure 4. qsort、Ackermann 和 matmul 程序在 sim 上的模拟结果

五．补充实验

1. GUI 界面和单步运行

使用 QT5.9 编写设计 GUI 界面，界面如下图 Figure 5 所示。界面固定显示寄存器堆（Register File），代码段（Text/Code）和感兴趣的数据段（Data of Interest）；此外，在载入一个 RISC-V 程序后还将切换至 ELF 界面显示 ELF 文件信息，并要求用户根据这些信息输入感兴趣的数据的名称。

运行方式如下，首先点击“File”按钮打开路径菜单，选择 RISC-V 程序；之后点击“Load Program”按钮读取 ELF 文件信息，点击后整个界面切换至 ELF 界面并显示 ELF 文件信息；在 ELF 界面右侧输入感兴趣的数据名称（数目上限为 5 个，也可以选择不输入）后点击“OK!”按钮，显示模拟器需要使用的信息，点击”OK!”按钮确认并加载程序。加载程序后，“JUST RUN!”和“Step-by-Step”按钮变为可以点击，点击“JUST RUN!”按钮，按照之前所述的方式循环 IF 和 ID 模拟直至终止或异常；点击“Step-by-Step”按钮，调用 IF 和 ID 一次进行单步模拟，若终止或异常，则停机不再运行。

整个程序请见 code/riscv-sim-ui，程序运行的过程如下图 Figure 5 所示。

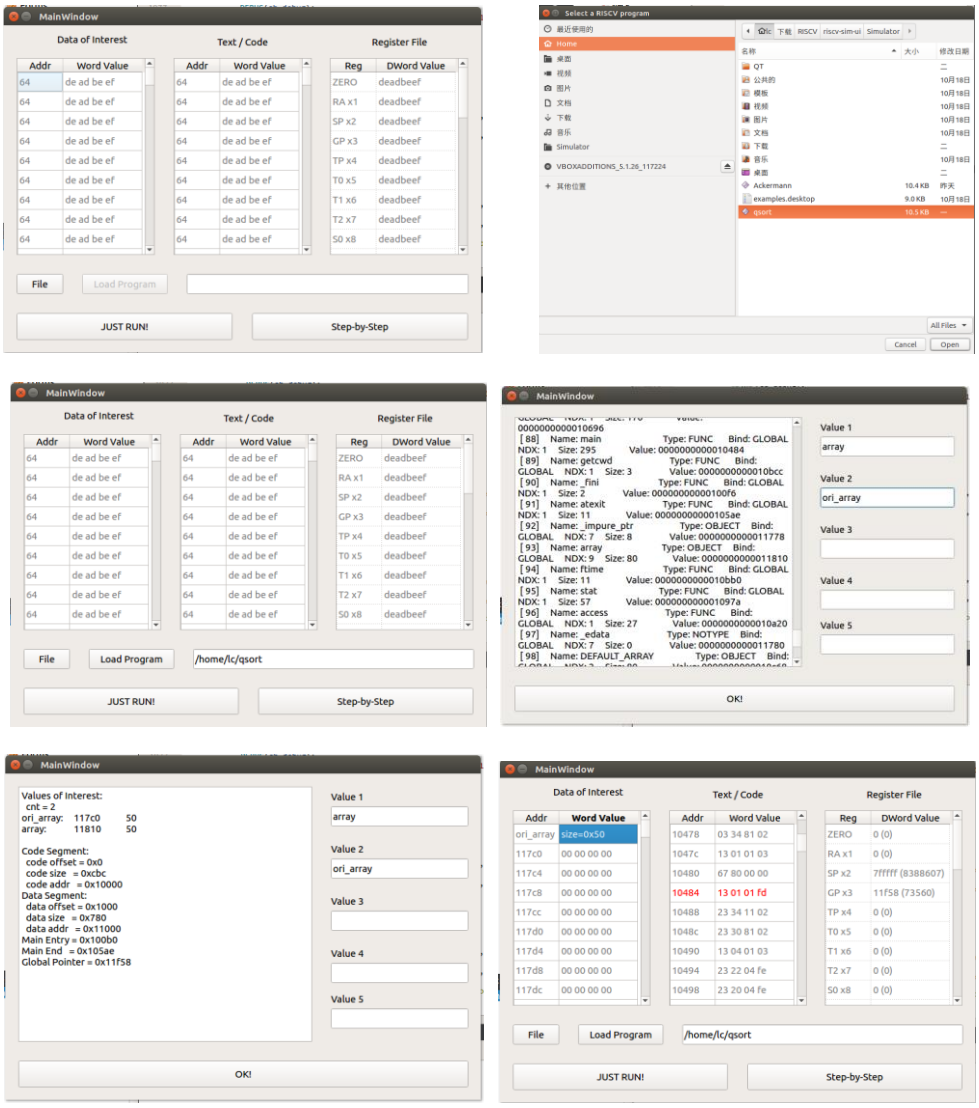


Figure 5. qsort 程序在 sim-ui 上的模拟结果

2. RV64 C 指令扩充

由于从 main 函数直接进入程序会很不自然，更妥当的做法应为从 ELF HDR 指出的程序入口进入程序。由于在 main 函数前的指令会有 16 位的 C 类压缩指令，因此需要对模拟指令进行扩充。由于会改变指令长度，因此 IF 和 ID 都需要有相应的扩充。

对 IF 的扩充为：首先读取 PC 所指地址的 1 个 byte，判断低 2 位是否为“11”，若是则说明该指令为 32 位指令，否则说明该指令为 16 位指令。对 32 位指令的操作不变，16 位指令改为 PC 加 2。除此以外，还需要向 ID 传递指令长度。

对 ID 的扩充为：首先读取 ID 传递来的指令长度，若为 32 位指令，则进入之前的操作步骤；若为 16 位指令，则进入新加入的 16 位指令译码操作。16 位译码操作与 32 位译码操作类似，参考 RISC-V 手册[2]进行译码，得到译码后直接跳转至对应的 32 位指令执行。

六．实验收获

1. 通过本次实验比较深入地对 ELF 文件格式进行了研究，对于 ELF 文件的链接和加载过程也有了比较形象的理解。

2. 对于 RISC-V 的 ISA 设计理念有了一些比较具象的理解，加强了对于 RISC 架构的认识。

3. 在一周内几乎从零开始，独立完成了 RV64I 模拟器项目——从设计到实现，并制作了一个比较能让人接受的 GUI 界面，锻炼了自己的能力，激发了自己的潜力。

七．不足之处

1. 过度依赖 Green Card 而忽视了原始文献——RISC-V 手册，因而造成了很多误读和 bug。

2. 截止提交时，指令模拟仍可能存在某些未修复的 bug，需要在之后的时间进行完善。

3. GUI 界面比较繁复冗杂，并且重复加载程序可能会导致内存不刷新的 bug，需要在之后的时间进行完善。

八．参考文献

[1] RISC-V RV64I Simple Green Card. LAB 2.1 参考材料

[2] Andrew Waterman, Yunsum Lee, David Patterson, Krste Asanovic. The RISC-V Instrucsion Set Manual Volume 1: User-Level ISA Version 2.1. CS Division, EESC Department, University of California, Berkeley. May 31, 2016.

[3] 施聪. UNIX/LINUX 平台可执行文件格式分析. Dec. 1, 2004.
https://www.ibm.com/developerworks/cn/linux/l-excutff/