计算机组成原理课中的工程师思维

杨 铮
清华大学

关键词：工程师思维　解决问题　计算机

每个人看到“工程师思维”这五个字都有自己的理解。在我看来，工程师思维中最重要的目标之一是要办成事，因此一切思维和方法都是面向解决问题的。许多学生申请学校时，在简历或个人陈述(Personal Statement) 中介绍自己具有 problem-solving 的能力。这个“problem-solving”，我认为就是工程师思维的一个体现。

面对具体问题，在特定的环境下思考针对这个问题的解决办法，即《孙子兵法》中的“势者，因利而制权也”。例如十二诡道中的后八个，“利而诱之，乱而取之，实而备之，强而避之，怒而扰之，卑而骄之，佚而劳之，亲而离之”——也就是俗话讲的“兵来将挡，水来土掩”。有办法并且坚信办法总比困难多，是工程师思维的重要原则。

计算机作为拥有信息表示、运算、存储以及其它信息处理能力的复杂系统，除了以计算机理论作为基石，其设计与实现还处处体现了工程师思维，特别是体现着工程师解决问题的思路。注意，是思路而不仅仅是结果。从这个角度考虑，学习“计算机组成原理”课程就是理解和掌握工程师思维的最便捷方式之一。

以有限的资源办成事

做任何事情都是有代价的，即使不是钱的代价，也会是其他方面的代价。工程师做事应先考虑代价。信息的表现形式多种多样，如整数、小数、文字、图片、音乐、视频等等，而计算机只能存储非 0 即 1 的比特串。因此，在计算机中表示信息，需要有一定的规则。以整数为例，我先考虑一下一个整数需要用多少比特来表示？整数的数量是可数无穷的，而计算机中的比特是有限的，只能表示一定范围的整数。尽管表示范围有限，但计算机中常见的整数表示方法（例如 32 比特补码表示）可以满足现实中很大一部分需求，而且处在表示范围内的整数都能精确表示。进一步考虑小数，情况就更加复杂，不仅范围有限，精度也有限。比如，有限的比特串就不能精确表示 0.1 这个重要的小数。换句话说，想精确表示 0.1 这个小数，多少比特都不够用。我们只能用有限的比特表示合理范围内的、合理精度的小数，通常，这个比特数是 32 位或 64 位。那么如何表示呢？一种思路是借鉴整数表示方法，将比特串分为两部分，用整数形式来分别表示小数的整数部分和小数部分。工程师进一步思考，能否在有限的 64 位比特上表示更大范围、更精确的小数？这个思考催生了浮点数，即使用二进制科学计数法（符号 s、尾数 M、阶数 E）来表示小数 $(-1)^s M2^E$，其中 M 和 E 分别编码为 *frac* 与 *exp* 存储在计算机中（如图 1）。有了浮点数，表达范围从不足 2^{64} 提升到约 $2^{2^{10}}$，表达精度从不足 2^{-64} 细化到最小 $2^{-2^{(52+10)}}$）。

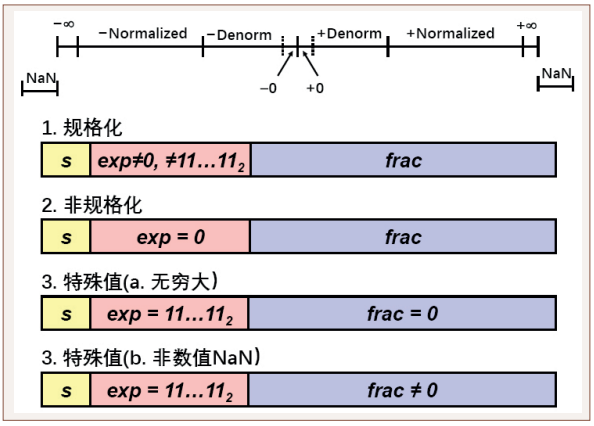

图1　IEEE 754标准中浮点数表示方法和范围

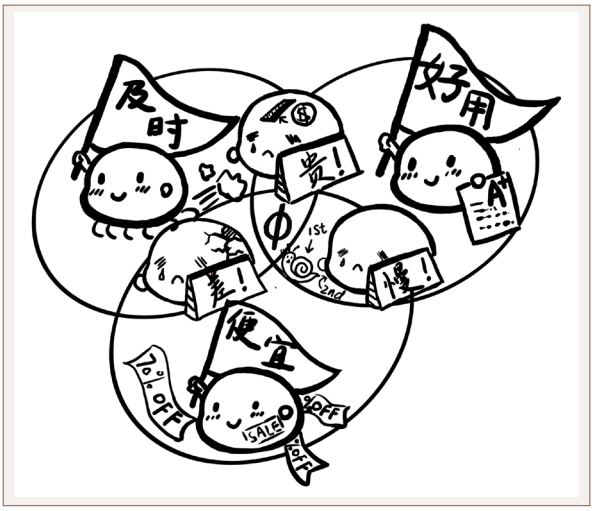

图2　在“好用、及时、便宜”三个维度中寻找平衡

再进一步，在64位比特中，是否还有利用空间呢？有。在二进制的科学计数法中，小数点前总是1，即都是1.xxx的形式。因此，IEEE 754浮点数标准规定，这个1也可以省略，节省出1比特的空间分配给小数点后的部分。这样就“凭空”增加了1位精度。然而，既然这个1是默认隐含的，那么0就无法表示了，这是个巨大的遗憾。所以，除了规格化情况(Normalized)，IEEE 754还规定了浮点数的非规格化(Denorm)情况，即当阶数是特定值时(00000000000,11个0)，没有这个默认隐含的1。这样一来，IEEE 754就可以精确表示0以及更加接近0的数值。到此为止，64比特的资源得到充分利用。

办成事需要资源，但没有人拥有无限的资源。学生有学生的难处，老师有老师的难处，校长有校长的难处；乞丐有乞丐的烦恼，马云有马云的左右为难，李彦宏有李彦宏的焦头烂额；神舟飞船还有预算限制，建不建对撞机也要经过论证……做任何事情的资源都是有限的，以有限的资源办成事（或以最小的代价办成事），需要的是真本事。

在多维度中寻找平衡

既然资源总是有限的而目标又是多元的，那么就需要在多维度中寻找平衡。无论是买东西、卖东西、上学、找工作，只要面临选择，就需牢记一般情况下“好用、及时、便宜”三者不可兼得（如图2）。如果想既便宜又好用，那就要接受慢；既便宜又及时，那就不好用；既好用又及时，那就贵。人生中的许多事情都是这个道理。

如果把“好用”替换成“存储容量大”，那么图3就比较准确地描述了计算机存储结构的设计思路。寄存器访问速度最快，但是存储容量最小且价格最贵；高速缓存访问速度比较快，容量稍大，价

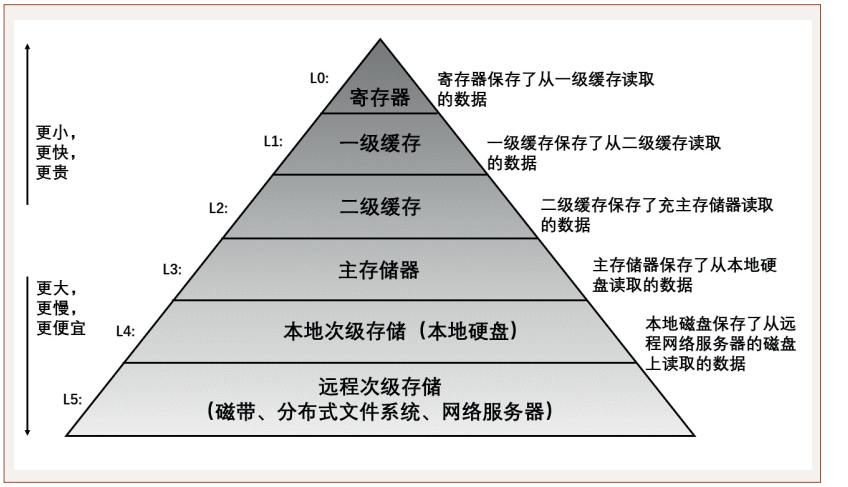

图3　存储器体系结构

格比较贵；主存储器访问速度一般，容量一般，价格也一般；磁盘访问容量最大，价格也便宜，但是速度慢。似乎没有两全其美的办法，但工程师也不愿简单地选择一种存储介质来达成在多个维度（速度、容量、价格）之间的平衡。

实际上，存储器系统是一个具有不同容量、成本和访问时间的存储设备的层次结构（如图3）。CPU内的寄存器保存着最常用的数据。靠近CPU的小的、快速的高速缓存存储器作为缓冲区域，负责暂存原本存储在相对慢速的主存储器中的一部分数据和指令。同样地，主存是暂时存放那些存储在容量较大的、慢速磁盘上的部分数据。而这些磁盘常常又作为存储在通过网络连接的其他机器的磁盘或磁带上的数据的缓冲区域。

聚焦到高速缓存，也处处体现出平衡的艺术。各级高速缓存通常按照组、行、块的方式组织，块大有助于利用程序的空间局部性，但块越大意味着行越少，不利于时间局部性，不命中惩罚也提高了（直接映射(direct mapped cache)就是行数等于1的极端情况）；相连度提高（行数增加）可以降低冲突不命中带来的抖动，有利于程序的时间局部性，但是实现比较复杂，需要额外的控制逻辑，成本较高，也会增加命中时间（全相连(full associative cache)就是行数最多的极端情况）。因此，需要选择合适的块大小及行数，在各项参数中保持合理的平衡。当然，为不同级别的高速缓存选择不同的配置，也是灵活性的体现。例如，在Intel Core i7中，L1和L2是8路组相连（行数=8），L3是16路组相连（行数=16）。

未知问题转化为已知问题

我曾问过一个数学牛人，解决一道数学题的思路是怎样的。他回答说，其实最先想到的是有没有见过这道题，或者是类似的题目。这个回答太坦诚了——我原以为是“数学思维”之类的“高大上语言”，后来发现，能快速解出大部分考试题目就是因为见过。千万不要小看这个“见过”，不仅需要记忆力，还需要透过现象看本质的洞察力。有些人是见过但没有记住，大部分人是见过但找不到联系。

将未知问题转化为已知问题是工程师思考问题的方式。在翻译机器指令时，就需要反复使用这种转化能力。机器级指令只有跳转指令(jmp)、条件跳转指令(jxx)，不支持高级语言中的分支指令（if-then-else，switch等）以及循环指令（do-while，while，for等）。假设我们已经掌握了将do-while循环翻译为机器级指令的方法，则无须再重新探索while或者for循环翻译成机器指令的方法，因为while循环是在do-while循环的基础上先判断一下循环条件，而for循环也是在while循环的恰当地方加上循环变量初始化与循环变量的更新（如图4）。这样转化的好处是逻辑清晰，从而提高准确性。

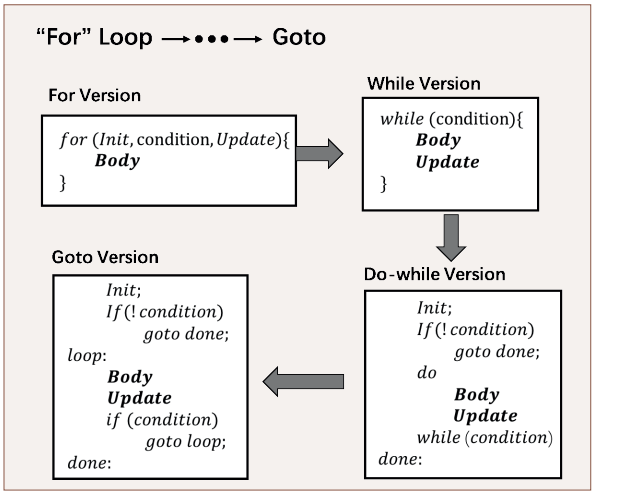

图4 do-while、while、for循环的转化

大问题分解为小问题

“如何把大象放到冰箱里”这个经典的问题，就是将大问题分解为小问题的例子。通常，标准答案是分为三步，即“把冰箱门打开，把大象塞进去，再把冰箱门关上”。这三步中最受人诟病的是第二步，因为难度大，又没有说清楚如何解决。但是换个角度看，这样三步走的策略至少把冰箱门的问题解决了，部分解决也是解决。第二步还可以继续分解，然后再解决其中一部分问题。一个困难问题的

解决，整体上必然像剥洋葱一样层层展开，局部上需要深耕细作以及灵光闪现。

在“计算机组成原理”课程中，最难理解的部分是CPU流水线的设计与实现。如果我们按照“大问题分解为小问题”的思路重新整理一下这部分内容，就可以用一句稍长的话来解释：顺序执行指令的处理器效率太差（问题），需要引入流水线技术使得多条指令同时执行（出路）；但是指令的数据相关与控制相关会造成流水线运行结果不正确（问题），需要插入空操作指令来间隔两条相邻的指令（出路）；大量无效空操作降低处理器效率，让处理器退化为顺序执行（问题），采用数据转发机制，将线路上还没有写入寄存器的数据用作下一条指令的输入（出路）；数据转发不能解决所有数据相关和控制相关（问题），特殊情况要使用暂停和气泡机制（出路）。总结一下，随着一个问题的解决，又暴露出了一个新问题，在层层解析的过程中，问题规模逐渐变小，可解决的情况不断增加，直至整个问题的解决。这一过程当然并不容易，需要清醒的大局观又专注于细节，但是经历这一过程把问题解决后，所获得的快乐也是不一般的。

遵循章法

我高中时认为，当时的物理老师在解题时有些循规蹈矩，甚至有点死板。以受力分析问题为例，无论多简单的题目，都要画坐标系，然后在坐标系中画出重力、摩擦力、牵引力等等。当时我觉得很多题目明明一目了然，何必每次都画，那么麻烦。上大学后我辅导一个高中生做物理题，题目是一根绳子拉着一个物体静止在斜面上。让我意外的是，他胡乱画出了扯力、顶力、拉力等等，这也让我彻底改变了高中时对那位老师的看法。

我终于明白高中物理老师的做法是多么高明，复杂问题是不能单纯依靠直观思维来解决的，往往需要从简单的情况中提炼出章法，再循序渐进，把章法练到纯熟，才有能力解决更复杂的问题。物理老师那看似繁琐的重复，其实就是在培养章法，把握问题的核心，做到了这一点，再复杂的问题，都可以一眼看到本质，而不会受到困扰和迷惑。

寄存器的使用就要严格遵守规矩。在调用函数时，哪些寄存器是调用者负责保存的，哪些是被调用者负责保存的，都要有规矩（如图5）。这些规矩不是严格推理出来的，而是约定俗成的。《格列佛游记》[1]中记载了吃鸡蛋应该从大端还是小端打破的争端，并由此引发出了两个国家的战争，造成一位皇帝送了命、一位皇帝丢了王位、成千上万人流离失所。根据这一故事，计算机中的两种不同的数据存储方式分别命名为“大端”和“小端”，不同的计算机软硬件系统采用不同的方式。在一个特定

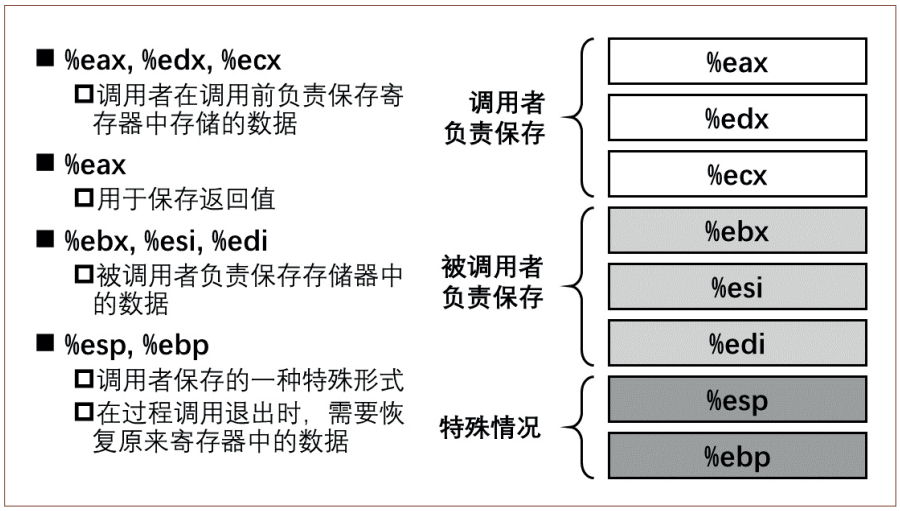

图5 寄存器使用惯例

[1]《格列佛游记》是1726年出版的一部长篇游记体讽刺小说，由英国作家乔纳森·斯威夫特创作。小说讲述了英国外科医生格列佛周游“小人国”“大人国”“飞岛国”“慧骃国”的奇遇。

的软硬件环境下，根据约定的规则来存储数据；不同软硬件环境交互时，做好数据转换工作。

用词准确也很重要，这也是章法的一部分。互联网的五层架构模型，从上到下依次是应用层、传输层、网络层、链路层、物理层。数据以一定分组的比特为基本单位，在每一层上进行传输。这个数据的基本单位在应用层称为消息 (message)，在传输层称为数据段 (segment)，在网络层称为数据包 (datagram)，在链路层称为数据帧 (frame)，在物理层称为符号 (symbol)、比特 (bit) 等。如果不特指某一层，用 packet 是挺合适的。不用或者乱用这些词只会增加理解的难度，没有任何好处。

所谓工程师思维，总结起来，就是“有办法、有限制、有取舍、有分解、有转化、有章法”。现代社会中，工程师思维越来越重要，对不做工程的人亦然。在愈发复杂的社会分工协作链条中，人们的付出与回报的间隔越来越长，目标和行动看起来越来越缺乏直接联系。以前，原始人打到猎物就能吃饱，打不到就饿着；今天，辛苦工作一个月，月底才能拿到工资；辛苦写了一堆代码，产品没交付甚至没盈利就拿不到奖金。在漫长的从具体行动到达成目标的路途上，没有工程师思维来规划旅程，恐怕是举步维艰的。

社会发展滚滚向前，以后我们每个人的生活，就是一个人要像一支队伍，一件事要干成一项工程。■

（本文最初是写给我所教授的“计算机组成原理”课程的学生，语言更活泼一些，希望可以激起学生的学习兴趣。本文虽然经过修改，仍不免会有失严谨，也请大家多多指正。）

杨　铮

CCF专业会员、普适计算专委会委员。清华大学软件学院副教授、博士生导师。主要研究方向为物联网，包括传感网、Mesh网络、RFID、室内定位、群智感知、移动数据挖掘等。

yangzheng@tsinghua.edu.cn