RISC-V 体系结构编程与实践 勘误

亲爱的小伙伴，

感谢您选购和阅读《RISC-V 体系结构编程与实践》，感谢您对奔跑吧 Linux 社区的长期支持！由于我们水平有限，出书时间仓促，难免有不少勘误，我们深感歉意，也恳请大家多多包涵，也欢迎给我们提出批评和建议！我们会不久的将来第 2 版中继续完善和优化！如果发现有新的勘误或者有好建议，请通过如下方式与我们联系：

- 邮件：runninglinuxkernel@126.com
- 微信号：runninglinuxkernel
- 在 github/gitee 上提 PR：
  https://gitee.com/benshushu/riscv_programming_practice
  https://github.com/runninglinuxkernel/riscv_programming_practice

请读者在阅读本书之前，一定要先阅读书最前面的“本书约定”一文。

1. 配套资料

本书提供海量的配套资料。

- QEMU 实验平台 - VMware/VirtualBox 镜像
- 实验参考代码
- 免费视频
- 本书涉及的芯片资料

本书配套的实验参考代码：

https://gitee.com/benshushu/riscv_programming_practice
https://github.com/runninglinuxkernel/riscv_programming_practice

下载方法：

关注“奔跑吧 linux 社区”微信公众号，在微信公众号里输入“risc-v”获取下载地址。

本书配套的视频课程。***文字不如声音，声音不如视频，欢迎订阅我们的 RISC-V 视频课程。***

1. 扫码订阅 RISC-V 课程，支持在线播放。

2. 淘宝小店：http://shop115683645.taobao.com/

视频课程试看：

https://space.bilibili.com/277836343/channel/seriesdetail?sid=3055480

读者如果想购买与本书实验适配的开发板，推荐韦东山团队制作的 DongShanPI D1s 开发板。该开发板相关的技术问题，请咨询韦东山团队。

购买地址：（注：该商品由韦东山团队发货和提供售后服务）

https://item.taobao.com/item.htm?&id=695941169493

支持 D1S 的实验参考代码分支是：（该分支由韦东山团队负责维护）

https://gitee.com/benshushu/riscv_programming_practice/tree/DongShanPI_D1/

更多资料和资源请关注奔跑吧 linux 社区微信公众号。

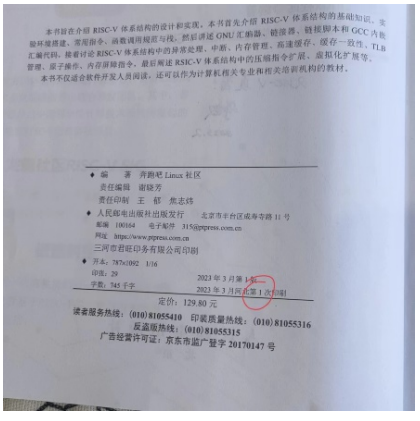
作者为本书创建了微信交流群，想加入微信读者交流群的小伙伴，请先加我们客服微信号：runninglinuxkernel。请出示购买纸质图书的截图，以方便我们邀请您进读者群。

2. 勘误

勘误日志：

2.1 P1 勘误

P1 勘误里列出的是需要更正的勘误。**（Fixed 列表示第几次印刷已经修订），请读者核对购买的书是第几次印刷的。**



列出的勘误按发现的时间为顺序

页数	原文	更正	Fixed
375		图 18.23 需要更正，见下面的插图。	3
383-384		代码 45 行注释改成：计算 C2 代码 56 行注释改成：计算 C3	3
386		图 18.30 需要更正，图上最上方 v9 改成 v8，如下所示： vfmacc.vf v8, ft2, v2	3
406	hlv{x}.{b|h|w|d}{u} rd, offset(rs1)	b 和 h 之间应该用“|”	3
411		图 20.16 需要更正，图中的 SGEIP 改成 SGEIE，VSEIP 改成 VSEIE	3
427		图 20.21 需要更正，图中的 imm[11:0] 更正为 imm[11:5]	3
436	例如，Mem8 addr 表示内存地址 addr 中的 8 位数据。	Mem8 addr 改成 Mem8 (addr)	3
139	第 8.4 章 spec 寄存器的值	笔误，改成： sepc 寄存器的值	3
74	根据 ra 寄存器的返回地址，跳转到上一级函数（即 func1()函数通过 JAR 指令调用 add_c()函数的下一条指令）。	根据 ra 寄存器的返回地址，跳转到上一级函数（即 func1()函数通过 JAL 指令调用 add_c()函数的下一条指令）。	3
30	表 2.1 的信息都可以在 NEMU 配置菜单中配置。	表 2.3 的信息都可以在 NEMU 配置菜单中配置。	3
38	表 2.5 中第 4 行 0b0001 预分频寄存器中的高 8 位	0b0001 改成 0b001	3
61	倒数第二段： 在第 1 行中，**交换 tp 和 sscratch 寄存器的值，即**读取 sscratch 寄存器的旧值	将红色那边半句去掉	3
386		图 18.31 需要修正	3
65	实验 3-3 从 0x802 0000 地址复制 32 字节到 0x8021 0000 地址处	0x802 0000 改成 0x8020 0000	4
132	mpec：记录发生异常的指令地址。	mepc：记录发生异常的指令地址。	4
51		表 3.3 有改动，见下面红色部分	4
6		图 1.5 中的“x8/s01/fp”改成“x8/s0/fp”	4
7	另外，s0 寄存器可以用作栈指针（Frame Pointer，FP）	另外，s0 寄存器可以用作栈帧指针（Frame Pointer，FP）	4
71	把局部变量 a 和 b 分别存储到 SP 与 SP+8 位置上，如图 4.4（d）所示	把局部变量 b 和 a 分别存储到 SP 与 SP+8 位置上，如图 4.3（d）所示	4

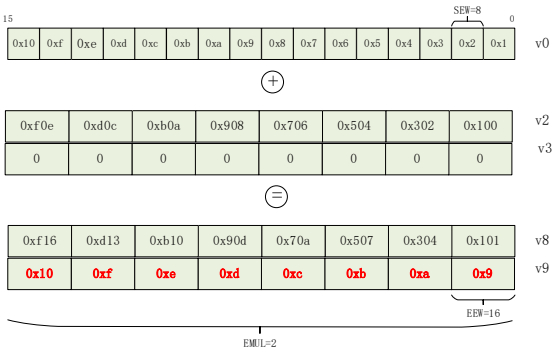


图 18.23 修改见红色部分

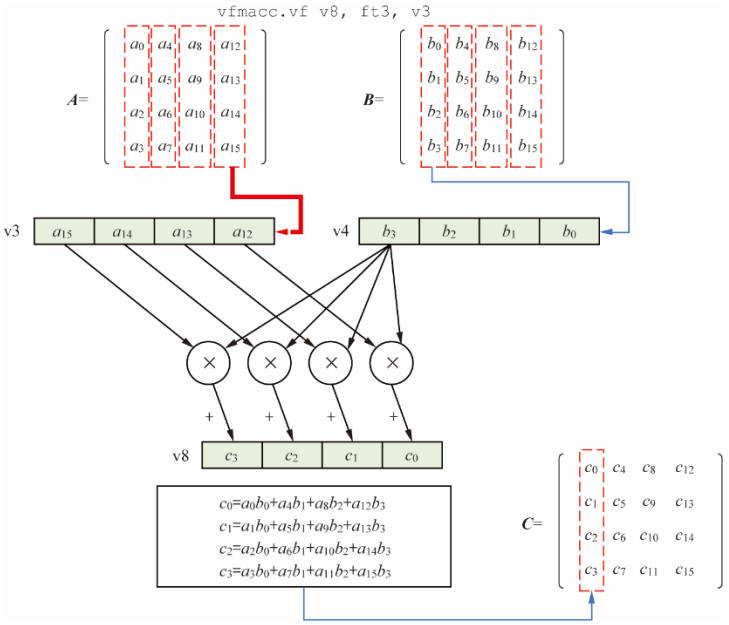


图 18.31 见红色加粗的地方

表 3.3 与 PC 相关的加载和存储伪指令

伪指令	指令组合	说明
la rd, symbol（非 PIC）	auipc rd, delta[31 : 12] + delta[11] addi rd, rd, delta[11:0]	加载符号的绝对地址。 其中 delta = symbol−pc
la rd, symbol（PIC）	auipc rd, delta[31 : 12] + delta[11] l{w|d} rd, delta[11:0](rd)	加载符号的绝对地址。 其中 delta = GOT[symbol]−pc

表 3.3 修改见红色部分

2.2 P2 补充

这里列出的不是勘误，是有益的补充，可能会对读者理解有帮助。

页数	原文	补充
262	第 14.2 章	在 14.2 章补充如下内容 LR 和 SC 指令是配对使用，SC 指令成功的条件，一是当前保留集有效，二是保留集中包含的数据被成功更新或者写入。无论 SC 指令执行成功与否，当前 CPU 包含的保留集都被无效掉。 有如下情况 SC 指令会失败： 1. 如果 SC 写入的地址，不在与之配对的 LR 指令组成的保留集范围内。 2. 如果在 LR 和 SC 指令范围内，执行了另外一条 SC 指令，不管写入任何地址，都会导致 SC 指令执行失败。 3. 如果在 LR 和 SC 指令范围内，执行了另外一条 store 指令。这相当于是在 LR/SC 配对的序列中，又插入了一条 store 指令，并且这条 store 指令是对 LR 的加载地址进行 store 的话，那 SC 指令会执行失败。如果这条 store 指令不是对 LR 加载地址进行 store 的话，SC 指令不会失败。 4. 如果另外一个 CPU 对当前 LR 和 SC 的保留集地址进行写入操作。 5. 如果另外一个外设（不是 CPU）对 LR 加载的数据进行写入 RISC-V 架构对 LR/SC 序列做了一些约束，不符合这些约束的 LR/SC 序列不能保证在所有 RISC-V 处理器中都能成功：

		1. LR/SC 的循环 loop 中最多包含 16 条指令 2. LR/SC 序列包括 RV64I 指令或者压缩指令，但是不包括加载、存储、向后跳转、向后分支、JALR、FENCE、以及 SYSTEM 指令。 3. LR/SC 序列可以包含向后跳转的 retry 重试。 4. SC 指令的地址必须与同一个 CPU 执行的最新 LR 的有效地址和数据大小相同。
262	第 14.2 章	如果在 LR 和 SC 序列中发生了进程切换，那么 LR/SC 还能成功吗? 目前不少 RISC-V 芯片采用类似 arm 的 exclusive monitor 的方式，那么在异常返回时，RTL 代码，也就是 CPU 的 verilog 代码里是 可以清除 local 监视器。从另外一个角度看，一个进程的切换，必定发生了一次时钟中断。中断也是异常的一种。所以，这样就可以保证进程切换时，LR 申请的 reservation set 被作废了。 另外还可以采用软件方式来清除 local 监视器。例如最新的 Linux 内核，在异常返回的时候， 对异常返回地址执行了一条 SC 指令（读异常返回地址的值，然后用 SC 指令往异常返回地址里写 刚才读回来的内容），这样就可以无效掉 LR/SC 组成的保留集，达到清除 reservation set 的目的 在 RISC-V 架构手册卷 1 的第 8.2 章里有提到， 使用 SC 指令对一块可以被抓破的内存进行写入（也可以是任何地址。Linux 内核异常返回时的做法是：读异常返回地址的值，然后用 SC 指令往异常返回地址里写 刚才读回来的内容），用来无效掉已经加载的保留集，主要用途： 1. 进程切换 2. 修改了 VA->PA 映射，并且这页面包含了一个活跃的保留集。
45	第 3.2 章	RV64 指令集支持 64 位宽的数据和地址寻址，为什么指令的编码宽度只有 32 位？ 对这个问题的回答，需要添加下面红色的部分，这样更加完整。 因为 RV64 指令集是基于寄存器加载和存储的体系结构设计，所有的数据加载、存储以及处理都是在通用寄存器中完成的。RISC-V 一共有 32 个通用寄存器（通用寄存器都是 64 位宽的，可以处理 64 位宽的数据），即 x0～x31，例如，x0 寄存器的编号为 0，以此类推。因此，在指令编码中使用 5 位宽（2^5＝32），即索引 32 个通用寄存器。对于精简指令集来说，大部分运算指令最多采用 3 个寄存器的方式来表示，例如加法指令 rd=rs1+rs2，那么在指令编码中只需要 15 个位宽即可以索引 3 个通用寄存器，剩余的位宽可用于指令分类等。因此 32 位宽的指令编码已经绰绰有余。使用 32 位宽指令编码比 64 位宽指令编码更有利于提高指令密度，减少代码尺寸，从而提高指令高速缓存的命中率，提升程序执行效率。 加载和存储指令通常采用 12 位立即数加上 2 个通用寄存器的方式来表示，下面以 lw 加载指令为例，说明其指令编码的布局。
281	规则 1 规则 2	规则 1 类似流水线中的 WAW（Write After Write，先写后写）， RAW（Read After Write，先读后写）数据冲突。 从规则 1 推导可知，如果两条连续的 store 指令或者 load+store 指令组合，它们访问的地址不重叠，那么这两条指令可以乱序执行。 规则 2 是经典的“load-load”问题。两条 load 指令访问相同地址，如果它们乱序执行，当它们之间存在一条 store 指令并且修改了该地址的值，这样会导致前面的 load 指令得到了最新的值，后面的 load 指令反而获取了旧值。 从规则 2 推导可知：如果两条连续的 load 指令访问的地址不重叠，那么这两条指令可以乱序执行。
284	例 15-17	第 1 行和第 2 行存在地址依赖关系，根据规则 13，第 1 行和第 3 行的执行次序不能重排，因为第 3 行的存储指令不能在地址依赖还没解决之前执行，有可能存在 a1=s0 的情况，即第 2 行和第 3 行的访问了相同的内存地址，这样就违背了规则 1。 另外这里第 3 行代码可以改成：sw **t1**, 0(s0)，这样更能体现规则 13 的精髓，避免第 2 和第 3 行存在数据依赖的影响。

1. 第 4 季视频课程

欢迎小伙伴订阅笨叔录制的第 4 季视频课程：RISC-V 体系结构编程与实践，我们全面深入解读 RISC-V 指令集、体系结构、陷阱等。订阅视频课程，请扫码。

第 4 季课目录：

第4季 奔跑吧Linux社区 视频课程

RISC-V体系结构编程与实践

主讲：笨叔

课程名称	进度	时长（分钟）
第1课：课程介绍	完成	20
第2课：RISC-V体系结构介绍	完成	47
第3课：RISC-V处理器微架构	完成	48
第4课：搭建RISC-V开发环境	完成	30
第5课：RISC-V指令集	完成	128
第6课：RISC-V函数调用规范	完成	40
第7课：RISC-V GNU AS汇编器	完成	42
第8课：RISC-V GNU 链接脚本	完成	90
第9课：RISC-V GNU 内嵌汇编	完成	52
第10课：RISC-V异常处理	完成	80
第11课：RISC-V中断处理	完成	52
第12课：RISC-V内存管理	完成	116
第13课：内存管理实验讲解	完成	36
第14课：cache基础知识	完成	78
第15课：缓存一致性	完成	96
第16课：RISC-V TLB管理	完成	54
第17课：RISC-V原子操作	完成	93
第18课：RISC-V内存屏障	未录制	
第19课：BenOS操作系统相关知识	未录制	
第20课：RVV可伸缩矢量计算	未录制	
第21课：RISC-V压缩指令	未录制	
第22课：RISC-V虚拟化	未录制	
		总计18.5小时

更多精彩内容马上献上....
微信公众号：奔跑吧Linux社区