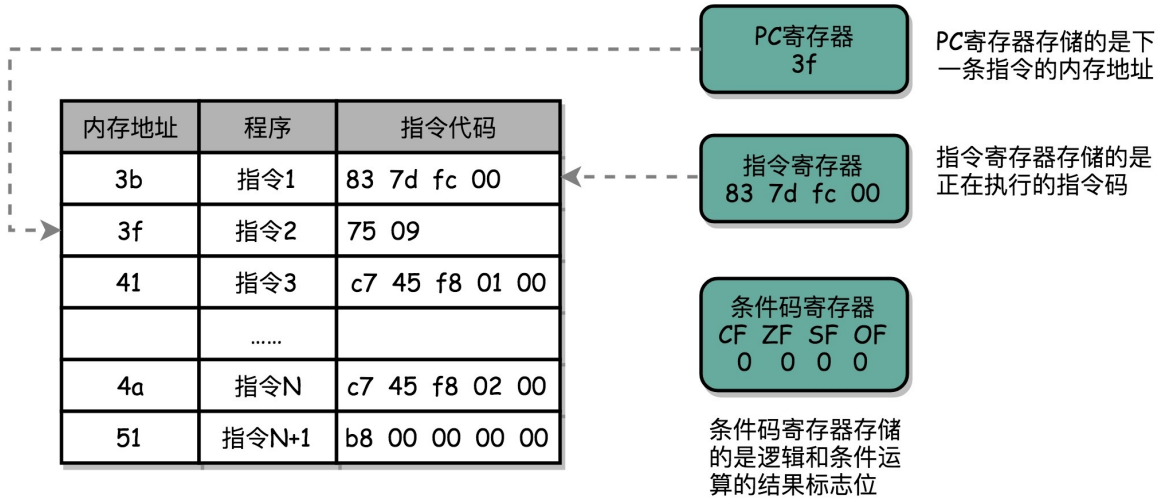
29-CISC和RISC：为什么手机芯片都是ARM？

我在第5讲讲计算机指令的时候，给你看过MIPS体系结构计算机的机器指令格式。MIPS的指令都是固定的32位长度，如果要用一个打孔卡来表示，并不复杂。

指令类型	6位	5位	5位	5位	5位	6位	解释
R	opcode	rs	rt	rd	shamt 位移量	funct 功能码	算术操作、逻辑操作
I	opcode	rs	rt	address/immediate 地址/立即数			数据传输、条件分支、立即数操作
J	opcode	target address 目标地址					无条件跳转

MIPS机器码的长度都是固定的32位

第6讲的时候，我带你编译了一些简单的C语言程序，看了x86体系结构下的汇编代码。眼尖的话，你应该能发现，每一条机器码的长度是不一样的。



Intel x86的机器码的长度是可变的

而CPU的指令集里的机器码是固定长度还是可变长度，也就是**复杂指令集**（Complex Instruction Set Computing，简称CISC）和**精简指令集**（Reduced Instruction Set Computing，简称RISC）这两种风格的指令集一个最重要的差别。那今天我们就来看复杂指令集和精简指令集之间的对比、差异以及历史纠葛。

CISC VS RISC：历史的车轮不总是向前的

在计算机历史的早期，其实没有什么CISC和RISC之分。或者说，所有的CPU其实都是CISC。

虽然冯·诺依曼高屋建瓴地提出了存储程序型计算机的基础架构，但是实际的计算机设计和制造还是严格受硬件层面的限制。当时的计算机很慢，存储空间也很小。《人月神话》这本软件工程界的名著，讲的是花了好几年设计IBM 360这台计算机的经验。IBM 360的最低配置，每秒只能运行34500条指令，只有8K的内存。为了让计算机能够做尽量多的工作，每一个字节乃至每一个比特都特别重要。

所以，CPU指令集的设计，需要仔细考虑硬件限制。为了性能考虑，很多功能都直接通过硬件电路来完成。为了少用内存，指令的长度也是可变的。就像算法和数据结构里的赫夫曼编码（Huffman coding）一样，常用的指令要短一些，不常用的指令可以长一些。那个时候的计算机，想要用尽可能少的内存空间，存储尽量多的指令。

不过，历史的车轮滚滚向前，计算机的性能越来越好，存储的空间也越来越大了。到了70年代末，RISC开始登上了历史的舞台。当时，UC Berkeley的大卫·帕特森（David Patterson）教授发现，实际在CPU运行的程序里，80%的时间都是在使用20%的简单指令。于是，他就提出了RISC的理念。自此之后，RISC类型的CPU开始快速蓬勃发展。

我经常推荐的课后阅读材料，有不少是来自《计算机组成与设计：硬件/软件接口》和《计算机体系结构：量化研究方法》这两本教科书。大卫·帕特森教授正是这两本书的作者。此外，他还在2017年获得了图灵奖。

CISC	RISC
以硬件为中心的指令集设计	以软件为中心的指令集设计
通过硬件实现各类程序指令	通过编译器实现简单指令组合，完成复杂功能
更高效地使用内存和寄存器	需要更大的内存和寄存器，并更频繁地使用
可变的指令长度，支持更复杂的指令长度	简单、定长的指令
大量指令数	少量指令数

RISC架构的CPU究竟是什么样的呢？为什么它能在这么短的时间内受到如此大的追捧？

RISC架构的CPU的想法其实非常直观。既然我们80%的时间都在用20%的简单指令，那我们能不能只要那20%的简单指令就好了呢？答案当然是可以的。因为指令数量多，计算机科学家们在软硬件两方面都受到了很多挑战。

在硬件层面，我们要想支持更多的复杂指令，CPU里面的电路就要更复杂，设计起来也就更困难。更复杂的电路，在散热和功耗层面，也会带来更大的挑战。在软件层面，支持更多的复杂指令，编译器的优化就变得更困难。毕竟，面向2000个指令来优化编译器和面向500个指令来优化编译器的困难是完全不同的。

于是，在RISC架构里面，CPU选择把指令“精简”到20%的简单指令。而原先的复杂指令，则通过用简单指令组合起来来实现，让软件来实现硬件的功能。这样，CPU的整个硬件设计就会变得更简单了，在硬件层面提升性能也会变得更容易了。

RISC的CPU里完成指令的电路变得简单了，于是也就腾出了更多的空间。这个空间，常常被拿来放通用寄存器。因为RISC完成同样的功能，执行的指令数量要比CISC多，所以，如果需要反复从内存里面读取指令或

者数据到寄存器里来，那么很多时间就会花在访问内存上。于是，RISC架构的CPU往往就有更多的通用寄存器。

除了寄存器这样的存储空间，RISC的CPU也可以把更多的晶体管，用来实现更好的分支预测等相关功能，进一步去提升CPU实际的执行效率。

总的来说，对于CISC和RISC的对比，我们可以一起回到第4讲讲的程序运行时间的公式：

$$程序的CPU执行时间=指令数 \times CPI \times Clock\ Cycle\ Time$$

CISC的架构，其实就是通过优化**指令数**，来减少CPU的执行时间。而RISC的架构，其实是在优化CPI。因为指令比较简单，需要的时钟周期就比较少。

因为RISC降低了CPU硬件的设计和开发难度，所以从80年代开始，大部分新的CPU都开始采用RISC架构。从IBM的PowerPC，到SUN的SPARC，都是RISC架构。所有人看到仍然采用CISC架构的Intel CPU，都可以批评一句“Complex and messy”。但是，为什么无论是在PC上，还是服务器上，仍然是Intel成为最后的赢家呢？

Intel的进化：微指令架构的出现

面对这么多负面评价的Intel，自然也不能无动于衷。更何况，x86架构的问题并不能说明Intel的工程师不够厉害。事实上，在整个CPU设计的领域，Intel集中了大量优秀的人才。无论是成功的Pentium时代引入的超标量设计，还是失败的Pentium 4时代引入的超线程技术，都是异常精巧的工程实现。

而x86架构所面临的种种问题，其实都来自于一个最重要的考量，那就是指令集的向前兼容性。因为x86在商业上太成功了，所以市场上有大量的Intel CPU。而围绕着这些CPU，又有大量的操作系统、编译器。这些系统软件只支持x86的指令集，就比如著名的Windows 95。而在这些系统软件上，又有各种各样的应用软件。

如果Intel要放弃x86的架构和指令集，开发一个RISC架构的CPU，面临的第一个问题就是所有这些软件都是不兼容的。事实上，Intel并非没有尝试过在x86之外另起炉灶，这其实就是我在第26讲介绍的安腾处理器。当时，Intel想要在CPU进入64位的时代的时候，丢掉x86的历史包袱，所以推出了全新的IA-64的架构。但是，却因为不兼容x86的指令集，遭遇了重大的失败。

反而是AMD，趁着Intel研发安腾的时候，推出了兼容32位x86指令集的64位架构，也就是AMD64。如果你现在在Linux下安装各种软件包，一定经常会看到像下面这样带有AMD64字样的内容。这是因为x86下的64位的指令集x86-64，并不是Intel发明的，而是AMD发明的。

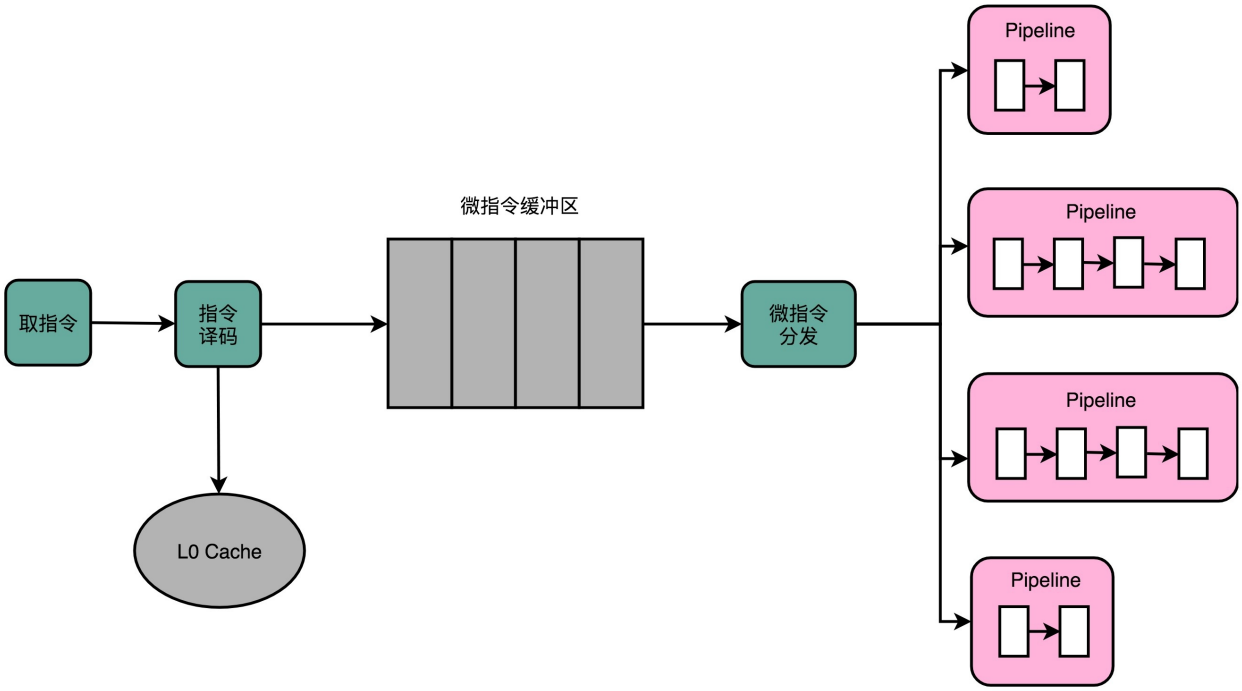
```
Get:1 http://archive.ubuntu.com/ubuntu bionic/main amd64 fontconfig amd64 2.12.6-0ubuntu2 [169 kB]
```

在Ubuntu下通过APT安装程序的时候，随处可见AMD64的关键字

花开两朵，各表一枝。Intel在开发安腾处理器的同时，也在不断借鉴其他RISC处理器的设计思想。既然核心问题是要始终向前兼容x86的指令集，那么我们能不能不修改指令集，但是让CISC风格的指令集，用RISC

的形式在CPU里面运行呢？

于是，从Pentium Pro时代开始，Intel就开始在处理器里引入了**微指令**（Micro-Instructions/Micro-Ops）**架构**。而微指令架构的引入，也让CISC和RISC的分界变得模糊了。



在微指令架构的CPU里面，编译器编译出来的机器码和汇编代码并没有发生什么变化。但在指令译码的阶段，指令译码器“翻译”出来的，不再是某一条CPU指令。译码器会把一条机器码，“**翻译**”成好几条“微指令”。这里的一条条微指令，就不再是CISC风格的了，而是变成了固定长度的RISC风格的了。

这些RISC风格的微指令，会被放到一个微指令缓冲区里面，然后再从缓冲区里面，分发给到后面的超标量，并且是乱序执行的流水线架构里面。不过这个流水线架构里面接受的，就不是复杂的指令，而是精简的指令了。在这个架构里，我们的指令译码器相当于变成了设计模式里的一个“适配器”（Adaptor）。这个适配器，填平了CISC和RISC之间的指令差异。

不过，凡事有好处就有坏处。这样一个能够把CISC的指令译码成RISC指令的指令译码器，比原来的指令译码器要复杂。这也就意味着更复杂的电路和更长的译码时间：本来以为可以通过RISC提升的性能，结果又有一部分浪费在了指令译码上。针对这个问题，我们有没有更好的办法呢？

我在前面说过，之所以大家认为RISC优于CISC，来自于一个数字统计，那就是在实际的程序运行过程中，有80%运行的代码用着20%的常用指令。这意味着，CPU里执行的代码有很强的局部性。而对于有着很强局部性的问题，常见的一个解决方案就是使用缓存。

所以，Intel就在CPU里面加了一层L0 Cache。这个Cache保存的就是指令译码器把CISC的指令“翻译”成RISC的微指令的结果。于是，在大部分情况下，CPU都可以从Cache里面拿到译码结果，而不需要让译码器去进行实际的译码操作。这样不仅优化了性能，因为译码器的晶体管开关动作变少了，还减少了功耗。

因为“微指令”架构的存在，从Pentium Pro开始，Intel处理器已经不是一个纯粹的CISC处理器了。它同样融合了大量RISC类型的处理器设计。不过，由于Intel本身在CPU层面做的大量优化，比如乱序执行、分支预测等相关工作，x86的CPU始终在功耗上还是要远远超过RISC架构的ARM，所以最终在智能手机崛起替代PC的时代，落在了ARM后面。

ARM和RISC-V：CPU的现在与未来

2017年，ARM公司的CEO Simon Segards宣布，ARM累积销售的芯片数量超过了1000亿。作为一个从12个人起步，在80年代想要获取Intel的80286架构授权来制造CPU的公司，ARM是如何在移动端把自己的芯片塑造成了最终的霸主呢？

ARM这个名字现在的含义，是“Advanced RISC Machines”。你从名字就能够看出来，ARM的芯片是基于RISC架构的。不过，ARM能够在移动端战胜Intel，并不是因为RISC架构。

到了21世纪的今天，CISC和RISC架构的分界已经没有那么明显了。Intel和AMD的CPU也都是采用译码成RISC风格的微指令来运行。而ARM的芯片，一条指令同样需要多个时钟周期，有乱序执行和多发射。我甚至看到过这样的评价，“ARM和RISC的关系，只有在名字上”。

ARM真正能够战胜Intel，我觉得主要是因为下面这两点原因。

第一点是功耗优先的设计。一个4核的Intel i7的CPU，设计的时候功率就是130W。而一块ARM A8的单个核心的CPU，设计功率只有2W。两者之间差出了100倍。在移动设备上，功耗是一个远比性能更重要的指标，毕竟我们不能随时在身上带个发电机。ARM的CPU，主频更低，晶体管更少，高速缓存更小，乱序执行的能力更弱。所有这些，都是为了功耗所做的妥协。

第二点则是低价。ARM并没有自己垄断CPU的生产和制造，只是进行CPU设计，然后把对应的知识产权授权出去，让其他的厂商来生产ARM架构的CPU。它甚至还允许这些厂商可以基于ARM的架构和指令集，设计属于自己的CPU。像苹果、三星、华为，它们都是拿到了基于ARM体系架构设计和制造CPU的授权。ARM自己只是收取对应的专利授权费用。多个厂商之间的竞争，使得ARM的芯片在市场上价格很便宜。所以，尽管ARM的芯片的出货量远大于Intel，但是收入和利润却比不上Intel。

不过，ARM并不是开源的。所以，在ARM架构逐渐垄断移动端芯片市场的时候，“开源硬件”也慢慢发展起来了。一方面，MIPS在2019年宣布开源；另一方面，从UC Berkeley发起的RISC-V项目也越来越受到大家的关注。而RISC概念的发明人，图灵奖的得主大卫·帕特森教授从伯克利退休之后，成了RISC-V国际开源实验室的负责人，开始推动RISC-V这个“CPU届的Linux”的开发。可以想见，未来的开源CPU，也多半会像Linux一样，逐渐成为一个业界的主流选择。如果想要“打造一个属于自己CPU”，不可不关注这个项目。

总结延伸

这一讲，我从RISC和CISC架构之前的差异说起，讲到RISC的指令是固定长度的，CISC的指令是可变长度的。RISC的指令集里的指令数少，而且单个指令只完成简单的功能，所以被称为“精简”。CISC里的指令数多，为了节约内存，直接在硬件层面能够完成复杂的功能，所以被称为“复杂”。RISC的通过减少CPI来提升性能，而CISC通过减少需要的指令数来提升性能。

然后，我们进一步介绍了Intel的x86 CPU的“微指令”的设计思路。“微指令”使得我们在机器码层面保留了CISC风格的x86架构的指令集。但是，通过指令译码器和L0缓存的组合，使得这些指令可以快速翻译成RISC风格的微指令，使得实际执行指令的流水线可以用RISC的架构来搭建。使用“微指令”设计思路的CPU，不能再称之为CISC了，而更像一个RISC和CISC融合的产物。

过去十年里，Intel仍然把持着PC和服务器市场，但是更多的市场上的CPU芯片来自基于ARM架构的智能手机了。而在ARM似乎已经垄断了移动CPU市场的时候，开源的RISC-V出现了，也给了计算机工程师们新的设计属于自己的CPU的机会。

推荐阅读

又到了推荐阅读的时间了，这次我们又要一起来读论文了。

想要了解x86和ARM之间的功耗和性能的差异，以及这个差异到底从哪里来，你可以读一读《Power Struggles: Revisiting the RISC vs. CISC Debate on Contemporary ARM and x86 Architectures》这篇论文。

这个12页的论文仔细研究了Intel和ARM的差异，并且得出了一个结论。那就是ARM和x86之间的功耗差异，并不是来自于CISC和RISC的指令集差异，而是因为两类芯片的设计，本就是针对不同的性能目标而进行的，和指令集是CISC还是RISC并没有什么关系。

课后思考

Intel除了x86和安腾之外，还推出过Atom这个面向移动设备的低功耗CPU。那Atom究竟是RISC还是CISC架构的CPU呢？

你可以搜索一下相关资料，在留言区写下你搜索到的内容。你也可以把今天的内容，分享给你的朋友，和他一起学习和进步。

精选留言：

- 靠人品去赢 2019-07-01 15:51:45
  这个intel之前的移动端芯片功耗过大而不能像PC端一样处于垄断地位。那问题来了，既然ARM构架在功耗上有优势，本身PC端功耗也是一个大问题（比如什么N卡电表倒转，农厂大火炉这样的梗），那为什么近些年PC端ARM构架没什么作为，是传说中的生态问题导致的吗？感觉以后不同种类终端都会统一起来，执行一个标准，处处运行会是一个趋势吗？

- 陈华应 2019-07-01 12:49:00
  开阔了视野。另外想插句题外话: 设计思想都是通用的，灵活应用，能选择最适合的才是关键

- 有铭 2019-07-01 09:26:28

Atom仍然是X86 CPU，所以它应该还是“带有微指令架构的CISC CPU”。不过Atom基本已经失败了。说明intel在低功耗CPU的设计上积累还是一般

- xindoo 2019-07-01 08:52:04
  atom是面向移动市场的，所以肯定会以低功耗为设计导向，我觉得肯定会大量参考rsic的设计，但atom貌似是可以运行windows的，应该还是x86的指令集。

- Sentry 2019-07-01 07:59:58
  按老师结尾部分的内容推测，应该是CISC了--