

[19] 中华人民共和国国家知识产权局



[12] 发明专利说明书

专利号 ZL 200480019877. X

[51] Int. Cl.

G06F 1/00 (2006.01)

G06F 9/42 (2006.01)

[45] 授权公告日 2008 年 1 月 9 日

[11] 授权公告号 CN 100361036C

[22] 申请日 2004. 7. 6

[21] 申请号 200480019877. X

[30] 优先权

[32] 2003. 7. 11 [33] FR [31] 03/08550

[86] 国际申请 PCT/FR2004/001755 2004. 7. 6

[87] 国际公布 WO2005/008451 法 2005. 1. 27

[85] 进入国家阶段日期 2006. 1. 11

[73] 专利权人 奥贝蒂尔卡系统股份有限公司

地址 法国巴黎

[72] 发明人 简 - 博纳德 · 费希尔

休格斯 · T. · D. · 拉克劳伊

[56] 参考文献

CN1212770A 1999. 3. 31

US5274817A 1993. 12. 28

CN1375084A 2002. 10. 16

审查员 于 平

[74] 专利代理机构 中国国际贸易促进委员会专利
商标事务所

代理人 郭思宇

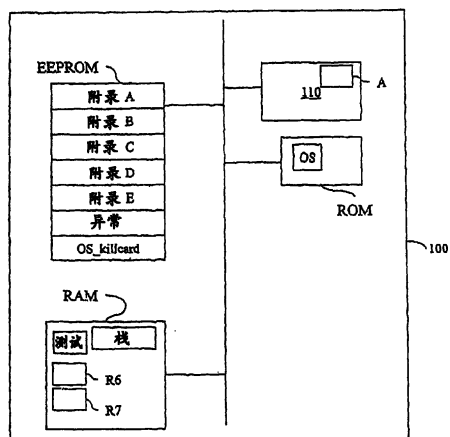
权利要求书 2 页 说明书 17 页 附图 1 页

[54] 发明名称

保护计算机程序尤其是在微电路卡中的计算机程序的执行安全的方法

[57] 摘要

本发明涉及保护计算机程序执行安全的方法，包括：一个向该程序的一个指令栈中推入一个预定值的步骤；和一个从该栈出栈的步骤，在必要的情况下，该出栈步骤适于容许检测一个执行异常。



1. 一种使计算机程序的执行安全的安全保障方法，其特征在于包括：

使预定值进入该程序的指令栈中的入栈步骤，该预定值是异常处理函数的地址；

在该程序正常执行时，从所述指令栈中删除所述预定值而不执行所述异常处理函数的步骤；以及

从所述栈出栈的步骤，其间，如果所述预定值出栈，则执行所述异常处理函数。

2. 根据权利要求1的安全保障方法，其特征在于：所述入栈步骤和出栈步骤分别与所述程序的至少一个指令子集的元素相关联。

3. 根据权利要求2的安全保障方法，其特征在于：所述元素分别是括号系统中的开括号和闭括号。

4. 根据权利要求2的安全保障方法，其特征在于：所述出栈步骤与所述程序或所述程序的子程序的执行返回指令相关联。

5. 根据权利要求1的安全保障方法，其特征在于：所述程序是用编程语言编写的，该语言包含第一指令和/或第二指令，该第一指令的执行实施所述入栈步骤，该第二指令的执行实施所述出栈步骤。

6. 根据权利要求5的安全保障方法，其特征在于：第二指令结束所述程序或所述程序的子程序。

7. 根据权利要求1的安全保障方法，其特征在于：所述预定值代表所述程序的关键指令子集。

8. 根据权利要求1的安全保障方法，其特征在于：它包括异常处理步骤，如果在所述出栈步骤过程中出栈的值不同于所述预定值就执行该异常处理步骤。

9. 根据权利要求1的安全保障方法，其中，所述程序包含至少一个对子程序的调用，其特征在于：所述入栈步骤在所述调用之前实施，并且，在所述子程序执行期间，从所述栈中删除所述预定值。

10. 根据权利要求 1 的安全保障方法，其中所述程序包含至少一个对子程序的调用，其特征在于：所述入栈步骤在执行所述子程序期间实施，并且，在执行所述子程序之后，从所述栈中删除所述预定值。

11. 根据权利要求 1 的安全保障方法，其特征在于：所述异常处理函数是操作系统破坏函数。

12. 一种受到安全保护的电子实体，其特征在于包括：

使预定值进入程序的指令栈中的入栈装置，该预定值是异常处理函数的地址；

在该程序正常执行时，从所述指令栈中删除所述预定值而不执行所述异常处理函数的删除装置；以及

从所述栈出栈的装置，其被配置为如果所述预定值出栈，则执行所述异常处理函数。

13. 根据权利要求 12 的电子实体，其特征在于它是微电路卡。

保护计算机程序尤其是在微电路卡中的计算机程序的执行安全的方法

技术领域

本发明涉及一种保护计算机程序执行安全的方法并涉及一种应用这样一种方法的被安全保护了的电子实体。

本发明尤其被用于保护微电路卡(也被称作“智能卡”)的安全。

在下文中,计算机程序的“安全保护”意指:

- 检测旨在修改计算机程序的正常行为的恶意攻击;以及
- 任何旨在使计算机程序,尤其是在强烈干扰的环境,例如卫星,中执行的程序,或者对可靠性要求极高的计算机程序,例如控制心脏植入物的程序,的运行可靠的处理;

另一方面,“计算机程序”意指任何程序,无论哪种计算机语言,无论用的哪种存储装置。非限制性地举例而言,计算机程序可以是用机器语言、汇编语言、C、C++、Java、VHDL 编写的。程序可以存储在永久性存储器中,例如 ROM 或 EEPROM 存储器中或硬盘中,或存储在易失性存储器中,例如 RAM 类型的存储器中。还可以用集成电路,例如 FPGA 类型的,或用 ASIC 电路(专用集成电路)将程序固化。

本发明容许检测旨在修改在被安全保护了的电子实体,例如微电路卡、被安全保护了的 PCMIA 卡(例如 IBM4758 卡)、USB 钥匙、或在页面上集成了无触点芯片的护照,上执行的计算机程序的执行进程的攻击。

本发明尤其容许检测通过干扰电子实体的运行的攻击,例如“通过制造错误的攻击”(英文的“fault Attack”,即“错误攻击”)类型的攻击。

这些攻击旨在非法修改寄存器、存储器、或总线的内容或对这些内容的读取,或旨在迫使处理器不执行或错误地执行计算机程序的某些指令。被攻击的计算机程序于是可能以与设计时所预定的方式非常

不同的方式运行。

这些攻击可能通过各种方式实施，其中已知的有：

- 通过在处理器的电源端子上产生一个电压峰值；
- 通过突然升高其温度；
- 通过迅速改变其时钟频率或其电源电压；
- 通过在构成它的硅部分上施加闪光、激光射线、或电磁场。

背景技术

根据现有技术，对于保护计算机程序的安全，尤其是对于抵御通过在微电路卡中制造错误的攻击，业内人士有各种各样的方法。

第一种方法在于：在微电路卡的元件中安装容许检测这种攻击的传感器。

然而，这种方法的有效性有限，因为实际上不可能在该元件的整个表面布置上传感器。此外，这些传感器也是由硅构成的，所以也有可能干扰这些传感器或修改它们传输的信息。

第二种已知的并在大多数微电路卡操作系统中应用的安全保护方法基于“信号量（semaphore）”的使用。一种这样的方法包括：

- 一个在执行关键指令集期间修改存储器的内容的步骤；和
- 一个检验步骤，在这个检验步骤期间，通过读前述存储区的内容，从而检验前述的修改是否实施了。

如果存储区没有被修改，这说明修改步骤没有实施，因此，前述关键指令集没有正确执行。

我们将注意到：术语“信号量”在本文中表示一个与编程领域内并发进程已知的概念不同却具有相同名称的概念。

该第二种方法由软件实施，没有前述第一种方法的弊病。

然而，已经知道，传统上这些信号量是由驻留在工作存储器(RAM)中的变量来实现的，它们的操作（定位、读取）比较慢或者耗费存储空间。当程序在象智能卡这样的资源(内存、计算能力，.....)有限的系统上执行时，该限制尤其不利。本发明力求得到一种没有前述弊病的软件安全保护方法。

发明内容

为此，本发明涉及一种使计算机程序的执行安全的安全保障方法，该方法包括：使预定值进入该程序的指令栈中的入栈步骤，该预定值是异常处理函数的地址；在该程序正常执行时，从所述指令栈中删除所述预定值而不执行所述异常处理函数的步骤；以及从所述栈出栈的步骤，其间，如果所述预定值出栈，则执行所述异常处理函数。

本发明还涉及一种受到安全保护的电子实体，其特征在于包括：使预定值进入程序的指令栈中的入栈装置，该预定值是异常处理函数的地址；在该程序正常执行时，从所述指令栈中删除所述预定值而不执行所述异常处理函数的删除装置；以及从所述栈出栈的装置，其被配置为如果所述预定值出栈，则执行所述异常处理函数。

我们将在此提醒：指令栈是一个用于临时保存数据的存储区。用两个在下文中分别称作 PUSH 和 POP 的专用指令将数值推入栈中及从栈中弹出。

这些指令只操作固定长度的数值，例如八位字节的。

栈的使用遵循 LIFO(“后进先出”)类型的算法。

已经知道，它尤其存储进程的返回地址(例如 80x86 汇编语言的 RET 指令)。

根据本发明的安全保护方法因而使用执行栈来存储一个容许检测一个执行异常的数值。

执行栈一方面具有快速的读写存取速度，另一方面很少耗费存储空间，因此根据本发明的安全保护方法尤其适合于保护在资源有限的系统上执行的计算机程序的安全。

这种对指令栈的新颖使用具有其它的优点，将在以后描述。

在一个最佳实施方式下，入栈步骤和出栈步骤分别与所述程序的至少一个指令子集的元素相关联。

例如，入栈步骤可以与打开文件的指令“open(文件)”相关联，而出栈步骤可以与关闭该文件的指令“close(文件)”相关联。

该特征尤其有利，因为它容许通过例如用编辑器将入栈和出栈操作与前述元素，即前面例子中的“open”和“close”指令，关联起来，而

使安全保护指令的编写自动化。

根据该最佳实施方式的一个第一变型，指令子集的元素分别是一个括号系统的一个开括号和一个关括号。

为此，我们提醒，根据语言理论以及如计算机语言的业内人士所知道的，当一段文本包含同样多的开括号和关括号，而且该文本的任何开头都包含数量多于或等于关括号的数量的开括号时，我们面对的就是一个括号系统。

根据这个尤其有利的特征，入栈步骤和出栈步骤可以分别与如下指令相关联：

- “(”和“)”；或
- “{”和“}”；或
- “begin”和“end”；或
- “repeat”和“until”。

在该最佳实施方式的另一种变型下，出栈步骤与程序或该程序的一个子程序的一条执行返回指令相关联。

该特征有利地容许使用传统上在一个程序或子程序的(当执行 **return** 指令时)返回时实施的正常出栈操作来检测一个执行异常，如果在这种场合下出栈的数值与在该程序正常执行时本应该出栈的数值不一致的话。

根据另一个特征，程序由这样一种编程语言编写：该语言包含一个第一指令，该第一指令的执行实施入栈步骤；和/或一个第二指令，该第二指令的执行实施所述出栈步骤。

在该实施方式下，一些新的指令被合并到编程语言中，这些指令每条都有一个专有函数，以及为了程序的安全保护的要么一个入栈函数，要么一个出栈函数。

再回到上文中简短引入的例子，可以创造一条取名为“**open**(文件)”的新指令，该新指令容许既打开文件又将一个预定的数值推入程序的指令栈。

因此，编程人员确信：每次打开文件时，安全保护函数都被应用

了，甚至无需他操心，也不需要特别的软件工具。

优先地，第二指令结束程序或该程序的一个子程序。

该实施方式具有与在前文中引入的实施方式方式相同的优点，在前文的这个实施方式中，入栈和出栈指令与程序的指令子集的元素相关联，而不是与这些元素合并。因此，此处的实施方式将不在后面详细描述。

在一个最佳实施方式下，预定值代表程序的一个关键指令子集。

当安全保护方法被用于保护程序的多个指令子集的安全时，该特征尤其有利。

该特征容许在入栈步骤期间检测是否一个特定的指令子集被正确执行了，而不是其执行会导致另一个预定值入栈的另一个指令子集被执行。

业内人士将容易地领会到：该特征可以用于保护一个(C 语言中的“if”，“then”，“else”类型的)测试的各个分支的安全，在每一个分支中，一个预定的不同的数值被入栈，而在该测试末实施出栈步骤。

当程序调用一个子程序时，该特征还容许在执行子程序期间确信进入的子程序是响应这个调用的，而不是响应一个通过制造错误的攻击的。

应用该特征的两个例子将在以后参照附录 A 和 C 被详述。

根据另一个特征，根据本发明的安全保护方法包括一个如果在出栈步骤中出栈的数值不同于预定的数值就执行的异常处理步骤。

有利地，该特征容许一旦一个攻击有修改程序的正常执行，尤其是该程序的一个函数的调用和返回，的后果就应用一个异常处理步骤。该安全保护方法因而尤其有效。

例如，异常处理可以在一个微电路卡使用安全保护方法的情况下旨在通过破坏该卡的操作系统而使该卡无效。

应用该特征的三个例子将在后面参照附录 A、C、D 被详述。

在程序包括至少一个对一个子程序的调用的一个特殊实施方式下，入栈步骤在该调用之前实施，并且，在该子程序执行期间，预定

值被从栈中删除。

该特征因此容许检验子程序是否确实执行了以及是否正确地执行了。

事实上，如果对该子程序的调用被跳过了，或者如果出栈步骤没有执行，指令栈中将保存着被入栈的预定值。

以后该数值的出栈将导致检测出执行异常，正如下文中参照附录 B 和 C 所阐明的那样。

有利的是，在该特殊实施方式下，预定值可以是一个异常的处理函数的地址。

因此，如果在执行子程序期间预定的数值没有出栈，例如由于一个旨在使该子程序不执行的攻击，那么以后由处理器将该数值弹出栈就将导致应用该处理函数。以后将在附录 B 中详述一个例子。

该特征容许如果程序遭受任何一个其后果为避免执行子程序的攻击就触发处理函数。该特征因而对保护关键函数，例如一个验证过程，的安全尤其有用。

在程序包括至少一个对一个子程序的调用的另一个特殊实施方式下，入栈步骤在执行子程序期间实施，而预定值在执行该子程序之后被删除。

该特征因此容许检测该子程序的返回是否正确执行。

事实上，如果该子程序的返回被干扰了，指令栈将保存着被入栈的预定值。

该特殊实施方式将被参照附录 D 详述。

有利的是，在该特殊实施方式下，预定值可以是一个异常的处理函数的地址。

因为前文中提及的理由，该特征容许如果程序遭受了任何一个旨在避免执行子程序的攻击就触发处理函数。该特征因而对保护关键函数，例如一个验证过程，的安全尤其有用。

被参照附录 E，一个应用该特征的例子被给出。

本发明还力求得到一种计算机系统可读的、可能完全或部分可拆

卸的信息载体,尤其是 CD-ROM 或磁载体,例如硬盘或软盘,或者如电信号或光信号这样的可以传输的载体,这种信息载体包括一个计算机程序的一些指令,容许在计算机系统加载和执行该程序时应用如上文中简述的安全保护方法。

本发明还力求得到一种存储在一个信息载体上的计算机程序,该程序包括一些指令,容许在计算机系统加载和执行该程序时应用如上文中简述的安全保护方法。

本发明还力求得到一种包括应用如上文中简述的安全保护方法的装置的、被安全保护了的电子实体及一个微电路卡。

信息载体、计算机程序、和微电路卡专有的优点和特征与上文中关于根据本发明的安全保护方法所叙述的那些相同,在此将不提醒了。

附图说明

一阅读随后对特殊实施方式的描述,本发明的其它方面和优点将更清晰地显现出来,该描述仅仅是作为非限制性的例子给出,并参照附录 A 至 E,这些附录包括五个被按照本发明安全保护了的计算机程序的例子。

这些程序用 C 语言和 80c51 汇编语言编写。为了便于描述,每一行前面都有一个被包含在字符串“/*”和“*/”之间的注释。

对一个最佳实施方式下的、根据本发明的微电路卡的描述将参照图 1 实施。

具体实施方式

附录 A 包含一个计算机程序的、被编为/*a1*/至/*a33*/的 33 行指令,该计算机程序的执行被一个根据在最佳实施方式下的本发明的安全保护方法保护。

确切地说,行/*a1*/并不是一条指令。它象征地表示这样一个事实:附录 A 除那些用于保护该程序的安全的指令以外还可能包含一定数量的指令代替字符串“...”。它表示与本发明无关的一组指令。

行/*a2*/包括一条指引语句：`#pragma asm`，向编译程序表明随后的指令行是用 80c51 汇编语言编写的。

行/*a3*/包括这样一条指令：其执行将应用把预定值 0(用十六进制符号表示)推入附录 A 的程序的指令栈中的步骤。为简单起见，在下文中我们将说在行/*a3*/把数值 0 入栈。

然后，在行/*a4*/把数值 1 入栈。

在此处描述的最佳实施方式下，预定值 00h 和 01h 分别代表用两个八位字节编码的数值 1(用十六进制符号表示)的高八位字节和低八位字节。

行/*a5*/包括一条指引语句 `#pragma endasm`，向编译程序表明随后的指令不再是用 80c51 汇编语言编写的了，而是用 C 语言编写的。

与行/*a1*/相似的行/*a6*/和/*a7*/表示一组与本发明无关的指令。

行/*a8*/包括这样一条指令：在这条指令中测试变量“test”的内容是否等于“VRAI（真）”。已经知道，如果在执行附录 A 程序时这样的测试成立，那么处理器将执行指令/*a9*/至/*a23*/，作为对行/*a8*/的测试的响应。

否则，它将直接执行行/*a24*/的指令。

行/*a9*/与前述的行/*a2*/相同。

行/*a10*/和/*a11*/与曾经描述过的行/*a3*/和/*a4*/相似。它们容许把用两个八位字节编码的数值 1（用十六进制符号表示）分两次入栈。

行/*a12*/与前述的行/*a5*/相同。

与前述的行/*a1*/相似的行/*a13*/和/*a14*/表示一组与本发明无关的指令。只要在行/*a14*/结束时使指令栈处于指令/*a13*/之前的状态，这些指令当然可以操作指令栈。

行/*a15*/与前述的行/*a2*/相同。

行/*a16*/包含这样一条指令：其执行实施一个指令栈出栈的步骤，出栈的数值被存储在寄存器 A 中。为简化起见，下文中我们将说

在行/*a16*/出栈入寄存器 A。

因而，在指令/*a16*/的结束时，寄存器 A 存储着最后入栈的数值，这是由于栈按照 LIFO 的机制运行。

行/*a17*/包含一条容许将寄存器 A 的内容与数值 02H 比较的指令。正常情况下，如果程序在其自从行/*a11*/的指令末的执行期间没有遭受到攻击的话，寄存器 A 的内容就包含着在行/*a11*/的指令期间入栈的数值 02H。

因此，行/*a16*/容许按照本发明检测一个执行异常。

如果在行/*a17*/的比较步骤中发现寄存器 A 的数值不等于数值 02H，附录 A 的程序就在行/*a18*/的指令期间分支到“异常”的地址。

在此处描述的实施方式下，该“异常”的地址是根据本发明的安全保护方法的异常处理步骤的地址。在实践中，“异常”的地址是一个直接可以由处理器解释的、用十六进制符号表示的地址。

反之，如果在行/*a17*/的比较步骤中发现寄存器 A 存储着数值 02H，附件 A 的程序就执行行/*a19*/的指令。

行/*a19*/至/*a21*/与前述的行/*a16*/至行/*a18*/相似：

- 在行/*a18*/处出栈入寄存器 A；
- 在行/*a20*/处比较寄存器 A 与数值 00H，数值 00H 对应于在行/*a10*/处入栈的预定值；和
- 如果寄存器 A 在执行行/*a20*/的指令时不包含数值 00H，那么在行/*a21*/的指令期间分支转移至“异常”的地址。

反之，如果寄存器 A 包含数值 00H，程序执行与前述的行/*a5*/相同的行/*a22*/的指令。

与前述的行/*a1*/相似的行/*a24*/和/*a25*/表示一组与本发明无关的指令。

行/*a26*/至/*a33*/与前述的行/*a15*/至行/*a22*/相似：

它们包括出栈步骤/*a28*/和/*a30*/，这些步骤容许如果栈被破坏了而且就在执行行/*a27*/之前该栈不包含分别在行/*a4*/和/*a3*/入栈的预定值 01H 和 00H，就检测出程序的一个执行异常。

综上所述，分别由行/*a6*/至行/*a25*/和行/*a13*/至行/*a14*/构成的两个指令子集被安全保护了。

由行/*a6*/和行/*a25*/构成的指令子集由于如下步骤而被安全保护：

- 用两个八位字节编码的预定值 1 入栈(行/*a3*/和/*a4*/)的步骤；和
- 行/*a27*/和行/*a30*/的出栈步骤。

同样，由行/*a13*/和行/*a14*/构成的指令子集由于如下步骤而被安全保护：

- 用两个八位字节编码的预定值 2 入栈(行/*a10*/和/*a11*/)的步骤；和
- 行/*a16*/和行/*a19*/的出栈步骤。

这种实施绝不是限制性的，预定值 1 和 2 也可以是相同的或随机地选择。

附录 B 包含一个计算机程序的、被编为/*b1*/至/*b28*/的 28 行指令，该计算机程序的执行被一个根据在最佳实施方式下的本发明的安全保护方法保护。

行/*b1*/和/*b2*/构成由 C 语言编写的函数“function”的声明的前两行，该函数既不包含输入参数也不包含返回值。行/*b11*/包含该函数的声明的最后一行指令。

与前文中参照附录 A 描述的行/*a1*/相似的行/*b3*/表示与本发明无关的一组指令。

行/*b4*/与前文中参照附录 A 描述的行/*a2*/相同。

在行/*b5*/和/*b6*/的指令中，分两次实施将一个用两个八位字节编码的预定值入栈的步骤，在该最佳实施方式下，该预定值是一个异常处理函数 OS_killcard 的地址。在实践中，“OS_killcard”的地址是一个可以由处理器直接解释的、用十六进制符号表示的地址。

例如，在微电路卡中使用安全保护方法的情况下，函数 OS_killcard 可以通过破坏卡的操作系统来禁止卡的运行。

行/*b7*/与前文中参照附录 A 描述的行/*a5*/相同。

与前文中参照附录 A 描述的行/*a1*/相似的行/*b8*/表示与本发明无关的一组指令。

行/*b9*/包含一条调用一个关键函数“fonction_critique”的指令，该指令的代码将被参照行/*b12*/至/*b28*/描述。

已经知道，调用一个子程序自动导致该子程序的返回地址入指令栈。该返回地址由两个八位字节编码，因而占用栈的两个寄存器。在此处描述的例子中，该地址与行/*b10*/的指令的地址相对应，该行应该在函数“fonction_critique”返回时被执行。

一方面，行/*b12*/和/*b13*/，及另一方面，行/*b28*/，构成函数“fonction_critique”的声明的前两行和最后一行，该函数既不包含输入参数也不包含返回值。

执行行/*b12*/和/*b13*/的指令后，入了指令栈的最后四个数值按照时间顺序是：

- 函数 OS_killcard 的地址的高八位字节(行/*b5*/);
- 函数 OS_killcard 的地址的低八位字节(行/*b6*/);
- 行/*b10*/的第一条指令的地址的高八位字节；和
- 行/*b10*/的第一条指令的地址的低八位字节。

与前文中参照附录 A 描述的行/*a1*/相似的行/*b14*/表示与本发明无关的一组指令。

与前文参照附录 A 的行/*a13*/和/*a14*/所描述的相同，我们假定这些指令使指令栈处于它在指令/*b14*/之前所处的状态。

行/*b15*/与前文中参照附录 A 描述的行/*a2*/相同。

在行/*b16*/处，出指令栈入寄存器 A，该寄存器 A 的内容接着在步骤/*b17*/被保存到寄存器 R7 中。

同样，在行/*b18*/处，再一次出指令栈入寄存器 A，寄存器 A 的内容在步骤/*b19*/被保存到寄存器 R6 中。

因而，根据前面的描述，在附录 B 的程序正常执行的情况下，在行/*b19*/的指令执行结束时，寄存器 R6 和 R7 分别包含：

- 行/*b10*/的第一条指令的地址的高八位字节；和
- 行/*b10*/的第一条指令的地址的低八位字节。

然后，在行/*b20*/和/*b21*/，两次出指令栈入寄存器 A，这在附录 B 的程序正常执行的情况下就相当于在执行子程序“fonction_critique”期间从指令栈删除函数 OS_killcard 的用两个八位字节编码的地址。

在行/*b22*/处，往寄存器 A 中存入寄存器 R6 的内容，即行/*b10*/的第一指令的高八位字节，这个值在行/*b23*/的步骤中入指令栈。

同样地，在行/*b24*/和/*b25*/处将行/*b10*/的第一指令的低八位字节入栈，该八位字节被存储在寄存器 R7 中。

行/*b26*/与前文中参照附录 A 描述的行/*a5*/相同。

与前文中参照附录 A 描述的行/*a1*/相似的行/*b27*/表示与本发明无关的一组指令。

行/*b28*/是子程序“fonction_critique”的最后一行。已经知道，它被一条“RETURN”或“RET”类型的指令翻译成汇编语言，该指令的执行导致程序跳转到指令栈的前两个寄存器中存储的地址。

因而，如果程序在执行期间没有遭受到攻击，它就转移到行/*b10*/的第一指令，该指令的地址在行/*b23*/和/*b25*/入栈了。

与前文中参照附录 A 描述的行/*a1*/相似的行/*b10*/表示与本发明无关的一组指令。

行/*b11*/结束函数“function”。

综上所述，在附录 B 的特殊实施方式下，将函数 OS_killcard 的地址入栈的步骤在调用子程序“fonction_critique”之前实施，而在行/*b20*/和/*b21*/处，在执行该子程序期间，该地址被从栈中删除。

因此，该实施方式容许控制子程序“fonction_critique”是否确实被执行了。

例如，如果对该子程序的调用被干扰过，或者，更通常地，该调用没有实施，那么指令栈将保存着函数 OS_killcard 的值，而该值在以后的出栈，例如在一条执行返回的命令期间出栈，就导致检测出该

执行异常，以及执行异常处理函数 OS_killcard。

附录 C 包含一个计算机程序的、被编为/*c1*/至/*c32*/的 32 行指令，该计算机程序的执行被一个根据在最佳实施方式下的本发明的安全保护方法保护。

行/*c1*/至/*c11*/与参照附录 B 描述的行/*b1*/至/*b11*/相似，唯一的不同之处在于：将占用两个八位字节、用十六进制编码的预定值 05F1H，而不是函数 OS_killcard 的地址，入指令栈(行/*c5*/和/*c6*/)。

在调用子程序 fonction_critique 之前，该入栈操作步骤也被实施。

在该特殊实施方式下，该预定值 05F1H 表示由行/*c12*/至/*c19*/的指令构成的子集。

行/*c12*/至/*c19*/与参照附录 B 描述的行/*b12*/至/*b19*/相似。

因此，在附录 C 的程序正常执行的情况下，在行/*C19*/的指令执行结束时，寄存器的 R6 和 R7 分别包含着与函数“fonction_critique”的返回地址相对应的、行/*c10*/的第一条指令的地址的高八位字节和低八位字节。

然后，在行/*c20*/出指令栈入寄存器 A，然后，在行/*C21*/该寄存器的内容与十六进制数值 F1H 比较。

正常情况下，如果程序没有遭受过攻击，尤其是在调用函数“fonction_critique”的时刻没有遭受攻击，那么寄存器 A 就包含着在行/*c5*/的指令期间入栈的数值 F1H。

因此，根据本发明，行/*c20*/的出栈步骤容许检测一个执行异常。

如果在行/*c21*/的比较步骤中发现寄存器 A 的值不等于 F1H，附录 C 的程序就在行/*c22*/的指令期间转移到“OS_killcard”的地址。尤其，这种情况发生在一个导致未经调用而执行函数“fonction_critique”的通过制造错误的攻击结束时。

因而，在根据本发明的安全保护方法的这种实施方式下，如果在指令/*c20*/的出栈步骤中弹出一个与在指令/*c6*/处入栈的预定值 F1H 不同的数值时，异常处理程序 OS_killcard 就被应用了。

反之，如果在行/*c21*/的比较步骤中发现寄存器 A 存储着数值 F1H，附录 C 的程序就执行行/*c23*/的指令。

行/*c23*/至/*c25*/与前述的行/*c20*/至/*c22*/相似：

- 在行/*c23*/处出栈入寄存器 A；
- 在行/*c24*/处比较寄存器 A 与数值 05H，数值 05H 是在行/*c5*/处入栈的预定值；和

- 如果在执行行/*c25*/的指令的时刻寄存器 A 不包含数值 05H，就在行/*c25*/的指令期间转移到“OS_killcard”地址。

反之，如果寄存器 A 包含数值 05H，程序就执行行/*c26*/的指令。

无论怎样，行/*c20*/和/*c23*/的指令的执行会从执行栈中删除预定值 05F1H。

行/*c26*至/*c29*/与前文中参照附录 B 描述的行/*b22*/至/*b25*/相似。

它们容许在执行行/*c17*/和/*c19*/的指令时将存储在寄存器 R6 和 R7 中的数值入指令栈，即分别为：

- 行/*c10*/的第一条指令的地址的高八位字节；和
- 行/*c10*/的第一条指令的地址的低八位字节。

行/*c30*至/*c32*/与前文中参照附录 B 描述的行/*b26*/至/*b28*/相似。

因而，如果未曾有过攻击，程序就转移到行/*c10*/的第一指令，而在行/*c27*/和/*c29*/，该指令的地址已经入了栈。

与前文中参照附录 A 描述的行/*a1*/相似的行/*c10*/表示与本发明无关的一组指令，而行/*c11*/则结束附录 C 的函数“function1”。

在该实施方式下，数值 05F1H 本可以是一个异常处理函数的地址。该特殊实施方式容许加强程序的安全保护，因为即使在执行行/*c20*/至/*c25*/的测试期间发生了一次攻击，也会通过后来应用该异常处理函数而检测到该攻击。

作为变型，可以使用多个异常处理函数的地址，其中的每一个各是一个与一组关键指令相关联的预定值。

附录 D 包含一个计算机程序的、被编为/*d1*/至/*d32*/的 32 行指令，该计算机程序的执行被一个根据在最佳实施方式下的本发明的安全保护方法保护。

在该特殊实施方式下，在行/*d4*/处，程序包含一个对子程序“fonction_critique”的调用。

该调用自动导致该子程序的返回地址，即行/*d5*/的指令的地址，入栈。

在子程序“fonction_critique”的行/*d20*/至/*d23*/期间，把指令栈的前几个值，即该子程序的用两个八位字节编码的返回地址，存储到 R6 和 R7 中。

然后，在行/*d24*/和/*d25*/处把预定值 05F1H 入栈。

我们将注意到：在该实施方式下，该入栈步骤在执行子程序“fonction_critique”期间实施。

最后，在执行行/*d27*/和/*d29*/的指令期间，将寄存器 R6 和 R7，即包含着行/*d5*/的指令的地址的寄存器，的内容入栈，如上文所讲解的那样。

因而，附录 D 的程序在子程序“fonction_critique”结束时转移到行/*d5*/。

在执行行/*d5*/的指令之前，指令栈的前两个数值在正常情况下是在行/*d24*/和/*d25*/处入栈的预定值 05H 和 F1H。

与前文中参照附录 A 描述的行/*a1*/相似的行/*d5*/表示与本发明无关的一组指令。我们假定这些指令使指令栈处于它在指令/*d5*/之前所处的状态。

行/*d7*/至/*d12*/与前文中参照附录 C 描述的行/*c20*/至/*c25*/相似：

- 在行/*d7*/和/*d10*/处出栈入寄存器 A；
- 在行/*d8*/和/*d11*/处比较寄存器 A 与预定值 F1H 和 05H；
- 如果在执行行/*d9*/(分别地/*d12*/的指令的时刻寄存器 A 不包含数值 F1H(分别地 05H)，就在指令/*d9*/(分别地/*d12*/期间转

移到“OS_killcard”地址。

因此，例如如果在/*d7*/出栈步骤中，弹出一个与预定值 F1H 不同的值，那么异常处理子程序 OS_killcard 就被应用。

我们将注意到：在该实施方式下，从执行栈中删除预定值 05F1H 是在执行子程序“fonction_critique”之后被实施的，而不是在执行另一个子程序期间发生的一次攻击之后而被实施的，该攻击具有执行行/*d6*/至/*d13*/的后果。

因而，这种应用容许确信执行行/*d6*/至/*d13*/的指令是在执行子程序“fonction_critique”之后被实施的。

行/*d14*/和/*d15*/结束附录 D 的程序。

附录 E 包含一个计算机程序的、被编为/*e1*/至/*e28*/的 28 行指令，该计算机程序的执行被一个根据在最佳实施方式下的本发明的安全保护方法保护。

行/*e1*/至/*e5*/和行/*e12*/至/*e28*/分别与参照附录 D 描述的行/*d1*/至/*d5*/和/*d16*/至/*d32*/相似，唯一的不同之处在于：将异常处理函数 OS_killcard 的地址，而不是预定值 05F1H，入地址指令栈(行/*e20*/和/*e21*/)。

该入栈步骤还在执行子程序“fonction_critique”期间被实施。

因而，在子程序“fonction_critique”结束时，附录 E 的程序转移到行/*e5*/。

正常情况下，在执行行/*e5*/的指令之前，指令栈的前两个数值是函数 OS_killcard 的高位地址和低位地址，这些预定值已经在行/*e21*/和/*e21*/处入栈。

在执行行/*e7*/和/*e8*/的指令期间，这些值出栈。

该特殊实施方式容许确信函数“fonction_critique”确实是在被调用之后执行的，而不是由于一个通过制造错误的攻击而执行。

事实上，在相反的情况下，在一个子程序的执行返回的必然时刻，函数 OS_killcard 的地址的出栈容许检测一个执行异常，尤其是通过应用该函数。

行/*e10*/和/*e11*/结束附录 E 的程序。

图 1 表示根据一种最佳实施方式下的本发明的微电路卡 100。

为简化起见，只有微电路的内容被表示出来，并且是以示意的方式表示的。

已经知道，根据本发明的微电路卡 100 还包括微电路卡的一些常用的硬件和软件部件，即尤其是一个半刚性材料的载体和供电装置。这里将不对这些部件作描述。

根据本发明的微电路卡 100 包括一些实施如前文中参照附录 A 至 E 所描述的安全保护方法的装置。

在此处所描述的最佳实施方式下，这些装置包括：一个处理器 110，尤其，该处理器与一个 EEPROM 类型的非易失性存储器、一个包含一个指令栈 STACK 的随机存取存储器 RAM、以及一个包含一个操作系统 OS 的只读存储器 ROM 相连接。

尤其，半易失性存储器 EEPROM 包含附录 A 至 E 的各程序，这些程序被处理器 100 读，以便执行它们。

存储器 EEPROM 还包含两个子程序“异常”和“OS_killcard”。

在执行附录 A 至 E 的程序时，寄存器 R6、R7 和 test 被存储在读写存储器 RAM 中。

在此处描述的实施方式中，寄存器 A 是处理器 110 的累加器。

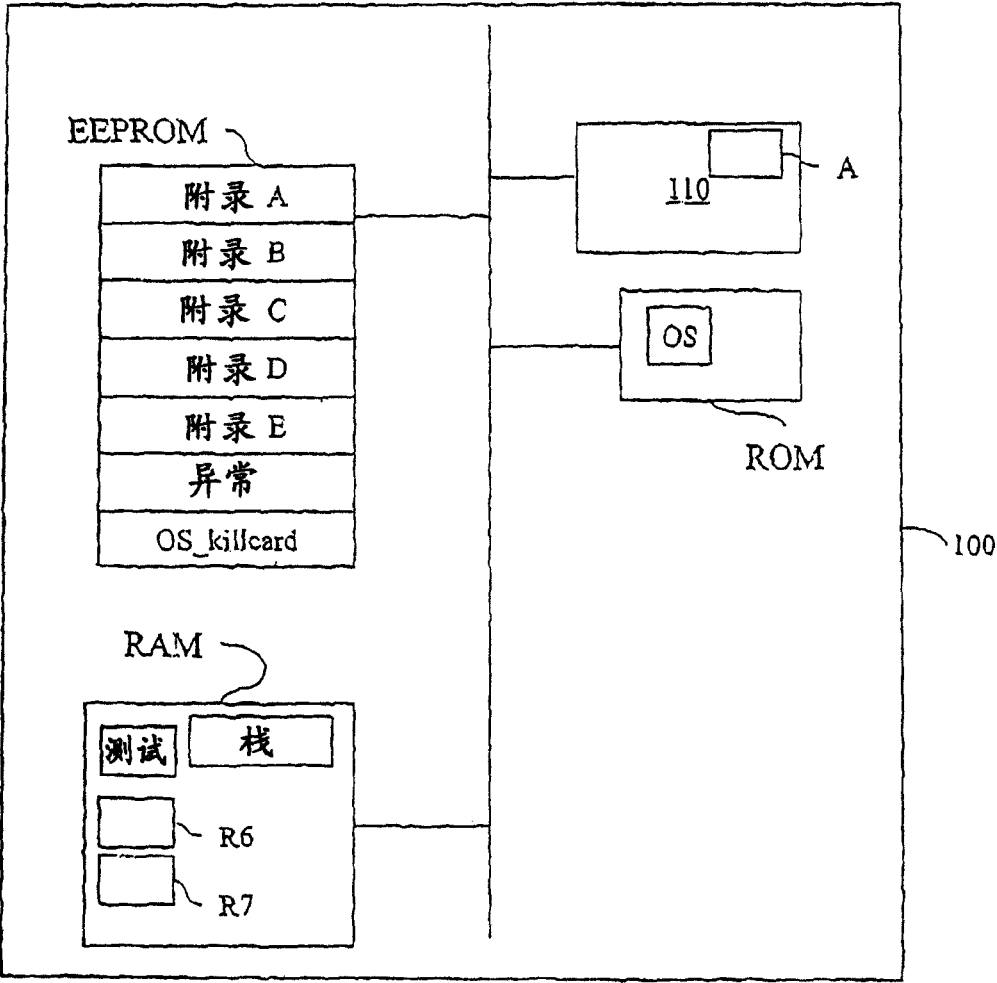


图 1