       The Case for the Reduced Instruction Set Computer

David A. Patterson

加州大学伯克利分校计算机科学系

David R. Ditzel

贝尔实验室计算科学研究中心

介绍  
计算机体系结构最主要的目标之一就是设计比之前产品具有更高成本效益的计算机。成本效益包括用于制造计算机的硬件成本、软件花费、在初始硬件和随后程序调试及测试所需的费用。如果我们回顾计算机家族的过往历史，我们会发现最常见的体系结构的变化倾向于朝向更复杂的机器。假设这一额外的复杂性对于新型产品的成本效益产生积极的平衡作用。在这篇论文中我们认为这一趋势并不总是有益于成本效益的，事实上，弊大于利。我们将针对RISC机和CISC机就成本效益进行验证。这篇论文将讨论下一代超大规模集成电路计算机作为RISC机较CISC机而言，将更加有效。  
随着复杂性的与日俱增，IBM系统/3向IBM系统/38过渡，DEC PDP-11向VAX11过渡。复杂程度一向由控制存储的容量表明，对于DEC而言从PCP-11的256\*56直到现在VAX11/780的5120\*96，复杂性一直在增加。  
   
复杂性增加的原因  
为什么计算机变得更加复杂？我们可以想象得到以下几个原因：  
1.         存储速度 vs CPU速度。John Cocke曾经说过复杂性开始于CPU701到709的过渡时期。701CPU速度是核心主存储器速度的十倍。这使得任何用子程序实现的原语要比用指令实现的原语慢得多。另外浮点逐渐称为709架构的意外收获。709的复杂程度的加深使得较701而言成本效益更高。从那时起，很多“高等级”的指令开始增添到计算机中以提升性能。值得注意的是这一趋势的开始时由于速度的不平衡。但架构师仍无法明确这一速度失衡是否依旧存在；  
2.         微代码和大规模集成电路。与硬布线控制相比，微程序控制使得复杂架构的设计更加节约成本。在60年代末70年代初，集成电路技术的提出使得几乎在任何一种情况下微程序控制的方式都是最经济的方式。一旦决定使用微程序控制的方式，拓展指令集的花费非常微小；仅仅需要存储部分字节。由于控制存储器的大小总是2的几次幂，有时指令集可以无额外硬件开销而仅仅扩展微程序填充控制存储器的代价，变得更加复杂。另外，实现技术的发展导致了传统的子程序在计算机架构实现中的广泛应用。这类指令包括字符串的编辑、整型与浮点型的转换以及数学操作比如多项式求值。  
3.         代码密度。早期的计算机，存储器的价格非常昂贵。因此，总是需要编写非常压缩的程序节约成本。通过增加指令集的复杂性来增加代码密度通常是一把双刃剑，随着更多的指令以及更多的寻址模式需要更多的位来表示。有证据表明代码压缩可以很容易通过清除原始指令集来实现。由于代码压缩变得越来越重要，额外10%的存储通常要比为了所谓的“架构创新”而榨干10%的CPU资源便宜得多。这对于大型CPU而言花费的仅仅是必要的额外电路的封装，但对于单片CPU而言而由于需要控制PLA，性能更可能降低了而非提升了。  
4.         营销策略。不幸的是，电脑公司的主要目标并不是设计成本效益最高的计算机；而是通过销售计算机获得最多的利润。为了销售计算机制造商必须说服消费者认同他们的计算机比其他竞争者的更为优越。更复杂的指令集通常是更好计算机的主要“营销”证据。为了保住他们的工作，架构师必须在保持更新更好的设计。无论实际用途或者成本效益抑或是指令集的复杂程度，指令数目以及它们的“力量”总是被用来宣传一种架构。在某种意义上制造商或者设计师不应该为消费者没有去询问复杂性和成本的问题而受到责备。在硅资源充分的条件下，优质的微处理器总是被用来作为抽奖卡片，因为其实际利润来自于吸引消费者购买相对大规模的存储以适应相对便宜的CPU。  
5.         向上兼容性。与营销战略相一致的是向上兼容的日趋需要。向上兼容意味着改进设计的主要方式是增加新的通常是更复杂的特性。很少会出现指令或者寻址模式从架构中被一出，这导致了在系列机中指令的复杂性和数量的逐渐增长。新的架构常常倾向于囊括已知成功竞争者处理器中的所有和可能是因为架构师和客户没有真正抓住“优秀”指令集的精髓。  
6.         支持高级语言。随着高级语言的使用日益流行，制造商变得渴望提供更有效的指令来支持高级语言。不幸的是几乎很少有证据可以证明更复杂的指令集可以更好支持高级语言。与之相反，我们认为在很多情况下复杂指令集更加不利于支持高级语言。为了支持高级语言而做出的努力是值得赞赏的，但是我们感觉总是没有抓住问题的本质。  
7.         多道程序设计。分时系统需要计算机能够响应中断，并且有能力停下当前的进程稍后重新开始。存储管理以及分页额外需要可以在指令执行结束前停止并在稍后重新执行。尽管这些都未能对指令集的设计产生巨大的影响，但是他们对于实现都有直接影响。复杂的指令以及寻址模式增加了需要在任何中断保存的状态。保存这一状态通常导致影子寄存器的使用以及大规模复杂伪代码的增加。如果没有复杂指令以及寻址模式的负面影响，这一机器复杂性将大大消失。  
   
CISC如何应用？  
软件成本的增加导致了对于高级语言依赖的增加。其中一个影响是编译器正在代替汇编器决定机器执行哪些指令的地位。编译器经常无法利用复杂指令，或者无法使用汇编语言程序验证引以为傲的小技巧。编译和汇编程序员也理所当然的忽略指令集不适用于给定时间域权衡的部分。这导致了总是仅有很小一部分架构被应用。  
例如，对IBM360的特定编译器的测试发现10条指令可以用来解释80%需要执行的指令，16条指令可以解释90%，21条指令可以解释95%，30条指令可以解释99%【Alexander75】。另一项有关各式各样的编译和汇编语言程序表明将CDC-3600指令集减小到现有规模的1/2或1/4，几乎不会损失任何灵活性【Foster71】。Shustek关于IBM370指出经过多次观察发现，很少一部分程序即可以解释大部分程序。比如，COBOL程序，仅执行可用183条指令的84条指令，而其中48条指令解释程序的99.08%，26条指令解释程序的90.28%【Shustek78】。相似的统计结果还会在寻址模式的测试中发现。  
   
CISC机实现的结果  
科技的快速发展以及实现CISC机产生的困难导致了一些有趣的影响。  
1.         更快速的存储。半导体技术的发展已经对CPU和主存速度不匹配的假设造成了印象。半导体存储器速度快而廉价。最近在很多系统中应用的高速缓冲存储器更加降低了CPU和主存速度的差异；  
2.         不合理的实现。可能复杂架构的大部分不寻常的方面都很难有合理的实现。这里我们指的是特殊的、有目的的指令并非总是比一系列简单指令执行得快。其中一个例子有Peuto和Shustek在IBM370发现【Peuto，Shustek77】。他们发现在少于4个寄存器的情况下一系列装载指令比一条多装载指令快得多。另一个例子来自于VAX-11/780。INDEX指令过去常常用来计算数组元素的地址，与此同时检查是否满足边界条件。但是我们在VAX11/780机中发现，它使用一系列简单指令（COMPARE、JUMP LESS UNSIGNED、ADD、MULTIPLY）代替这一“高级”语言指令，并且速度可以提高到45%。此外，如果编译器可以利用数组小标最小为0的优势，速度可以提高到60%甚至更多。显然更少的指令并不意味着更快的速度，“高级语言”指令也不意味着更快的速度。  
3.         设计时间的延长。有时会被忽略的成本之一是开发新架构的时间。尽管CISC机开发的成本可能较低，但是设计的时间却被大大延长。从设计到交付PDP-1仅仅花费了DEC六个月的时间，但是却需要花费至少三年的时间完成诸如VAX等机器的设计交付周期。漫长的设计时间对于最终实现产生巨大的影响。机器或者沿用三年内的旧的技术，或者设计师必须尝试语言新的技术并努力成为应用这种技术制造机器的先锋。显然设计时间的减少对于最终机器的实现非常有益。  
4.         设计错误的增加。复杂指令集的主要问题之一是调试设计，这通常意味着从微程序控制移除错误。尽管很难记录，对于IBM360家族而言这些修改正是面临的主要问题，因为几乎所有的机器都使用只读存储器。370则使用可修改的控制存储，也许可能是为了减小硬件的开销，但最可能是吸取了来自IBM360修改错误的经验。控制存储从装有微代码的软盘加载，这与操作系统相似。当bug被修复时，新的软盘用来升级微代码的版本并进行替换。VAX11/780设计团队意识到潜在存在的微代码错误。他们的解决方案是使用可编程逻辑阵列和1024字节的可写控制存储来修正微代码的错误。幸运的是DEC更乐于公开他们的经历，所以我们已经了解到超过50个错误被更正。很少人相信错误都被发现了。  
5.         RISC和VLSI。单片VLSI计算机设计使得上述有关CISC问题叫多片SSI设计而言更值得批评。如下因素表明精简指令计算机是一种合理的设计方向。  
(1)       实现的灵活性。实现的灵活性很大程度上依赖于在单一芯片上实现完整CPU设计的能力。复杂的架构较精简架构而言，更难在现有技术的条件下实现单一芯片的设计。一个很好的例子是DEC的VAX系列机。尽管最终的模型令人印象深刻，但是复杂的架构使得在现有的设计规则下很难将它实现在单一芯片上，或者说根本不可能。VLSI技术的发展最终使得单一芯片的实现称为可能，单也仅仅在不复杂并且等价于32位架构功能的情况可以实现。因此早期RISC机有利于实现。  
(2)       设计时间。设计难度是VLSI计算机成功的主要因素。如果VLSI技术每两年都可以使片密度至少翻倍，那么设计和调试需要仅仅两年的设计可以使用更搞笑的设计，并且较花费四年的设计而言提升更多的性能。由于新任务的周转时间通常以月份为计算单位，每个错误的更正都会使产品的最终交付延期1/4。普遍延期1-2年，比如Z8000和MC68000。  
(3)       速度。有关性能的最终测试是检测产品执行给定算法的速度。芯片资源的有效使用以及新技术在减少调试时间的应用可以大大提升芯片的速度。RISC速度的提升来源于设计的精简。简单的寻址模式以及指令将导致控制结构的简单。这将导致更小的控制PLA、更小的微代码存储、机器关键路径更少的门。所有这些都可以导致更快的最小周期。如果寻址模式和指令对速度的提升达到10%，额外的对速度的提升将超过10%。目前，我们没有证据显示复杂指令集在这方面也有同样的效力。  
(4)       芯片资源的更好使用。如果你有芯片，为什么不实现CISC？对于给定的芯片关于哪些可以实现由很多折衷。较CISC而言，我们觉得芯片可以通过设计更吸引人的RISC机获得更高的收益。比如，我们认为如果使用片上cache【Patterson，Sequin80】、更大更快的晶体管甚至流水线代替硅，整个系统性能将大幅度提升。由于VLSI技术的发展，RISC建构总是可以领先于同层次的CISC架构。当CISC架构可以在单芯片上实现时，RISC可以应用流水线技术；当CISC应用流水线技术时，RISC开始采用片上cache，等等。对于先进的技术CISC机往往更难实现。  
   
支持高级语言的计算机系统  
一些人总认为简化架构意味着对高级语言支持的倒退。最近的调查【Ditzel，Patterson80】显示“高级”架构并不必须是实现高级语言计算机系统的最重要方面。高级语言计算机系统可以由如下特性定义：  
(1)       使用高级语言编程、调试或者其它用户/系统交互；  
(2)       根据高级语言源程序发现并报告语法错误和执行时间；  
(3)       没有任何从用户编程语言到内部语言的转换；  
另外唯一重要的特性是软硬件协同保证程序员总是通过高级语言与机器交互。在编写或调试程序时程序员无需了解计算机实现的底层情况。只要满足需要，目标就可以实现。这使得使用CISC架构或者RISC架构实现高级语言系统没有任何区别。  
我们从编译器获得的经验启发我们当指令集变得简单统一时，编译器的负担就会大大减轻。能够支持高级功能的复杂指令通常很难由编译器生成。由于指令等级的增长复杂指令逐渐趋向于实现“错误”的功能。这是因为功能变得异常特殊以至于指令对其它操作几乎没有任何用处。复杂指令能够最终由小部分较低等级的指令代替，却几乎没有或仅有一点儿性能的损失。为CISC机设计编译器的时间额外增加，这是因为在生成复杂指令的机器码时更容易产生bug。  
虽然有一些证据显示复杂指令集编译器更容易设计，但是这些编译器却经常没能实现预期的目标。原因如下：首先，过多的使得有很多方式可以实现给定的基础操作，这位编译器和编译器设计人员造成了很大的疑惑。其次，许多编译器设计人员声称他们在处理合理的实现而事实上他们并没有。比如，在VAX11/780机上使用PUSHL R0指令将寄存器压入栈要比使用MOVL R0，-(SP)指令慢的多。对于几乎所有的复杂机器，我们都可以随便举出诸如此类的例子。必须特殊注意的是不能因为有这条指令就使用这条指令。除非彻底摧毁程序的可移植性和编译器的可用性否则这些问题我们被同种架构的不同模型所解决，这是由于相关指令时序的改变需要新的机器码生成器获取优化的指令生成。  
支持高级语言的愿望既包含对HLLCS（High-Level Language Computer System）的实现也包含减少编译器的复杂性。我们很少看到RISC机逐渐不如CISC机的例子，这使得我们可以得出这样的结论：合适的RISC机似乎比CISC机更适于支持高级语言。  
   
RISC架构的应用  
在Berkeley，关于RISC架构的调查在D.A. Patterson和C.H. Sequin的监管下已经持续了几个月。我们认为通过正确选择合适的指令集并且设计对应的结构使得简单指令集的实现更加迅捷。这将导致全局的程序执行速度获得大幅度的提升。这就是RISC的概念。RISC的实现也几乎比CISC需要更少的成本。如果我们可以证明简单的架构与CISC一样有效的支撑高级语言，比如VAX或者IBM S/38，我们就可以认为我们已经完成了有效的设计。  
在Bell实验室。一项由少部分个人参与的计算机设计的工程，以C程序作为测试标准，已经在贝尔实验室计算科学研究中心的监管下进行了很多年了。16位机的原型由A.G. Fraser设计并构建。32位架构由S.R. Boure，D.R. Ditzel和S.C. Johnson复杂研究。Johnson使用迭代技术提出计算机架构、编写编译器、测试结果，从而提出更好的计算机架构，已经重复这样的循环很多次了。尽管最初的意图并不是提出一种加单的设计。结果却是提出了与RISC类似的32位架构，其代码密度与PDP-11和VAX几乎相同。  
在IBM。毫无疑问最好的有关RISC的例子是801迷你计算机，由纽约约克镇高地的IBM研究中心研发。这项工程已经持续了几年并且已经投入大规模的设计团队探索RISC机与先进编译技术的协同。尽管很多他们早期成果的详细信息看起来很特殊。当benchmark PL/I的程序子集时，他们仍就可以获得5倍于IBM S/370模型168的性能。我们可以期待更详细的信息。  
   
结论  
毫无疑问有很多使用“特殊”指令大幅度提升程序性能的例子。但是我们却很少看到将其应用于整个系统时可以获得相同的优势。对于很多计算环境我们认为仔细的精简指令集将导致最高效的实现。当设计指令集时计算机架构师们应该经常问他们自己如下的问题：如果某指令很少产生，那么它是否合理、是否必须、是否可综合，比如Supervisor Call指令；如果指令很少产生并且可综合，那么由于花费大量时间做这样的操作它是否合理，比如浮点操作；如果指令是可综合的并且来自于很多基础指令，那么当指令精简后是否会对程序规模和速度造成影响；指令的实现是否无需额外的成本，比如使用未使用的控制存储或者使用ALU已经提供的操作；如果实现时无需成本的，那么在调试、记录以及未来实现的花费又会有多少；编译器是否能够容易生成这一指令。  
我们认为RISC可以减小复杂性提升性能并且适用于高级语言计算机系统。特别是我们认为VLSI计算机将从RISC概念中获得最多的好处。过去VLSI技术的快速发展一直被作为宣扬复杂架构的良药。在未来的十年我们仍然认为晶体管很珍贵。这以朝向复杂架构的趋势或许是提升计算机性能的一条途径，但是这篇论文却要提出另一条途径——RISC。