[19] 中华人民共和国国家知识产权局

[51] Int. Cl[7]
G06F 9/445
G06F 9/45

[12] 发明专利说明书

专利号 ZL 00811932.5

[45] 授权公告日 2005 年 9 月 28 日

[11] 授权公告号 CN 1220939C

[22] 申请日 2000.8.21 [21] 申请号 00811932.5
[30] 优先权
[32] 1999. 8.23 [33] FR [31] 99/10697
[86] 国际申请 PCT/FR2000/002349 2000.8.21
[87] 国际公布 WO2001/014958 法 2001.3.1
[85] 进入国家阶段日期 2002.2.22
[71] 专利权人 信用逻辑公司
地址 法国凡尔塞
[72] 发明人 X·莱尔奥
审查员 赵晓春

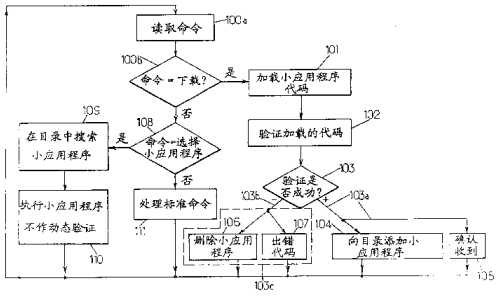
[74] 专利代理机构 北京市中咨律师事务所
代理人 于 静 杨晓光

权利要求书 5 页 说明书 39 页 附图 14 页

[54] 发明名称 管理协议，验证和转换下载程序片断的方法及对应的系统

[57] 摘要

本发明涉及管理协议，并涉及验证已经下载到便携系统的程序片断或小应用程序的方法。小应用程序的下载命令(100a，100b)被执行。一旦收到正响应，则读取小应用程序的对象代码(101)并逐条指令经过验证过程(102)。验证过程由一阶段组成，该阶段包括类型栈和寄存器类型表的初始化，它们表示在小应用程序代码的执行开始时便携系统的虚拟机状态；以及对每一目标当前指令逐条指令的验证，包括验证目标转移指令，目标异常处理程序调用或目标子程序调用的存在性，验证和更新指令对类型栈和寄存器类型表的效果。如果验证成功(103a)，小应用程序被寄存(104)并向下载驱动器发送(105)确认。否则，小应用程序被删除(106)。本发明适用于 Java 环境的便携系统。



ISSN1008-4274

知识产权出版社出版

1.一种用于管理程序片断的方法，程序片断是下载到装有可重写存储器的微处理器卡这种可重编程板载系统的，所述程序片断由对象代码即一系列指令组成，对象代码可由板载系统的微处理器通过虚拟机执行，虚拟机带有执行栈并带有通过这些指令处理的局部变量或寄存器，并使得能够解释这些对象代码，所述板载系统与终端互连，其特征在于该方法在所述板载系统级上至少包括:

a)检测用于下载这种程序片断的命令；并在对检测下载命令这一阶段为正响应时，

b)读取构成这种程序片断的对象代码并暂时存储这种对象代码;

c)使所有暂时存储在存储器中的对象代码经受验证过程，这一验证过程至少包括初始化类型栈和寄存器类型表阶段，它们表示所述虚拟机在开始执行暂时存储的对象代码时的状态，并且包括以下后继的阶段，通过对每一当前指令判断目标、转移指令目标、异常处理程序调用目标、或子程序调用目标的存在性，逐个指令地进行验证，并且包括验证和更新所述当前指令对类型栈和寄存器类型表的效果，并且在所述对象代码的成功验证的情况下，

d)把下载的程序片断记录在可用的程序片断的目录中，并在所述对象代码的不成功验证的情况下，

e)在所述板载系统上禁止所述程序片断的执行。

2. 如权利要求 1 所述的方法，其特征在于所述禁止执行的阶段 e)包括:

f)当省略在所述可用程序片断的目录中记录暂时记录的程序片断时，将其删除，以及

g)向所述终端发送出错代码。

3. 如权利要求 1 或 2 所述的方法，其特征在于，在对检测下载命令这一阶段 a)是负响应时，则包括:

b')检测一命令，该命令是从程序片断目录选择一可用程序片断；并在对检测选择可用程序片断的命令这一阶段为正响应时，

c')调用所述已被选择的可用程序片断；

d')通过虚拟机执行所述被调用的可用程序片断，当每一指令被执行时，对于变量类型、对于由所调用的可用程序片断处理的对象的访问权，或对于执行栈的溢出不进行动态验证，并且在对检测选择可用程序片断的命令这一阶段是负响应时，

e')前进到处理板载系统的标准命令。

4. 一种验证程序片断的方法，程序片断是下载到诸如装有可重写存储器的的微处理器卡这种可重编程板载系统的，所述程序片断由对象代码组成，并至少包含一个子程序即一系列指令，验证是通过板载系统的微处理器由虚拟机进行的，虚拟机带有执行栈及由这些指令处理的操作数寄存器，并使得能够解释这种对象代码，所述板载系统与阅读器互连，其特征在于所述方法在检测下载命令并把构成程序片断的所述对象代码存储到所述可重写存储器中之后，对于每一子程序：

α)通过数据进行使类型栈和寄存器类型表初始化的阶段，这些数据表示虚拟机在开始执行暂时存储的对象代码时的状态；

β)通过对每一当前指令判断目标、转移指令目标、异常处理程序调用目标或子程序调用目标的存在性，逐个指令地对所述暂时存储的对象代码指令进行验证；

γ)基于转移指令目标、子程序调用目标或异常处理程序调用目标的存在性，就所述当前指令对所述类型栈和所述寄存器类型表的数据类型的效果进行验证和更新，当寄存器类型表在所有指令的验证过程中没有被修改时，所述验证是成功的，并且验证过程逐个指令地进行直到寄存器类型表稳定而没有修改出现为止，否则验证过程被中断。

5. 如权利要求 4 所述的验证方法，其特征在于，在验证过程期间被处理的变量类型至少包含：

－对应于在程序片断中定义的对象类的类标识符；

－数值变量类型，至少包含类型 short，按 p 位编码的整数，以及用于跳转指令 JSR 的返回地址的类型 retaddr;

－与零对象的参照相关的类型 null;

－与对象相关的类型 object;

－第一特定类型⊥，表示所有类型的交集并且对应于值 0，nil;

－第二特定类型 T，表示所有类型的并集并且对应于任何值的类型。

6. 如权利要求 5 所述的方法，其特征在于，所有所述变量类型验证子类型分类关系：object ∈ T; short, retaddr ∈ T; ⊥ ∈ null, short, retaddr 。

7. 如权利要求 4 所述的方法，其特征在于，当所述当前指令是转移指令的目标时，所述验证方法包括验证类型栈是空的，在肯定的验证的情形下，验证过程对随后的指令继续进行，否则验证过程失败且程序片断被拒绝。

8. 如权利要求 4 中所述的方法，其特征在于，当所述当前指令是子程序调用的目标时，所述验证过程验证前一指令是无条件转移，子程序返回或异常的引起，在肯定的验证的情形下，验证过程进到通过 retaddr 类型实体，子程序的返回地址，重新更新变量类型栈，否则验证过程失败且程序片断被拒绝。

9. 如权利要求 4 所述的方法，其特征在于，在当前指令是异常处理程序的目标时，所述验证过程验证前一指令是无条件转移，子程序返回或异常引起，在肯定的验证的情形下，所述验证过程进到通过输入异常类型而重新更新类型栈，否则验证过程失败且程序片断被拒绝。

10. 如权利要求 4 所述的方法，其特征在于，在当前指令是多个不兼容转移的目标时，验证过程失败且程序片断被拒绝。

11. 如权利要求 4 所述的方法，其特征在于，在当前指令不是任何转移目标时，验证过程通过过渡到类型栈的更新而继续。

12. 如权利要求 4 所述的方法，其特征在于，当前指令对类型栈的效果的验证阶段至少包含：

－验证类型执行栈至少包含象当前指令包含的操作数那么多的项的阶段;

－非栈存储及验证栈的顶部项的类型是这一指令的操作数的操作数的类型的子类型的阶段;

－验证类型栈上有足够的存储空间存在以便进到对当前指令结果进行栈存储的阶段;

－在栈上把指定给这些结果的数据类型进行栈存储的阶段。

13. 如权利要求 12 所述的方法，其特征在于，在当前指令是对地址 n 的寄存器进行读取的指令时，验证过程包括:

－通过读取寄存器类型表中的项 n，验证这一读取结果的数据类型;

－通过使对应于这一当前指令操作数的栈的项作非栈存储，并通过对这一结果的数据类型作栈存储，判断当前指令对类型栈的效果。

14. 如权利要求 12 所述的方法，其特征在于，在当前指令是对地址 m 的寄存器进行写入的指令时，验证过程包括:

－判断当前指令对类型栈和已写入地址 m 的这一寄存器的操作数的类型 t 的效果;

－以紧接在先前存储的类型之上以及已写入地址 m 的这一寄存器的操作数的类型 t 之上的类型，代替地址 m 处的寄存器类型表的类型项。

15. 一种板载系统，该系统能够通过下载的程序片断被重编程，该系统至少包括一个微处理器，一个随机访问存储器，一个输入/输出装置，一个电可重编程非易失存储器以及一个永久性存储器，其中装有主程序和虚拟机，该虚拟机使用所述微处理器使得能够执行主程序及至少一个程序片断，其特征在于，所述板载系统包含至少一个管理和验证装置，用于根据权利要求 1 到 3 之一所述的用于管理下载的程序片断的方法而管理和验证下载的程序片断，所述管理和验证装置安装在永久性存储器中。

16. 一种板载系统，该系统能够通过下载的程序片断被重编程，该系统至少包括一个微处理器，一个随机访问存储器，一个输入/输出装置，一个电可重编程非易失存储器以及一个永久性存储器，其中装有主程序和虚

拟机，该虚拟机使用所述微处理器使得能够执行主程序及至少一个程序片断，其特征在于，该系统包含至少一个验证装置，用于根据权利要求 4 到 14 之一所述的验证方法而验证下载的程序片断。

管理协议，验证和转换下载程序片断的方法及对应的系统

本发明涉及用于管理的协议，对下载的程序片断验证和转换的方法，以及对应的系统，特别着重于就存储器和计算能力而言具有很少可用资源的板载数据处理系统。

参见图 1a，一般来说，通常认为板载数据处理系统 10 包括微处理器 11，永久存储器，诸如包含可执行程序代码的非可写存储器 12，及包含存储在系统中的数据的 EEPROM 型可重写非易失永久存储器 15，易失性随机访问存储器 14，程序在执行时在其中存储其中间结果，以及允许系统与其环境交互的输入/输出装置。在板载数据处理系统由板卡型微处理器卡组成的情形下，则输入/输出装置 15 由串行链路组成，允许卡与诸如读卡器终端等终端通信。

在传统的板载数据处理系统中，系统所执行的程序代码在系统构成期间，或至少当后者在交付最终用户之前被定制时，是固定的。

然而，更精致的板载数据处理系统已经实现，这些系统是可重编程的，诸如 JavaCard 型微处理器卡。至于先进的类型，这些可重编程系统通过下载程序片断，增加了系统投入服务之后强化程序的可能性。常常由“小应用程序”所指的这些程序片断，在本说明书中不加区别地指小型应用程序或程序片断。对于 JavaCard 系统更为详细的说明，宜参照由 SUN MICROSYSTEMS INC. 公司发布的文件，并特别是在 www(World Wide Web) 网址 http://java.sun.com/products/javacard/index.html，可用的电子文档，JavaCard 技术文章，1996 年 6 月。

图 1b 示出这种可重编程板载数据处理系统的结构。这种结构类似于传统的板载系统，所不同在于，可重编程板载系统还能够通过其输入/输出装置之一接收小应用程序，然后在其永久性存储器 13 中存储这些程序，然后从该存储器这些程序可被执行，作为对主程序的补充。

由于不同板载数据处理系统之间的可移植性，小应用程序以对于标准

虚拟机的代码形式呈现。这种代码是不能直接由微处理器 11 执行的，而必须由虚拟机 16 以软件术语解释，该虚拟机是由驻留在非可写永久性存储器 12 中的一个程序组成的。在上述 JavaCard 卡的例子中，所使用的虚拟机是一 Java 虚拟机子集。至于有关 Java 虚拟机的规范及所使用的虚拟机的说明，宜参见由 Tim LINDHOLM 和 Frank YELLIN 公布的著作，标题为“The Java Virtual Machine Specification(Java 虚拟机规范)”，Addison-Wesley 1996，并参见由 SUN MICROSYSTEMS INC.发布的文件“JavaCard 2.1 Virtual Machine Specification”，电子可用的文档在 www 网址 http://java.sun.com/products/javacard/JCVMSpec.pdf，1999 年 3 月。

向工作中的板载数据处理系统下载小应用程序的操作造成相当大的安全问题。偶然或甚至是故意地不良写入的小应用程序可能错误地修改系统上呈现的数据，妨碍主程序正确地或在正确的时间执行，或修改先前下载的其它小应用程序，造成它们不能使用或有害。

由计算机黑客所书写的小应用程序也可能泄漏存储在系统中的机密信息，例如在银行卡的情形下的访问代码等信息。当前，就纠正小应用程序安全问题已经提出三种解决方案。

第一个解决办法是使用加密签字，以便只接收来自可信团体或人员的小应用程序。

在上述银行卡的例子中，只接收带有发卡银行的加密签字的小应用程序，并由卡执行，在下载操作过程中任何其它非签字小应用程序都被拒绝。而一个卡的恶意用户，由于没有来自银行可用的密钥，于是不可能执行卡上非签字且危险的小应用程序。

这第一个解决办法可用于所有小应用程序来自同一单一来源的情形，例如上述的银行。这种解决办法难以用于小应用程序来自若干来源的情形，诸如在银行卡的例子中，卡的制造商，银行，通过银行卡管理服务的团体，提供客户诚信程序并合法提出向卡下载小应用程序的大型商业配送组织。对于小应用程序的电子签字所必须的密钥，在这些各种经济参与者之间共享或持有，则造成主要的技术，经济和法律上的问题。

第二个解决办法是在小应用程序执行期间，对访问和键入进行动态检验。

在这一解决办法中，在小应用程序执行期间虚拟机进行一定数目的检验，诸如：

－检验对存储器的访问：对存储区中每一读和写，虚拟机检验小应用程序对于对应数据的访问权限；

－数据类型的动态验证：对来自小应用程序的每一指令，虚拟机验证满足对数据类型的约束。举例来说，虚拟机可以对诸如有效存储地址等数据进行特别的处理，并通过整数/地址转换或对地址的算术运算防止小应用程序产生无效存储地址；

－检测栈溢出及对虚拟机的执行栈的非法访问，在一定的条件下，就前面的检验机制来说，它们可能会干扰其操作。

这第二个解决办法允许在安全的条件下执行范围广泛的小应用程序。然而，其缺点是由动态检验范围所引起的，使执行明显变慢。为了达到降低这种变慢，某些这些验证可由微处理器本身承担，然而其代价是增加其本身的复杂性，并因而增加板载系统的成本。这种验证还增加了对系统随机访问和永久性存储器的需求，原因是增加的与处理的数据相关所必须的类型信息。

第三个解决办法是在下载期间，对小应用程序的代码进行静态验证。

这个解决办法中，这种静态验证模拟小应用程序在数据类型级的执行，并一劳永逸地确定，小应用程序的代码符合虚拟机所要求的数据类型和访问控制规则，而不会引起栈溢出。如果这一静态验证成功，则小应用程序能够被执行，而不必动态验证符合这一规则。在静态验证过程失败的情形下，板载系统拒绝“小应用程序”，且不允许其随后的执行。对于上述第三个解决办法更详细的说明，宜参见以上引述的由 Tim LINDHOLM 和 Frank YELLIN 公布的著作，参见由 James A. GOSLING 发表的文章，标题为“Java Intermediate Byte Codes(Java 中间字节代码)” ACM SIGPLAN 会议文集，中间表示法研讨会(IR'95)，页 111－118，1995，1 月，及 1998 年 5 月 5 日授予的美国专利 5,748,964。

与第二个解决办法相比，第三个解决方案的优点是小应用程序执行速度快得多，因为在执行期间虚拟机不进行任何验证。

然而第三个解决方案的缺点是，不论就进行这一过程所必须的代码量，包含验证中间结果必须的随机访问存储器的大小以及计算时间来说，代码的静态验证过程复杂而成本高。举例来说，集成到由 SUN MICROSYSTEMS 出售的 Java JDK 系统的代码验证有大约 50kbytes 的机器代码，且其随机访问存储器的耗用量正比于(Tp+Tr)×Nb，其中 Tp 表示最大栈空间，Tr 表示寄存器最大数，而 Nb 表示由子程序使用的转移目标最大数，也广泛地由小应用程序方法指定。这些存储器需求大大超过当今大部分板载数据处理系统的资源容量，特别是市售微处理器卡。

已经提出第三个解决方案的几种变形，其中小应用程序的写入程序向验证程序除了发送小应用程序代码之外，还发送一定量特定的补充信息，诸如预计算的数据类型或正确数据类型的预估计验证。对于对应的操作模式更为详细的说明，宜参见由 Eva ROSE 与 Kristofer HØGSBRO ROSE 公布的文章，“Lightweight Bytecode Verification(轻量字节代码验证)”，Proceedings of the Workshop Formal Underspinning of Java，1998 年 10 月，以及由 George C. NECULA 发表的 “Proof-Carrying Code(证明进位代码)”，Proceedings of the 24th ACM Symposium Principles of Programming Lanuages(第 24 届 ACM 专题研讨会 编程语言原理)，页 106 - 119。

这种补充信息使得能够更快地验证代码并稍微降低了验证程序的代码量，然而没能降低对随机访问存储器的需求，或甚至有相当的增加，特别是在正确数据类型预估计验证信息的情形下。

本发明的目的是要纠正上述先有技术的缺点。

具体来说，本发明的一个目的是实现用于管理下载程序片断或小应用程序的协议，允许后者通过诸如微处理器卡，这种具有很少可用资源的板载数据处理系统执行。

本发明的另一目的还在于实现验证下载的程序片断或小应用程序的方法，其中当小应用程序被下载时，进行小应用程序代码的静态验证过程，

这一过程可能至少在原理上与上述第三个解决办法类似，但是其中引入了新的验证技术，因而允许在微处理器卡和其它低性能板载数据处理系统所能提供的存储器大小和计算速度值限制内，执行这种验证。

本发明的另一目的是实现对所获得的通常类型的程序片断进行转换的方法，例如，通过对标准的程序片断或小应用程序使用 Java 编译程序，如果优先满足作为本发明目的的验证方法的验证准则，目的是在当前微处理器卡或板载数据处理系统水平下，加速验证和执行小应用程序的过程。

最后，本发明的另一目的是产生一种板载数据处理系统，该系统允许实现上述用于管理的协议，及实现上述验证下载程序片断的方法及数据处理系统，如上所述，允许实现把通常的程序片断或小应用程序转换为标准化的程序片断或小应用程序的方法。

作为本发明的目的，用于管理可重新编程板载系统的下载程序片断的协议，特别用于装有可重写存储器的微处理器卡。程序片断由对象代码构成，这是由板载系统的微处理器可执行的一系列指令，通过装有由这些指令处理的执行栈和装有局部变量或寄存器的虚拟机执行，并使得能够解释这种对象代码。板载系统与终端互连。

值得注意的是，在检测用于下载程序片断的命令时，它至少是在板载系统级构成的。在对检测下载命令的阶段有正响应时，系统进而读取构成程序片断的对象代码，并把这种对象代码存储在可重写存储器中。存储在存储器中的所有对象代码逐个指令地经过验证过程。验证过程至少有以下阶段，即初始化类型栈和寄存器类型表的阶段，这表示虚拟机在开始执行暂时存储的对象代码时的状态，以及后继的阶段是，对每一当前指令逐个指令验证目标，转移指令目标，异常处理程序目标的存在性，并在于验证和更新当前指令对类型栈和寄存器类型表的效果。在对象代码不成功验证的情形下，当省略在可用程序片断的目录中记录程序片断时，作为本发明的目的协议在于删除暂时记录的程序片断，并向阅读器发送出错代码。

作为本发明的目的，验证下载到板载系统的程序片断的方法特别用于装有可重写存储器的微处理器卡。程序片断由对象代码组成，并包含至少一个子程序，一系列指令，可由板载系统的微处理器通过装有由这些指令

处理的执行栈及操作数寄存器的虚拟机执行，并能够解释这种对象代码。板载系统与阅读器互连。

值得注意的是，在检测下载命令和在可重写存储器中存储构成程序片断的对象代码之后，对于每一子程序该方法是执行这样一个阶段，即通过数据初始化类型栈和寄存器类型表，数据表示虚拟机在开始执行暂时存储的对象代码时的状态，在于通过对每一当前指令辨别转移指令目标，异常处理程序调用目标，或子程序调用目标的存在性，执行逐个指令地验证暂时存储的对象代码，并在于基于转移指令目标，目标子程序的调用或异常处理程序调用的目标的存在性，执行当前指令对类型栈和寄存器类型表的数据类型的效果的验证和更新。当在所有指令验证过程中寄存器类型表没有被修改时，验证是成功的，验证过程逐个指令进行到寄存器类型表稳定，没有出现修改为止。否则，验证过程被中断。

作为本发明的目的，把程序片断的对象代码转换为同一程序片断的标准化对象代码的方法，用于程序片断的一种对象代码，其中每一指令的操作数属于由这一指令处理的数据类型，执行栈不出现任何溢出现象，并对每一转移指令，在这一转移时的栈的变量类型与这一转移的目标处相同。所获得的标准化对象代码是这样的，即每一指令的操作数属于由这一指令处理的数据类型，执行栈不出现任何溢出现象，并在每一转移目标指令处执行栈为空。

值得注意的是，该方法在于，对所有的对象代码指令，在这一指令执行之前和之后，以执行栈的数据类型注释每一当前指令，通过分析与这一指令相关的数据流而计算注释数据，在于在这些指令内和当前指令内，检测执行栈非空的转移的存在性，检测操作是基于分配给每一当前指令的栈变量类型的注释数据而进行的。在出现检测到非空执行栈时，该方法进而在于，在这些转移或这些转移目标的任一侧向转移栈变量插入指令，以便在这一转移之前，把执行栈的内容清空到临时寄存器，并在这一转移之后从临时寄存器重建执行栈，并且否则在于不插入任何转移指令。

于是，这一方法使得能够对于同一程序片断获得标准化对象代码，其中在没有对程序片断的执行有任何修改时，执行栈在每一转移指令及转移

目标指令处为空。

此外作为本发明的目的，把程序片断的对象代码转换为同一程序片断的标准化对象代码的方法，用于程序片断的对象代码，其中每一指令的操作数属于由这指令处理的数据类型，且已由这一对象代码的指令写入寄存器的给定类型的操作数，通过这一对象代码的另一指令以相同的给定的数据类型从这同一寄存器重新读出。所获得的标准化对象代码是这样的，即操作数属于由这一指令处理的数据类型，在全部标准化对象代码中同一种数据类型分配给同一寄存器。

值得注意的是，该方法在于，对于对象代码的所有指令，在这指令执行之前和之后以寄存器数据类型注释每一当前指令，注释数据通过对与此指令相关的数据流的分析被计算，并在于通过把这些原始寄存器划分为分开的标准化寄存器，对采用不同类型的原始寄存器重新分配。一个标准化寄存器被分配给使用的每一数据类型。对处理使用标准化寄存器的操作数的指令进行重新更新。

作为本发明的目的，用于管理程序片断的协议，验证程序片断的方法，把程序片断的对象代码转换为标准化对象代码的方法及对应的系统，在开发可重编程板载系统，诸如微处理器卡，特别是在 Java 环境中，找到应用。

在阅读说明并细看以下附图时对它们将能够更好地理解，其中除了图 1a 和 1b 与先有技术相关之外，它们是：

－图 2 示出一流程图，表示用于管理下载到可重新编程板载系统的程序片断的协议，

－图 3a 示例表示根据本发明目的的验证下载程序片断方法的一流程图，

－图 3b 示出一示意图，表示通过作为本发明目的下载程序片断的管理方法和验证方法实现的数据类型与子类型的关系，

－图 3c 表示如图 3a 所述的与管理转移指令相关的验证方法的细节，

－图 3d 表示如图 3a 所述的与管理子程序调用指令相关的验证方法的细节，

－图 3e 表示如图 3a 所述的与管理异常处理程序目标相关的验证方法

的细节

－图 3f 表示如图 3a 所述的与管理不兼容转移的目标相关的验证方法的细节，

－图 3g 表示如图 3a 所述的与转移目标的不存在的管理相关的验证方法的细节，

－图 3h 表示如图 3a 所述的与管理当前指令对类型栈效果相关的验证方法的细节，

－图 3i 表示如图 3a 所述的与管理用于读取寄存器的指令相关的验证方法的细节，

－图 3j 表示如图 3a 所述的与管理用于向寄存器写入的指令相关的验证方法的细节，

－图 4a 示出一流程图，表示把程序片断的对象代码转换为同一程序片断的带有空栈的转移指令分别还有转移目标指令的标准化对象代码的方法，

－图 4b 示出一流程图，表示使用类型寄存器，把程序片断的对象代码转换为同一程序片断的标准化对象代码的方法，单一特定数据类型属于每一寄存器，

－图 5a 表示实现图 4a 中所示转换方法的细节，

－图 5b 表示实现图 4b 中所示转换方法的细节，

－图 6 表示用于开发标准化程序片断的系统以及可重编程微处理器卡完整结构的功能示意图，允许实现根据本发明的目的管理协议和验证程序片断的方法。

一般来说应当指出，作为本发明的目的，管理协议和验证与转换下载程序片断的方法，及对应的系统可被实现，是由于软件体系结构用于在带有很少资源的板载数据处理系统，特别是诸如微处理器卡上，小应用程序的可靠的下载和执行。

一般来说应当指出，以下的说明涉及本发明在 JavaCard 型可重编程微处理器卡的场合的应用，比较本说明前面开头提及的来自 SUN MICROSYSTEMS INC.公司可得的电子文档 JavaCard Technology。

然而，通过下载以虚拟机代码书写的小应用程序本发明可用于任何可重编程的板载系统，该虚拟机包含执行栈，局部变量或寄存器，其执行模型类型明确，小应用程序代码的每一指令只用于特定的数据类型。以下将参照图 2 更为详细地说明，作为本发明的目的，管理下载到可重编程板载系统的程序片断的协议。

参照上述图示可见，构成程序片断或小应用程序的对象代码由一系列指令组成，这些指令借助于上述虚拟机可由板载系统微处理器执行。虚拟机使得能够解释上述对象代码。板载系统例如通过串行链路与终端互连。

参照上述图 2，作为本发明的目的的管理协议，至少是在阶段 100a，100b 在板载系统中检测下载这一程序片断的命令。于是，阶段 100a 可由读取上述命令的阶段组成，而阶段 100b 可由测试已经读取的命令及验证下载命令的存在性的阶段组成。

在对上述检测下载命令的阶段 100a，100b 有正响应时，作为本发明的目的的协议继而是在阶段 101 读取构成相关程序片断的对象代码，并把上述对象代码暂时存储在板载数据处理系统的存储器中。上述暂时存储的操作能够在板载系统的可重写存储器执行，或如果适宜而有足够的容量时在随机访问存储器中执行。读取对象代码并暂时在可重写存储器中存储的阶段在图 2 中被表示为加载小应用程序代码。

然后，上述阶段之后是阶段 102，该阶段是向上述对象代码的验证过程逐个指令提交所有暂时存储的对象代码。

验证过程至少是由初始化类型栈和数据类型表阶段构成，数据类型表示虚拟机在开始执行暂时存储的对象代码的状态，并在于后继阶段是逐个指令进行验证，这是通过对每一被指定为 I_i 的当前指令，辨别诸如已指定 CIB 的转移指令目标，异常处理程序调用目标或子程序调用目标的存在性。就当前指令 I_i 对类型栈和对寄存器类型表的效果进行验证和更新。

当在阶段 103a 验证已经成功时，作为本发明的目的的协议在阶段 104 是在可用的程序片断目录中记录下载的程序片断，并在阶段 105 向阅读器发送正的接收确认。

另一方面，在阶段 103b 对象代码不成功验证的情形下，作为本发明的

目的的协议是在阶段 103c 禁止在板载系统执行任何暂时记录的程序片断。禁止阶段 103c 能够以各种方式实现。作为非限制性例子，这一阶段可以在阶段 106 删除临时记录的程序片断，而不把这一程序片断记录在可用的程序片断目录中，并在阶段 107，向阅读器发送一出错代码。在阶段 106 和 104 之后阶段 107 和 105 能够分别或者按顺序实现，或者对它们进行多任务化操作实现。

参见同一图 2，在对阶段 100b 检测下载命令阶段为负响应时，作为本发明的目的的协议是在阶段 108 检测一命令，以便从程序片断的目录选择可用的程序片断，并在对阶段 108 的响应为正时，则已经检测了可用程序片断的选择，在阶段 109 调用这一被选择的可用程序片断以便执行这一程序片断。跟随阶段 109 之后是阶段 110，由虚拟机执行所调用的可用程序片断，当每一指令被执行时，对变量类型，对调用的可用程序片断处理的对象的访问权，或执行栈的溢出，进行动态验证。

在阶段 108 获得负响应的情形下，这一阶段是检测一命令，以便选择被调用的可用程序片断，作为本发明的目的的协议进到阶段 111，以便处理板载系统的标准命令。

考虑到没有对类型或访问例如 JavaCard 类型对象权限的动态验证，要指出验证的这种缺失并不影响卡的安全性，因为小应用程序代码必须成功通过验证。

更具体来说，应当指出，正如作为本发明的目的的方法中所述，在微处理器卡或板载数据处理系统上所执行的代码验证，与对于 Java 虚拟机通常的代码验证，如本说明前面提及的标题为“Java 虚拟机规范”工作中所述相比较要有更多的选择。

然而，Java 虚拟机的任何代码，只要就传统的 Java 验证程序来说它是正确的，则就能够转换为一种等效的代码，这种代码能够成功地通过在微处理器卡上进行的代码验证。

然而可以想象，直接书写的 Java 代码，它满足上述在实现作为本发明的目的协议的场合所提及的验证准则，后者明显的目的也是实现把任何标准的 Java 代码自动转换为同一程序片断的标准化代码的方法，因而必定满

足上述已实现的验证准则。随后将在说明中详细叙述转换为标准化代码的方法及对应的系统。

现在将参照图 3a 及随后的图示，更为详细地说明根据本发明的目的验证程序片断或小应用程序的方法。

一般来说应当指出，作为本发明目的的验证方法，能够或者作为上述参照图 2 所述本发明的目的的管理程序片断的协议的部分来实现，或者独立地提供任何必须被判断的验证过程。

一般来说应当指出，程序片断由包括至少一个子程序的对象代码组成，更普遍指定的一个方法，是由板载系统的微处理器通过虚拟机可执行的一系列指令构成。

如图 3a 所示，对于每一子程序的验证方法在于执行阶段 200，这是通过数据使虚拟机的类型栈和寄存器类型表初始化，这些数据表示在开始执行作为验证目的对象代码时这一虚拟机的状态。如上参照作为本发明的目的的协议的实现所述，这一对象代码能够被暂时存储。

然后，上述的阶段 200 随后是阶段 200a，该阶段在于把当前索引为 i 的指令 I_i 的读取定位在对象代码的第一指令。阶段 200a 之后是阶段 201，该阶段在于逐个指令地进行上述对象代码的验证，这是通过对每一当前指令（指定为 I_i），辨别转移指令目标 CIB、的异常处理程序调用目标（指定为 CEM）、或子程序调用 CSR 的目标的存在性而进行的。

验证阶段 201 之后是阶段 202，该阶段验证并更新当前指令 I_i 对类型栈和寄存器类型表的数据类型的效果，这是对于由另一指令指向的当前指令，作为转移指令目标 CIB、子程序调用目标 CSR 目标或异常处理程序调用目标 CEM 存在性的函数。

对于当前指令 I_i 的阶段 202 之后是阶段 203，该阶段测试是否到达最后的指令，该测试书写为：

I_i =对象代码的最后指令?

对测试 203 的响应为负时，过程进到下一个指令 204，写为 i = i+1，并返回进到阶段 201。

应当指出，当在构成对象代码的所有指令 I_i 的验证期间寄存器类型表

没有被修改时，则上述在阶段 202 的验证已经成功。为此，提供了对寄存器类型表的稳定状态存在性的测试 205。这一测试写为：

∃? 寄存器类型表的稳定状态.

在对测试 205 正响应时，验证已经成功。

另一方面，在没有修改缺失通知的情形下，通过返回阶段 200a 验证过程被重复并被重新初始化。表明该过程保证在最大 NrxH 叠代之后结束，其中 Nr 表示使用的寄存器数目，H 表示与子分类关系有关的常数。

现在参照图 3b 给出与以上参照图 3a 所述验证过程的过程中处理的变量类型相关的各种指示。

上述变量类型至少包括对应于在受到验证的程序片断中定义的对象类的类标识符，数值变量类型至少包括 short 类型，按 p 位编代码的整数，这里 p 的值可以是 16，以及跳转指令 JSR 的返回地址的类型，这种地址类型定义为 retaddr，关于零对象参照的 null 类型，关于对象性质的 object 类型，表示所有类型的交集并且对应于值零 nul 的特定类型⊥，表示所有类型的并集并且对应的任何值类型的另一特定类型 T。

参照图 3b，应当指出所有上述变量类型要实现子分类关系：

object ε T;

short, retaddr ε T;

⊥ε null, short, retaddr

现在参照附录中表 T1 所示数据结构的第一个例子，给出如图 3a 所示验证过程的一个更具体的例子。

上述例子涉及以 Java 代码书写的小应用程序。

通过指向被验证的指令 I_i 的指针，验证过程访问形成受到验证的小应用程序的子程序的代码。

验证过程记录对应于上述表 T1 的例子中 saload 的当前指令 I_i 的执行栈的大小和类型。

然后验证过程通过其类型栈指针在类型栈中记录当前指令执行栈的大小和类型。

如本说明以上所述，这种类型栈反映了虚拟机在当前指令 I_i 处执行栈

的状态。在表 T1 所示的例子中，在将来执行指令 I_i 时，栈将包含三个项：对 C 类的对象的参照，对按 p=16 位编代码的整数表的参照，类型 short[]，以及类型 short 的 p=16 位的整数。这还示于类型栈中，该栈也包含所三个项：C，C 类对象类型，short[]，整数 p=16 位和 short 的表的类型，整数 p=16 位的类型。

另一值得注意的数据结构由整数型表组成，这种表反映虚拟机寄存器的状态，即是说存储局部变量的寄存器的状态。

继续来看表 T1 中所指的例子，指出寄存器类型表的项 0 包含类型 C，即在将要执行当前指令时 I_i = saload，保证寄存器 0 包含对 C 类对象的参照。

在验证期间处理并存储在寄存器类型表和类型栈中的各种类型表示在图 3b 中。这些类型包括：

－对应于在小应用程序中定义的特定对象类的类标识符 CB；

－基类型，诸如 short，按 P = 16 位编代码的整数，int1 和 int2，例如分别按 2p 位编代码的整数的最高和最低有效 p 位，或 retaddr，上述指令返回地址；

－类型 null，表示零对象的参照。

关于子分类关系，指出如果类型 T1 的每一有效值也是类型 T2 的有效值，则类型 T1 是类型 T2 的子类。类标识符之间的子分类反映了小应用程序的类之间的继承性体系。关于其它类型，子分类通过图 3b 所示的格定义，其中⊥是所有类型的子类型，并且所有类型是 T 的子类型。

参照上述表 T1，验证形成小应用程序的子程序的过程顺序如下。

验证过程独立于小应用程序的每一子程序进行。对于每一子程序，过程对相关子程序的指令进行一次或多次验证扫描。在附录的表 T2 中给出验证过程的伪代码。

验证子程序的过程以初始化表 T1 所示的类型栈和寄存器类型表开始，这种初始化反映了在所考察的子程序执行开始时虚拟机的状态。

类型栈最初是空的，栈指针等于零，且寄存器类型以子程序的参数类型初始化，表示这样的事实，即虚拟机扫描这些寄存器中的这一子程序的

参数。由子程序分配的寄存器类型初始化为数据类型⊥，表示这样的事实，即在子程序执行的开始，虚拟机把这些寄存器初始化为零。

然后，对子程序的指令和每一当前指令 I_i 进行一个或多个验证扫描。

在实施验证扫描或连续几次扫描结束时，例如，验证过程判断在验证扫描期间，包含在附录表 T1 所示的寄存器类型表中的寄存器类型是否已经改变。在没有改变时，验证终止并向主程序返回一个成功代码，使得能够在在图 2 所示的管理协议的阶段 105 发送正的接收确认。

如果出现对上述寄存器类型表的改变，则验证过程重复验证扫描直到包含在寄存器类型表中的寄存器类型稳定为止。

现在将参照图 3c 到 3j 说明进行一次或多次直到寄存器类型表稳定的验证扫描的特有的顺序。

对于每一当前指令 I_i，进行以下验证：

参照图 3a 在阶段 201，如上所述，验证过程判断当前指令 I_i 是否是转移指令、子程序调用或异常处理程序调用的目标。通过检验子程序及与这一子程序相关异常处理程序代码中的转移指令，进行这一验证。

参见图 3c，该图示以阶段 201 开始，在当前指令 I_i 是转移指令的目标时，这一条件通过由 I_i = CIB 表示的测试 300 实现，这一转移是无条件的或有条件的，验证过程通过测试 301 在子程序这一点检验类型栈为空。在对测试 301 为正响应时，通过标记继续 A(continue A)的上下文继续阶段继续进行验证过程。在对测试 301 响应为负时，类型栈不为空，验证失败且小应用程序被拒绝。这种失败是由失败(Failure)阶段表示的。

参见以继续 A 阶段开始的图 3d，在当前指令 I_i 是子程序调用目标时，这一条件由测试 304 I_i=CSR 实现，在测试 305 中验证过程验证前一指令 I_{i-1} 在序列中没有继续。当前一指令是无条件转移，子程序返回或引起异常时，这一验证是由测试 305 实现的。阶段 305 的测试标记如下：

$I_{i-1} = IB_{unconditional}$，返回 RSR 或引起 L-EXCEPT。

在对测试 305 有负响应时，验证过程在失败阶段失败。另一方面，在对测试 305 正响应时，验证过程重新初始化类型栈，使得它只包含 retaddr 类型的一个项，即上述子程序的返回地址。如果在阶段 304 当前指令 I_i 不

是子程序的调用目标，则验证过程在继续 B 阶段处的上下文继续。

参见图 3e，在当前指令 I_i 是异常处理程序的目标时，这一条件由标记为 I_i = CEM 的测试 307 实现，其中 CEM 表示异常处理程序的目标，这一条件是由测试 307 实现的，标记为：

I_i = CEM

在对测试 307 正响应时，过程通过测试 305 验证前一指令是无条件转移，子程序返回或异常的引起，标记为：

$I_{i-1} = IB_{unconditional}$，返回 RSR 或引起 L-EXCEPT。

在对测试 305 正响应时，验证过程进到在阶段 308 的重新更新类型栈，这是通过输入标记为 EXCEPT 类型的异常类型进行的，阶段 308 之后是上下文继续阶段，即继续 C。在对测试 305 负响应时，验证过程以标记为失败(Failure)的阶段失败。

参图 3f，在当前指令 I_i 是多个不兼容转移目标时，这一条件由测试 309 实现，它标记为：

I_i = 不兼容 XIBs

例如不兼容转移是无条件转移和子程序调用，或两个不同的异常处理程序。在对测试 309 正响应时，转移是不兼容的，验证过程以标记为失败的阶段失败，且程序片断被拒绝。在对测试 309 负响应时，验证过程以标记为继续 D 的阶段继续。测试 309 以本说明中前面提及的继续 C 阶段开始。

参见图 3g，在当前指令 I_i 不是任何转移目标时，这一条件由以上述继续 D 开始的测试 310 实现，这一测试标记为

I_i ∃? 转移目标，

其中∃表示存在符号，

在对测试 310 负响应时验证过程继续，过渡到阶段 311 的类型栈的更新，阶段 311 和对测试 310 的正响应之后是阶段 202 处的上下文继续阶段，该阶段在以上参照图 3a 的说明中已说明。

现在参照图 3h，给出在上述阶段 202 验证当前指令对类型栈的效果的阶段的更为详细的说明。

根据上述图示，这一阶段能够至少包括一个验证阶段 400，类型执行

栈至少包含为当前指令包含的操作数那样多的项。这一测试阶段 400 标记为:

Nbep ≥ NOpi

其中 Nbep 表示类型栈的项数，而 Nopi 表示包含在当前指令中的操作数的数目。

在对测试 400 正响应时，这一测试之后是使类型栈非堆栈的阶段 401a，以及 401b 验证，在栈的顶部项的类型是上述当前指令操作数类型的子类型。在测试阶段 401a，指令 i 的操作数类型标记为 Topi，在栈的顶部项的类型标记为 Targs。

在阶段 401b，验证对应于子分类关系 Topi 的 Targs 子类型的验证。

在对测试 400 和 401b 负响应时，验证过程失败，这由到失败阶段的通路表示。另一方面，在对测试 401b 正响应时，验证过程继续，并在于进行:

-验证存在足够的类型栈存储器空间，以进行当前指令结果的栈存储。这一验证阶段是由测试 402 实现的，标记为:

栈-空间≥结果-空间

其中表达式每一边表示对应的存储器空间。

在对测试 402 负响应时，验证过程失败，这由失败阶段表示。另一方面，在对测试 402 正响应时，这时验证过程进到在阶段 403 对指定给结果的数据类型进行栈存储，栈存储是在指定给这些结果的数据类型栈上进行的。

作为一个非限制性例子应当指出，为了实现图 3h 验证当前指令对类型栈的效果，当前指令由对应于读取在整数表中按 p=16 位编代码的整数元素的 Java saload 指令(这整数表由整数表和表中的整数索引定义)，以及读取这表中这一索引处的整数结果构成，对于这一当前指令，验证过程检验类型栈至少包含两个元素，在类型栈顶部的这两个元素分别是 short[]和 short 的子类型，进到非栈存储过程，并然后进到对作为结果类型的数据类型 short 进行栈存储的过程。

此外，参见图 3i，为了实现验证当前指令对类型栈的效果的阶段，在当前指令 I_i 是标记为 IR 的地址 n 的寄存器的读指令时，这一条件由标记

为 $I_i = IR_n$ 的测试 404 实现，在对上述测试 404 正响应时，验证过程在阶段 405 通过读取寄存器类型表中的项 n 验证这一读取结果的数据类型，然后通过对对应于这一当前指令操作数的栈的项的非栈存储的操作 406a，以及通过 406b 对这一结果数据类型的栈存储，确定当前指令 I_i 对类型栈的效果。指令 I_i 的操作数标记为 OP_i。阶段 406a 和 406b 之后是返回上下文继续，即继续 F。在对测试 404 负响应时，验证过程以上下文继续，即继续 F 继续进行。

参见图 3j，在当前指令 I_i 是标记为 IW 的地址 n 的寄存器的写指令时，这一条件由标记为 $I_i = IW_m$ 的测试实现，在对测试 407 正响应时，验证过程在阶段 408 判断当前指令对类型栈和写入地址 n 的寄存器的操作数的类型 t 的效果，然后在阶段 409，以紧靠在先前存储的类型之上和写入地址 n 的寄存器的操作数类型 t 之上的类型，替换地址 n 处的寄存器类型表的类型项。阶段 409 之后是返回上下文继续，即继续 204。在对测试 407 负响应时，验证过程由上下文继续，即继续 204 继续进行。

作为一个例子，在当前指令 I_i 对应于把类型 D 的值写入地址 1 的寄存器，且寄存器 1 的类型在指令验证之前为 C 时，则寄存器 1 的类型由类型 object 替换，这是图 3b 所示的类型格中高于 C 和 D 的最小类型。

同样地作为一例，在当前指令 I_i 是对寄存器 0 的内容进行栈存储的指令 aload-0 的读取，且寄存器类型表的项 0 为 C 时，则验证程序把 C 栈存储到类型栈。

现在将参照附录中表 T3 和 T4，给出验证在 Java 环境中书写的子程序的一例。

表 T3 表示对应于包含在这一表中的 Java 子程序的特定 JavaCard 代码。

表 T4 表示寄存器类型表和类型栈在每一指令验证之前的内容。对各种指令操作数的类型约束都可观察到。在由箭头表示的指令 5 转移到指令 9 之后以及上述转目标 9 之前栈都是空的。当检验到指令 1 在寄存器 1 存储类型 null 的值时，其初始时为⊥的寄存器 1 的类型变为 null，即 null 和⊥的上界，然后当处理指令 8 在寄存器 1 中存储类型 short[]的值时，变为

类型 short[]，即类型 short[]和 null 的上界。如果在第一个验证扫描期间寄存器 1 的类型已经改变，则进行第二次扫描，分配在第一次结束时获得的寄存器类型。这第二次验证扫描如同第一次扫描那样成功，且不改变寄存器类型。于是验证过程成功终止。

现在参见附录中的表 T5 给出验证过程对不正确代码的四个例子的失败情形的各种例子:

-在表 T5 的点 a)，作为例子所给出的代码的目的是使用关于指点器的算术过程试图形成一个无效的对象参照。通过指令 2 sadd 的变元类型的验证它被拒绝，该指令要求这两个变元为类型 short。

-在表 T5 的点 b)和 c)，代码的目的是要进行两种试图，以便把任何整数转换为一个对象参照。在点 b)，寄存器 0 与类型 short，指令 0 一同使用，而指令 5 与类型 null 一同使用。于是，验证过程向记录 0 指定类型 T，并当寄存器 0 作为指令 7 处的类型 object 的结果而被返回时，检测类型错误。

-在表 T5 的点 c)，一组类型转移“if ... then ... else ...”用来在栈的顶部留下由整数或对象参照构成的结果。验证过程拒绝这种代码，因为在从由箭头表示的指令 5 向指令 9 的转移时检测出栈不是空的。

-最后在表 T5 的点 d)，代码包括一个循环，该循环在每一次叠代的效果是要在栈的顶部栈存储另外的一个整数，于是在一定数目的叠代之后将引起栈溢出。验证过程拒绝这种代码，在由返回箭头表示的从指令 8 向指令 0 向后转移时通知栈为非空。在转移点栈不是空的。

以上参照表 T3，T4 和 T5 给出的各种例子表明，作为本发明的目的的验证过程是特别有效的，它可适用于小应用程序，并特别适用于其子程序，对于它们，栈类型，类型栈的先前空字符，以及到转移指令或转移目标的条件都分别满足。

明显地，这种验证过程蕴含着满足这些准则的写入对象代码，这些对象代码可能对应于上述表 T3 中的子程序。

然而，为了保证不一定满足作为本发明的目的的方法的验证准则的现有的小应用程序和小应用程序的子程序的验证，特别关于在 Java 环境中书

写的小应用程序和子程序，本发明的目的是要建立把这些小应用程序或子程序转换为标准化小应用程序或程序片断的方法，使得能够成功地进行作为本发明的目的的验证方法以及实现这种方法的管理协议的验证测试。

为此，本发明的目的是实现用于转换形成小应用程序的传统的对象代码的方法和程序，这种相关的小应用程序在生成时，它能够在板载系统或微处理器之外实现这种方法和这种转换程序。

作为一个纯粹的示例，现在在 Java 环境的框架下说明，作为本发明的目的的把代码转换为标准化代码的方法。

通过现有的 Java 编译程序产生的 JVM 代码满足各种准则，现叙述如下：

C1：每一指令的变元实际上属于这一指令预期的类型；

C2：栈不溢出

C'3：对于每一转移指令，这一转移处的栈类型与对于这一转移可能的目标处的类型相同；

C'4：在代码的一点写入寄存器的以及在代码的另一点从同一寄存器再读取的类型 t 的值，总是以相同的类型 t 被再读取；

准则 C'3 和 C'4 被为验证所提供的对象代码所验证，作为本发明的目的的验证方法的实现蕴含着它们将由以下准则 C3 和 C4 代替：

C3：在每一转移指令处及在每一转移目标处栈是空的；

C4：相同的寄存器由子程序所有的代码的同一类型使用；

参见上述的准则，应当指出，Java 编译程序只保证了较弱的准则 C'3 和 C'4。作为本发明目的的验证过程及对应的管理协议事实上保证了更严格的准则 C3 和 C4，使得能够保证小应用程序执行和管理的可靠性。

包括代码向标准化代码的转换的标准化的概念可表现出各个方面，就以准则 C3 和 C4 代替准则 C'3 和 C'4 来说，依照作为本发明的验证过程，能够被独立地实现，以保证在每一转移指令处及每一转移目标处栈为空，并还保证对小应用程序打开的寄存器分类型，且为相关小应用程序执行所指定的单一数据类型对应于每一打开的寄存器，或另一方面，同时满足了作为本发明目的的整个验证过程。

以下将以所述的两种不同的实现方式说明，把对象代码转换为本发明

中所述的标准化代码的方法，第一种实现方式对应于把满足准则 C1，C2，C`3，C`4 的对象代码转换为满足准则 C1，C2，C3，C`4 的标准化对象代码，它们对应于带有空转移指令或转移目标的标准化代码，然后，如所述在第二实现方式中，其中满足相同初始准则的传统对象代码被转换为满足准则 C1，C2，C`3，C4 的标准化对象代码，它们例如对应于使用被分类的寄存器的标准化代码。

现在将参照图 4a 说明作为本发明的目的的代码转换方法的第一实现方式。在图 4a 所示的实现方式中，认为初始的传统代码满足准则 C1+C2+C`3，而认为作为转换结果获得的标准化代码要满足准则 C1+C2+C3。

根据上述图示，对于代码或子程序的每一当前指令 I_i，转换方法在阶段 500，在这一指令执行之前和之后对每一指令以栈的数据类型进行注释。注释数据标记为 AI_i，并对其以相关的当前指令关系 $I_i \leftrightarrow AI_i$ 相关联。通过分析与这一指令相关的数据流计算出注释数据。指令执行之前和之后的数据类型分别标记为 tbe_i 和 tae_i。通过分析数据流计算注释数据是业内专业人员所熟知的一种传统的计算方法，故在此不再详述。

在阶段 500 所实现的操作表示在附录的表 T6 中，其中对于包含 12 个指令的小应用程序或小应用程序的子程序，引入了由寄存器类型和栈类型构成的注释数据 AI_i。

然后上述的阶段 500 之后是阶段 500a，该阶段使索引 I 定位在第一指令 $I_i = I_1$。阶段 500a 之后是阶段 501，该阶段是在指令中及每一当前指令 I_i 中检测标记为 IB 的转移或其执行栈非空的转移目标 CIB 的存在性。这一检测 501 是通过一种测试实现的，该测试是基于分配给每一当前指令的栈变量的类型的注释数据 AI_i 而进行的，对于当前指令该测试标记为：

I_i 是 IB 或 CIB 且栈(AI)?空。

在对测试 501 正响应时，即呈现出检测到非空执行栈，则上述测试之后是在这些转移 IB 一侧或在转移目标 CIB 一侧插入转移栈变量指令的阶段，以便在这一转移之前把执行栈的内容清空到暂时寄存器，并在这一转移之后从暂时寄存器重新建立执行栈。在图 4a 中该插入阶段标记为 502。这阶段之后是阶段 503，以测试到达最后的指令，标记为

I_i = 最后的指令?

在对测试 503 为负响应时，进行 504 的增量 i=i+1，进行到下一个指令而返回阶段 501。在对测试 503 为正响应时起动一结束阶段。在对测试 501 为负响应时，在没有插入转移指令之下，转换方法通过向阶段 503 转移而继续进行。如图 4a 所示，把传统的代码转换为带有具空栈转移指令的标准化代码的方法的实现，使得能够在不对程序片断的执行作任何修改的情形下，获得对于相同初始程序片断的标准化对象代码，其中在每一转移指令处和每一转移目标指令处栈变量的栈为空。 在 Java 的环境情形下，在栈和寄存器之间转移数据的指令是 Java 虚拟机的 load 和 store 指令。

现在返回表 T6 中引入的例子，该转换方法在指令 9 处检测到栈非空的一个转移目标。然后该方法在导致上述指令 9 的转移指令 5 之前插入一指令 istore 1，以便在寄存器 1 中保存栈的内容，并保证该栈在转移时为空。对称地，在指令 9 之前插入一指令 iload 1，以便建立栈的内容完全与转移之前一样。最后，在指令 8 之后插入指令 istore 1，以保证栈在导致指令 9 的两个通路上平衡。在表 T7 中示出这样进行的向标准化代码转换的结果。

现在将参照图 4b 说明作为本发明的目的的转换方法的第二个实现方式，所在的情形是，初始的传统对象代码满足准则 C1+C`4，而标准化对象代码满足准则 C1+C4。

参见上述图 4b，应当指出，按这一实现方式的方法在所述的阶段 500，与图 4a 所示的情形几乎相同，是在这一指令执行之前和之后以寄存器数据类型注释每一指令 I_i。同样地，通过分析与这指令相关的数据流计算出注释数据 AI_i。

然后，注释阶段 500 之后是进行配寄存器重新分配的阶段，该阶段标记为 601，重新分配的进行是通过检测以不同类型使用的原始寄存器，并把这些原始寄存器划分为分开的标准化寄存器，一个标准化寄存器分配给使用的每一数据类型。阶段 601 之后是阶段 602，该阶段重新更新处理使用上述标准化寄存器的操作数的指令。阶段 602 之后是上下文继续阶段 302。

参见在表 T6 中给出的例子，应当指出，转换方法检测标记为 r0 的序号 0 寄存器用于两种类型，即 object，指令 0 和 1，以及 int，指令 9 及随后的指令。然后该方法将把原始寄存器 r0 划分为两种寄存器，即用于 object 类型的寄存器 0，及用于 int 类型的寄存器 1。然后通过把它们转换为对记录 1 的参照而重写对 int 类型的记录 0 的参照，获得的标准化代码表示在附录的表 T8 中。

要注意，以非限制性方式在参见上述表 T8 引入的例子中，新的寄存器 1 同时用于通过把寄存器 0 划分为两种寄存器而使栈标准化及分类的寄存器的生成。

现在，在一优选的与图 5a 相关的非限制性例子中，将更为详细地说明把传统的代码转换为带有具图 4a 所示空栈的转移指令的标准化代码的方法。

这一实现方式涉及阶段 501，该阶段在于在指令内及当前指令 I_i 内，检测转移 IB 或分别地栈为非空的转移目标 CIB 的存在性。

在判断栈为非空的目标指令之后，这一条件在阶段 504a 标记为 I_i 栈≠空，转换过程在上述阶段 504a 使一组新的寄存器与这些指令相关联，每一在这些指令处有效的栈单元一个。于是，如果 i 表示其相关的栈类型非空的转移目标的序号，且是类型 $tp1_i$ 到 tpn_i，其中 n>0，栈非空，则转换过程分配 n 个新的未使用的寄存器 r_1 到 r_n，并使它们与对应的指令 i 相关联。该操作在阶段 504a 处实现。

阶段 504a 之后是阶段 504，该阶段在于检验每一已检测的序号 i 的指令，并在测试阶段 504 鉴别转移目标 CIB 或转移 IB 的存在性。阶段 504 以由以下所示的测试形式示出：

$\exists$? CIB, IB, andI_i = CIB。

在序号 i 的指令是由以上等式表示的转移目标 CIB，且在这一指令处的栈变量的栈非空，即对测试 504 的正响应的情形下，对由转移组成的序号 i-1 的每一个前边指令，异常的引起或程序返回，这一条件在测试阶段 505 实现，表示为：

I_{i-1} = IB, EXCEPT 引起，Prog. 返回。

被检测的序号 i 的指令只能由转移访问。在对上述测试 505 正响应时，转换过程执行阶段 506，该阶段是在相关的被检测的序号为 i 的指令之前插入来自新的寄存器集的一组 load 类型的加载指令。插入操作 506 之后是 507 使所有指向被检测的序号为 i 的指令的转移重新定向到第一插入的加载指令 load。插入和重新定向操作示于附录中表 T9 中。

对于按顺序连续进行的序号 i-1 的每一个前边指令，即序号 i 的当前指令可同时由一转移和来自前面指令访问，这一条件由测试 508 实现，并由以下关系表示：

$I_{i-1} \rightarrow I_i$

以及

$IB \rightarrow I_i$

转换过程在于阶段 509，该阶段在被检测的序号 i 的指令之前插入备份一组新的寄存器的备份指令 store，以及从这一组新的寄存器加载的一组加载指令 load。然后阶段 509 之后是阶段 510，该阶段是把所有指向被检测的序号 i 的指令的转移重新定向到第一插入的加载指令 load。

在被检测的序号 i 的指令是向一已判断的指令转移的情形下，对于任何由无条件转移构成的被检测的序号 i 的指令，这一条件由测试 511 实现，标记为：

$I_i = IB_{uncondit}$

如图 5a 所示转换过程在对测试 511 正响应时，在阶段 512，在检测的序号 i 的指令之前插入多个备份指令 store。作为一例如表 T11 所示，转换过程在指令 i 之前插入 n 个指令 store。指令 store 寻址寄存器 r_1 到 r_n，其中 n 表示寄存器的数目。这样，备份指令与每一新的寄存器相关联。

对于由条件转移组成的每一被检测的序号 i 的指令，以及对于由这一条件转移指令处理的操作数大于 0 的数目 mOp，这一条件由测试 513 实现，标记为：

$I_i = IB_{condit}$

其中 mOp>0

在对上述测试 513 正响应时，转换过程在阶段 514，在被检测的序号 i 的

指令之前，在序号 i 的被检测指令的 mOp 操作数及 n 随后的值的栈变量的栈的顶部，插入标记为 swap_x 的置换指令。置换操作使得能够在栈变量的栈顶部收集 n 个值以便备份在新的寄存器组 r_1 到 r_n。阶段 514 之后是阶段 515，是在序号 i 的指令之前插入一组备份指令 store 以便备份新的寄存器集 r_1 到 r_n。上述插入阶段 515 本身之后是阶段 516，该阶段是在被检测的序号 i 的指令之后插入一组加载指令 load 以便从新的寄存器组 r_1 到 r_n 加载。对应的插入操作集在附录的表 12 中示出。

为了完整性并参见图 5a，应当指出，在对测试 504 负响应时，转换过程的继续是由上下文继续阶段即阶段 503 实现的，对测试 505，508，511 和 513 的负响应本身也是通过上下文继续阶段即阶段 503 由转换过程的继续跟随的，以及这同样适用于在上述重新定向阶段 507 和 510 及插入阶段 512 和 516 之后的操作的继续。

现在将参照图 5b 给出，使用图 4b 所示的已分类型的寄存器，使对象代码标准化并将其转换为标准化对象代码的方法更为详细的说明。更具体来说，这一实现方式涉及阶段 601 的这一非限制性的优选实现方式，以便通过检测用于不同类型的原始寄存器而重新分配寄存器。

参见上述图 5b，要指出的是上述阶段 601 在于在阶段 603 判断标记为 ID_j 的每一寄存器 r_j 的生命区间。这些生命区间，表示为“live range”或“webs”，对一个寄存器 r 定义为局部迹的最大集，使得寄存器 r 在这些迹的所有点处都是生存的。这些概念更为详细的定义宜参见由 Steven S. MUCHNIK 编辑的著作，标题为“Advanced Compiler Design and Implementation”，16.3 节，Morgan KAUFMANN,1997。阶段 603 由以下关系表示：

$$ID_j \leftrightarrow r_j$$

其中所述对应的的生命区间 ID_j 与每一寄存器 r_j 相关联。

上述阶段 603 之后是阶段 604，是在阶段 604 判断标记为 tp_j 的每一生命区间 ID_j 的主数据类型。对于寄存器 r_j 生命区间 IDj 的主类型是由属于上述生命区间的备份指令 sotre 存储在这一寄存器 r_j 中的数据类型的上界定义的。

阶段 604 本身之后是阶段 605，这阶段在于在如以上阶段 603 和 604 定义的生命区间之间建立一种干扰图，这种干扰图由非有向图构成，其每一顶点由生命区间构成，而如果顶点包含对其它顶点的寄存器寻址的备份指令则其在图 5b 中标记为 $a_{j1,j2}$ 的两个顶点 ID_{j1} 和 ID_{J2} 之间的弧存在，或反之也然。图 5b 中，干扰图的结构被示意性表示出，基于业内专业人员所知道的计算技术能够实现这种结构。对于这类图的结构更详细的描述，宜参见由 Alfred V. AHO, Ravi SETHI 及 Jeffrey D. ULLMAN 发表的著作，标题为“Compilers: principles, techniques, and tools”, Addison-Wesley 1986, Section 9.7。

阶段 605 之后，图 5b 所示的标准化方法在阶段 606，通过在顶点对的两个顶点没有同一相关的主数据类型时，在干扰图的所有顶点对之间添加弧，转化分配给干扰图中每一寄存器 r_j 的数据类型的唯一性。应当理解到，分配给每一寄存器的数据类型的唯一性的字符转化明显对应于转化并把准则 C4 纳入干扰图，这一准则在说明的前面提及。然后，上述阶段 606 之后是阶段 607，其中进行干扰图的示例化，这种示例化更一般地表示为如通常的技术所述的干扰图的绘图阶段。在阶段 607 期间，转换过程向每一生命区间 ID_{jk} 指定一检寄存器号码 rk，使得干扰图中两个邻接的区间收到不同的寄存器号码。

能够基于任何适当的过程实现这一操作。作为一非限制性例子，指出一种优选的过程可以是：

a) 在干扰图中选择最小阶顶点，最小阶定义为邻接顶点最小数，并从图中取消它。这阶段能够被重复直到图变为空。

b) 每一先前取消的顶点按它们取消的反序被重新引入干扰图，最后被取消的顶点是首先引入的顶点，即按取消顺序的反序顺序进行。于是能够向每一被重新引入的顶点指定不同于指定给所有邻接顶点号码的最小寄存器号码。

最后，通过示于图 4b 中的阶段 602，转换和重新分配过程向相关小应用程序的子程序的代码中的寄存器重新写入访问指令。在对应的的生命区间内访问给定的寄存器由访问不同的寄存器代替，其号码是在示例化阶段被指

定的，还指定绘图阶段。

现参照图 6 给出板载数据处理系统以及小应用程序开发系统的详细说明，该板载数据处理系统使得能够实现如本发明的目的中所述的程序片断或小应用程序的管理协议和验证过程。

关于带有标号 10 的对应的的板载系统，回忆起这一板载系统就是包括如图 1b 中所示主要组件的可重编程型系统。认为上述的板载系统通过串行链路与终端互连，终端本身例如通过局域网连接，如果有适当的远程网络，则连接到标号为 20 的小应用程序开发计算机。在板载系统 10 上运行一主程序，该主程序读取并执行终端在串行链路上发送的命令。此外，微处理器卡的标准命令，诸如 ISO 7816 协议的标准命令，能够被实现，且主程序识别两个附加的命令，一个用于小应用程序的远程加载，另一个用于选择先前已经加载到微处理器卡的小应用程序。

根据本发明的目的，主程序的结构是这样实现的，即遵循参照图 2 的说明中如上所述用于管理下载程序片断的协议，包含至少一个用于管理和验证下载的序片断的程序模块。

此外，遵循在参照图 3a 到 3j 的说明中以上所述的验证方法，该程序模块还包含一子程序模块以便验证下载程序片断。

因此，存储器的结构，特别是非可写永久性存储器 ROM，被这样修改，使得除了主程序之外特别包含如上所述的一个协议管理和验证模块 17。最后，关于 EEPROM 型的非易失可重写存储器，最好包含一个小应用程序目录，标记为 18，使得能够实现作为本发明的目的的管理协议和验证过程。

参见同一图 6，应当指出，根据本发明的目的的小应用程序开发系统，实际上如说明中以上所述使得能够把传统的满足 Java 环境框架中的准则 C1+C2+C`3+C`4 的对象代码，转换为同一程序片断的标准化对象代码，与传统的 Java 编译器 21 相关，包含标记为 22 的一个代码转换模块，正如以上参照图 4a，4b和5a和5b的说明中的第一实现方式和第二实现方式所述，该模块进行把代码转换为标准化代码。事实上应当理解，一方面，原始对象代码标准化为带有空栈的转移指令的标准化对象代码，并使用划分类型的寄存器转换为标准化代码，另一方面，如说明中以上所述，使得能够满

足由作为本发明的目的的验证方法施加的验证准则 C3 和 C4。

代码转换模块 22 之后是 JavaCard 转换器 23，该转换器使得能够保证通过远程或局域网向终端，并通过串行链路向微处理器卡 10 传输。于是，图 6 所示的小应用程序开发系统 20 使得能够把由 Java 编译器 21 从小应用程序的 Java 源代码产生的已编译的类文件，转换为等效的但是符合额外的约束 C3，C4 的类文件，这些约束是由板载微处理器卡 10 的管理协议和验证模块施加的。这些已转换的类文件在卡上由标准的 JavaCard 转换器 23 转换为可下载的小应用程序。

现在给出作为本发明的目的的协议成分，方法和系统集合值得注意的各种组成部分，只供参考。

与说明书引言中提及的先有技术验证过程相比较，作为本发明的目的的验证方法值得注意的表现在于，该方法把验证的着重点集中在操作数的划分类型的性质上，即遵从与每一指令相关的类型约束且没有栈溢出，它们对于每一小应用程序的执行的可靠性是至关重要的。其它的验证就可靠性而言没有显现出重要性，特别是在第一次读取代码之前代码正确初始化每一寄存器的验证。反之，作为本发明的目的的验证方法，是通过在方法被初始化时把所有来自的虚拟机的寄存器初始化为零而操作的，以便保证读取非初始化的寄存器不会削弱卡的可靠性。

此外，由作为本发明的目的的验证方法施加的要求，如所述其中在每一转移或转移目标指令处栈必须是空的，这保证了在转移执行之后，以及程序已经转移到的指令执行之前，栈处于相同的空的状态。这种操作方式保证了栈处于一致的状态，而不论通过相关的子程序或小应用程序的代码被跟随的执行程序如何。这样，即使存在转移或转移目标，也保证了栈的一致性。先有技术的方法和系统中必须在随机访问存储器中保留每一转移目标的栈类型，这必须有正比于 TpxNb 的随机访问存储器的量，TpxNb 是所使用的执行栈大小与代码中转移目标数的乘积，与此相反，作为本发明的目的的验证方法只需要验证期间指令时间的执行栈类型，并不在代码的其它点在存储器中保持这种栈类型。因而，与 Tp 成正比，而与 Nb 无关的，于是与子程序或小应用程序的代码的长度成正比的随机访问存储器的量，

即可满足作为本发明的目的的方法。

准则 C 中所述的要求，其中所述给定的寄存器必须在子程序所有代码中与同一类型使用，保证了上述代码不会以不一致的方式使用一寄存器，例如，在程序的一点向寄存器写入 short 整数，而在程序的另一点作为对象参照读取之。

在先有技术中所描述的验证过程中，特别是在先前提及的由 Tim LINDHOLM 和 Frank YELLIN 编辑的标题为 "The Java Virtual Machine Specification" 的 Java 规范中，为了保证上述通过转移指令使用的一致性，必须在随机访问存储器中保持每一转移目标处的寄存器类型的表的拷贝。这种操作必须有正比于 $T_r x N_b$ 的随机访问存储器的量，其中 T_r 表示由子程序使用的寄存器数目，而 N_b 表示这一子程序的代码中转移目标数。

与此相反，作为本发明的目的的验证过程在寄存器类型的全局表上进行操作，而无需在随机访问存储器中保持代码的不同点处的拷贝。因而为了实现验证过程所需的随机访问存储器与 T_r 成正比而与 N_b 无关，于是与相关的子程序代码长度成正比。

如所述在所有的点，即在相关代码的每一指令处，给定的寄存器要与同一类型使用这样的限制，相当程度地简化了子程序的验证。反之，在先有技术的验证过程中，在没有这种限制的情形下，验证过程必须建立严格的栈规则，并必须关于一定的寄存器类型多方面验证子程序体。

总之，与先有技术相比，作为本发明的目的的验证过程，一方面使得能够降低执行验证方法的程序代码量，另一方面，能够降低在验证操作期间随机访问存储器的消耗，在作为本发明的目的的验证过程的情形下复杂度为 $O(T_p+P_r)$ 形，而不是对于先有技术验证过程的 $(O(T_p+P_r) x N_b)$，然而却提供了关于被验证代码执行的可靠性的同样的保证。

最后，把原始的传统的代码转换为标准化代码是通过代码的本地化的转换实现的，而无需向验证器组件，即微处理器卡或板载数据处理系统传输额外的信息。

关于图 4b 和 5b 中所述的重新分配寄存器的方法，这种方法不同于已知的先有技术的方法，如美国专利 4,571,678 和 5,249,295 中具体所述的

方法，不同在于以下事实：

－寄存器重新分配保证了同一寄存器不会分配给两个带有不同主类型的区间，这样就保证了给定的寄存器在所有代码中与同一类型使用；以及

－在以上文献中所述的现有的寄存器分配算法假设固定数目的寄存器，并试图使在寄存器和栈之间的转移最小化，称为“溢出(spills)”，然而，作为本发明的目的中所述的寄存器重新分配以寄存器总数可变的框架操作，其结果是当进行使寄存器总数最小化的过程时，不必在寄存器与栈之间进行转移。

作为本发明的目的的管理下载到板载系统的程序片断的协议，以及分别验证这种下载的程序片断对象代码及转换这种下载的程序片断对象代码的方法，当然能够以软件实现。

因而，本发明还涉及能够直接加载到可重新编程的板载系统的内部存储器的计算机程序产品，这种板载系统使得能够下载由对象代码即一系列指令组成程序片断，这种对象代码可由板载系统的微处理器以虚拟机的方式执行，该虚拟机装有执行栈和本地寄存器或通过这些指令处理的变量，使得这种代码能够被解释。对应的计算机程序产品包含对象代码部分，以便当这种板载系统与终端互连且这种程序由这种板载系统以虚拟机的方式执行时，执行用于管理下载到这种板载系统的程序片断的协议，如以上说明中所述的图 2 和 6 所示。

本发明还涉及一种计算机程序产品，该产品可直接加载到诸如带有可重写存储器的微处理器卡，这种可重新编程的板载系统的内部存储器。这种计算机程序产品包含对象代码部分，以便执行对下载到这种板载系统的程序片断进行验证的的阶段，如以上说明中参照图 3a 到 3j 图所示和所述。当这种板载系统与终端互连，且这种程序通过这种板载系统由虚拟机执行时，执行这种验证。

本发明还涉及一种计算机程序产品；这计算机程序产品包含对象代码部分，以执行把程序片断的对象代码转换为这同一程序片断的标准化对象代码的转换方法的各阶段，如图 4a，4b，5a，5b 所示，及本说明中以上所述。

本发明还涉及一种计算机程序产品，该产品记录在可用在可重新编程的板载系统中的介质上，例如装有可重写存储器的微处理器，这种板载系统使得能够下载由对象代码组成的程序片断，这种对象代码可由这种微处理器以虚拟机方式执行，虚拟机装有执行栈及通过这些指令处理的本地变量或寄存器，以便允许对这种对象代码进行解释。上述的计算机程序产品至少包含可由板载系统的微处理器通过虚拟机读取的程序模块，以便命令执行用于对被下载的程序片断进行下载管理的过程，如图 2 所示并如以上说明中所述，一种可由微处理器通过虚拟机读取的程序模块，以便命令执行用于逐个指令验证构成这种程序片断的对象代码的过程，如以上说明中参照图 3a 到 3j 所示和所述，以及一种可由这种板载系统通过虚拟机读取的程序模块，以便把这种程序片断的对象代码转换为同一程序片断的标准化对象代码之后或在没有这样作的情形下，命令执行被下载的程序片断，如图 2 所示。

上述的计算机程序产品还包含可由微处理器通过虚拟机读取的一个程序模块，以便在上述程序片断不成功验证过程的情形下，命令在板载系统上禁止执行该程序片断，如以上参照图 2 的说明中所示和所述。

附录

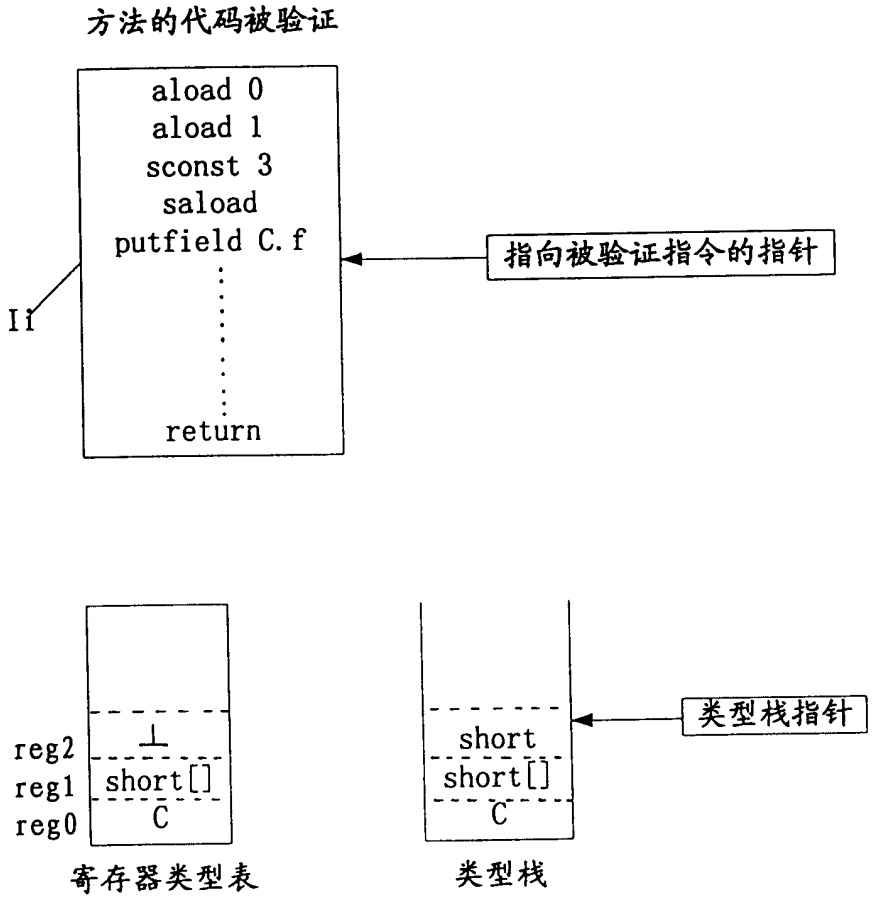

方法的代码被验证
aload 0
aload 1
sconst 3
saload
putfield C.f
return
If
指向被验证指令的指针
reg2
reg1
reg0
⊥
short[]
C
寄存器类型表
short
short[]
C
类型栈指针
类型栈

表 2

验证程序模块的伪代码

验证程序模块的伪代码

所使用的全局变量:

T_r 当前方法要求的寄存器数

T_p 当前方法要求的栈最大容量

$tr[T_r]$ 寄存器类型表(图 4 中的 402)

$tp[T_p]$ 栈类型(图 4 中的 403)

pp 栈指针(图 4 中的 404)

chg 指示 tr 是否已被改变的标志

初始化 $pp \rightarrow 0$

初始化来自方法的 *n* 个变元的类型 $tp[0] \ldots tp[n-1]$

初始化 $tp[n] \ldots tp[T_r-1]$ 为⊥

初始化 *chg* 为真

当 *chg* 为真时:

复位 *chg* 为假

定位在方法的第一指令

当没有到达方法的结束时:

如果当前指令是一转移指令的目标

如果 $pp \neq 0$, 验证失败

如果当前指令是一子程序调用的目标

如果先前的指令按顺序继续进行, 则失败

取 $tp[0] \leftarrow$etaddr 以及 $pp \leftarrow 1$

如果当前指令是 C 类异常处理程序:

如果先前的指令按顺序继续进行，则失败

作 $tp[0] \leftarrow C$ 以及 $pp \leftarrow 1$

如果当前指令是不同种类的目标:

验证失败

判断指令变元的类型 $a_1 \ldots a_n$

如果 $pp<n$，则失败(栈溢出)

对于 $i = 1, \ldots, n$

如果 $tp[pp-n-i-1]$ 不是 a_i 的子类型，则失败

作 $pp \leftarrow pp-n$

判断指令结果的类型 $r_1, \ldots, r_m$

如果 $pp+m \geq T_p$，则失败(栈溢出)

对于 $i = 1, \ldots, m$，作 $tp[pp+i-1]$？r_i

作 $pp \leftarrow pp+m$

如果当前指令是向一寄存器 r 写入:

判断向记录写入的值的类型 t

作 $tr[r] \leftarrow lower\ bound\ (t, tr[r])$

如果 $tr[r]$ 已经改变，作 $chg \leftarrow$真

如果当前指令是一转移:

如果 $pp \neq 0$，验证失败

进到下一个指令

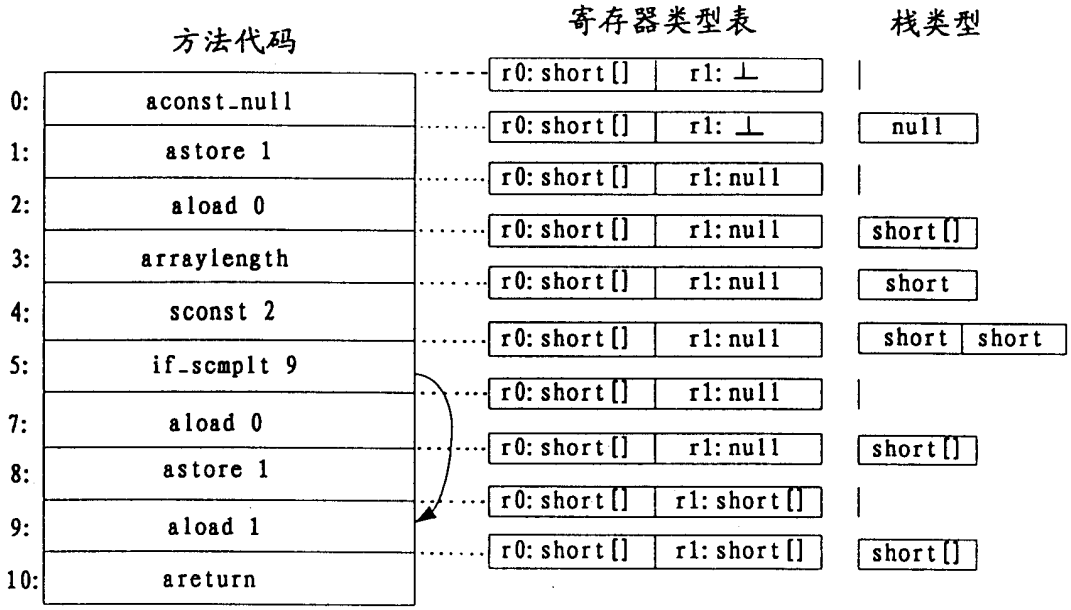
返回验证成功代码

表 T3

```
static short [] meth (short [] table )
 {
     short [] result        =null;
     if (   table length      >=2) result=table;
        return table
}
```

表 T4

方法代码上的第一次叠代:



对方法代码上的第二次叠代:

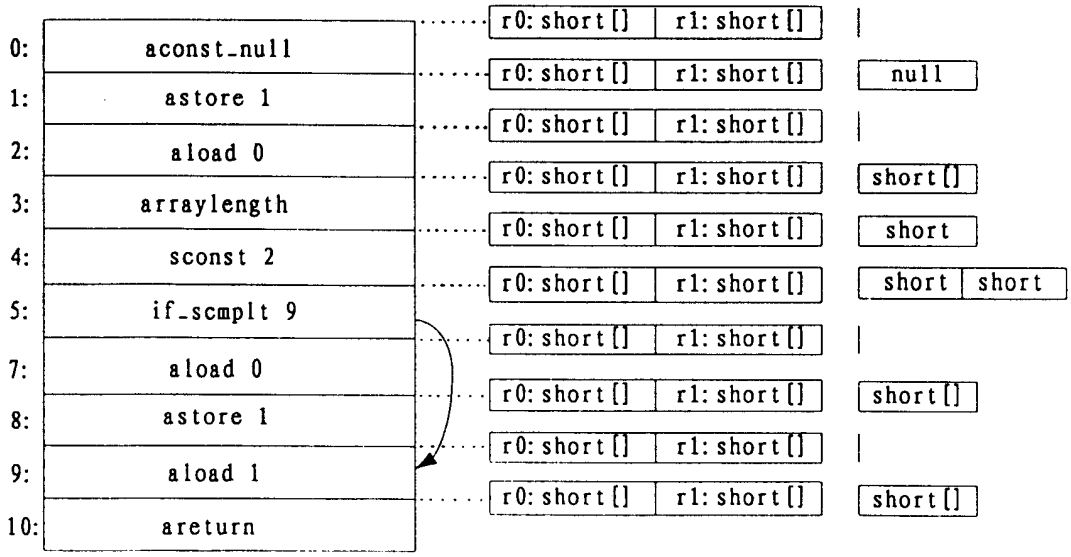
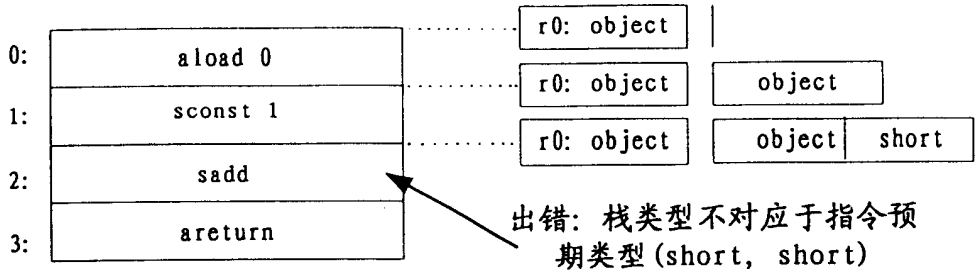
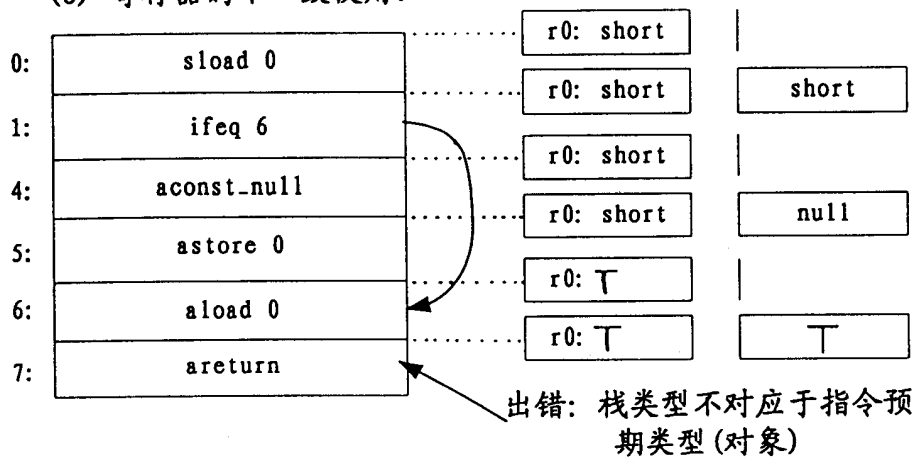


表 T5

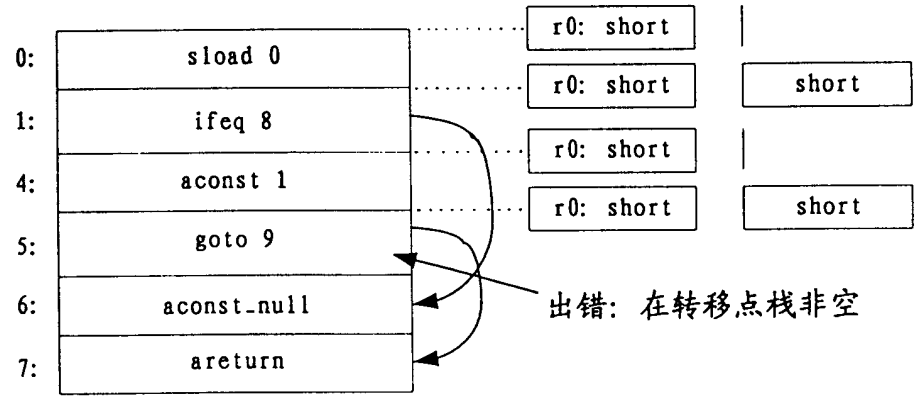
(a) 对一个指令变元的类型约束的违反:



(b) 寄存器的不一致使用:



(c) 转移引入栈水平的不一致:



(d) 在循环内栈溢出:

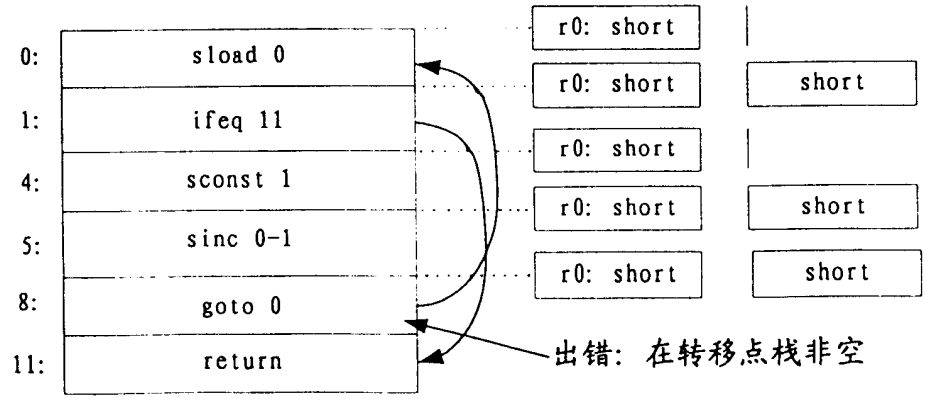


表 T6

(a) 方法的初始代码，通过寄存器和栈类型注释

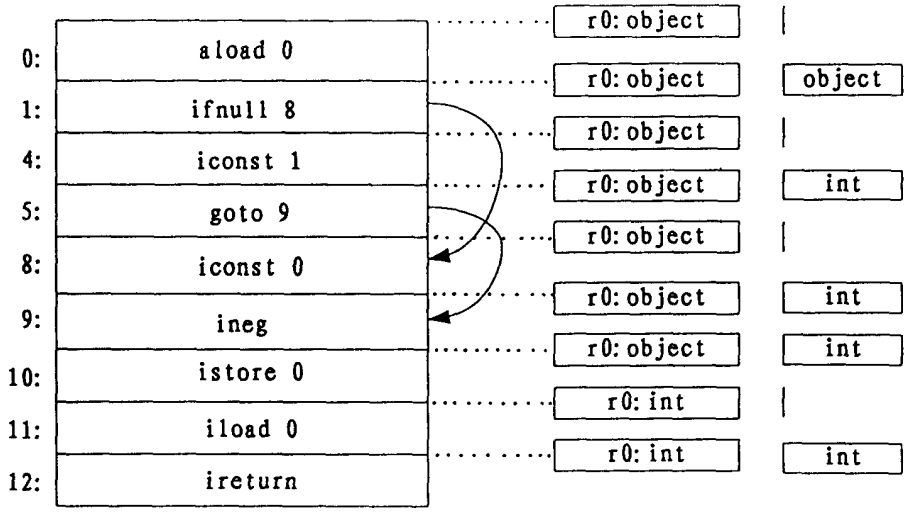


表 T7

(b) 在转移5—>9处栈的标准化之后的方法代码：

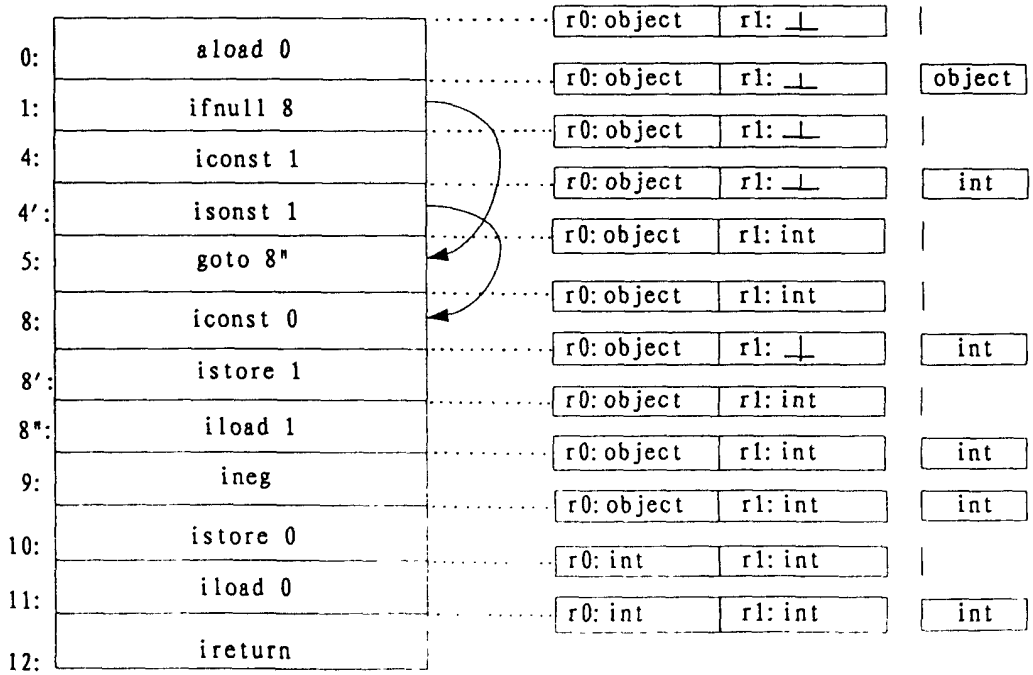


表 T8

(c) 在寄存器重新分配之后的方法代码：

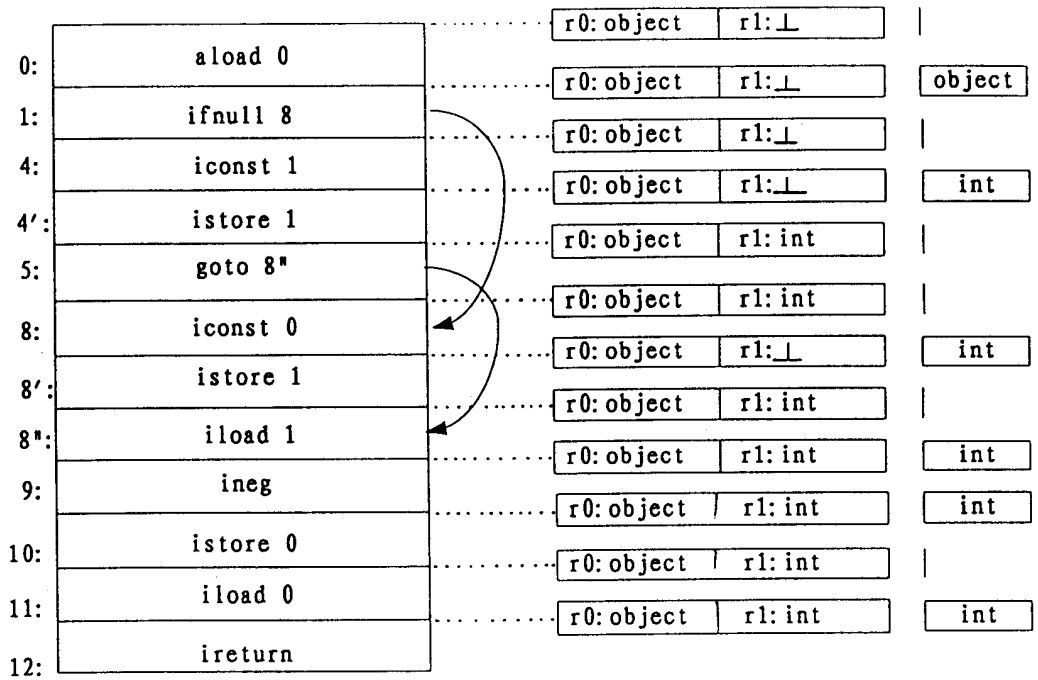


表 T9

(a) 转移目标，在顺序中前一指令没有继续：

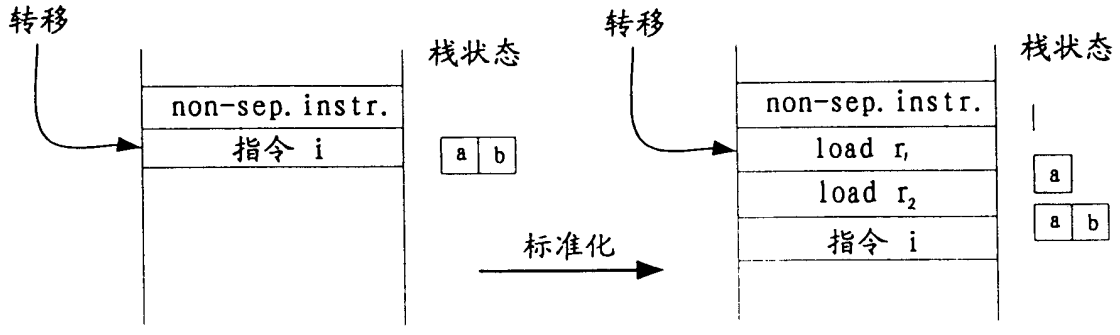


表 T10

(b) 转移目标，在顺序中前一指令继续：

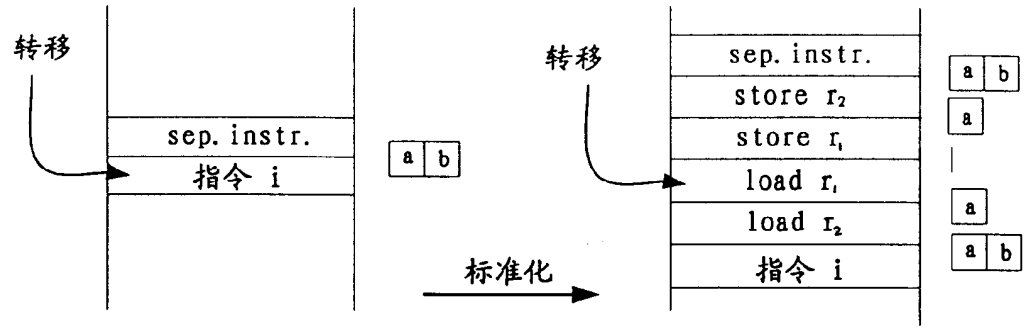


表 T11

(c) 无变元的无条件转移：

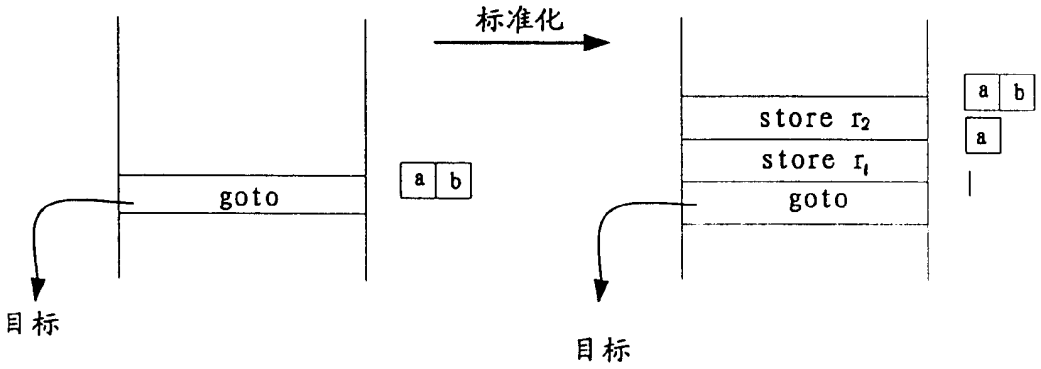
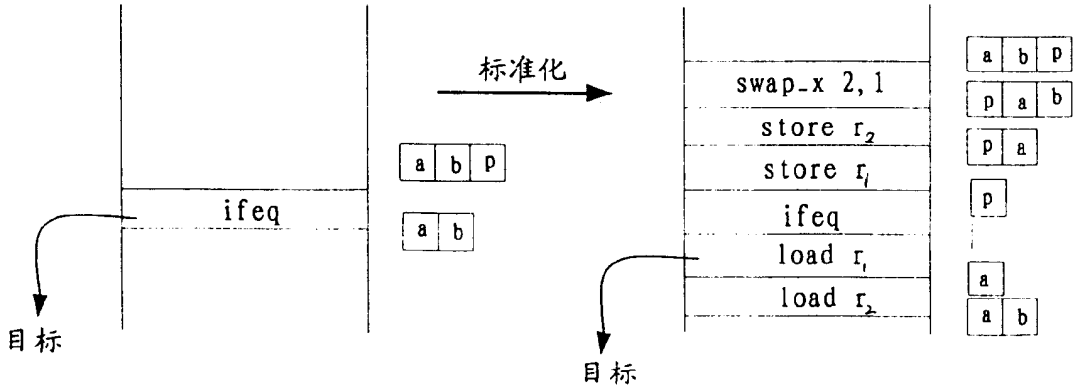
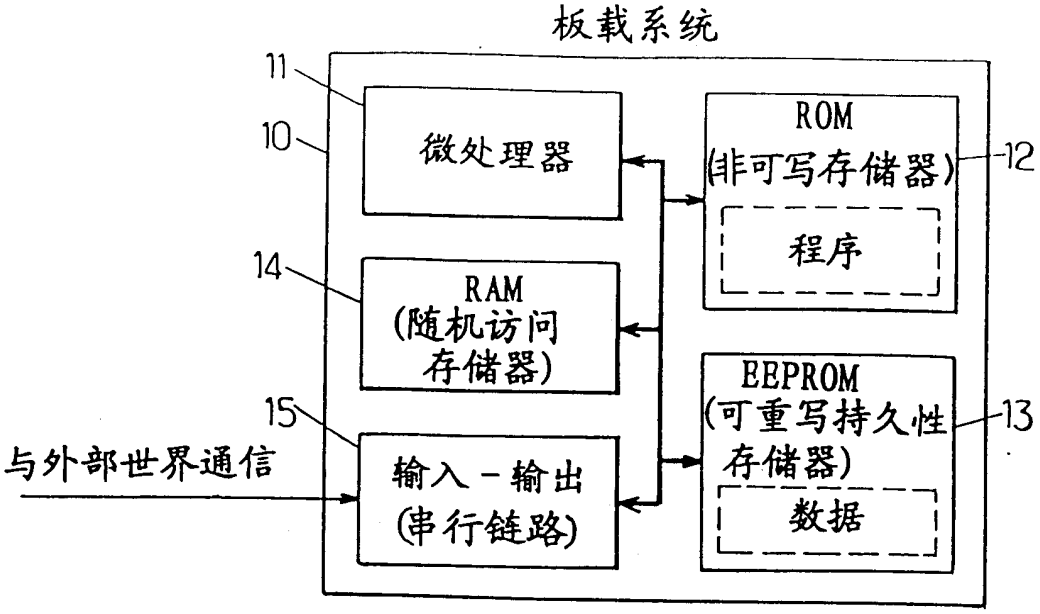


表 T12

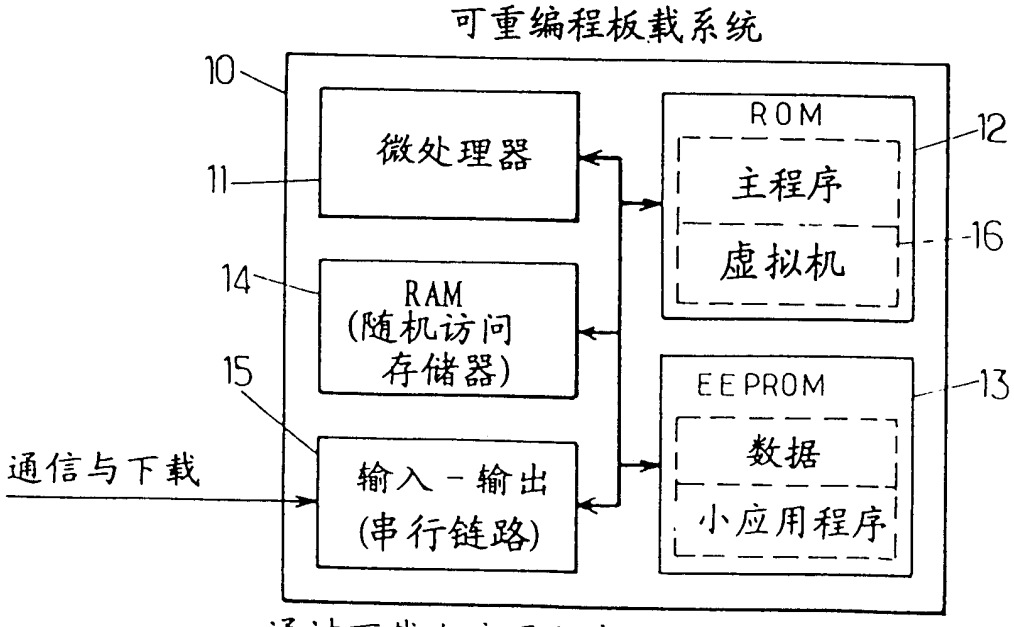
(c) 带一个变元的条件转移：





非可编程的板载系统

图 1a (先有技术)



通过下载小应用程序可重编程的板载系统

图 1b(先有技术)

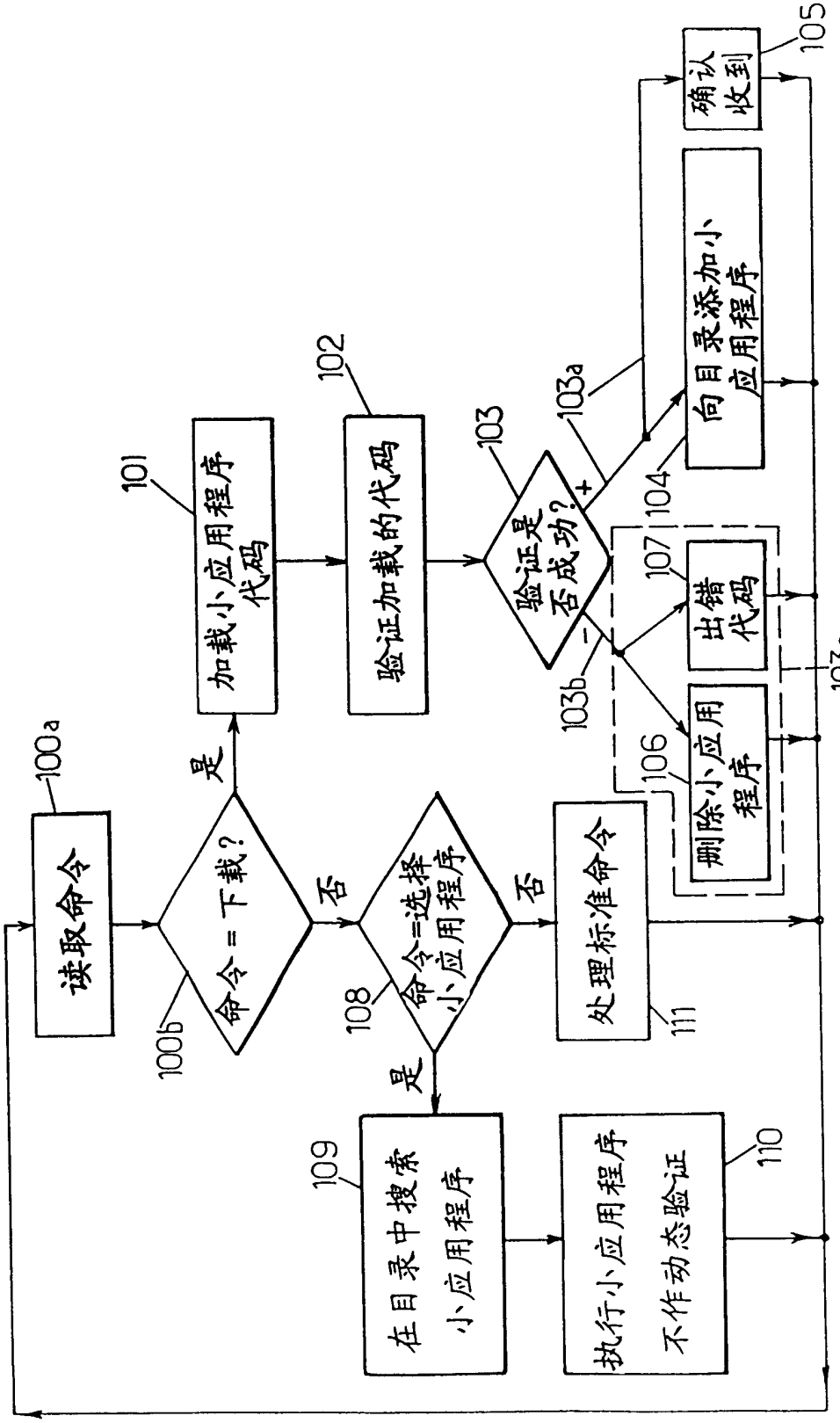

读取命令
100a
命令＝下载？
100b
是
否
加载小应用程序代码
101
验证加载的代码
102
验证是否成功？
103
+
-
103a
103b
向目录添加小应用程序
104
确认收到
105
出错代码
107
删除小应用程序
106
103c
命令=选择小应用程序
108
是
否
处理标准命令
111
在目录中搜索小应用程序
109
执行小应用程序不作动态验证
110

图 2 管理协议

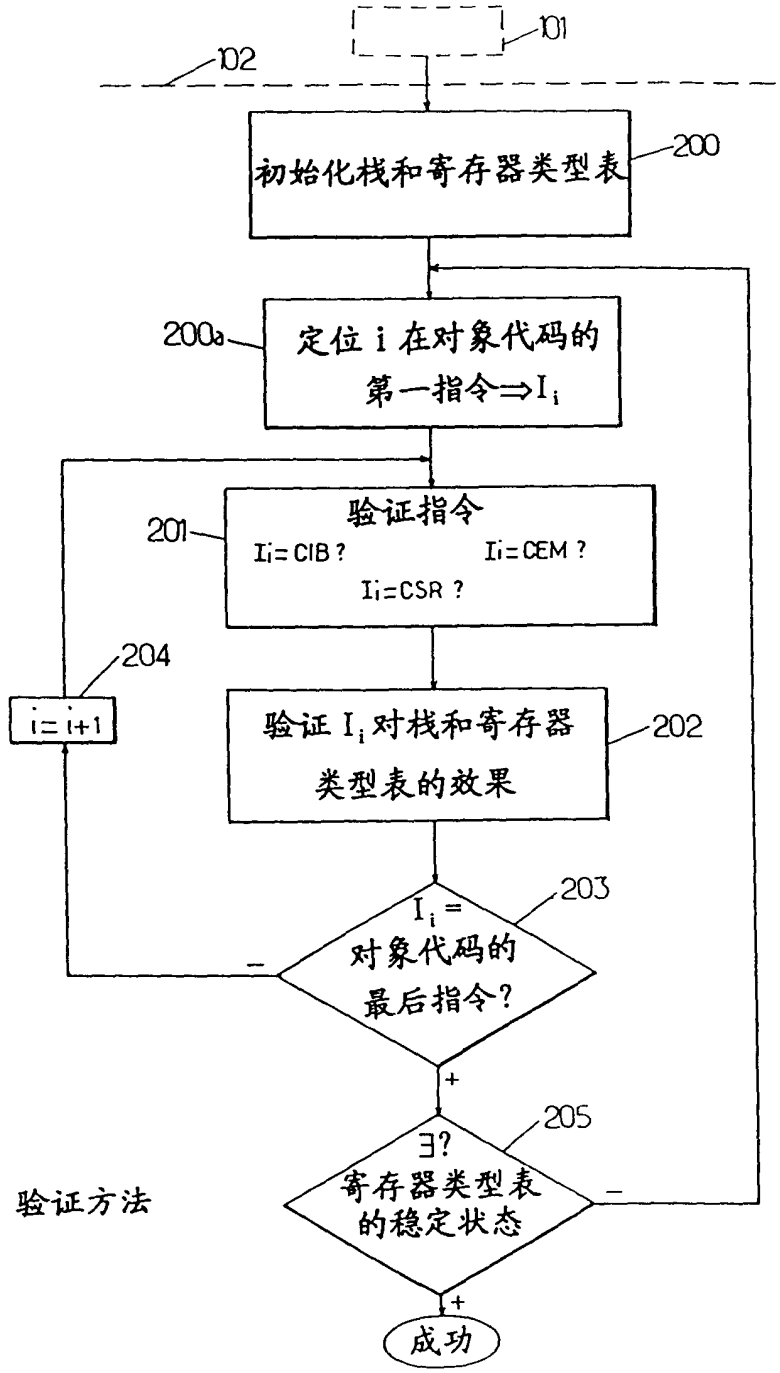

101
102
初始化栈和寄存器类型表
200
200a
定位 i 在对象代码的
第一指令⇒I_i
验证指令
201
I_i=CIB ?
I_i=CEM ?
I_i=CSR ?
204
i=i+1
验证 I_i 对栈和寄存器
类型表的效果
202
I_i =
对象代码的
最后指令?
203
−
+
∃?
寄存器类型表
的稳定状态
205
−
+
成功

图 3a 验证方法

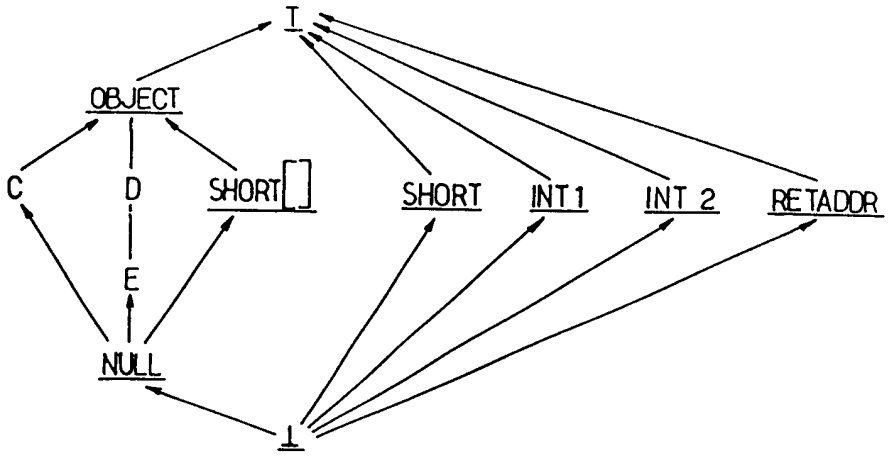


图 3b 数据类型和子类型关系

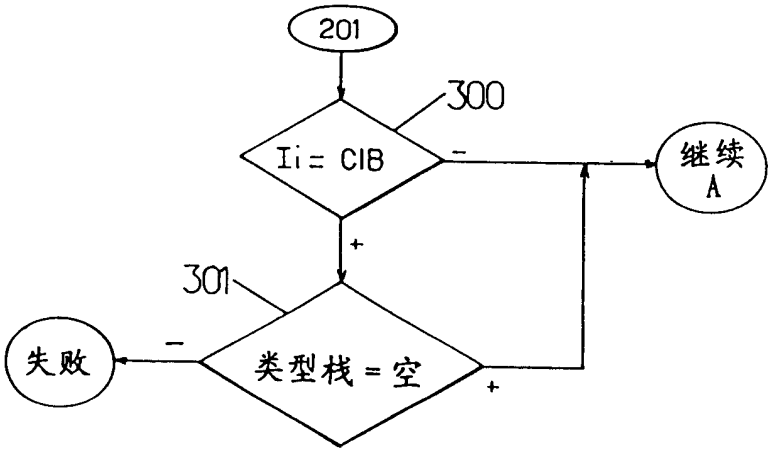


图 3c 验证：转移指令的管理

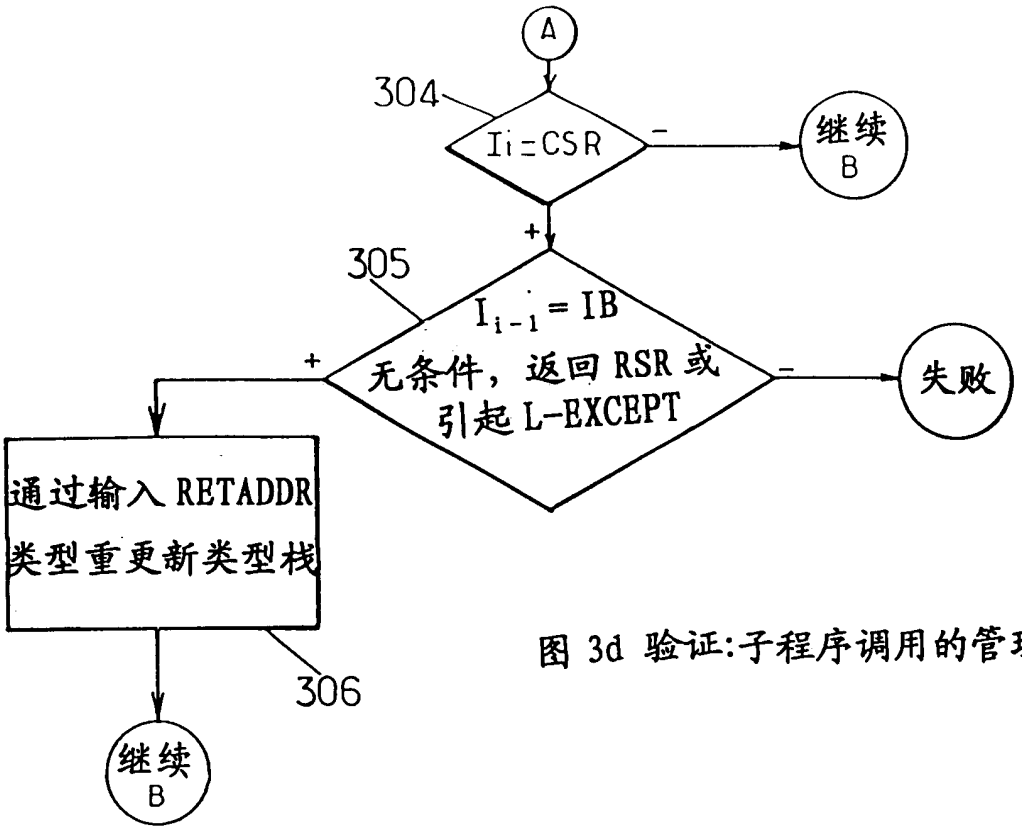


图 3d 验证:子程序调用的管理

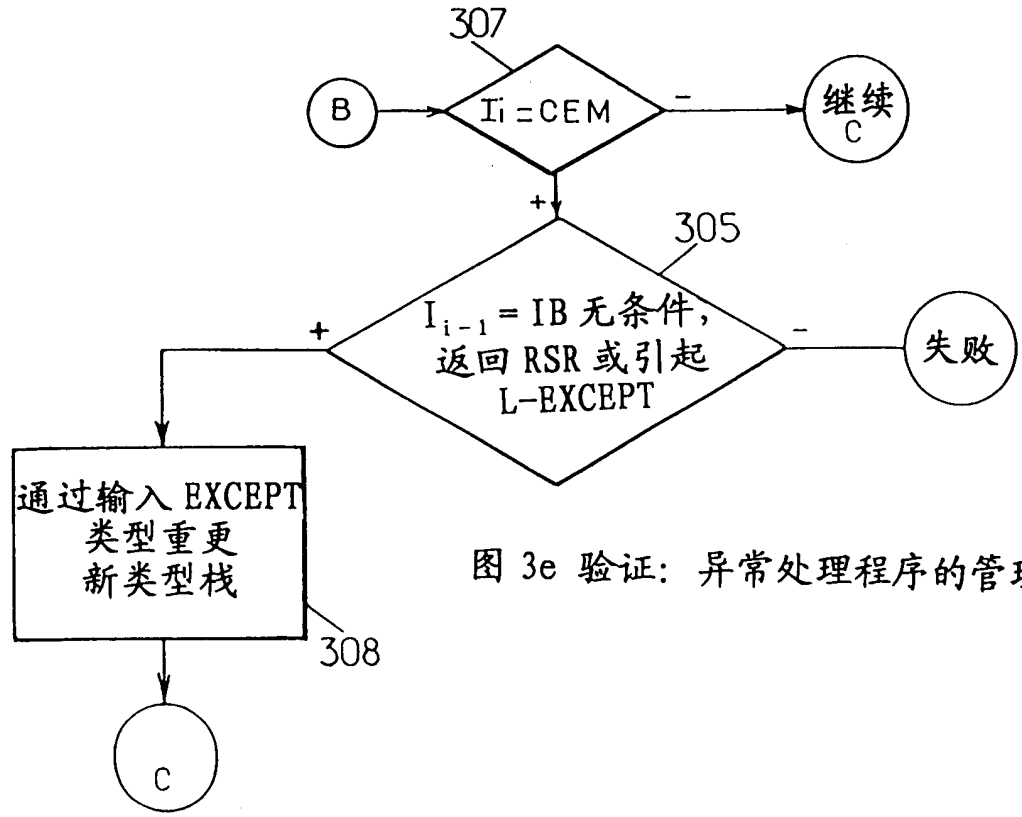


图 3e 验证: 异常处理程序的管理

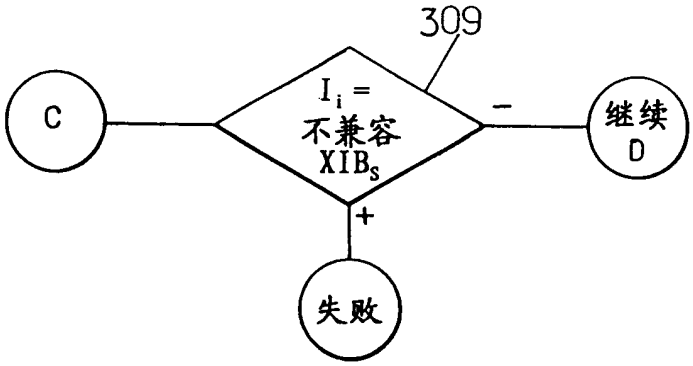


图 3f 验证：不兼容转移目标的管理

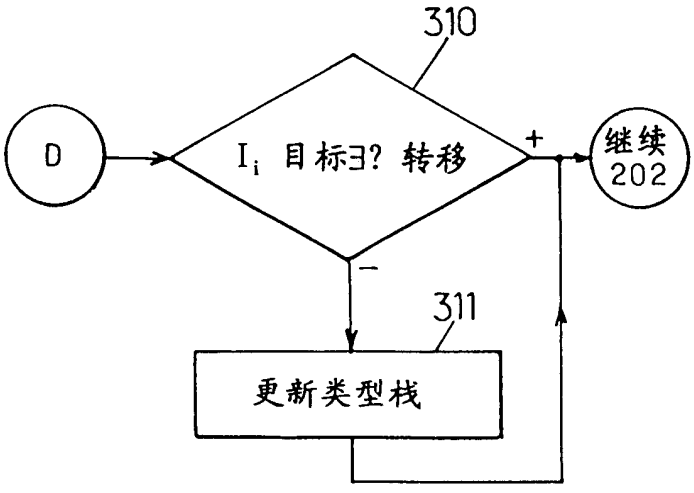


图 3g 验证：无转移目标的管理

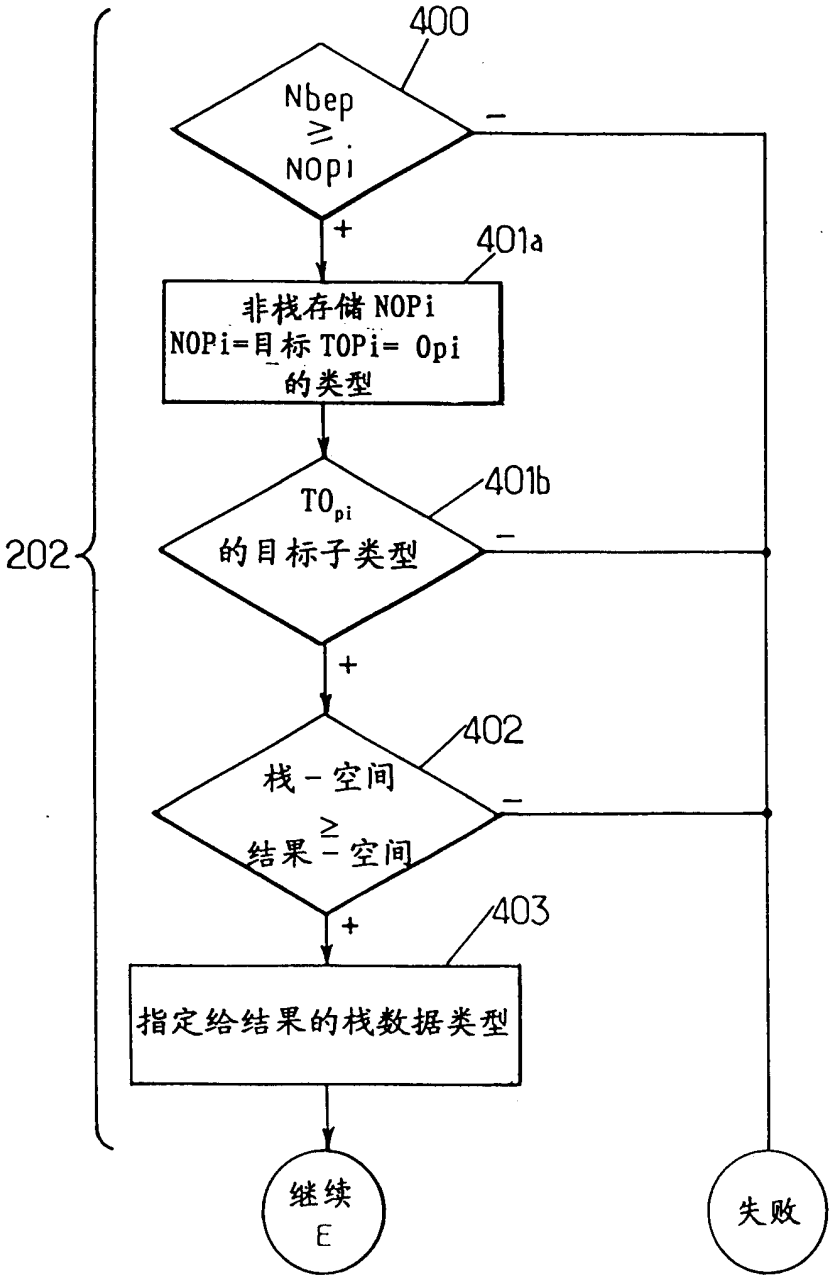


图 3h 验证：当前指令对类型栈的效果

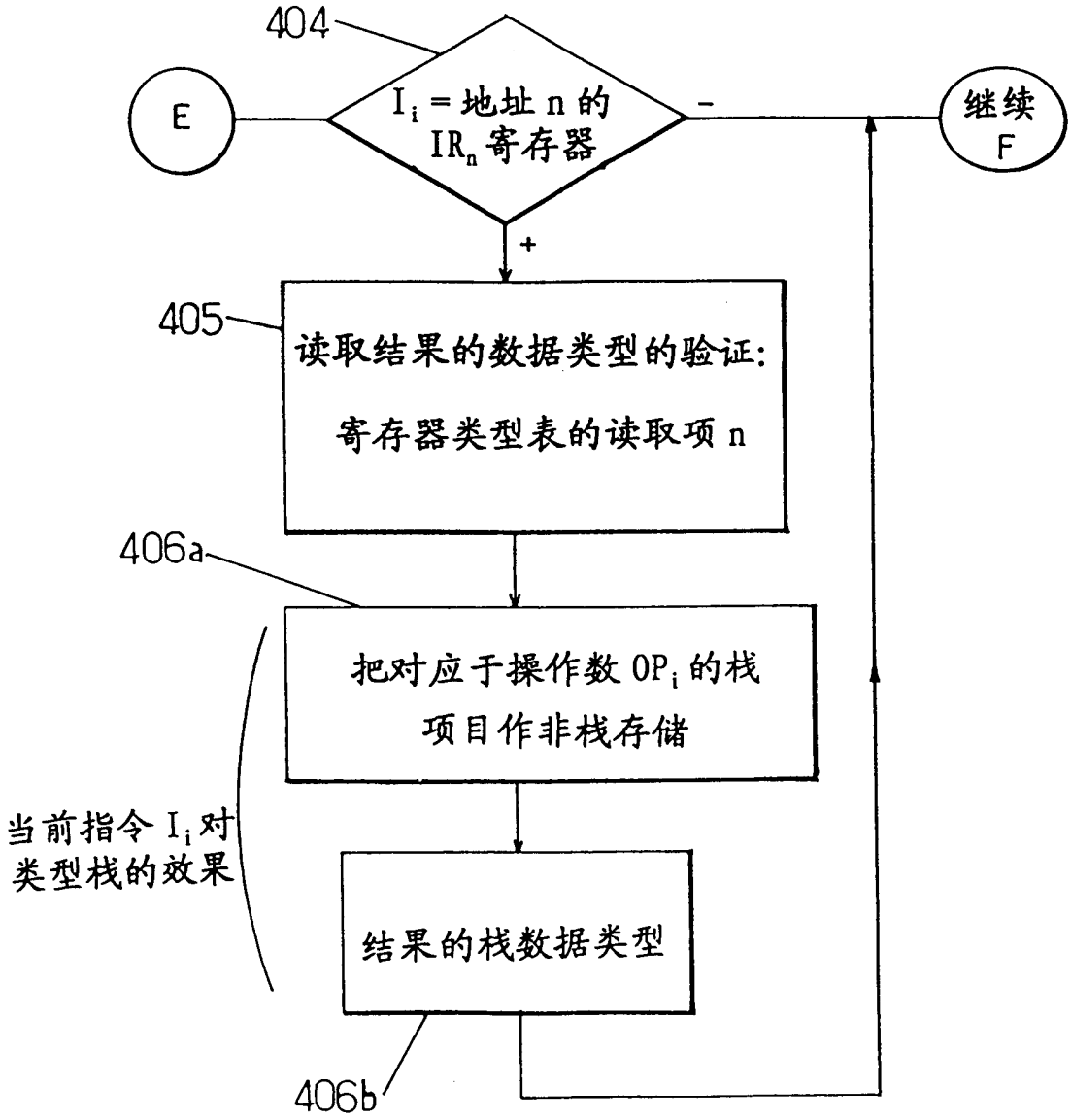


图 3i 验证: 读取寄存器的指令管理

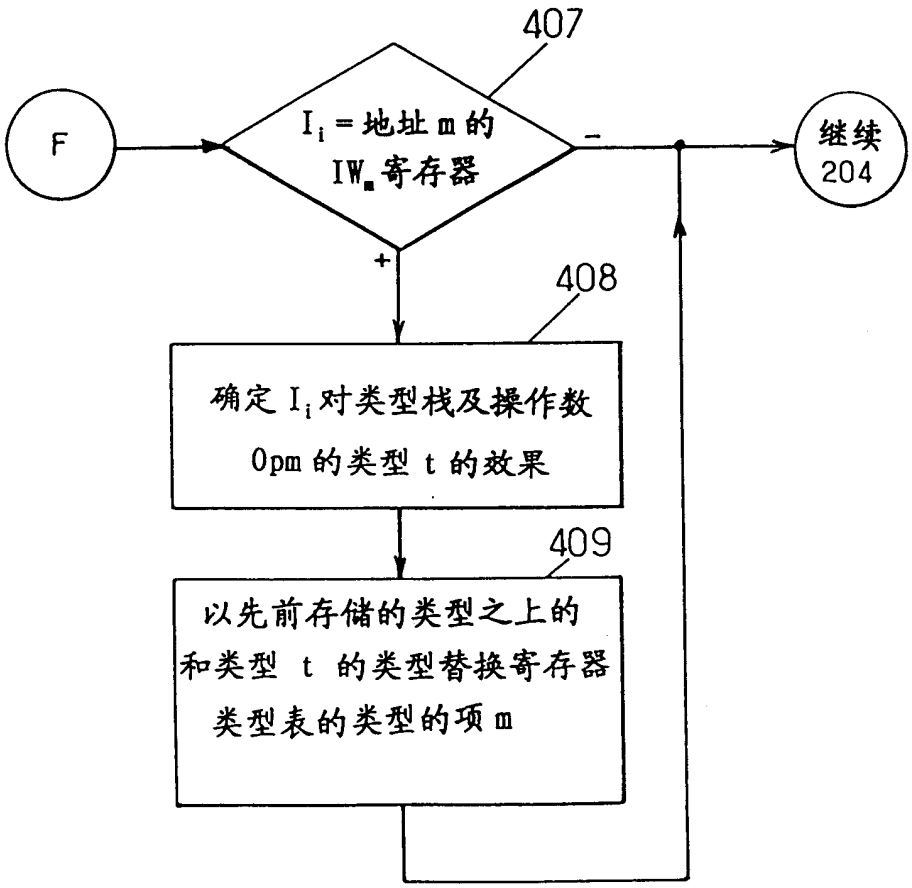


图 3j 验证: 写寄存器的指令管理

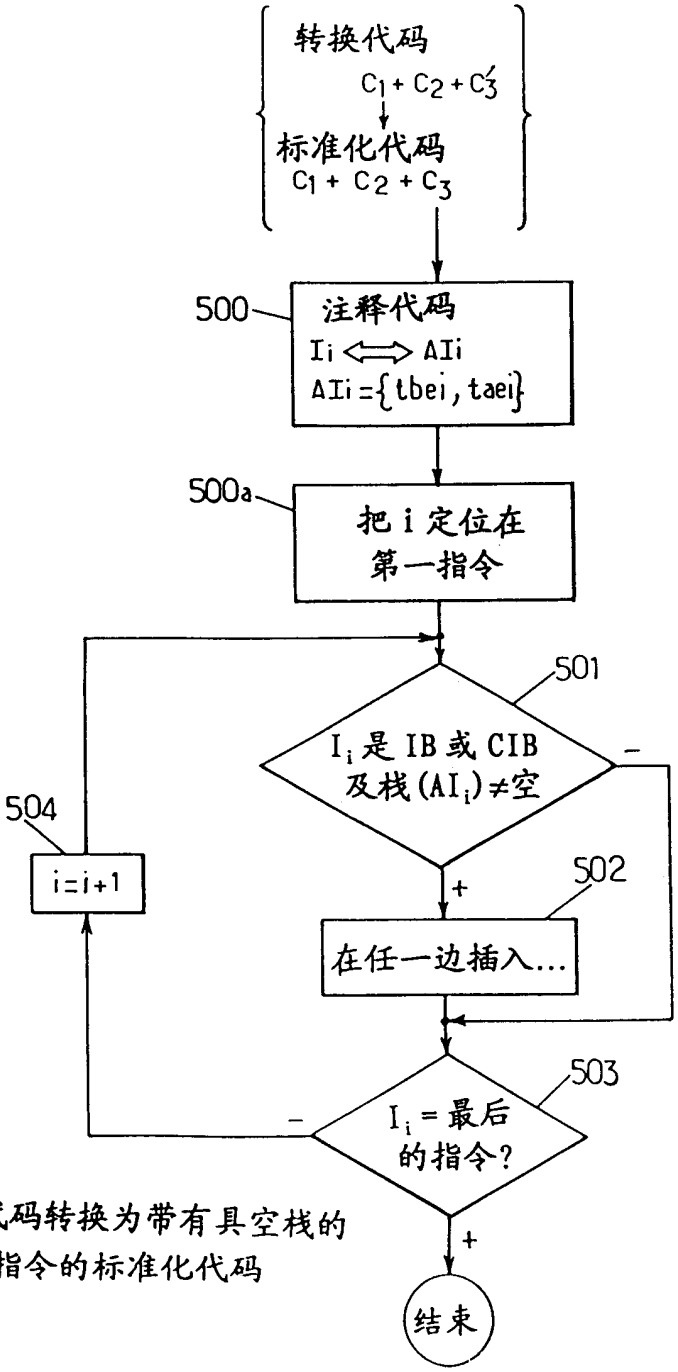


图 4a 把代码转换为带有具空栈的转移指令的标准化代码

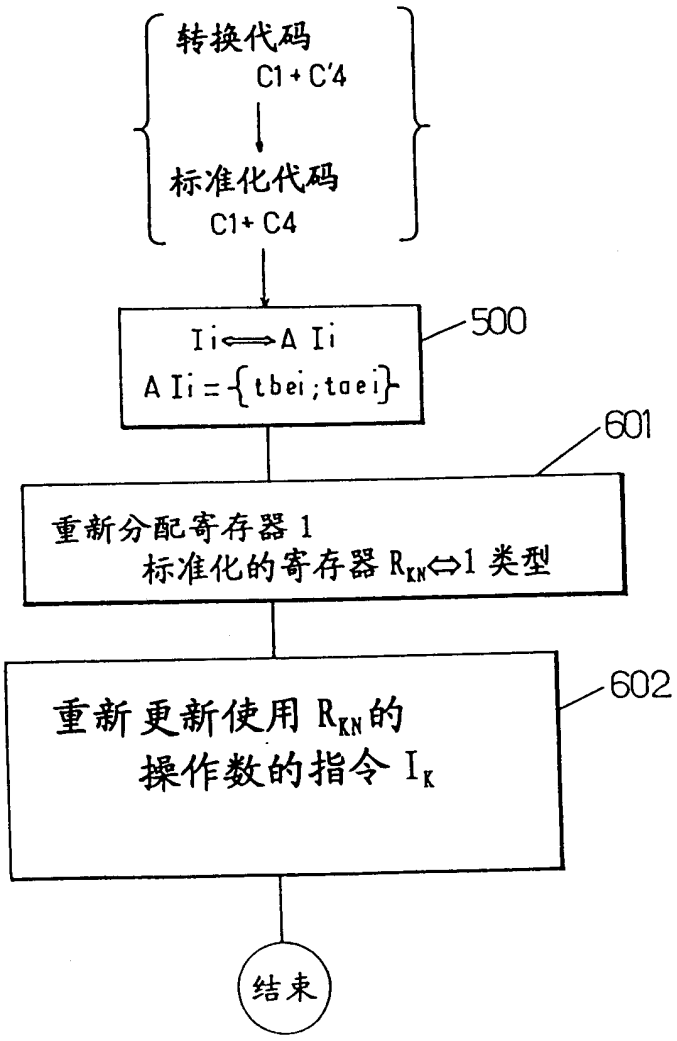


图 4b 使用类型寄存器把代码
转换为标准化代码的方法

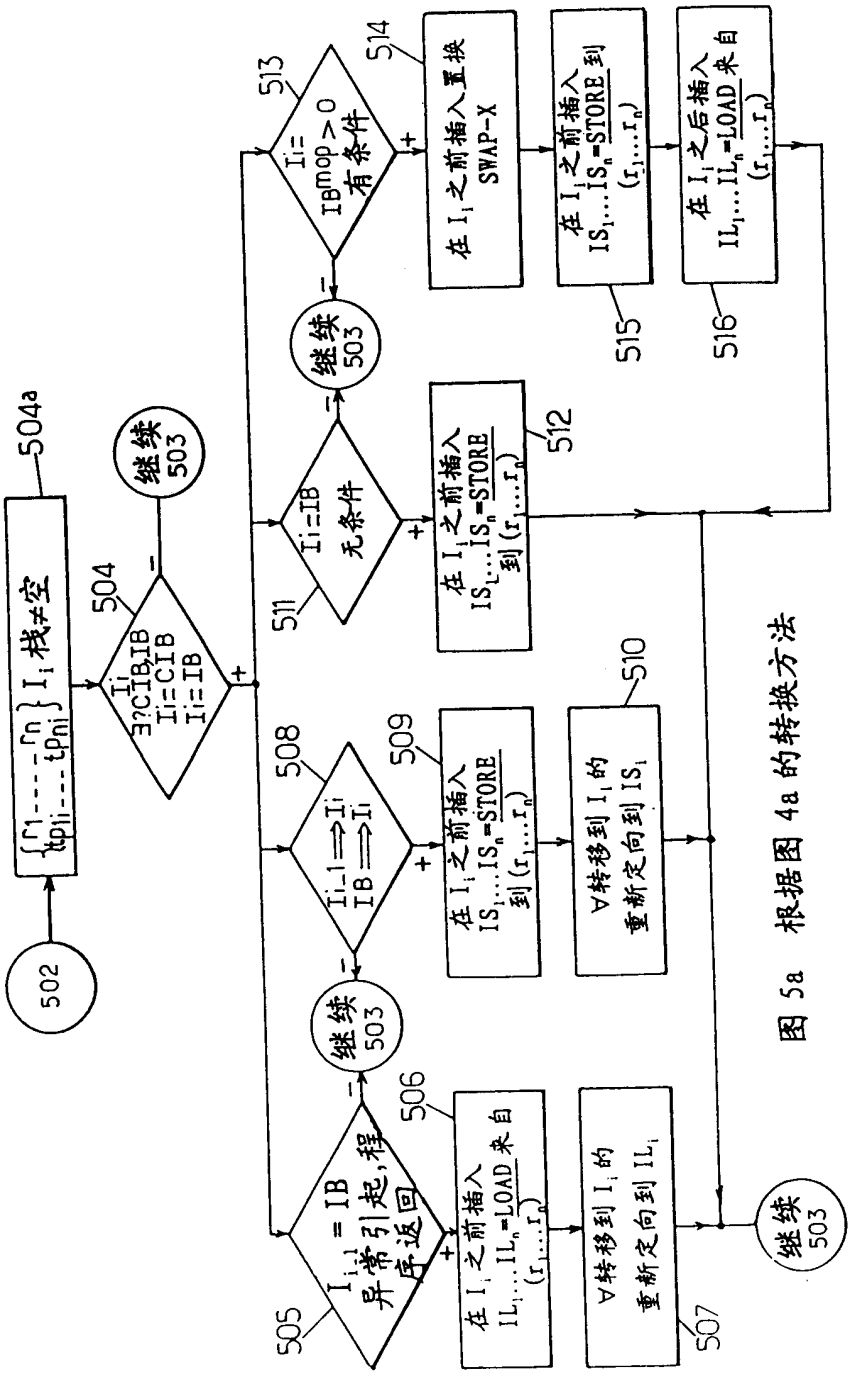


图 5a 根据图 4a 的转换方法

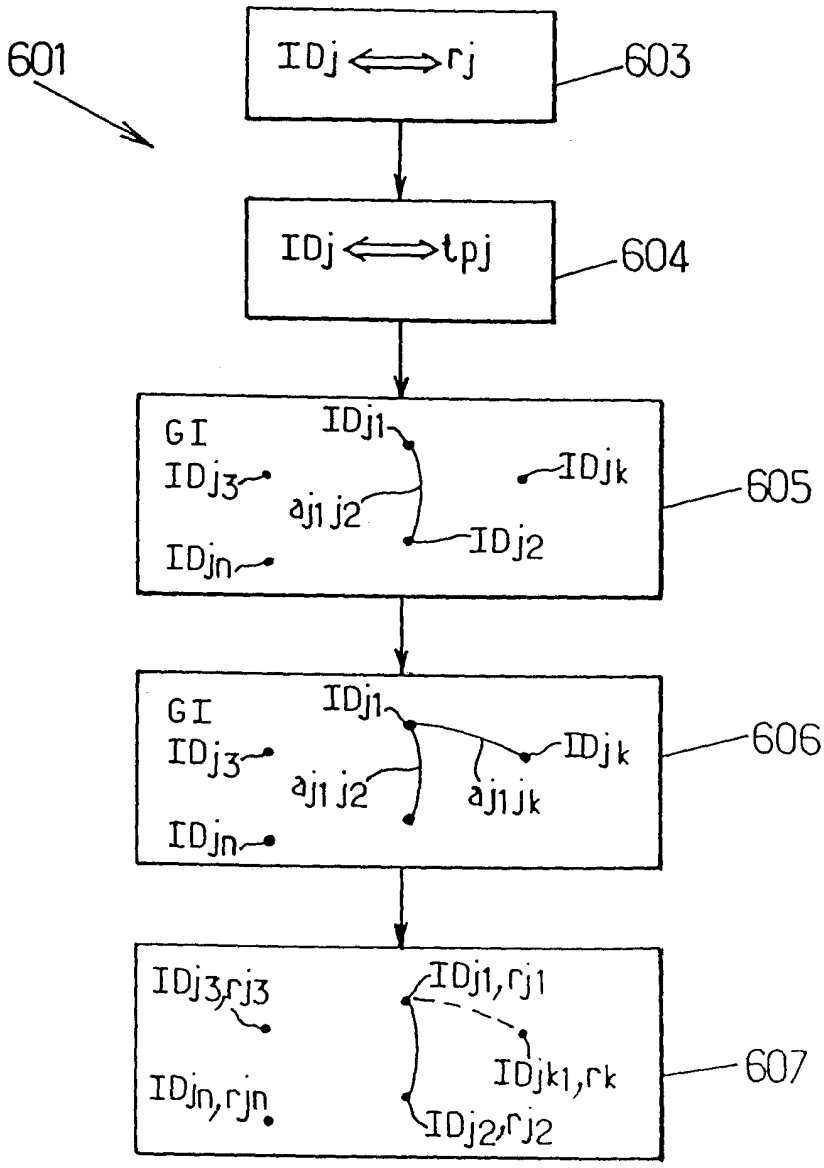


图 5b 根据图 4b 的转换方法

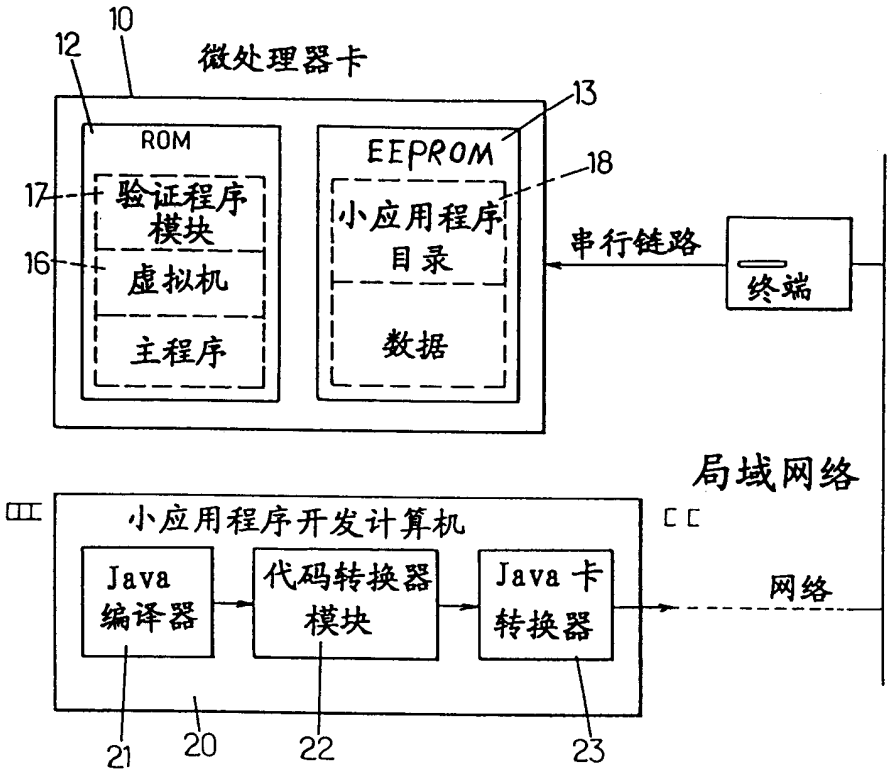

10
12
微处理器卡
13
ROM
EEPROM
18
17
验证程序模块
小应用程序目录
串行链路
16
虚拟机
终端
主程序
数据
局域网络
小应用程序开发计算机
Java编译器
代码转换器模块
Java卡转换器
网络
21
20
22
23

图 6