S. F.

我们有必要读这两份将近 40 年的论文吗？

40 年，改革开放也是 40 年，对年轻的计算机科学，是半条命的时间了。40 年前严肃讨论的计算机领域相关问题，是今天应该关注的事情吗？不光我有这个疑问，我们宿舍学霸们竟然都不能区分哪种计算机是 C，哪种是 R。有意思吧？为什么会这样呢？可能是因为，这种计算机类型的划分方法并不会给当前的用户们带来极大的、体验上的差异。

但是，这个毕竟是作业，我还是看看吧，拜读一下。看后我的答案是：是。这种讨论对我们理解计算机系统依然有意义。原因如下：

1. 问题依旧还在

今天，我们真的还在讨论这个问题，还在 R 和 C 之间徘徊，你说奇怪不？

手机和大部分的嵌入式设备（PMD）使用的是 RISC，比如手机的 ARM，ARM 核心（IP 核）针对不同应用领域不同，又可分为 A，R，M 三个系列，AVR、PIC、MIPS、MSP430，这些单片机芯片都是 RISC 结构。TI 的 DSP 芯片，也属于 RISC。当然，C51 单片机是 CISC。

桌面 PC 却依旧沿用 CISC（包括苹果的 MAC、还是 WIN10），以前的苹果机是 POWER PC 芯片的，他是 RISC。还有再早点的 MAC 是 68000，也是 RISC。

大部分服务器是 CISC。有些小型的服务器，文件服务器，小网页服务器，路由器，会用 ARM 或者 MIPS。

巨型计算机是啥有 CISC 的天河（天河三改成飞腾 RISC，是吗？MIPS？），也有 RISC 的太湖（DEC 的 alpha）。当然也有使用 GPU 的 nVidia 的 GPU，应该归入 RISC。

那么，有没有能够满足所有场景的计算机体系结构？你难道不会这样问自己吗？还没有解决这个问题之前，我们确实还要严肃讨论它？

2. 寻找谁让我们复杂，又让我们简单?

是谁，是哪些事情，驱动着我们走向复杂？又是谁，让我们考虑应该从复杂回归简单？为了回答这些疑问，我们背对未来，总结归纳走过的路显得尤为重要。

面对复杂的指令集系统，我们回望计算机的发展史，找到为什么我们会这么做，我们为什么会走上让系统复杂到不能被人类理解的道路上。回到原点考虑看待手上面对的棘手问题，试图找到出路。

3. 讨论是复杂还是精简，其实也是在讨论对世界看法的问题

RISC~断舍离。CISC~洗剪吹。

你要怎么理解这个世界，怎样解决这个世界里的各种问题?

是做一个复杂系统，整体解决所有问题(比如做个能打电话、看电影、听音乐的“手机”)，还是把世界拆解成细小单元（买手机、MP3、MP3)，小步快跑、不急不躁?

对问题求解的看法不同，有人认为要集中力量办大事做复杂的小型机集中处理问题，也有人认为，降低成本使用工业化的方法把计算机送入寻常百姓家。不置对错，只说历史。

引言

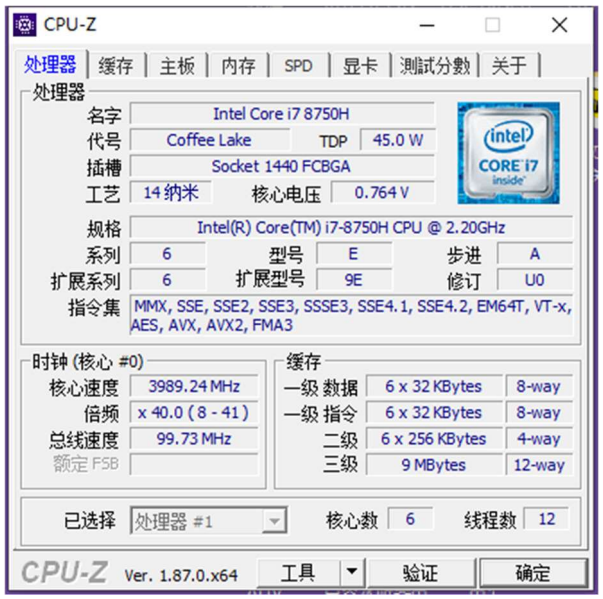
探究为什么计算机越做越复杂?

这种为解决技术发展不平衡导致的计算机复杂度极度攀升，是否会导致，事与愿违的尴尬局面?

何才能实现计算机系统的成本效益最大化? Cost-effectiveness:

1. 工业生产的 cost
2. 编程的 cost
3. 体系结构设计与调试的 cost
4. 硬件初始化和后续的程序段? ? 不是人话，不好理解，但是无所，后面没提。
5. 开发周期和时间
6. 功耗，芯片面积

回望计算机发展史，并举了个例子 DEC PDP-11 到 VAX11，说明我们的计算机系统确实是越来越复杂。好我查了一下，现在的 x86 片子，有多少指令:



MMX：57；SSE：70；

为什么我们会越来越复杂？

文章中，CPU 从简单到复杂的原因，罗列如下：

1. CPU 和 MEM 的速度不匹配，不平衡
2. 微程序和大规模集成电路设计提升
3. 过分追求代码密度，节省内存使用
4. 市场策略
5. 向后兼容性
6. 对高级语言的支持
7. 对多程序流的支持（主要说了中断这个事）

1. CPU 和 MEM 的速度不匹配，不平衡

2. 微程序和大规模集成电路设计提升

3. 代码密度

上面这三个事情其实是一条线上的蚂蚱，我试试能不能一口气把他说完。

为什么二者速度不匹配导致了系统的复杂？因为程序在内存，数据在内存，取指、取数、存数都要访问内存，就像，疯狂动物城里的树懒 Flash（内存），兔子（CPU）再快也是没用的。怎么办？

1. 哈弗双总线的结构，I、D 分开访问，现在的很多 RISC 都是这么做的，单片机都是。就是带宽翻翻。
2. 引入 cache，一开始是外挂，后面变成集成在 CPU 里，还有 I、D 分开，双总线访问。（这是第 1 点）

3. 引入微代码技术（就是第 2 点），把复杂的程序段，封装成指令，比如一个 float 运算的程序段（20 行放在内存），之前在内存里，现在封装成微程序放在 CPU 里，一条指令拿下。这个微代码技术，从原理上讲，特别像是“函数调用”，就是把需要常用的程序段放在 CPU 里，但是这里是用硬件实现。当然，这需要 VLSI 技术的支持。
4. 引入协处理器，一开始是外挂，后面也集成到了 CPU 里，比如 387 浮点协处理器，后面 PSP 游戏机还有“雾化加速器”一种特殊的 GPU。想想打游戏的时候，什么时候会卡顿？？出现雾化效果的时候最容易卡！为什么？
5. 还有各种各样的内存管理机制的引用，比如“页式＋虚拟内存”的引入为了，给每一个程序一种假象，“我在独占计算机”，就是支持多任务的操作系统，那么好吧，需要加上 MPU，内存保护机制（自陷指令），访存越界检查机制，TLB。是不是这些都是你当年看见都想吐的东西？？
6. 还有南北桥引入，显卡引入、流水线技术、多核技术等等，现在的 x86 甚至集成了个显卡进去，诡异吧?

   总之，上面种种，每一个都会让 CPU 及周边或者整个计算机系统规模翻倍复杂。

当然，当年做复杂指令的另一个目的是让程序缩短，占用更少的内存。本来是一个 100 行的 float 乘法，现在是一条指令。你说程序是不是会变短了？但是，作者，大学教授，不差钱，他说，10%的内存价格（cost）要远比压榨 CPU 的 10%的性能，更便宜！

4. 市场策略

好吧，你一个老师，又不卖电脑，为啥要说“市场策略”，看看他说什么？？

*不幸的是：*电脑是拿来卖的，不是拿来看的。追求最效费比最优、性能最优的计算机系统，并不是那些满身铜臭味的硅谷商人的初心。“复杂”比“精简”更有噱头，一代更比一代强，

变成了一种默认选择！你要不断复杂，才好吹牛，才能让别人掏钱，也才能应付上级领导检查。

我的观点，恰恰相反。

我不否认学术研究的伟大意义，我甚至认为毕达哥拉斯把根号 2 扔河里淹死是一件很浪漫的事情。但是，对于计算机这门科学或者产业，能卖出去，才是伟大的一步。我认为，Godson 狗剩和 Intel x86 最大的差别，其实是，在几十年前 x86 已经开启了“正反馈循环”机制，它的产出可以换成钱，养活它的新研，而不是等着领导审批。对 Intel 领导只有一个：市场。毕竟，我们先要让自己活下来。

5. 向上兼容性？？up

其实这个概念，作者有点问题，可能是他那时候还没认识到这个事情吧。兼容两种：向上兼容（向前）、向下兼容（向后）：

向上兼容，烂电脑上写的程序，能在好电脑上跑。

向下兼容，好电脑上写的程序，要在烂电脑上跑。

这两句话，要求极高，不但是要求“同一时期的、性能不同的计算机”，而且是要求“不同时期的计算机”能够跑同一段程序。当然，是一种体系下的。

x86 就是这样。为什么这样？为什么不“轻装简行”再出发？为什么背着这么大一个包袱？而且越滚越大？原因还是同上面说的，电脑是为了卖的，你今天说昨天卖给你的电脑我不再支持了，你要买我今天新出的，那还会有人愿意再买你的东西吗？这可以认为是企业的一种责任吧。

吼吼，还真有，我们的手机怎么解决的在截然不同的硬件平台上(高通、台积电、x86、海思麒麟)运行同一个程序的问题？？虚拟机技术。你的程序和计算机指令无关，比如 Android 的程序策略，尽量和硬件无关，这也是为什么 Android 程序执行的效率差一些。

6. 对高级语言的支持

这个我会留到后面说。

7. 对多程序流的支持

这里提到了中断，主要是说，中断后，需要 CPU 自动完成的保护现场操作，这些是对要对程序员透明的操作，这种透明，增加了 CPU 的复杂度。但是好像这个并不复杂，可能是，每个我用过的 CPU 上都有“中断功能”的缘故。现在的中断，还能动态软件设置优先级（抢占和响应优先级)。有些片子，还能做到，中断响应时间严格可控。(针对实时性要求较高的应用场景）

除了中断这个事情，其实，多程序流的问题非常复杂，只是作者那个时候还没有“敢”考虑这些。比如，多个核心的问题，他们如何共享 cache，私有 cache 内部数据如何同步更新。还有单核多线程的问题、流水线的问题，这些我也不是非常明白，故放下不表。

CISC 复杂指令集，带来的那些事儿

1. 更快速的内存读写速度

引入了 cache 等手段，部分解决了，CPU 和内存之间的速度不平衡问题。不再多说。

2. 不合理的实现

作者举例，在某些（一打）情况下，使用简单短指令“合成”复杂指令功能，会得到更好的运行效果（时间短)。

我觉得，如此比较，并没有任何意义，因为，这种测试的条件也是相当“复杂”、“特殊”，

你用一个“复杂”、“特殊”测试方法或者例子，去论证“复杂指令”不如简单指令，逻辑本身就很有问题，也许是我想多了。

3. 超长的设计与调试时间

cost，我更愿意翻译成，代价，我们都在反复权衡各种代价。

比如，钱，别说这个东西不差钱，那是你不知道钱是啥，如果不缺钱，就应该造 1W 条航母。再比如，设计的复杂度，大小、尺寸、功耗，我做了这么多年的板子，对于更新缓慢的系统，建议，充分实现、留足冗余、力求简洁，这个原则就是再规避风险，让系统合适场景使用即可。最后比如，时间，这是最容易被忽视的成本。因为我们总在低头干活，却不知道活是什么，你的时间就被浪费了。当然，还有学习成本等等因素。我们都是在不断做出权衡，不管有意无意。

具体到作者的观点：

那时候（80 年代），设计一款 CISC 的计算机系统，需要 4 年的时间。

而那时的摩尔定理几乎让大规模集成电路技术，每两年翻一番，可以认为两年一代大规模集成电路技术；而 CISC，4 年设计一个产品出来，那么产品在投放市场那一刻，所用的 VLSI 技术已经是上一代的了。

我认为，这个观点是有点问题的，因为当时还没用计算机设计计算机的技术（计算机辅助设计，CAD、EDA），大部分的图都是手工在纸上绘制的。新一代的设计工具出现，设计效率也在提升。

事实证明，80 年之后的 Intel 几乎是按照摩尔定律提供新的 CPU 的。当然，这个速度再放缓。我们的手机更新也和摩尔定律合拍，一年软件大改，一年硬件大改。

4. 设计风险的大幅提高

复杂度的提升，设计风险就会指数上升，因为，这是一个相互关联的复杂系统，除了各个部件的复杂度导致的 bug，还有部件之间的设计风险，这点比较容易理解。

微指令技术，其实，从这个层面上也是为了减小大规模集成电路设计后可能会存在 bug 而引入的。因为，你只要能保证每个微指令、微操作正确，就能，通过更改微代码（控存），修复 bug。

早期芯片的控存，为了降低成本，在正式大量出货前，使用的是 OTP 存储器，只能写一次。后面，有了可以修改控存的产品，这里面有个 FPGA，大家可以关注一下，它是空白磁带（光盘），是可以造芯片的芯片。很多人可能都知道。但是，你不一定知道还有一种片子叫做 FPAA（现场可编程模拟阵列，使用的是开关电容原理，有兴趣可以关注下）。

作者认为，那时，给出的控存修改方法，体验都不太好。（experiences，乔布斯口头禅）

这其实并不能作为 CISC 风险高的原因，因为，这种风险也会存在在 RISC 芯片中。当然，CISC 复杂，风险高一些，也是正常情况。

我认为，真正的风险是：有些极少数用到的复杂指令，用到的特殊微操作，对应的那块电路逻辑实体，一直开着电，却不经常用，又一直点着火，浪费电不说，它的测试肯定不如简单指令做的完整。

CISC 被设计出来后，我们是怎样好好使用它的？

很遗憾，我们没有怎么用它。

作者引用了几个例子，在证明，只有很少的指令在被大量使用。那些独特的、复杂的指令，被设计出来，但是被使用的概率却极低。这些复杂指令可是占了大部分 CPU 面积和设计调试

资源哟。这是不是很傻。

这点，我确实也是这么想的。但是，我觉得，另一篇论文的论述也非常精彩，下面转成人话：

1. 你光说，不同指令用的频率不同，但是你没有说，在不同的应用里，到底是那些指令用的多，那些指令用的少哟。
2. 比如 A 系统里 top10 的指令和 B 系统里 Top10 的指令，可是不一样的哟。
3. 既然我要做一个通用的计算机，那么，我要全面考虑所有应用的可能哟。

克己复礼，小国寡民，走向 RISC

复杂之后，断舍离，剩下的，拆成 RISC。就像把武术套路拆成，弓步、马步、歇步和拳、掌、沟。武功再高，基本功也就这些。这就是 RISC。

当然这种拆解，并不是要我们退回到计算机的原始部落时期，而是，在借助 VLSI 及大量 CISC 优势技术的前提下，向前回归。

复杂的任务拆解交给谁？聪明的编译器。编译器不够聪明怎么办？让它聪明起来。

1. RISC 实现的可能性和先进性

在 40 年前，作者认为，只有使用 RISC 技术，才能尽早的把整个 CPU 及配套 cache、FPU、SDRAM 控制器等等部件，都封装到一个单独的芯片中。封装在一起，能有什么好处？

1. 接插件、连线的增长，都会导致主频的提升困难，也会增加系统不可靠的风险，毕竟，接插件、电源、功率器件，是一个电子系统里经常出毛病的部件。
2. 我小时候，还见过外挂的 FPU、cache 的 PC 机，你也能从第一章节里看到，每一次把新的部件装入 CPU，计算机的性能就会突飞猛进。

作者认为，RISC 能让这一切来得更早。不过可惜，他低估了摩尔定律的速度。现在的 x86 芯片不但装了上面这些，还装了好几层 Cache、多个 core、系统控制单元，甚至还装了一个小操作系统（MINIX，世界上装机量最大的系统，不知道吧？哈哈）

2. RISC 设计与调试时间优势

RISC 简单，所以设计调试时间短，可以控制在 1-2 年一代产品，和摩尔定律琴瑟和音，紧跟 VLSI 潮流。

举的例子实在有点意思，Z8000，MC68000（后继还有 DragonBall），街机、MD 游戏机、MAC、Palm，都是这个做的。但是吧，很可惜，这俩 RISC 都死了。X86 还活着。

3. RISC 速度

不服，看跑分。给一个算法任务，看谁的最快才是王道。但是，作者在这里也只是在道理上完成 RISC 会比 CISC 的原理推导（那怪第二篇论文，上来就在说，你没做过能够达到我性能水平的 RISC，你凭什么说我的就不好？）：

1. 更快的设计周期，更好的使用硅片的面积，更早的应用 VLSI 新的技术，就能有更好的 CPU 速度，和程序运行速度。
2. 更少的引址模式和指令，可以大量简化控制电路结构，就可以有更少的 PLA 电路，更小的微代码及控存。电路的“延迟线”越长，数字逻辑电路的竞争风险越大，基准频率就不能往上提升。
3. 减少关键路径上的逻辑门数量，是提高 CPU 频率的重要手段。

4. RISC 更好的利用硅片面积

这点，已经揉到前面了，不多说。我估计作者也是东拉西扯，没话说了。

对“高级语言计算机系统”的支持

实话实说，我就没看懂，啥事这篇文章说的“高级语言计算机系统（HLLCS)”，难道是支持高级语言，比如，就是支持 Basic、C、C++这样的高级语言的计算机系统吗？现在看如此基本的事情，80 年代，都说的这么复杂吗？还是我理解浅薄了？反正，作者提出里三点要求：

1. 用高级语言书写、编译、调试程序，以及人机接口交互（现在多是图形界面交互)。
2. 在高级语言的源代码中，发现、报告（标注）代码中的警告或执行错误。(这点现在也是 IDE 的基本功能）
3. 不需要一个明显的翻译机（编译器？）完成高级语言到内部机器语言的转换。(这点，我最不理解，难道那时候的电脑还需要额外的机器完成 C 到 ASM 的翻译吗？这么神奇？）

作者抛出观点：

只要上面这三个目标需求被满足就行了。没必要，让计算机指令和高级语言一一对应。也可以化整为零，化成简单高效的指令，小步快跑，难道不好?

作者观察那时候的编译器，发现：

如果指令集简单并且规范，那么，写编译器的通知就会比较轻松。复杂指令被编译器生成使用的比例想到少；复杂指令在使用的时候容易被编译器用错；因为这些复杂指令实在太特殊了，特殊到只有某一种特殊计算才能用到他们，而对其他操作都是无效的。

比如，教授研究巨型机的，我研究搬砖的，假如有天全世界就只用一台计算机就行了，那我还能找到饭吃，教授不能。

过量的，能够实现同一功能的指令，或者指令序列，会让编译器或者其开发者困惑，在某一类情况下，我到底要用哪一种指令呢?

我们现在 CISC 的指令并没有做到“一条指令的名字和它应该具有的属性相统一”。比如 XX 里，压栈时间，比 MOV 指令还要慢，这个实在让人很难理解。

其实同样的问题，也会出现在 RISC 里面。起初，RISC 的那一组通用寄存器，也会让编译器迷惑，这么多寄存器，我该用那一个好呢？？这个问题，目前有一些标准化的答案，后面会有提到。

当前的 RISC 工作进展

Berkeley、Bell、IBM

结论

你能找到很多例子，证明，一个特殊的指令，可以改进某一段程序运行的速度。但我们很少看见，有例子可以证明，这些复杂指令能够让系统整体性能提升。因此有必要对 CISC 完成必要剪裁。剪裁时，应该问自己：

1. 这条指令是否是一条不常用的指令？是否确定无疑的合理？并且不能被简单指令合成？
   比如自陷指令
2. 是不常用的，可以被合成的，就是累赘，就该删除，比如 float（我不这么想）
3. 如果可以被合成，那么要考虑扇区后，是否对代码量和速度又严重的影响
4. 这条指令，是否能被 CPU 其他执行部件作为“副产品”直接生成产出？

很有必要断舍离，达到最大化性能提升。建议，C 和 R 努力实现 HLLCS，然后比较一下孰优孰劣。

没有提到的关键问题

功耗低、面积小

这是我认为，在移动端，或者嵌入式领域，ARM 或者 AVR 或者其他的 RISC 能够打败 X86 的主要原因。

通用寄存器的数目，RISC 寄存器数目远大于 CISC

RISC 越多越好，这个事情并不是以开始就被人们意识到的，因为，多了选择就多，人难选择，编译器更难选择。后面，有了“图染色法”等等问题，让编译器更聪明，才体现了 RISC 的价值

90 年代，那个第二篇论文的作者道格拉斯，CISC 的坚定支持者，发论文，说，RISC 的通用寄存器还是牛逼的。

RISC、CISC 都能打太极

CISC 的架构调整也在不停的调整，RISC 也是。

你能看到，RISC 也在用微代码技术，我听沈立老师的公开课，一直认为 RISC 的控制逻辑完全是用组合/时序逻辑电路搭起来，但其实不是。

你也能看到，CISC 的代表 x86 系统，在奔腾的后继体系结构中，应用了多种技术，按照网上大神的说法，现在 x86 留下的只是 ISA 的外科，其内核，已经和 RISC 趋向一致。

让我想到一句话，活到极致的人都是雌雄同体的，既有女人的柔软，又有男人的坚毅。反正，就是这么一说，放到这里，我也不咋想得明白。

我的读后感

1. 当你遇到一个问题，你要想办法解决它，你会用各种手段，即便手段可能会部分解决问题，但它100%会带来更多新的问题。即便你暂时还没有发现新的棘手问题，那么他在复杂度上的提升，也总会让你看不清原有的问题是什么。
2. 尽量少拿例子，完成证明。因为，你总能找到，跳大神治好的病人，也能找到做手术牺牲的病人。有时候同一个例子能找到无数理由。考虑事情满足自洽、他恰、续恰是最重要的，不需要例子来点缀，“例子”只是带入计算的过程，或者方便理解的手段。
3. 不要确信你的预测，你的预测都可能都不靠谱，要背对未来，面对历史，完成总结。
4. 由俭入奢易，由奢入俭难。但天生我材必有用，看你做的东西给谁用，要解决什么问题。