维普资讯 http://www.cqvip.com

硬件体系结构与程序设计语言的相互支撑

N. Wirth

摘要

程序设计语言与操作系统引入抽象概念使程序员可以忽略实现细节。支撑抽象的主要目的不只是提高实现效率，而且也提供必要的保护以避免违背抽象。由于对效率的狂热追求，第二个目的被忽视了。种种迹象表明，近来一些声称为简单而有效的设计是通过把代码生成与适当的保护这两者的复杂问题推卸给编译程序来达到效率的。

软件设计与硬件设计的共同特点是其复杂性，这是通过通常的测试与模拟所控制不了的。硬件设计可以采用那些类似于程序设计中所使用的证明技术，从而利用程序设计方法学的研究成果。

支撑抽象

计算机的用途是执行计算输出结果。我们希望在公理和抽象的数学框架内利用语言这种形式体系来规定计算。计算机以特定的机器代码形式解释程序。语言与计算机体系结构的设计应该协调一致，使得程序正文到代码的翻译成为可能、有效和可靠。结论是把计算机、语言和编译程序视为一体，把这三个成分设计成一个和谐的单元。

长期以来，程序设计的概念一直影响着机器的设计。我们可以用四个例子来进行说明。其中每一个例子针对一种特定的抽象并在计算机内产生了对应的表示。

数 由于用二进制序列来作为数的具体表示，就可能把算术运算作为指令系统中的单独一组指令，程序员不再需要知道基本表示法。这种抽象是由体系结构支撑的。

数组 变量的数组概念促使引进变址寻址方式与变址寄存器，这就有可能有效地实现数组的抽象。

过程 把算法分解成由若干部分组成的层次结构这一概念导致引入了子程序调用与返回指令。它们支撑了过程与函数的抽象。

参数与局部变量 过程的一个很重要的性质是它们拥有局部对象。局部变量(及参数)的实现是用相对寻址方式与基地址寄存器支撑的，它们有效地存取、分配与释放局部对象。

抽象的本质在于程序员可以忽略(即抽象掉)其实现细节。如果不确实如此，抽象就失去了价值。因此，机器设计师必须提供保护以防违背支配抽象的规则。我们现在简单地研究一下怎样清楚地理解这一基本要求。

算术运算 如果算术运算的结果违背了所选取的数的表示法的基本规则(即在溢出情况下)，那么这种算术运算显然必定失败。然而，在绝大多数系统中，溢出问题往往忽略掉了。算术运算是不可信赖的，它充其量只能看作是取模算术运算。而且，在绝大多数计算机上，如果整数除法的被除数为负数，那么也违背了既定的数学规则，即对所有x与y规定

$(x \text{ DIV } y)*y+(x \text{ MOD } y)=x$ and $0<=(x \text{ MOD } y)<y$。通常实现的整数除法都是关于零对称的。这种"除法"不仅相当无用，而且也不能用简单的移位指令来有效地实现除以2的幂的除法。

数组 在抽象意义上，对不存在的数组元素的引用是没有定义的，从而必定导致明显的失败。然而，在绝大多数已投入使用的系统中都表现数组界

维普资讯 http://www.cqvip.com

作检查。使用无效下标会产生一个随机值，在赋值时这会导致灾难。抽象是不可信赖的。只是近来才在体系结构中引入了下标界检查指令，把所谓的"开销"减少到一定程度，为关心效率的程序员所能接受。

过程 经过长期争论，人们普遍认为，如果从过程抽象中去掉不自然的限制，那么过程就必须是可递归调用的。递归的意思是，每一次调用时都要为局部对象分配存贮单元并在每一次返回后释放。有些现代体系结构为此提供了特殊指令。这些特殊指令的一个主要任务就是检查所请求的空间是否可用。然而，这种检查确确实实还是被忽略掉了：这种抽象可能容易被违背，从而也就不可信赖。

在所有这些情况下，编译程序设计者都面临进退两难的选择。可以生成指令序列使程序值得信赖但效率低，以便绕过硬件缺陷，从而提供正确的实现；也可以将看得见的体系结构的缺陷留给程序员去处理，从而在设想由语言提供的抽象层上进行妥协。在这两种情况下，体系结构的不适应性终会暴露。因此，所需要的不仅仅是要支撑作为语言基础的抽象，而且还需要一个坚实的基础，使抽象正确、一致而且完备。

最后之争：速度

过去，人们对抽象的安全性往往作了一些牺牲，这是由于实现者低估了实现的正确性与完备性的重要，也由于计算机设计者一直把效率作为首要目标。不幸的是，他们对效率的看法似乎集中在那些可以应用传统度量尺度的特性上，即指令周期与内存容量。

为提高指令系统的适用性所做的工作主要是测量已编译代码中出现的操作频率与短操作序列的频率。把经常遇到的序列编码成单独的指令，就会导致相当复杂的指令与寻址方式(例如，见[1])。由此产生的体系结构叫做复杂指令型计算机(CISC)。这种计算机的优点是取指令时间与代码长度都相当小，而缺点是编译程序必须识别出生成这种简写指令的可能性。现在甚至在某些情况下几乎有不使用指令的趋势，于是就需要既笨重又缓慢的"优化"编译程序。CISC 处理机一般是用微程序设计技术实现的：这种处理机包含一个由(微)处理器解释的(微)程序，其体系结构要充分适合于解释微程序，因为它是唯一被经常执行的程序。

每一指令都需要几个周期，这已被看成是效率的严重障碍。因此，人们期望对 CISC 概念来一个反叛。缩减指令型计算机 (RISC) 就标志着回复到单级实现，在许多方面这使人们想起25年前使用的指令系统。如果没有其它原因，人们可能欢迎降低编译程序复杂性。但是，其缺点是代码密度低。已证明，扩充代码是完全可以接受的，这是因为内存容量不再受到严重限制，而且用来取指令的附加时间通过流水线技术，利用并发性可以进行弥补[2]。最近的设计甚至可以整行存取存贮单元并可使用包含在内存芯片（视频随机存取存贮器）中的长移位寄存器。一味凭借蛮力大量使用硬件成分不采用新技术，这种补救途径是有问题的，至少从数学角度看是如此。

必须区分 RISC 模式的两个特性。最初 RISC 被看作是一种每指令只需要一个时钟周期的机器（象处理外部设备一样异步处理内存访问）[3]。后来又把相当大的寄存器文件运用作为 RISC 的基本特性添加进来。这种添加实际上导致了二级存贮器的概念并要求编译程序能记录下变量的位置及其副本。人们很快就不再对更简单的编译程序抱有希望，事实上，如果用 RISC 实现高级语言，那么就不得不大量使用复杂的编译程序，这是十分明显的。RISC 设计者必须注意不要只是想把复杂问题交给编译程序，而使体系结构简单化。

最近对代码生成算法与各种计算机代码密度的比较研究表明，各种体系结构的编译程序代码生成部分的长度各不相同，相差可达3倍之多。代码密度变化范围为1到4倍。但是，这项研究的一个重要结论是，编译程序复杂的原因主要不是由于指令复杂，而是

维普资讯 http://www.cqvip.com

由于体系结构的不规则性[4]。如果把68000与32000处理机(两者都属于复杂指令型计算机类)的代码生成程序比较一下，就特别明显。例如，导致不规则性的原因有：对某些指令允许某种寻址方式，而对其它指令则不允许；对某些指令可以使用某种格式，而对另一些与之密切相关的指令则使用另一种格式；对某些指令可以指定自动符号扩充，而对其它指令则不允许；限制某些运算到部分寄存器；等等。

68000的代码生成算法差不多长达两倍。这定量地反映了这种处理机的编译程序设计者所必须忍受的苦恼。使设计者受到莫大委屈是由于这样的事实：对众多的体系结构不规则性都缺乏明确的证明。这些真的就是设计者的怪念头吗？我真诚地希望，RISC这一缩写词将来代表的是规则(regular)指令型计算机，而不是缩减指令型计算机。

程序设计与硬件设计

我们总在考虑硬件与软件设计两个领域的复杂性控制问题。设计者尽力把设计做得精细，直到他无能为力为止，这也许是一条人性法则。客户不应轻易接受，或者根本不接受。例如，复杂的编译程序不仅体积庞大，执行速度慢，而且通常也是不可靠的。再者，所生成的代码序列也不可预测；当对源程序进行微小的修改时，代码序列可能变动很大。欲预测执行时间是完全不可能的，实时系统程序员不得不使用汇编语言。

资源缺乏迄今成为硬件设计模式过于杂乱的保护伞。现在不再是这样了，因此，硬件设计与软件设计也受到了相同现象的的干扰。为什么不！计算机实际上也是用程序与算法作为电路实现的。许多计算机已做得相当复杂，描述它们所需的纸张数量要比它本身所需硅片多百万倍。现在是到了产品描述成本比产品装配成本还要高的时候了。这是荒谬的。

1968年，当"软件危机"这一术语刚刚杜撰出来时，硬件设计被当作值得模仿的范例[1]。主张精确的规格说明、细心的设计、完全的测试是硬件项目的特性。令人怀疑的是，今天是否仍可以这样认为。另一方面，程序设计技术其间也取得了一定的进展。主要是，人们认识到完全测试根本不适合于复杂程序。由于对涉及无数晶体管的设计进行完全测试同样是不适合的，因此现在大概到了硬件设计者应了解程序设计领域的时候了，这样可以使他们熟悉新的算法推理方法。最后对电路模拟花大量工作，也越来越不能继续作为我们信赖用以控制重要过程的设备的基础。

程序设计的进展主要依靠建立算法的推理方法来加深对算法现象的理解，尤其不以静态观点的机械理解。程序不再被看作是输入到计算机中去的客体，也不再被看作是只有通过对各种输入数据集进行逐步的解释才能理解的"处方"，而被看作是适于数学分析与逻辑推理的静态正文。这种重点的根本性转移主要以E.W.Dijkstra的工作为核心。与正文类似的是电路图，而对程序的逐步解释则类似于从晶体管到晶体管的信号追踪。考虑到电路中的并发成分比比皆是，进行追踪是绝对徒劳无益的。可是模拟器仍按这种极不现实的范例在建造。

计算可以表征为随时间变化的状态，状态用相关变量的集合来表示。一个程序规定状态变换序列。状态可以用逻辑命题与谓词(亦叫做断言)来刻划，其变元就是程序变量。因此，每个语句S都可看作是某个断言P的变换器，它把该语句执行前成立的断言P变换为执行后成立的断言R。

$$\{P\}\ S\ \{R\}$$

"q:=e"这一基本赋值语句由这样一个公理来决定：结果断言R在赋值后有效，是指在赋值前断言P成立，这里P是在R中把(每一个自由出现的)q用e代替得到的。

硬件中与程序变量相对应的客体是寄存器，变量的赋值对应于寄存器中的信号锁存，

维普资讯 http://www.cqvip.com

赋值所用的时间对应于时钟周期。"q:=d"的执行过程对应于把时钟脉冲边沿作用于保持q的D寄存器(图1)。

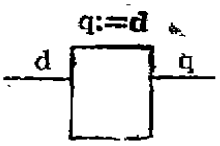

图 1 寄存器

对于重复语句，在重复序列开始给出的断言P在该序列的末尾仍必须成立。这种断言叫做(循环)不变式，它的成立与该语句序列已执行次数无关。

我们使用一个求T个值$e_0, e_1, \cdots, e_{T-1}$($T \geqslant 0$)的和的简单程序例子来说明，该程序用到一个计数器变量t和一个用于存放部分和的变量q。

```
t:=0; q:=0;
WHILE t<T DO
{p}q:=q+e[t]; t:=t+1
END
{R=P &(t=T)}
```

不变式P定义为

$$P:\ q:=e_0+e_1+\cdots+e_{t-1}$$

它与终止条件止t=T结合起来断言预期结果：

$$R:\ q:=e_0+e_1+\cdots+e_{T-1}$$

注意，对结果的推导并不涉及基于执行步的任何推理，而基于对静态程序正文的分析以及赋值与程序合成规则的运用。

这一算法可以用计数器电路来实现。假定t代表以时间表示的离散步数，因此e_t就表示在时间(时钟区间)t期间(输入)信号e的值。信号q表示和$q=e_0+e_1+\cdots+e_{t-1}$，这是电路中的不变式。

该电路由一个寄存器与一个加法器组成(图2)。如果使用二进制编码就需要$\log_2 T=N$个二进制寄存器(注意，为简单起见，假定e的取值只为0与1)。加法器(对应于表达式q+e)是一种组合电路。由于采用二进制数表示，即

$$s=2^{N-1}*s_{N-1}+\cdots+2^2*s_2+2*s_1+s_0$$

两数之和$s=q+e$可以按数位用等式

$$s_i=q_i+c_i \pmod 2$$

(c_i表示进位)

$$c_{i+1}=q_i*c_i$$

$$c_0=c$$

来定义。和与积被用2取模，因此，可分别用逻辑运算符XOR与AND表示。该电路的图形形式示于图3。图4给出了对C的定义中的递归展开后得到的形式

$$c_{i+1}=q_i*q_{i-1}*\cdots*q_0*e.$$

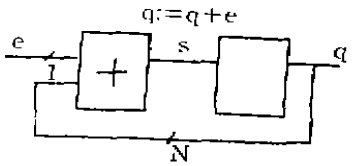

图2 计数器

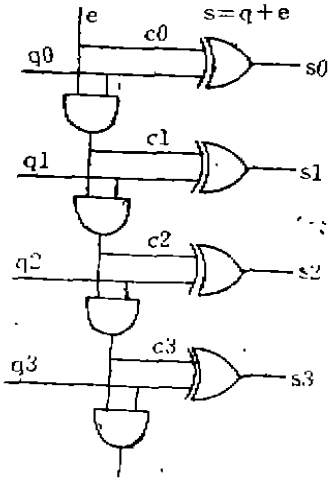

图3 加法器

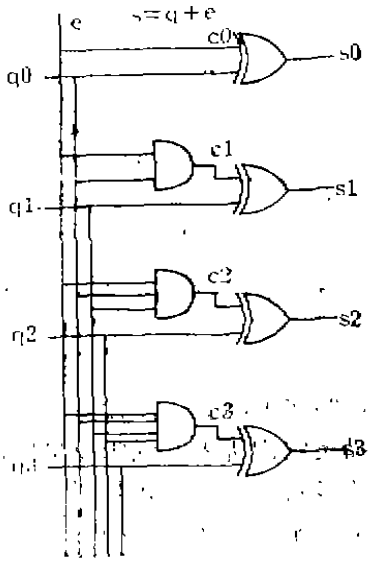

图4 加法器

(下转15页)

维普资讯 http://www.cqvip.com

[12] G.Goos, W.A.Wulf "DIANA : An Interrnediate Language for Ada" Lectures Notes in Computer Science, Vol. 161, 1983.

[13] M.Rueher, "From Specification to Design : An Approach Based on Rapid Phototyping", Fourth International Workshop on Software Specification and Design, 1987.

[14] H.K.berk "Formal Methods of Program Verification and specification" 1982.

[15] Burstull, R.M., Goguen, J.A. "The Semantics of CLEAR, A Specification Language", Lecture Notes in Computer Science, Vol.86, 1980.

[16] Guttag, I.V. "Notes on Type Abstraction", Proc. Specification of Reliable Software Conf., 1979.

[17] Bjorner, D. and Jones, C.B., "The Vienna Development Method: the Meta-Language", Springer-Verlag 1978.

[18] J.Jprgenson, S.V.Plam "Preliminary Definition of the RALSE Specification Langusge", Esprint 315 1988.

[19] Welch, T."Rapid Prototyping System", 1986.

[20] D.C.INCE, S.HEKMATPOUR, "Software Prototyping Progress and Prospects", Informations and Software Technology.Vol 29, No.1, 1987.

[21] G.MICHAELSON, "Interpreter Prototypes from Language Definition Style Specication", Information and Software Technology. Vol.30, No.1, 1988.

[22] Erin meiling, Chris W.George, "The RAISE Language, Method and Tools", 1987.

[23] Wayne K.Frey "Computer Aided Software Engineering (CASE) Environment ISSUES" 1987.6.

[24] 齐治昌"软件开发新模式对APSE的影响" 计算机工程与科学 1988.4.

（上接68页）

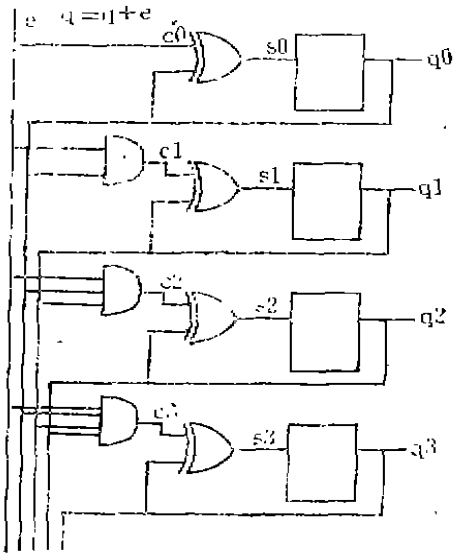

图5 计数器

用图4替代图2中的加法器就得到了图 5 所示的完整而详细的电路，后者就是众所周知的同步二进制计数器。我们特别注意到在组合电路中的等式s=q+e与顺序电路中的赋值q:=q+e之间微妙而根本的区别。

上述内容的寓意是，电路应该用正文形式定义，而实际电子电路可以据此按照一组固定的规则得到。对物理现象的模拟仍然需要，但可以把它简化为单个时钟周期期间的信号传播，可以把它简化为有限的几种情况。正文形式必须用硬件描述语言给出。最重要的是，这种语言必须经过严格定义，可以使用数学变元。其用途不是实现模拟，而是进行证明。

情况似乎是这样：不仅可以用适当的体系结构支撑程序设计语言与操作系统，而且反过来也可以用程序设计语言支撑体系结构的设计。将来大概不只是支撑设计，而且有可能用程序设计语言设计体系结构。

参考文献（略）

〔徐宝文 译自《ACM SIGPLAN Notices》1987, Vol.22, No.10, 示 瓦校〕