RISC-V 指令集扩展检测机制现状

郑鉉壬
i@zenithal.me

2022-07-06

碎片化的指令集扩展

当你说你是 RISC-V 的时候...

- 单字母：RV{32,64,128}{I,E}MAFDQCVH
- 发行版：RV64GC, G 代表 IMAFDZicsr_Zifencei
- 多字母
  - Zicsr, Zifencei, Zicntr, Zihpm, Zihintpause, Zihintntl, Zicbom, Zicbop, Zicboz
  - Zmmul, Zam, Zawrs, Zfh, Zfhmin, Zfinx, Zdinx, Zhinx, Zhinxmin, Zca, Zcf, Zcb, Zcmb, Zcmpe, Zcmt
  - Zba, Zbb, Zbc, Zbs, Zbkb, Zbkc, Zbkx, Zknd, Zkne, Zknh, Zksed, Zksh, Zkn, Zks, Zkt, Zk, Zkr, Zpbpo, Zpsfoperand, Zpn, Zjid
  - Zve32x, Zve32f, Zve64x, Zve64f, Zve64d, Zve, Zvl32b, Zvl64b, Zvl128b, Zvl256b, Zvl512b, Zvl1024b, Zvl, Zv
  - Smstateen, Sscofpmf, Sstc, Smepmp, Svinval, Svnapot, Svpbmt, Sm1-12, Ss1-12, Sv57, Hypervisor, Ssmpu, Smepmp, Shcntrenw, Shtvala, Shtvdcc, Shatpa, Shgatpa, Sdext, Sdtrig
  - Ziccif, Ziccrse, Ziccamoa, Zicclsm, Zic64b, Za64rs, Za128rs, Ssccptr, Sstvecd, Sstvala, Ssu64xl, Ssu32xl, Sstvecv, Ssptwad, Ssube,
  - XBitmanip, XCrypto, XHwacha, XPulp, Xthead, Xventanacondops, X...
  - RVWMO, Ztso

现有检测机制

M 模式

故事要从 misa 开始讲起

- M 模式 ISA 寄存器 `misa`
- 低 26 位分别表示英语字母表中 26 个字母
- 背景：RISC-V 的扩展名由单字母表示（至少在设计之初这样就能满足使用了）
- 问题一：只能表示 26 个扩展，不能满足日渐增长的扩展量需求
  - 解决办法：`mconfigptr`
  - 先介绍 M 模式的机制
- 问题二：只提供给 M 模式程序使用，例如 OpenSBI/RustSBI
  - 是否有 sisa, uisa? 没有，内核和用户空间得用另一套方法
  - 之后介绍内核和用户空间的机制

与 x86 CPUID 的联系

- 提到扩展发现，不得不提到 x86 的 CPUID
- CPUID 以 MSR 为参数（类似于 CSR），返回相应信息
- 其实 `misa` 非常接近这种使用方式
- `csrr t0, misa`
- 这可以复用已有的 `csrr` 指令，而不需要单独的类似 CPUID 的指令
- （这非常 RISC）
- 所以 RISC-V 不是没有 CPUID 指令，只不过不够好用罢了
- 要变好用，那么多加 CSR，要么…

更多 CSR 还是 MMIO

- 更多 CSR
  - CPUID 的老路
  - 优点是简单直接，缺点如下
  - 寄存器消耗较高，对嵌入式场景不友好
  - 每个扩展一个 CSR：消耗更大
  - 几个扩展塞到一个 CSR 里面：怎么分配每个 CSR 的每个位
  - RVI 又不是号码分配局，并不方便把两个无关扩展塞在一块
  - CSR 访问不方便虚拟化（RISC-V 主打经典可虚拟化）
- MMIO
  - 更加灵活！
  - 但格式咋办，这是大家都在吵的事情
  - （现在都还没定下来，所以标题叫检测机制现状）
  - 安卓水人选择了这条路

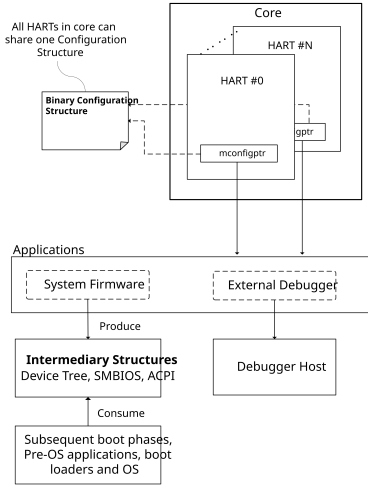
于是有了 mconfigptr[1]

- 安卓水人提了一个曾经名为 `Smdisc` 的扩展（现在变成必须实现的，所以没有了该名字）
- M 模式配置指针 `mconfigptr`
- 指向配置文件所在的物理地址
- 注意也是 M 模式的软件才能访问
- 配置文件格式长啥样?

[1] https://github.com/riscv/riscv-isa-manual/pull/697

mconfigptr 配置格式（草稿）[2]

- 图来自该项目
- 该配置被 `mconfigptr` 指向
- 使用 ASN.1 编码
- 换句话说，OpenSBI 里面要嵌入一个 ASN.1 解析器
- 缺点：ASN.1 较为复杂
- 可以被固件用于生成设备树/SMBIOS/ACPI



[2] `https://github.com/riscv/configuration-structure`

M 模式总结

- `misa`: 类似 CPUID，但不够用
- `mconfigptr` 与配置格式：较为复杂，尚未确立标准
- 好在无论 M 态怎么变，内核（S 态）都不被影响（真的吗）

内核

内核：设备树

- 先从大家最熟悉的设备树开始讲起
- 用过 RISC-V 板子的都知道，启动内核的时候要给内核传一个设备树
- 这也是一个二进制文件，描述 CPU 和外设等一系列设备
- 常被嵌入式平台所使用
- 里面有 `riscv,isa` 这一项，包含一个 ISA 字符串
- 例如：
  `rv64imafdc_zifencei_zicsr_zba_zbb_zbc_zbkb_zbkc_zbkx_zbs_zknd_zkne_zknh_zk`
- 所有的启动器/内核都要包含一个该字符串的解析器
- （这非常容易出错）

设备树的生成

- 内核会自己维护一份设备树，编译的时候用 dtc 生成
  - 例如 `arch/riscv/boot/dts/starfive/jh7100-beaglev-starlight.dts`
  - 之后将生成的 dtb 拷贝到 `/boot` 并填写 U-Boot 配置文件即可使用
- 一些硬件生成器也会根据配置生成 dts
  - Rocket-Chip: `src/main/scala/tile/BaseTile.scala`，只生成 `imafdcbv`
  - 香山: `src/main/scala/xiangshan/XSDts.scala`，只生成 `imafdc`

内核：ACPI/SMBIOS（草稿）

- 服务器/PC 更经常使用 ACPI/SMBIOS 来获取设备信息
- 两者都是在内存中的表状结构
- ACPI 有一个 RHCT (RISC-V Hart Capabilities Table) 表，用来放 ISA 字符串
- SMBIOS 中存放了各个 hart 的 `misa` 的拷贝（显然不够用，也绕过了 misa 的设计目标）
- 均为草稿[3]，更多信息在 riscv-platforms-spec[4]
- 大家对如何表述这么多扩展还是很不确定
  - 要么照搬 ISA 字符串的路
  - 要么使用之前的配置格式（所以标准化流程在某种程度上被之前的项目阻挡住了）
  - 要么发明新的格式（也非常痛苦）

[3] `https://github.com/riscv/riscv-acpi` 与 `https://github.com/riscv/riscv-smbios`
[4] `https://github.com/riscv/riscv-platform-specs`

内核总结

- 设备树: ISA 字符串
- ACPI：ISA 字符串（草稿）
- SMBIOS：`misa` 的拷贝（草稿）
- 好在无论内核怎么变，用户空间都不被影响（真的吗）

用户空间

用户空间：cpuinfo

- 用户空间如何获取扩展信息
- 并没有系统调用（ecall）可以使用（那样是变相的 CPUID）
- 一种方式是 `/proc/cpuinfo`
- 之前的 Linux 会直接输出设备树中的 `riscv,isa`
- 现在 Linux 会解析设备树并输出它认识的扩展
- 这其实是 uapi，而这个 uapi 不稳定
- （所以用户空间的每个程序也需要解析 ISA 字符串了）[5]

[5] rocksdb 在构建脚本里面解析 cpuinfo：`https://github.com/facebook/rocksdb/pull/9366`

用户空间：HWCAP

- 另一种方式是 HWCAP
- （也是我最近在推进的方向，尚未合并）
- 大家或许听过 glibc-hwcaps，即一份 glibc 里面有适合不同架构的代码
- 在内核加载该二进制的时候，会传入一个 `AT_HWCAP`，用户程序可以通过 `getauxval` 获取
- 这里只需要和内核约定好哪个位表示哪个扩展，所以是内核 uapi 的一部分
- 缺点：只有 64 位，也不够用
- （我想尝试推出一个 `AT_HWCAPV` 的机制这样可以传一个变长数组）

用户空间：SIGILL

- 还有一种方式是 SIGILL
- 即用户空间尝试执行某个指令
- 硬件发现不支持后抛出非法指令异常，交给内核
- 内核最终发送 SIGILL 信号到用户空间
- OpenSSL 用该机制检测其他架构的密码学指令
- 问题有几个
  - 新架构还要使用这种黑魔法，实在是不体面
  - 探测的消耗较高

用户空间：环境变量

- 还有一种方法：从环境变量获取 ISA 字符串
- 类似于 `/proc/cpuinfo`
- 算是一个折中，可以用而且用户可控
- OpenSSL 用 `OPENSSL_riscvcap` 检测 RISC-V 的密码学指令

用户空间总结

- ecall: 并不存在
- cpuinfo: ISA 字符串，但不稳定
- HWCAP: 不够用
- SIGILL: 不体面
- 环境变量：ISA 字符串，可控但不够自动化

探测机制总结

- 事实标准就是 ISA 字符串
- 其他机制要么是 ISA 字符串的变种，要么更复杂
- 所有的 RISC-V 程序，如果需要利用一个扩展，要么编译期就确定好（不可移植），要么需要解析 ISA 字符串
- 可惜还没有 `riscv-software-src/riscv-isa-string-parser` 这样一个标准库出现
- （有兴趣的可以尝试折腾出来）

案例分析

案例分析：密码学扩展（Zk）

- 我的本科毕社是做标量密码学扩展（Zk）的硬件单元[6]
- 这个单元的存在怎么传递给用户空间?
- 考虑如下路径
  - Rocket 生成设备树
  - Linux 解析设备树然后产生 HWCAP
  - OpenSSL 解析 HWCAP 然后动态选择密码学加速实现
- 所以我在推动往 Linux 的 HWCAP 加入密码学扩展相关支持（尚未合并）[7]
- 同时也在推动 OpenSSL 的相关工作（已经合并）[8]

[6] https://github.com/chipsalliance/rocket-chip/pull/2950
[7] https://lore.kernel.org/linux-riscv/YqYz+xDsXr%2FtNaNu@Sun/
[8] https://github.com/openssl/openssl/pull/18197

案例分析：Zicsr 与 Zifencei

- RISC-V 非特权集指令集有几个版本：2.2, 20190608 和 20191213
- 在 20190608 中，Zicsr/Zifencei 从基础指令集中拆了出来
- 但 GCC 等工具链之前还在 2.2 的版本
- 最近的 GCC 工具链升级（例如 binutils 2.38）升级到了 20191213[9]
- 故而在编译时两者不契合，gcc 的 `-march=rv64gc` 中不包含 zicsr/zifencei，binutils 会直接收到 `-march=rv64imafdc`，会导致 binutils 不认 `csrrw` 等指令
- 据 Arch 小队反映，该问题尚未完全修好
- （相信不少同学已经经历过这个错误）

[9] `https://lkml.org/lkml/2022/1/24/537`

案例分析：Zicsr 与 Zifencei （继续）

- 受此启发，设备树中至少应该要包含 zicsr/zifencei
- 已经给 QEMU 的设备树生成提交补丁（已合并）[10]
- 内核也需要在 cpuinfo 中暴露该扩展（尚未发补丁，最近 linux-riscv 的补丁处理量较少）
- 工具链也应该可以直接从 cpuinfo 中获取信息（尚未调查，例如 `-march=native`）

[10] `https://lore.kernel.org/qemu-devel/YoTqwpfrodveJ7CR@Sun/`