中山大学学报 (自然科学版)
第 36卷 增 刊 ACTA SCIENTIARUM NATURALIUM Vol. 36 Suppl.
1997年 6月 UNIVERSITATIS SUNYATSENI Jun. 1997

独立于技术的机器接口及其实现*

刘星成 印 鉴
(中山大学无线电电子学系, 广州 510275) (中山大学计算机科学系)

摘 要 阐述了一种新的技术—— 独立于技术的机器接口(TMI),使原有的软件不必重写或修改就能运行,且能充分利用机器上新的资源. 并说明了 TMI的实现方法.

关键词 应用程序接口, 独立于技术的机器接口, 微码, 程序模板, 结构, 对象定义表
分类号 TP 303

随着计算机软硬件技术的不断提高、数量的迅速增大, 势必产生这样一个问题: 在软硬件技术发展以后, 如何保护原有用户的投资, 而且能够采用新的技术? 假定有 8个可供程序员随意使用的 16位寄存器,则用汇编语言写成的每一个程序都能识别、并依赖于这些寄存器. 随着技术的进步, 用同样的费用可获得 16个 (或更多) 32位的寄存器. 如果事先考虑了这些变化, 也只能利用其中的 8个原有的寄存器. 如果没有考虑寄存器在数量和位数上的扩充, 则使用到这些寄存器的每一条指令都必须修改, 否则, 原来的程序就不能运行在新的硬件机器上. 独立于技术的机器接口 (TMI) 就是在这样的背景下提出的.

1 独立于技术的机器接口

有三种机器结构: 处理器 – 中心结构、API– 中心结构和高级机器结构.

处理器 – 中心结构, 这是传统的、最通用的结构, 它允许程序员直接使用硬件接口. 显然, 这种结构的微小变化都需要重写软件.

API– 中心结构, 这是以应用程序接口 (API) 为基础的结构. 所有应用程序能利用这种接口来访问操作系统的服务, 不必与具体的硬件或软件相联系. 然而, 这样的标准 (如 POSIX) 问题一直还在定义之中. 而且应用程序开发商们迫不及待地定义他们自己的 API, 开发应用程序. 结果, 等到将来的标准定义充分后, 所有应用程序又要重新编写才能满足新的标准.

高级机器结构, 它是建立在独立于技术的机器接口 (TMI) 基础上的. 硬件以及所有必须知道硬件详情的操作系统均位于 TMI层之下. 升级到 64位的 RISC技术之后, 操作系统和所有应用程序都会立即成为 64位软件,而不必重写什么东西来利用这 64位的硬件.

1.1 TMI的结构

TMI的结构与硬件无关, 它是系统的逻辑接口, 不是物理接口. TMI结构为操作系

* 收稿日期: 1997-03-11 刘星成, 男, 33岁, 讲师

统和所有应用程序提供完备的 API集.完备的 TIMI指的是系统或应用程序无法“绕过” TIMI层, 它们与硬件和 TIMI层以下的一些系统软件通讯的唯一方法是借助于 TIMI层. 这个特点使得 TIMI结构与 API-中心结构完全不同. API-中心结构允许应用程序 “绕过” API, 并依赖于根本的硬件和软件.

那么,如何定义完备的 API集,使操作系统和所有应用程序不经修改而永远运行呢?答案是: 新的应用程序增加后,支持这些应用程序的新的 API也必须定义到 TIMI中,即 TIMI是可扩充的. 又由于所有旧的 API保持不变,故原来编写的应用程序就受到 TIMI层的保护而能正常运行.

TIMI结构由两部分组成: 指令集和指令所作用的操作数.有些操作数是传统计算机结构中具有的位 (bit) 操作数和字节 (byte) 操作数. 其它操作数是称为对象 (objects) 的复杂数据结构.

利用对象, 应用程序和系统软件就不再要知道数据结构的准确格式. 此格式包含于对象本身, 且在该对象之外是不可见的. 因此, 数据结构的任何变化都不会影响到应用程序和系统软件. 软件也独立于根本的结构.

TIMI指令不能直接执行,它们要先编译成硬件指令集,再转换成可执行的程序. TIMI指令集与用于编译程序的公共中间形式相似. 高级语言编译程序产生 TIMI形式的程序. TIMI层之下的翻译器将这种形式的程序进行优化,产生具体的接口指令 (如 IMPI指令或 PowerPC指令)

1.2 TIMI接口的特点

传统的机器指令是对寄存器和存贮器的内容以及指令本身提供的立即数进行操作的. 由于指令能看到三个空间: 地址空间、 I/O空间和寄存器空间,因此,它们与这些物理结构有关. 这些物理结构改变了, 就引起指令的变化. 因此, 原有程序的转换就成为主要问题.

图 1是 TIMI机器接口. 它也有操作码和操作数, 与传统机器接口不同, TIMI指令类似于现代高级语言编译程序中的中间形式; 数据结构 (对象) 是机器构成的一部分. 但是, 最主要的差别不是指令或操作码本身, 而是指令之中所用的操作数. 在传统的机器中, 操作数有寄存器、存贮器和立即数. 而 TIMI接口的机器中,操作数分为立即数和对象,不包含寄存器和存贮器.

对内部指令集的改变不会影响到用户应用程序. 因为这些改变先送到翻译器, 然后利用程序模板重新翻译每个程序. 最后,新的指令流被封装回对象. 所有这一切都在 TIMI层以下完成, 不要用户干涉.

当用户将硬件升级后, 安装的也是一新版本的翻译器. 当程序第一次运行时, 系统根据对象头标志判定这是个旧版本的程序, 从而, 重新翻译. 但这种重新翻译的过程只发生一次, 在对程序的后续调用中使用的是新产生的代码, 不会再次翻译一遍. 因此, 硬件升级后,原来的程序能继续运行,只是第一次运行时速度慢,但后来的运行速度会大大提高.

2 TIMI的实现

2.1 程序模板

程序模板包括几个信息片, 它包括信息头、 TIMI指令流、用户数据和对象定义表 (ODT). 图 2示出了指令流和对象定义表, 其中的指令流是一个 TIMI指令的范例, 即传

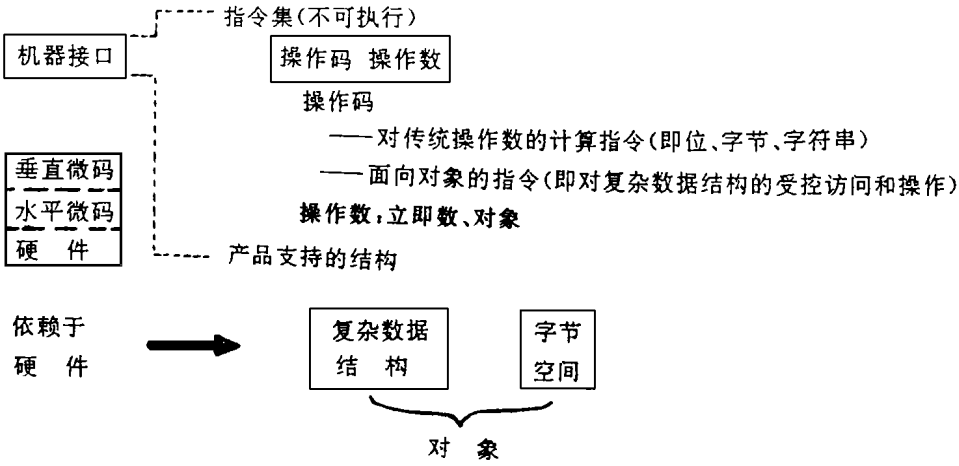

图 1 TIMI接口

统的三操作数的指令—— 数字求和指令. 它包含一个操作码和三个用于确定三个操作数的数值.这些值都用作对对象定义表的索引.图中所示的指令要求: 操作数 6加到操作数 2上, 和存于操作数 3中.

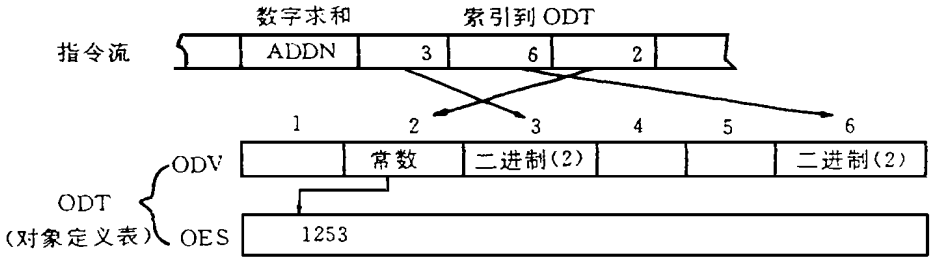

图 2 指令和对象定义表

对象定义表由两部分组成. 第一部分是对象定义表方向矢量 (ODV), 它包含程序中每个操作数的条目, 每个条目的长度固定, 因此, 指令流中的数值可用作对对象定义表方向矢量的索引. 对象定义表方向矢量中的条目描述了操作数. 本例中, 操作数 6是一个两字节长的二进制数, 操作数 3是另一个两字节长的二进制数, 操作数 2是常数. 常数和其它类型的操作数长度可以不同. 这就是需要对象定义表的第二个原因. 对象定义表条目字符串 (OES) 是长度可变的操作数, 它不适于存入对象定义表方向矢量. 对象定义表方向矢量字段的内容指向 OES中字符串的开始. 本例中, 操作数 2是一个值为 1253的常数.

从本例看出 TIMI指令的几个特点:

(1) 对象定义表规定了参加运算的操作数的格式, 翻译器负责进行数据转换.

(2) TIMI不是一个可执行的接口. 本例中, 操作数 3和操作数 6都与数字无关. 对象定义表方向矢量条目等价于变量说明. 没有存贮器保存该变量, 因此, 翻译器必须完成编

译, 并将变量赋值到寄存器或存贮器.

(3) 本例是一个很常用的计算指令.对对象进行操作的指令有相似的格式,只是对象定义表要指明如何找到该对象.

2.2 TIMI的指令格式

图 3示出了 TIMI指令格式. 从图中可知, 一个指令含有一个操作码、一个可选的扩展操作码、一个或多个操作数, 也可以没有操作数. TIMI接口在设计上是可扩展的, 因此, 指令数允许增多, 指令中的操作数也允许增加. 操作码以及扩展操作码是 16位字段. 每个用于索引到对象定义表方向矢量的操作数字段达 24位, 而且, 还可以根据实际需要进行调整. 因此, 程序可包含多达 2^{24} (1 600多万) 个不同的操作数.

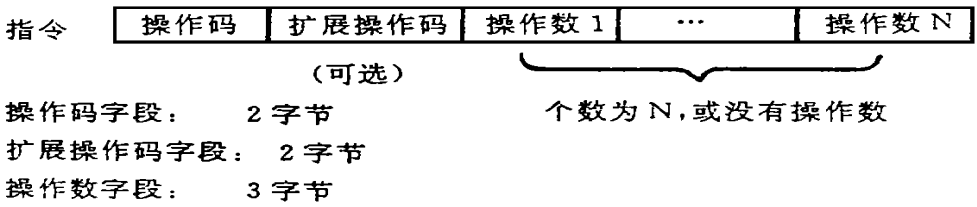

图 3 TIMI指令格式

2.3 TIMI操作码

TIMI操作码共 16位. 第 0~ 1位保留, 第 2位定义分枝目标, 第 3位规定该指令是计算格式还是非计算格式. 如果是非计算格式, 则操作码中的第 5~ 15位用于功能编码. 如果是计算格式 (如数字求和指令), 则第 5~ 7位进一步提供该指令的信息, 第 8~ 15位用于该计算指令的功能编码.

在计算格式中有两位说明扩展操作码, 第 4位决定是否有扩展操作码, 第 5位说明如何使用扩展操作码. 第 6位表示四舍五入, 这一般出现在浮点运算中. 在商用机器 (如 AS/400) 中, 使用的是十进制四舍五入, 十进制数作为浮点数. 第 7位表示是否使用短格式指令, 这只对计算格式有意义.

2.4 扩展操作码

TIMI扩展操作码在指令中占 16位, 有分枝形式和指针形式两种. 设置中的第 4位决定是否有扩展操作码. 有的话, 第 5位可用来规定分枝或指针形式.

对于分枝形式, 扩展操作码分为 4个 4位字段. 这些字段决定该指令的分枝能力. 所有 TIMI计算指令都可以包括作为执行部分的条件分枝. 换句话说, 根据计算结果, 可以从指令流的其它某个部位取出下一条 TIMI指令.

考虑扩展操作码中第一个 4位字段. 如果该字段包含数字 1 (二进制 0 001), 则意味着当指令调用的计算结果为正值时才分枝. 如果该字段数值包含数字 2 (二进制 0 010) , 则意味着当结果为负值时才分枝. 如果该字段数值为 4 (二进制 0 100), 则意味着当结果为 0时才分枝. 也有非零、非正、非负、不是非零的值. 对于不同类型的指令, 位的组合方式即使相同, 也可以有不同的意义. 例如, 比较指令与求和指令的位组合意义就不同.

如果第一个 4位字段规定的分枝条件满足了, 则分枝目标就在指令中最后一个操作数

的后面. 如果分枝条件不满足, 则执行下一条顺序指令. 由于扩展操作码中有 4个 4位字段, 每个字段规定一个分枝条件, 因此, 每个计算指令多达 4个分枝条件、 4个分枝目标. 如果不需要 4个, 则在字段中填 0, 表示没有分枝.

对于指针形式, 它与上类似. 不同的是: 条件满足时不是分枝, 而是设置指针. 与分枝选项相同, 每条指令多达 4个指针, 指针目标在最后一个操作码之后.

3 讨 论

由 TIMI提供的技术独立性是极其重要的, 因为它不必改变用户应用程序或操作系统, 就可以增加新的硬件, 并能立即加以充分利用.

除技术独立性之外, TIMI还可以扩展. 系统更新时, 能增加新的指令和功能. 例如, AS/400上可以增加 Single UNIX Specification API, 这样, 某些 Unix应用程序就可以移植到 AS/400上运行. 由于 TIMI支持 API应用程序的需要, 因此, 它是以应用为中心的接口, 需要新的应用程序时, 可方便地增加 API. 由于 TIMI扩充能力很强, 故 TIMI的生命力极为强大. 它已经在 AS/400、 RISC/6000等机型的应用中取得了极大的成功, 相信今后会有更多的机型采用 TIMI这种新技术.

参 考 文 献

1 Frank G Soltis. Inside the AS/400, Colorado, USA: Duke Press, 1996

2 IBM International Support Centers. Upgrading to AS/400 Advanced Series PowerPC AS, (First Edition), 1996

Technology-Independent Machine Interface and Its Implementation

*Liu Xingcheng** *Yin Jian*

Abstract The paper presented a new kind of technology, Technology-Independent Machine Interface (TIMI), which enabled original softwares to run smoothly without rewriting or revising them and enabled new resources on the machine to be fully utilized. The paper discussed the implementation of this technology.

Keywords application program interface, technology-independent machine interface, micro-code, program template, architecture, object definition table

* Department of Radio and Electronics, Zhongshan University, Guangzhou 510275