电子报/2020 年/9 月/20 日/第 006 版
电子文摘

# 利用 RISC-V 创建自定义处理器(一)

湖北　朱少华

由于同时具有开源自由与标准化的优势,RISC-V 基金会吸引了广泛的业内关注。RISC-V 核心规范稳定且正处于被认可的顶峰,其软核、硬核 CPU 以及芯片、开发板和工具都已实现商用,很多大公司已经开始采用 RISC-V 来取代他们的定制架构。该架构的一个关键特性是 CPU 开发人员可以根据需求调整 RISC-V 功能;而无需牺牲为基本标准所创建的工具与库的适用性。这种适用性的关键在于要了解 RISC-V 模块化指令集架构。

RISC-V 最初是加州大学伯克利分校精简指令集计算机(RISC)设计工作的第五次迭代,但它很快地从学术研究演变为一场寻求重新定义电子行业处理硬件设计方法的运动。目前的情况是,系统开发人员必须选择通常为特定应用领域优化过的专有 CPU 架构,或设计自己的 CPU 架构。然而,要追求自己的设计,开发人员就必须放弃现有 CPU 所形成的广泛的支持生态系统。一个折衷的方法是:在保留大部分支持生态系统的同时,调整专有 CPU 架构来实现定制。不幸的是,由于专有架构涉及高额的许可费用,这种折衷方法对于许多设计团队来说不切实际。

RISC-V 计划是在保留标准化优势的同时,为设计人员提供可定制、创新的替代方案。为此,RISC-V 基金会一直维持并推动模块化、开源 RISC-V 处理器指令集架构(ISA)的社区开发,旨在满足从嵌入式系统到服务器集群等各种应用需求。该架构的规范提供免费下载,开发人员可以自由实现基于 ISA 的设计而无需支付许可费。与其他一些开源计划一样,开发人员也没有义务向其他人开放他们的设计。它是开源 ISA,但如果开发人员希望的话,个人设计、硬件架构以及定制都可以保持专有。

该计划目前发展势头强劲,已经有 RISC-V 芯片和内核商用和开源。SiFive、GreenWaves Technologies 和 Microsemi 等公司都已开发了基于 RISC-V 架构的开发板。开发工具、软件库和操作系统端口(包括 Linux)都是当目 RISC-V 支持生态系统的组成部分。但是,要利用全部这项支持,将其用于定制设计,首先需要认真研究 RISC-V ISA 的架构。

基本规格

RISC-V ISA 的定义包含两个关键文档:用户级 ISA 规范和特权 ISA 规范。其中包括基本要求和丰富的标准化、模块化扩展。标准扩展是模块化的,在 CPU 设计中实现任何给定的标准扩展都不会干扰任何其他标准扩展的实现。但是,某些扩展可能基于其他扩展,需要将基本扩展作为所需扩展的一部分来实现。

RISC-V 的整体设计基于寄存器,所有操作以及对一般存储空间的加载存储访问都是通过调用 31 个通用寄存器进行。指令集对 32 位、64 位和 128 位地址空间进行了定义,还定义了一个额外的、精简寄存器数的 32 位指令集,专门针对嵌入式系统设计的较少门数实现。除了存在附加指令来处理较长字长,以及寄存器大小与地址空间匹配以外,这些变体的指令编码也都相同。

图 1 以示意图的形式给出了核心规范与标准扩展的交互方式。RISC-V 基金会现已冻结许多规范,从而确保基于它们的实现在 ISA 演进的过程中始终有效。这样能够确保今天编写的软件可以永远在类似 RISC-V 核心上运行。某些扩展指令集仍处于草案模式中,因此将来可能发生变化。还有一些被保留,即留下了一些占位符以待未来开发。例如,32 位和 64 位基本整数 ISA 已经冻结,而 128 位和嵌入式变体仍处于草案中。

基本整数指令集(I)是构建其他所有指令的基础,必须包含于任何实现中。除了基本整数 ISA 以外,所有标准 RISC-V 实现必须至少包含特权 ISA 的机器级部分;管理员(supervisor)级(S)和管理程

序(hypervi-sor) 部分则属于标准扩展。然而,特权级 ISA 定义成,如果开发人员愿意,他们可以实现自定义形式的特权代码执行,而不会影响基本整数 ISA。

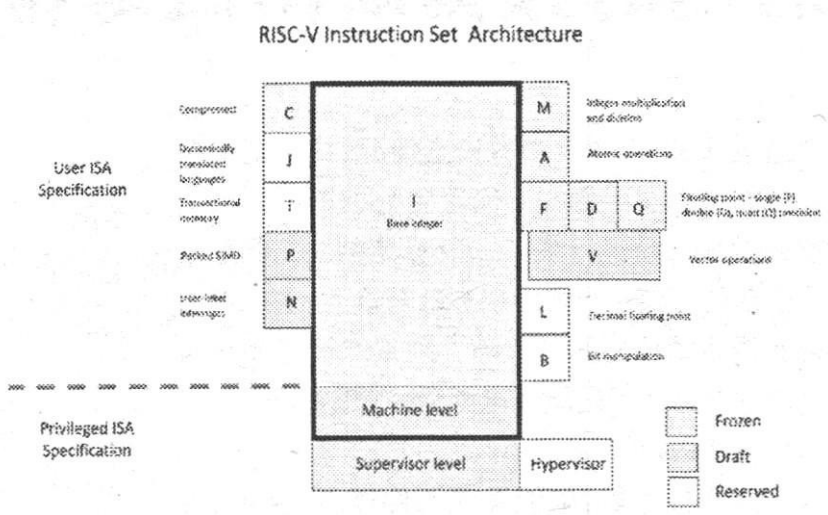


图 1 RISC-V ISA 构成了一个模块化的指令集合,CPU 设计可以使用它来实现,而彼此之间互不干扰。

使用标准扩展来增强基本功能

基本整数 ISA(I)和机器级特权 ISA 提供了基本通用 CPU 所需的所有功能。但是开发人员也可以通过向 ISA 添加扩展来增强这些基本功能。开发人员可以自定义扩展,但是已经有很多标准扩展,RISC-V 基金会技术任务组对其进行管理,从而确保了它们对设计社区具有广泛的吸引力,而且其指令不会与其他标准扩展冲突。因此,开发人员可以自由地在其设计中包含他们需要的任何标准扩展,而无需担心指令编码中会有冲突。这些标准扩展包括:

· M-对两个整数寄存器中保存的值进行乘法和除法的指令(冻结)

· A-对存储器进行原子读、改、写操作的指令,以支持同步(冻结)

· F、D 和 Q-符合 IEEE 754-2008 算数标准的(F)单精度、(D)双精度和(Q)四精度浮点计算的指令。每种精度的扩展取决于当前的较低精度扩展。(冻结)

· G-由于包含基本整数规范(I)以及 M、A、F 和 D 标准扩展指令集的实现非常受欢迎,基金会将这个集合定义为 G,并将 G 配置设置为正在开发的编译器工具链的标准目标。(冻结)

· V-向浮点扩展添加向量指令的指令(草案)

· L-十进制浮点计算指令(保留)

· B-位操作指令(保留)

· N-处理用户级中断的指令(草案)

· P-支持包裹单指令多数据指令的扩展(保留)

· T-支持事务内存操作的指令(保留)

· J-支持动态翻译语言的扩展(保留)

· C-支持压缩指令执行。基本整数(I)规范要求指令字长 32 位,并在存储器中的 32 位边界上对齐。实现 C 标准扩展提供了 16 位常用操作的编码,使 CPU 设计可在自由混合的 32 位和 16 位边界上对齐工作,从而使代码长度缩减.25%至 30%。它可以以任何基本整数位宽和用任何其他标准扩展实现。(冻结)

· S-特权 ISA 的 supervisor 级扩展(草案)

与用户级 ISA—样,特权 ISA 使设计人员可以决定所包含的复杂度。该规范定义了两组 ISA——机器级和 supervisor 级——以及为了支持 hypervisor 功能,而为相应指令保留了一个占位符。反过来,这些指令集又使开发人员能够支持最多三个代码可操作的特权级别(图 2)。单个特权级别(机

器级)意味着所有运行的代码都可以完全访问系统资源,例如在简单嵌入式系统中运行的单个应用程序。具有两个特权级别(需要机器和 supervisor ISA 指令),可以支持部分系统资源与应用程序代码隔离,而增强软件安全性,并使操作系统支持多个并发应用程序。三个特权级别则可支持类 Unix 操作系统和 hypervisor 等。

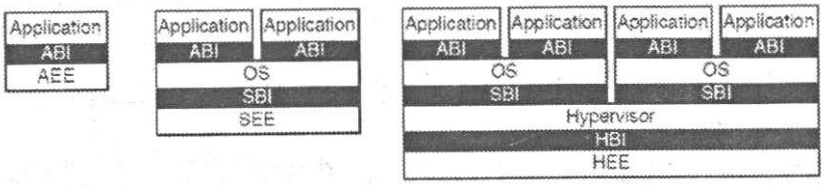


图 2:两个特权 ISA 级别一起可以支持多种软件配置,包括简单的应用执行环境(AEE)、具有 supervisor 执行环境(SEE)的单个操作系统上的多个应用程序,以及具有 hypervisor 的多个操作系统。(来源:RISC-V 基金会)

凭借标准扩展和特权级别的所有可能组合,“ RISC-V ”的简单命名还不足以表征 ISA 的某个实际硬件实现。为了阐明程序员在采用给定硬件实现时可以访问哪些指令,基金会设计了一个核心命名法。如图 3 所示,该命名包含三个部分:所用基本规范(如 RV321、RV641 等)、所加标准扩展(如 M、F、A 等),以及每个元素的版本号(如版本 1.2 标为 1p2 等)。为简单起见,在很多情况下可以省略版本号(如 RV32IC、RV64G 等)。

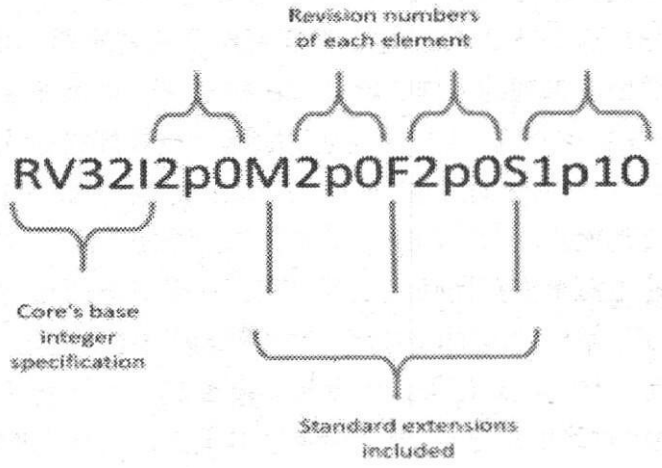


图 3 RISC-V 实现的名字完整地描述了其所支持的指令集。

该命名惯例也可以识别定制扩展。基本整数 ISA 的扩展使用开发人员选择的名称并采用 Xname 的形式,再加上适当的版本号。如果扩展涉及到特权 ISA 的 supervisor 级别,则形式为 SXname。

超越标准扩展

即使模块化标准扩展提供了设计灵活性,RISC-V 标准也没有提供很多开发人员可能希望的指令增强功能。例如,某个应用程序需要频繁对两个 16 位整数(例如两个 ADC 的测量值)进行向下取整求平均值运算,即(r1+r2)/2。若使用基本整数 ISA,计算该所需的平均值需要执行两条指令:整数加法和算术右移(后者实际上执行整数除以 2,并向下取整)。但如果采用一条可用一步执行两个操作的自定义指令,就可以加快应用程序的软件执行速度。如果采用 ISA 的标准指令格式,RISC-V ISA 可以轻松添加这样的指令。

RV32I 基本指令集遵循四种基本格式,如图 4 所示。R 类指令从两个源寄存器(rs1 和 rs2)取值,并以某种方式(add、XOR 等)对它们进行组合,形成一个值并将其存储到第三个寄存器,即目标寄存器(rd)中。Ⅰ类指令获取一个源值(rs1)和一个指令本身中所编码的 12 位值(imm),将其组合并存储到目标寄存器(rd)中。加载指令使用Ⅰ类格式,将源值和立即数组合来确定存储器地址,并将其内容

传送到目标寄存器。S 类指令从一个源寄存器取值,将其存储到第二个寄存器的内容和立即数组合所指向的存储器地址。S 类指令的 B 类变体格式相同,但使用这两个值来计算条件分支指令地址。U 类指令可使用大于 12 位的立即数,而 J 类变体使用该立即数来执行无条件跳转。

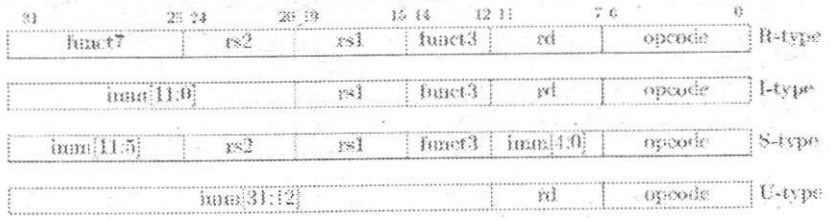


图 4 RISC-V 指令的四种基本格式。(来源:RISC-V 基金会)

这些指令格式的定义方法使得开发人员可以轻松地在混合指令集中添加自定义指令。所有指令都在低位包含一个 7 位操作码(opcode),除了 U 类格式外的所有指令在 12 至 14 位都有一个功能码(funct3)。R 类指令在第 25 至 31 位有一个第二功能码(funct7)。当涉及目标寄存器地址时,它始终位于第 7 至第 11 位,第一个源寄存器地址始终位于第 15 至 19 位,另一个源寄存器则位于第 20 至 24 位。这种代码和寄存器指针安排的一致性意味着,如果用户可以将其新指令映射为这些基本格式之一,并且可以以某个兼容的 RISC-V 设计开始,那么实现该指令的硬件设计几乎已经完成。大多数新操作(例如指令解码和寄存器数据访问)都存在于现有设计中,用户可以充分进行利用。

(未完待续)