1. 为什么要有RISC-V？

*简约是复杂的最终形式。 ——*列奥纳多·达·芬奇

* 1. 导言

RISC-V（“RISC five”）的目标是成为一个通用的*指令集架构（ISA）*：

* 它要能适应包括从最袖珍的嵌入式控制器，到最快的高性能计算机的各种规模的处理器。
* 它应该能兼容各种流行的软件栈和编程语言。
* 它应该适应所有实现技术，包括现场可编程门阵列（FPGA）、专用集成电路（ASIC）、全定制芯片，甚至未来的设备技术。
* 它应该对所有微体系结构样式都有效：例如微编码或硬连线控制;有序，解耦或无序管道; 单一或超标量指令问题;等等。
* 它应该支持广泛的专业化，成为定制加速器的基础，因为随着摩尔定律的消退，加速器的重要性日益提高。
* 它应该是稳定的，因为基础的指令集架构不应该改变。更重要的是，它不能像以前的专有指令集架构一样被弃用，例如AMD Am29000、Digital Alpha、Digital VAX、Hewlett Packard PA-RISC、Intel i860、Intel i960、Motorola 88000、以及Zilog Z8000。

RISC-V的不同寻常不仅在于它是一个最近诞生的指令集架构（它诞生于最近十年，而大多数其他指令集都诞生于20世纪70到80年代），而且在于它是一个开源的指令集架构。与几乎所有以前的架构不同，它的未来不受任何单一公司的浮沉或一时兴起的影响（这一点让许多过去的指令集架构都遭了殃）。它属于一个开放的，非营利性质的基金会。RISC-V基金会的目标是保持RISC-V的稳定性，仅仅出于技术原因缓慢而谨慎地发展它，并试图让它之于硬件如同Linux之于操作系统一样受欢迎。图1.1列出了RISC-V基金会最大的企业成员，作为其活力的证明。

* 1. 模块化与增量型ISA

*英特尔将其未来置于高端微处理器之上，但这还需要几年时间。为了对抗Zilog，英特尔开发了一款过渡产品，并给它起名为8086。它本应该是短命的，没有任何继任者，但事情并非如此。高端处理器姗姗来迟，等它最终出现时，它的性能并不如人意。因此，8086架构延续了下去——它变成32位处理器，最终演变为了64位处理器。它的名称不断变化（80186,80286，i386，i486，Pentium），但基础指令集保持不变。*

——Stephen P. Morse, 8086的架构师[Morse 2017]

计算机体系结构的传统方法是增量ISA，其中新处理器不仅必须实现新的ISA扩展，还必须实现过去的所有扩展。目的是为了保持向后的二进制兼容性，这样几十年前程序的二进制版本仍然可以在最新的处理器上正确运行。这一要求与同时发布新指令和新处理器的营销上的吸引力相结合，导致了ISA的体量随时间大幅增长。例如，图1.2显示了当今主导ISA的指令数量增长过程：80x86。它可以追溯到1978年，在它的漫长生涯中，它平均每个月增加了大约三条指令。

这个传统意味着x86-32（我们用它表示32位地址版本的x86）的每个实现必须实现过去的扩展中的错误，即便它们不再有意义。例如，图1.3描述了x86的ASCII Adjust after Addition（aaa）指令，该指令早已失效。

作为一个类比，假设一家餐馆只提供固定价格的餐点，最初只是一顿包含汉堡和奶昔的小餐。随着时间的推移，它会加入薯条，然后是冰淇淋圣代，然后是沙拉，馅饼，葡萄酒，素食意大利面，牛排，啤酒，无穷无尽，直到它成为一顿大餐。总的来说可能没什么意义，但食客可以在那家餐厅找到他们过去吃过的东西。坏消息是，用餐者必须为每次晚餐支付不断增加的宴会费用。

RISC-V的不同寻常之处，除了在于它是最近诞生的和开源的以外，还在于：和几乎所有以往的ISA不同，它是模块化的。它的核心是一个名为RV32I的基础ISA，运行一个完整的软件栈。RV32I是固定的，永远不会改变。这为编译器编写者，操作系统开发人员和汇编语言程序员提供了稳定的目标。模块化来自可选的标准扩展，根据应用程序的需要，硬件可以包含或不包含这些扩展。这种模块化实现了RISC-V袖珍化、低能耗的特点，而这对于嵌入式应用可能至关重要。RISC-V编译器被告知当前硬件包含哪些扩展后，便可以生成当前硬件条件下的最佳代码。惯例是把代表扩展的字母附加到指令集名称之后作为指示。例如，RV32IMFD将乘法（RV32M），单精度浮点（RV32F）和双精度浮点（RV32D）的扩展添加到了基础指令集（RV32I）中。

继续用我们刚才的类比来说，RISC-V提供的是菜单，而不是一顿应有尽有的自助餐。主厨只需要烹饪顾客需要的东西（而不是每次都做出一顿盛宴），顾客只需要按他们的订单付费。RISC-V无需仅仅为了营销而添加指令。RISC-V基金会会决定什么时候在菜单里添加新的选项。他们这样做只会出于技术原因，而且要在由软硬件专家组成的委员会进行额外的公开讨论以后才进行。这些菜单上的新选择仍是可选的，不会像增量ISA中那样成为未来所有实现的必要组成部分。

* 1. ISA设计 101

在介绍RISC-V这个ISA之前，了解计算机架构师在设计ISA时的基本原则和必须做出的权衡是有用的。如下的列表列出了七种衡量标准。页边放置了对应的七个图标，以突出显示RISC-V在随后章节中应对它们的实例。（印刷版的封底有所有图标的图例。）

* 成本（美元硬币）
* 简洁性（轮子）
* 性能（速度计）
* 架构和具体实现的分离（分开的两个半圆）
* 提升空间（手风琴）
* 程序大小（相对的压迫着一条线的两个箭头）
* 易于编程/编译/链接（儿童积木“像ABC一样简单”）

为了解释我们的意思，在这一节中我们会展示一些以往ISA所作出的选择。它们现在看起来是不明智的，而RISC-V通常会做出更好的决定。

**成本** 处理器通过集成电路实现，通常称为芯片或晶粒。它们叫做晶粒是因为，它们由一些单个的圆形晶片被切割成许多单独的片得到。图1.4显示了RISC-V处理器的晶圆。成本对晶粒面积十分敏感：

*cost ≈ f(die area2)*

显然，晶粒越小，每个晶圆上能切割出来的晶粒越多。晶粒的大部分成本来自于处理过的晶圆本身。不太明显的是，晶粒越小，产率（生产出的可用晶粒所占的比例）越高。原因在于目前的硅生产工艺会在晶圆上留下一些散布的小瑕疵。因此晶粒越小，有缺陷部分所占比重会越低。

架构师希望保持ISA的简洁性，从而缩小实现ISA的处理器的尺寸。我们将在随后的章节看到，RISC-V ISA比ARM-32 ISA简洁得多。就简洁性造成的影响举例，我们把使用相同大小缓存（16KiB）的RISC-V Rocket处理器和采用相同技术（TSMC40GPLUS）的ARM-32 Cortex A5处理器进行比较。RISC-V晶粒的大小是0.27mm2，而ARM-32晶粒的大小是0.53mm2。由于面积大一倍，ARM-32 Cortex A5的晶粒成本是RISC-V Rocket的约4（22）倍。即使晶粒的大小只缩小10%，成本也将以1.2（1.12）倍的比例缩小。

**简洁性** 鉴于成本对于复杂度的敏感性，架构师需要一个简单的ISA来缩小芯片面积。简洁性还能缩短芯片的设计和验证时间，而它们可能构成了芯片开发的大部分成本。这些成本必须算到芯片的成本当中。这个开销取决于发货芯片的数量。简洁性还能降低文档成本，让客户更容易了解如何使用这个ISA。

以下是ARM-32的ISA复杂性的一个明显示例：

ldmiaeq SP!, {R4-R7, PC}

该指令代表EQual上的LoaD Multiple，Increment-Address。它执行5次数据加载并写入6个寄存器，但仅在设置了EQ条件代码时才执行。此外，它将结果写入PC，因此它也执行条件分支。真不少！

具有讽刺意味的是，简单指令比复杂指令更容易被用到。例如，x86-32有一个enter指令，该指令本应该是在进入一个创建一个栈帧的过程中执行的第一条指令（见第三章）。大多数编译器用两条简单的x86-32指令来代替它：

push ebp # 将帧指针压入栈

mov ebp, esp # 把栈指针复制到帧指针

**性能** 除非是那些用于嵌入式应用的微型芯片，处理器的性能和成本通常都能成为架构师的关注对象。性能可以分解为如下三个因素：

即使一个简单的ISA可能在每个程序执行的指令数方面多于复杂的ISA，但它可以通过更快的时钟频率或更低的平均单条指令周期数（CPI）来弥补。

例如，运行CoreMark测试程序[Gal-On, Levy 2012]（100000次迭代）后，得到ARM-32 Cortex-A9的性能为：

对应地，RISC-V的BOOM实现的性能为：

在这个例子中，ARM处理器并没有比RISC-V处理器执行更少的指令。正如我们将要看到的，简单的指令也是最常用到的指令，因此ISA的简洁性是最为重要的指标。对于这个程序，RISC-V处理器在这三个因素中的每一个都获得了近10%的优势，它们加起来导致了近30%的性能优势。如果更简洁的ISA也能催生出更小的芯片，那么其性价比将非常出色。

**架构和具体实现的分离** 架构和实现之间最初的分离可以追溯到20世纪60年代，具体表现为：如果机器语言程序员要编写正确的程序，了解架构是必要的，而性能则不必关心。对于架构师来说，为了在性能和成本上对某一特定时间完成的实现进行优化，而在ISA中包含某些指令，有时候是一种诱惑。但如此做会给其他实现或者今后的实现带来负担。

延迟分支是MIPS-32 ISA的一个令人遗憾的例子。条件分支导致流水线执行出现问题，因为处理器希望下一条要执行的指令总是已经在流水线上，但它不能确定它要的到底是顺序执行的下一条（如果分支未执行），还是分支目标地址的那一条（如果执行了分支）。对于它们的第一个五级流水的微处理器，这种优柔寡断可能导致流水线一个时钟周期的阻塞。MIPS-32通过把分支操作重新定义在下一条指令之后发生，因此分支指令的下一条指令永远会被执行。程序员或编译器编写者要做的是把一些有用的指令放入延迟槽。

唉，这个“解决方案”没有帮助接下来有着更多流水级（因此在计算出分支结果之前取了更多的指令）的MIPS-32处理器，反而让MIPS-32程序员，编译器编写者，以及处理器设计者（因为增量ISA需要向后兼容，见1.2节）的生活变得更加艰难。此外，它让MIPS-32的代码变得更加难懂（参见第29页图2.10）。

虽然架构师不该放入有助于某个时间点的一个特定实现的功能，但他们也不应该放入阻碍某些实现的功能。例如，如上一页所述，ARM-32和其他一些ISA具有Load Multiple指令。这些指令可以提高单发射流水线设计的性能，但会降低多发射流水线的效率。原因在于这种直截了当的实现排除了与其他指令并行地调度Load Multiple的各个负载的可能，从而降低了这些处理器的指令吞吐量。

**提升空间** 随着摩尔定律（Moore’s law）的终结，对性价比进行重大改进的唯一途径是为特定领域（例如深度学习，增强现实，组合优化，图形等）添加自定义指令。这意味着如今的ISA必须保留操作码空间以供未来的提升。

在20世纪70年代和80年代，当摩尔定律如日中天的时候，很少有人考虑为未来的提升节省操作码空间。相反，架构师们重视长地址和立即数字段以减少每个程序执行的指令数（这是前一页上有关性能的方程式中的第一个因素）。

一个能说明缺少操作码空间的弊端的例子是，ARM-32的架构师后来试图通过向以前统一的32位ISA中添加16位指令来缩减代码长度，但根本就没有空间了。因此，唯一的解决方案是先用16位指令来创建一个新的ISA（Thumb），然后同时用16位指令和32位指令来组成另外一个ISA（Thumb-2），并用一个模式位在两种长度的指令间切换。为了切换模式，程序员或编译器会跳转到一个最低有效位为1的字节地址。这种方法能有效是因为16位和32位指令中的该位应该是0。

**程序大小** 程序越小，存储它所需的芯片面积就越小(这对于嵌入式设备来说可能是一个巨大的成本)。实际上，这个问题促使ARM架构师在Thumb和Thumb-2 ISA中追加了一些更短的指令。更小的程序还能减少指令缓存的未命中问题，从而节省了功耗（因为片外DRAM访问比片上SRAM访问耗能更多），也提高了性能。短的代码长度是ISA架构师的目标之一。

x86-32 ISA的指令可以短至1字节，也可以长达15字节。你可能会觉得x86的这种可变字节长度的指令写成的程序一定会比用一些ISA（比如ARM-32，RISC-V）中32位定长指令要更短。逻辑上，可变字节长度指令的程序也应该小于仅由16位和32位定长指令组成给的ISA（比如Thumb-2和使用RV32C扩展的RISC-V，参见第七章）。图1.5显示，当所有指令都是32位长时，ARM-32和RISC-V代码比x86-32长6%到9%，而令人惊讶的是，x86-32代码比同时提供16位和32位指令的压缩版本（Thumb-2和RV32C）大26%。

虽然使用新的可变字节长度指令的新ISA可能会导致比RV32C和Thumb-2更短的代码，但20世纪70年代第一个x86的架构师有不同的担忧。此外，考虑到增量ISA（第1.2节）对于向后二进制兼容性的要求，数百条新的x86-32指令比预期要长。它们承担了一到两个字节长的前缀，这迫使它们使用原始x86的有限的空余操作码空间。

**易于编程/编译/链接** 由于寄存器中的数据访问起来要比存储器中的快得多，编译器在寄存器方面一定要做得很好。这件事在有许多寄存器的时候变得更加容易。鉴于这一点，ARM-32有16个寄存器，而x86-32只有8个。大多数现代ISA（包括RISC-V）都有32个整型寄存器。有了更多的寄存器，编译器和汇编程序员的生活会毫无疑问地更轻松。

编译器和汇编语言程序员的另一个问题是弄清楚一个代码序列的执行速度。我们可以看到，一般每条RISC-V指令最多用一个时钟周期执行（忽略缓存未命中）。但正如我们之前看到的，ARM-32和x86-32都有需要很多个时钟周期执行（即使所有缓存都命中）的指令。此外，于ARM-32和RISC-V不同，x86-32的算术指令操作数可以在存储器中，而不是要求所有操作数都在寄存器里。复杂的指令和存储器中的操作数使得处理器的设计人员难以保证性能的可预测性。

ISA支持位置无关代码（PIC）非常有用，因为这样它就支持动态链接（参见第3.5节），原因在于在不同程序中共享库代码可以驻留在不同地址。PC相关的分支和数据寻址是PIC的福音。虽然几乎所有的ISA都提供与PC相关的分支，但x86-32和MIPS-32省略了与PC相关的数据寻址。

* 1. 全书的总览

本书假设您在RISC-V之前已经了解过其他指令集。如果没有，请查看我们基于RISC-V的相关入门架构书[Patterson和Hennessy 2017]。

第二章介绍了RV32I，它是RISC-V固定不变的基础整数指令集，是RISC-V的核心内容。第三章解释了第二章中没有介绍的其余RISC-V汇编语言内容，包括调用约定和一些用于链接的巧妙技巧。汇编语言包括所有符合规则的RISC-V指令和一些RISC-V指令集外的有用指令。这些伪指令是实际指令的巧妙变体，它们简化了编写汇编语言程序的过程，同时避免了使ISA复杂化。

接下来的三张阐述了RISC-V的标准扩展。当它们添加到RIV32I中的时候，我们统称RV32G（G代表一般）：

* 第四章：乘法和除法（RV32M）
* 第五章：浮点操作（RV32F和RV32D）
* 第六章：原子操作（RV32A）

第3页和第4页的RISC-V“参考卡”是本书中所有RISC-V指令（RV32G，RV64G和RV32 / 64V）的摘要。

第七章介绍了可选的压缩扩展RV32C，它是RISC-V优雅性的一个绝佳例子。通过把16位指令限制为现有32位RV32G指令的短版本，它们几乎是自由的。汇编程序可以选择指令大小，这使得汇编语言程序员和编译器忘记RV32C。将16位RV32C指令转换成32位RV32G指令的硬件解码器只需要400个门，这即使在最简单的RISC-V实现中也只占百分之几。

第八章介绍了矢量扩展RV32V。当与众多强力的单指令多数据（SIMD）指令（ARM-32，MIPS-32，x86-32）相比时，矢量指令成为了ISA优雅性的另一个例证。实际上，图1.2中添加到x86-32的数百条指令都是SIMD，还有数百条指令即将推出。RV32V甚至比大多数矢量ISA更简单，因为它将数据类型和长度与向量寄存器相关联，而不是将它们嵌入到操作码中。RV32V可能是大家从传统的基于SIMD的ISA转到RISC-V的最为突出的原因。

第九章展示了RV64G，它是RISC-V的64位地址版本。正如该章节所说的那样，RISC-V的架构师只需要拓宽寄存器，并加入一些字、双字或长版的RV32G指令，就可以把地址从32位扩展为64位。

第十章介绍了系统指令，说明了RISC-V如何处理分页以及Machine、User和Supervisor权限模式。

最后一章简要介绍了RISC-V基金会目前正在考虑增加的其它扩展。

接下来是本书最大的一个部分，附录A。它是按字母表顺序排列的指令集摘要。它定义了完整的RISC-V ISA以及上面提到的所有扩展，还有大概50页的全部伪指令。这是RISC-V简洁性的证明。

这本书的最后一部分是索引。

* 1. 结束语

此处冯诺依曼名言。

RISC-V是一个最新的，清晰的，简约的，开源的ISA，它以过去ISA所犯过的错误为鉴。RISC-V架构师的目标是让它在从最小的到最快的所有计算设备上都能有效工作。遵循冯诺依曼70年前的建议，这个ISA强调简洁性来保证它的低成本，同时有着大量的寄存器和透明的指令执行速度，从而帮助编译器和汇编语言程序员将实际的重要问题转换为适当的高效代码。

复杂度的一个标准是文档的大小。图1.6显示了以页数和单词数衡量的RISC-V、ARM-32和x86-32指令集手册的大小。如果你把读手册作为全职工作，每天8小时，每周5天，那么需要半个月读完ARM-32手册，需要整整一个月读完x86-32手册。有这样的复杂程度，大概没有一个人能完全理解ARM-32或x86-32。用这种常识来度量，RISC-V的复杂度只有ARM-32的,x86-32的到。实际上，包含所有扩展的RISC-V ISA摘要只有两页（参见参考卡）。

这个袖珍的，开源的ISA于2011年推出，现在由一个基金会提供支持。该基金会以长期讨论后严格依据技术理由添加可选扩展的方式来改进它。开源性让RISC-V的免费的，共享的实现成为可能，从而降低了成本，也减少了将不为人知的邪恶秘密隐藏在处理器之中的可能性。

然而，只有硬件不能组成一个系统。软件开发成本可能使硬件开发成本相形见绌。因此虽然稳定的硬件很重要，但稳定的软件更甚于此。这些软件需要包括操作系统，引导加载程序，参考软件和大众化的软件工具。基金会保证整个ISA的稳定性，而固定不变的基础指令集意味着核心的RV32I作为软件栈的目标永远不会改变。通过它的普适性和开源性，RISC-V可以挑战主流专有ISA的主导地位。

优雅是一个很少应用于ISA的词，但在阅读本书后，你可能会同意我们把它用于RISC-V。我们将用页边的蒙娜丽莎图标来凸显我们认为体现出优雅性的特征。

* 1. 更多请见

……