[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 9/38 (2006.01)
G06F 9/32 (2006.01)

[12] 发明专利说明书

专利号 ZL 200380110331.0

[45] 授权公告日 2009 年 4 月 29 日

[11] 授权公告号 CN 100483336C

[22] 申请日 2003.12.30
[21] 申请号 200380110331.0
[30] 优先权
[32] 2003. 6. 5 [33] GB [31] 0312977.2
[86] 国际申请 PCT/GB2003/005674 2003.12.30
[87] 国际公布 WO2004/109499 英 2004.12.16
[85] 进入国家阶段日期 2005.12.5
[73] 专利权人 ARM 有限公司
地址 英国剑桥郡
[72] 发明人 D·J·西尔 V·瓦斯金
[56] 参考文献
CN1347029 2002.5.1
审查员 詹芊芊

[74] 专利代理机构 中国专利代理(香港)有限公司
代理人 程天正 王 勇

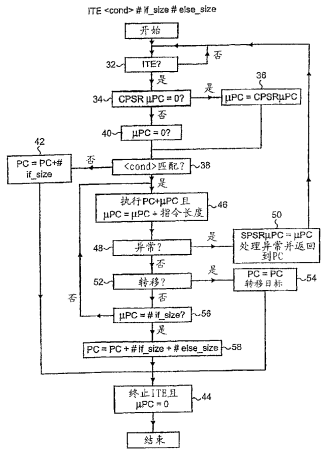
权利要求书 4 页 说明书 16 页 附图 5 页

[54] 发明名称

数据处理系统内处理数据的设备及方法

[57] 摘要

在数据处理系统(2)内部，提供断定指令(30)，该断定指令将条件动作添加到相关联程序指令(26，28)。



1. 用于处理数据(2)的设备，所述设备包括：

用来执行数据处理操作的数据处理逻辑（4，6，8，10）；

指令译码器(12)，其具有响应于程序指令而生成控制信号以控制所述数据处理逻辑去执行所述数据处理操作的电路; 以及

用于存储地址以指示正被执行的程序指令的存储位置的程序计数器寄存器，其中：

所述电路译码一断定指令来生成提供给所述数据处理逻辑的控制信号，以根据用于处理数据的设备的一个或多个条件状态，来控制所述数据处理逻辑去执行或不执行跟随在所述断定指令之后的一个或多个相关联程序指令，所述条件状态是通过所述断定指令之外的一个或多个程序指令的执行来设定的；且所述设备进一步包括

用于存储计数器值的被断定指令计数存储器，该计数器值指示受制于所述断定指令的一个或多个相关联程序指令中的多少个已经被执行，当受制于所述断定指令的一个或多个相关联程序指令被执行时，所述程序计数器寄存器继续存储对应于所述断定指令的地址；以及

一异常处理电路（18），所述异常处理电路在异常发生后存储所述计数器值，并在异常完成后重新启动在由所述计数器值指向的程序指令处开始的执行。

2. 如权利要求 1 所述的设备， 其中所述断定指令控制多个相关联程序指令。

3. 如权利要求 1 所述的设备，其中所述一个或多个相关联程序指令是非条件程序指令。

4. 如权利要求 1、2 和 3 中任何一个权利要求所述的设备， 其中所述一个或多个条件状态包括一个或多个条件码标记的值。

5. 如权利要求 1 所述的设备， 其中所述一个或多个条件状态是在以下之一评估的：

（i）刚一执行所述断定指令便评估一次；和

（ii）在执行每个相关联程序指令之前评估。

6. 如权利要求 1 所述的设备， 其中所述一个或多个相关联指令紧跟着所述断定指令。

7. 如权利要求 1 所述的设备， 其中所述断定指令与以下相关联：

（i）在所述一个或多个条件状态匹配一个或多个预定条件的情况下的、将要执行的一个或多个程序指令的条件匹配块；和

（ii）在所述一个或多个条件状态不匹配所述一个或多个预定条件的情况下的、将要执行的一个或多个程序指令的条件不匹配块。

8. 如权利要求 7 所述的设备， 其中所述断定指令指定以下的一个或多个：

（i） 所述一个或多个程序指令的条件匹配块的长度；和

（ii）所述一个或多个程序指令的条件不匹配块的长度。

9. 如权利要求 7 和 8 中的任一个权利要求所述的设备，其中所述断定指令指定所述一个或多个预定条件 。

10. 如权利要求 1 所述的设备，其中所述断定指令包括一个或多个字段，其中每个字段根据所述一个或多个条件状态与一个或多个预定状态的比较而指定是执行或是不执行各自的相关联程序指令。

11. 如权利要求 10 所述的设备，其中所述断定指令包括一字段，所述字段为每个各自的相关联程序指令指定所述一个或多个条件状态或所述一个或多个条件状态的互补是否与所述一个或多个预定状态进行比较，来确定是否执行所述各自的相关联程序指令。

12. 如权利要求 1 所述的设备， 其中所述一个或多个相关联指令在受制于所述断定指令时，指定不同的数据处理操作。

13. 如权利要求 12 所述的设备，其中根据所述断定指令内的可编程字段，禁阻所述一个或多个相关联指令对所述一个或多个条件状态作出任何改变。

14. 一种处理数据的方法，所述方法包括以下步骤：

利用数据处理逻辑执行数据处理操作；

利用指令译码器译码程序指令而生成控制信号来控制所述数据处理逻辑去执行所述数据处理操作，以及

存储一地址，所述地址指示程序计数器寄存器中正被执行的程序指令

的存储位置，进一步包括以下步骤：

通过所述指令译码器译码一断定指令而生成被提供给所述数据处理逻辑的控制信号，以根据所述用于处理数据的设备的一个或多个条件状态，控制所述数据处理逻辑来执行或不执行跟随在所述断定指令之后的一个或多个相关联程序指令，所述条件状态是通过所述断定指令之外的一个或多个程序指令的执行来设定的；

在被断定指令计数器寄存器中存储计数器值，所述计数器值指示受制于所述断定指令的所述一个或多个相关联程序指令的多少个已经被执行；以及

当执行所述一个或多个相关联程序指令时，在所述程序计数器寄存器中继续存储对应于所述断定指令的地址；以及

在异常发生后存储所述计数器值，并在异常完成后重新启动在由所述计数器值指向的程序指令处开始的执行。

15. 如权利要求 14 所述的方法，其中所述断定指令控制多个相关联程序指令。

16. 如权利要求 14 所述的方法，其中所述一个或多个相关联程序指令是非条件程序指令。

17. 如权利要求 14，15 以及 16 中的任何一个权利要求所述的方法， 其中所述一个或多个条件状态包括一个或多个条件码标记的值。

18. 如权利要求 14 所述的方法， 其中所述一个或多个条件状态在以下之一评估：

（i）刚一执行所述断定指令便评估一次；和

（ii） 在执行每个相关联程序指令之前评估。

19. 如权利要求 14 所述的方法， 其中所述一个或多个相关联指令紧跟着所述断定指令。

20. 如权利要求 14 所述的方法， 其中所述断定指令与以下相关联：

（i）在所述一个或多个条件状态匹配一个或多个预定条件的情况下、将要执行的一个或多个程序指令的条件匹配块；和

（ii）在所述一个或多个条件状态不匹配于所述一个或多个预定条件的情况下、将要执行的一个或多个程序指令的条件不匹配块。

21. 如权利要求 20 所述的方法， 其中所述断定指令指定以下的一个或多个：

（i）一个或多个程序指令的所述条件匹配块的长度；和

（ii）一个或多个程序指令的所述条件不匹配块的长度。

22. 如权利要求 20 所述的方法， 其中所述断定指令指定所述一个或多个预定条件。

23. 如权利要求 14 所述的方法，其中所述断定指令包括一个或多个字段，每个字段根据所述一个或多个条件状态与一个或多个预定状态的比较而指定是执行或是不执行各自的相关联程序指令。

24. 如权利要求 23 所述的方法，其中所述断定指令包括一字段，所述字段为每个各自的相关联程序指令指定所述一个或多个条件状态或所述一个或多个条件状态的互补是否与所述一个或多个预定状态进行比较，来确定是否执行所述各自的相关联程序指令。

25. 如权利要求 14 所述的方法，其中所述一个或多个相关联指令当受制于所述断定指令时，指定不同的数据处理操作。

26. 如权利要求 25 所述的方法，其中根据所述断定指令内的可编程字段，禁阻所述一个或多个相关联指令对所述一个或多个条件状态作出任何修改。

数据处理系统内处理数据的设备及方法

本发明涉及数据处理系统领域。更具体而言，本发明涉及这种数据处理系统内程序指令的条件执行。

提供响应于包括程序指令的指令集的数据处理系统是公知的，其中所述程序指令根据先前的处理操作的结果来有条件地执行或不执行。例如，ARM32 位指令集是完全条件的，因此所有的指令都包括指定在哪些条件下将禁止或允许执行的 4 位条件码字段。

虽然完全条件的 32 位 ARM 指令集在它的条件行为中是强有力的，但是每个指令内部的专用于条件码的 4 位代表该指令集的位编码空间的重要部分。在多数情况下，并没有完全利用所述条件码的全部能力，并且所述专用于这些条件码的 4 位也没有被很好的使用。

提供诸如由 ARM 处理器支持的 Thumb 指令集之类的非条件指令集也是公知的。不使任何指令位空间专用于条件码便允许 Thumb 指令更小，从而提高代码密度。然而，这种方法的缺点在于，在没有利用所述 32 位 ARM 指令集的条件码的能力的情况下，执行某些操作所需要的指令数目可能更多并且因此处理缓慢。

从诸如 PA-RISC 处理器之类的其他处理器提供指令、其可根据它们的结果而取消下一个指令是公知的。这种指令的示例是比较操作，该比较操作用来根据被比较的数字是相等还是不相等，来禁止或允许下一个跟随指令执行。虽然这种取消下一指令类型是可用的，但其具有有限的灵活性，因为它们只能用来根据其自身处理而取消下一个指令。

Pnevmatikatos D N等人的："Guarded Execution and Branch Prediction In Dynamic ILP Processors（在动态 ILP 处理器中的保护执行和转移断定）"公开了一种提供保护指令用于断定一个或多个跟随指令的系统。

从一个方面看，本发明提供用于处理数据的设备，该设备包括：

用来执行数据处理操作的数据处理逻辑；

一指令译码器，其具有用来响应于程序指令而生成控制信号以控制所

述数据处理逻辑去执行所述数据处理操作的电路；以及

用来存储地址以指示正被执行的程序指令的存储位置的程序计数器寄存器，其中

所述指令译码器译码一断定指令来生成提供给所述数据处理逻辑的控制信号，以根据用于处理数据的所述设备的一个或多个条件状态，来控制所述数据处理逻辑去执行或不执行跟随在所述断定指令之后的一个或多个相关联程序指令，其中所述条件状态通过所述断定指令之外的一个或多个程序指令的执行来设定；此外还包括

用于存储计数器值的被断定指令计数器寄存器，所述计数器值指示受制于所述断定指令的所述一个或多个相关联程序指令的多少个已经被执行，当执行受制于所述断定指令的所述一个或多个相关联程序指令时，所述程序计数器寄存器继续存储对应于所述断定指令的地址；以及

一异常处理电路，所述异常处理电路在异常发生后存储所述计数器值，并在异常完成后重新启动在由所述计数器值指向的程序指令处开始的执行。

本发明认识到，提供特定用途的断定指令可有很大的优势，因为这些可取决于更灵活指定的、处理器的条件状态而使得其他指令的执行是有条件的。因此，程序员和编译者可以更好的使用这种断定指令来产生更小和更快的程序。

为了便于所述处理系统的控制，正常的程序计数器寄存器是由被断定指令计数器寄存器补足的，所述被断定指令计数器寄存器用来存储计数器值，该计数器值指示受制于（subject to）断定指令的一个或更多个相关联程序指令中多少已经被执行。断定指令事实上在其中编码一个可达范围，指定它控制多少相关联程序指令、以及系统需要追踪执行已经通过此可达范围进展多远，以及何时已超过该可达范围。

此被断定指令计数器寄存器尤其是在异常处理以及从异常返回的情况下是有用的，由此在从异常返回后，所述被断定指令计数器寄存器可以被恢复并且它的值被用来确定被返回到的程序指令是否是使其行为被在前断定指令影响的那个。

虽然断定指令可能只影响一个其他指令，但在优选实施方案里，提供

所述断定指令来控制多个相关联程序指令的执行。

虽然断定指令可以结合相关联程序指令一起使用，所述程序指令是部分地或全部有条件的，但是本发明的优选实施方案是断定指令与无条件的相关联程序指令一起使用的方案。因此，所述断定指令能够将条件指令行为添加到另外的非条件指令集。

虽然断定指令使其对相关联程序指令的控制所基于的条件状态可以采用各种不同的形式，但是优选实施方案为了提供输入给所述断定指令，使用诸如，举例而言零标记、相等标记、进位标记等的条件码标记。

由断定指令评估以控制相关联程序指令的条件状态可以根据特定系统的要求在不同的时间被评估。有可能是条件状态可以刚一执行断定指令后就被评估，并且所述评估结果被用于控制所有的相关联程序指令。这样所具有的优势在于：使得有可能在指令流水线里及早完全评估断定指令，及此后可以以迅速且简化执行的方式从指令流水线移除所述断定指令。作为可选择的，也有可能是：条件状态可在每个相关联程序指令的执行之前被评估，使得所述条件码例如可对于一些相关联指令被通过，而对于其他的相关联指令则不被通过。这个方法的优势在于：可以以很好地适合许多普通类型的数据处理操作的方式来对条件行为进行优良程度的控制，其中可能期望它与这种条件指令行为一起执行。

虽然可以理解的是，所述相关联指令对所述断定指令可具有各种不同的相对位置，但优选实施方案是其中所述相关联程序指令紧跟着所述断定指令的方案。这对执行以及编程目的是方便的。

以各种不同的方式安排断定指令来控制相关联程序指令是可能的。在一个可能的优选实施方案里，将所述相关联程序指令安排在两个指令块里，其中一个这种块是在条件码被通过后来执行的，而另一个块是在条件码失败后执行的。这可以提供类似于指令类型 If...Then...Else 的行为。

在这种实施例里，虽然不同的指令块可以具有固定大小，但是优选的它们具有可变的大小，该大小是由断定指令自身内的字段指定的。这允许块大小中的灵活性，有利于断定指令的转移及其他行为的控制以及改善代码密度。

虽然有可能断定指令可专用于预定条件的特定集合，但是在优选实施

方案里，所述断定指令包括一个可变字段，通过该可变字段可以指定预定条件。

另一种给出有利的灵活度且尤其允许那些将通过或不通过其条件码的指令的交织的断定指令是这样一种断定指令，即：在其中断定指令包括一个或多个字段，每个字段根据一个或多个条件状态同一个或多个预定状态的比较来指定是要执行或是不执行各自的相关联程序指令。

尤其是在优选实施方案里，所述断定指令可以包括一字段，所述字段为每个各自的相关联指令指定：一个或多个条件状态或一个或多个条件状态的问候是否要与预定状态进行比较，以确定是否执行所述各自的相关联程序指令。因此，可以指定个体的相关联程序指令来执行所述"Then"条件或者所述"Else"条件。

有可能如同根据所述断定指令来控制相关联程序指令是执行或是不执行那样，所述断定指令也可用来修改所述相关联指令，使得当它们被执行时所述相关联指令具有不同的操作。这可以被看作是有效地扩展了所述指令集。

修改所述相关联程序指令的一个特定优选方法是：当它们受制于特别的、包含特定可编程字段的断定指令时，禁阻它们对一个或多个条件状态做任何变化。因此，条件码标记更新可以由适当的断定指令来禁止。

从另一方面看，本发明提供了一种处理数据的方法，所述方法包括以下步骤：

利用数据处理逻辑执行数据处理操作；

利用指令译码器译码程序指令而生成控制信号来控制所述数据处理逻辑去执行所述数据处理操作，以及

存储一地址，所述地址指示程序计数器寄存器中正被执行的程序指令的存储位置；进一步包括以下步骤：

通过所述指令译码器译码一断定指令而生成被提供给所述数据处理逻辑的控制信号，以根据所述用于处理数据的设备的一个或多个条件状态，控制所述数据处理逻辑来执行或不执行跟随在所述断定指令之后的一个或多个相关联程序指令，其中所述条件状态是通过所述断定指令之外的一个或多个程序指令的执行来设定的；

在被断定指令计数器寄存器中存储计数器值操作，所述计数器值指示受制于所述断定指令的所述一个或多个相关联程序指令的多少个已经被执行；以及

当执行所述一个或多个相关联程序指令时，在所述程序计数器寄存器中继续存储对应于所述断定指令的地址；以及

在异常发生后存储所述计数器值，并在异常完成后重新启动在由所述计数器值指向的程序指令处开始的执行。

从另一个方面看，本发明提供了一种计算机程序产品，所述计算机程序产品包括根据本技术用来控制处理数据的设备的计算机程序。

现在参考附图以仅为示例的方式来描述本发明的实施例，其中：

图 1 示意地示出了用来执行包括断定指令的程序指令的数据处理设备。

图 2 示意地示出了指令译码器响应断定指令的动作；

图 3 和 4 示意地示出了断定指令的一种形式，其中根据掩码值以及条件码来控制个体相关联程序指令执行或不执行；

图 5 是图表，示意地示出了另一种断定指令，其中相关联的指令块根据所述条件码执行或不执行；

图 6 是流程图，示意地示出了遵循图 5 所述类型的断定指令的执行的控制；

图 7 是用于执行上述技术的通用计算机设备的示意图。

图 1 示出了以处理器核心 2 的形式的数据处理设备，处理器核心 2 包括程序寄存器库 4 、乘法器 6 、移位器 8 、加法器 10、指令译码器 12、指令流水线 14、预取装单元 16、中断控制器 18、程序计数器寄存器 20 和断定指令计数器寄存器。可以理解的是，所述处理器核心 2 一般地包括本技术领域的技术人员所熟知的很多其他电路元件，但是为了简明，将这些电路元件从图 1 中省略掉了。

操作中，从存储器(未示出)处取装程序指令 I，并将其提供给指令流水线 14，程序指令在那里经过各种各样的流水线级，诸如取装、译码和执行，就如本技术领域所熟知的一样。将来自指令流水线 14 的译码级的指令提供给指令译码器 12，在那里指令用来产生控制信号，所述控制信号配置

并且控制数据处理操作/处理，所述操作/处理是由乘法器 6 、移位器 8 、加法器 10 和其它诸如 ALU（未示出）之类的电路元件来执行的，依据从所述程序寄存器库 4 读取的数据值来产生结果值，所述结果值要被写回到所述程序寄存器库 4 。预取装单元 16 用来控制指令地址产生，该指令地址用于取装程序指令 I 到指令流水线 14 中。程序计数器寄存器 20（可以是程序寄存器库 4 内的程序寄存器中的一个）提供正在执行的当前指令的地址值，或者对这个值具有预定偏移（例如+4）的地址。上下文寄存器 22 也提供一个值到预取装单元 16，该值在从中断返回后被使用，正如由中断控制器 18 控制的，使得受制于断定指令的相关联程序指令的一个部分完成的序列将仍受断定指令的影响，即使这种执行由异常的出现中断也如此。

图 2 示意地示出了第一示例实施例，其中断定指令 IT 是以程序指令流的形式馈入到指令译码器 12。当所述指令译码器 12 检测到这种断定指令时，就译码所述断定指令并且有效地将条件码添加到所述指令流内跟随所述断定指令的相关联程序指令。可以理解的是在这个例子里的所述指令流内的程序指令正常地是无条件指令，诸如 16 位 Thumb 指令，其正常地不具有与它们相关联的条件码。但是，支持 Thumb 指令集的处理器也支持完全条件的 32 位 ARM 指令集。当来自这个 ARM 指令集的指令正被译码时，那么它们都包括条件码，该条件码从所述指令译码器 12 向下继续沿指令流水线 14 传送以影响那些指令的后续执行或不执行。因此所述指令流水线 14 已经明了条件码，并且具有用于译码和响应与程序指令有关的条件码的结

构和机构。在一个整体的级别上可以看到，断定指令 IT 用来将条件码添加到在指令流内跟随其后的一个或多个相关联程序指令中，而它自己被从所述指令流中移走，因为除了这个条件码信息的添加功能之外它没有用于其它处理功能。

图 3 更详细地示出了图 2 的断定指令 IT。特别地，这个指令指定条件码"cond"和掩码值"xyx"，指示跟着 IT 指令的 1,2,3 或 4 程序指令是否受制于它的断定，以及如果是这样的话，就要施加所述条件码的极性。所述条件码或条件码的问候可以根据对应值 x，y，z 是否对相应的第二跟随、第三跟随以及第四跟随相关联程序指令存在以及这个标记的值来施加。图 3 示出了所述值 x，y，z 如何形成由值"1"来终止的掩码值 24。

图 4 示意地示出了这种指令的实例，其中四个相关联指令由断定指令来断定。这些指令的第一个被认为以"Then"形式断定，因此如果满足条件码则将执行所述指令。也可以指定第二、第三以及第四相关联指令依据“Then”条件或者"Else"条件来执行。在这个实例里，所述四个相关联程序指令具有形式 Then，Then，Else 和 Then。在掩码值 24 内，用于第二、第三和第四相关联指令的掩码值位 x，y，z 分别是 0，1，0，并且掩码值由值"1"来终止。

这个实例里指定的条件码是"1010"。这个条件码以其正常形式施加给第一相关联指令。然后所述掩码值左移一个位置，并且从所述掩码值移出的最高有效位被移至将施加到下一个指令的条件码的最低有效位位置。因此，施加到下一个指令的条件码可以根据所述掩码值内的最高有效位，在其正常形式和其补码形式之间进行转换。在这个实例里将看到，跟随所述断定指令的第一、第二和第四指令受制于所述正常条件码，而跟随所述断定指令的第三指令受制于所述条件码的补码。当检测到掩码值"1000"时，这表明已经到达最后的断定指令，并且所述断定指令的影响将不延伸到后来的程序指令。

在下文中给出了在图 3 和 4 里示出的指令类型的进一步描述：

IT

If - Then（IT）指令使存储器里的紧接的 N 指令是有条件的。跟随所述 IT 指令并且受其影响的 N 指令集被称为"IT 块"。

当在 IT 块内执行一些（或全部）指令时，可以重新定义该指令的功能性。例如，除 CMP/CMN/TST 之外的 Thumb 16 位指令可以没有标记设定。在这种情况下，IT 可与 AL 条件一起使用以在其自身上获取后者的效果。

可使用--个额外的 PSR 位（或专用 IT 状态寄存器）来保有 IT 状态。这将允许在 IT 块内中断。

以下实例中定义了具有 4 个指令的最大 IT 块大小的 Thumb 16 位 IT 指令。IT 块内的全部指令是由相同条件或条件（即 EQ 或 NE 等）的相反版本来断定的。

Thumb IT

语法

IT{<x>{<y>{<z>}}}<cond>

其中：

<x> 如果出现，是字母"T"和"E"的一个，并指出第二跟随指令将受 IT 指令影响。如果< x >是"T"（"Then"），那么附着于那个指令的条件是<cond>；如果< x >是"E"，那么它是<cond>的反，是由逻辑地反转<cond>的最低有效位而获得。如果< x >被省略（且由此< y >和< z >也被省略），那么只有一个跟随的指令受 IT 指令影响。

<y> 如果出现，是字母"T"和"E"的一个，并指出第三跟随指令将受 IT 指令影响。如果< y >是"T"（"Then"），那么附着于那个指令的条件是<cond>；如果< y >是"E"，那么它是<cond>的反，是由逻辑地反转<cond>的最低有效位而获得。如果< y >被省略（且由此< z >也被省略），那么少于 3 个跟随指令受 IT 指令影响。

<z> 如果出现，是字母"T"和"E"的一个，并指出第四跟随指令将受 IT 指令影响。如果< z >是"T"（"Then"），那么附着于那个指令的条件是<cond>；如果< x >是"E"，那么它是<cond>的反，是由逻辑地反转<cond>的最低有效位而获得。如果< z >被省略，那么少于 4 个跟随指令受 IT 指令的影响。

<cond>

是要施加到第一个跟随指令的条件，也可能是要施加到由< x >，< y >和<z>指定的其他跟随指令的条件。

<x>，<y>和<z>的值根据下列表格来确定指令的掩码字段的值：

<x>	<y>	<z>	掩码 [3]	[2]	[1]	[0]
省略	省略	省略	1	0	0	0
T	省略	省略	cond[0]	1	0	0
E	省略	省略	非 Cond[0]	1	0	0
T	T	省略	cond[0]	cond[0]	1	0
E	T	省略	非 cond[0]	cond[0]	1	0
T	E	省略	cond[0]	非 cond[0]	1	0
E	E	省略	非 cond[0]	非 cond[0]	1	0
T	T	T	cond[0]	cond[0]	cond[0]	1
E	T	T	非 cond[0]	cond[0]	cond[0]	1
T	E	T	cond[0]	非 cond[0]	cond[0]	1
E	E	T	非 cond[0]	非 cond[0]	cond[0]	1
T	T	E	cond[0]	cond[0]	非 cond[0]	1
E	T	E	非 cond[0]	cond[0]	非 cond[0]	1
T	E	E	cond[0]	非 cond[0]	非 cond[0]	1
E	E	E	非 cond[0]	非 cond[0]	非 cond[0]	1

注解：此语法的基本思想是，例如"ITTET EQ"表明第一、第二和第四跟随指令（由记忆中的"T"的位置指出）具有附着于它们的 EQ 条件，以及第三跟随指令（由所述"E"指出）具有一附着于它的 NE 条件。

操作

```
if CPSR [26,25, 15: 10]! = 0b00000000 then
   UNPREDICTABLE（不可预测）
else
   if mask = 0b0000 then
      UNPREDICTABLE
   else
      CPSR [15: 12] = cond
      CPSR [11,10, 26,25] = mask
```

//这些 CPSR 位的指令接指令更新然后使得

//发生 IT 指令的条件执行效果。

所述 8 个 IT 执行状态位如下地编码 256 个"指令间状态 "中的一个：

CPSR 状态

26	25	15	14	13	12	11	10	
P4	1	condbase			P1	P2	P3	对于条件（condbase,P1）、（condbase,P2）、（condbase,P3）和（condbase,P4），有条件地执行下 4 个指令
1	0	condbase			P1	P2	P3	对于条件（condbase,P1）、（condbase,P2）和（condbase,P3），有条件地执行下 3 个指令
0	0	condbase			P1	P2	1	对于条件（condbase,P1）和（condbase,P2）有条件地执行下 2 个指令
0	0	condbase			P1	1	0	对于条件（condbase,P1），有条件地执行下一个指令
0	0	N(非零)				0	0	不可预测的
0	0	0	0	0	0	0	0	正常执行

这些位对后续执行的效果是：

- MRS 和 SRS 指令将正常地读取这些位。MSR 指令将其正常地写到 SPSR，但如上所述，当写入 CPSR 时忽视所述写入值。依然如上所述，存在的异常返回指令（带有"复制 SPSR→CPSR"副作用的那些）和 RFE 指令将它们正常地写入 CPSR。
- 在 ARM 和 Java 状态执行期间，这些位应当是零（即如果其不为零，那么结果是 UNPREDICTABLE）。
- 异常入口序列使这些位从 CPSR 复制到 SPSR，然后在 CPSR 中清除（连同 J 和 T 一起），使处理器返回到正常 ARM 执行。
- 在 Thumb 状态执行期间：

1、当执行 IT 指令时这些位应该是零。（即，IT 指令被禁止在受其他 IT 指令影响的指令块内出现。）

2、当条件转移（branch）指令执行时，位[26,25,11,10]应该是零。

（即，条件转移指令被禁止在受 IT 指令影响的指令块内出现。然而，非条件转移指令可以出现在那里，且也将使得被当做条件对待：这个规则基本上被给出以避免由相同指令上的两个条件所引起的问题。}

3、对于没有由上述规则造成 UNPREDICTABLE 的全部指令，从当前"执行状态"位产生一条件如下：

if bits [26,25,11, 101] = 0b0000 then

/* 不在 IT 块里 */

condition is AL（条件是 AL）

else

condition is bits [15: 12], interpreting both 0b1110

and 0b1111 as"always"（条件是位[15:12]，解译 0b1110 和 0b1111 为“总是”）

/* 注解：因为在 ARMv5 中恢复"NV 空间"，所以这些解译已经标准化 */

此外，那些非 CMP、CMN 或 TST 指令且正常地设定条件标记的 16 位 Thumb 数据处理指令被修改如下：

if bits [26,25,11, 10] ! = 0b0000 then

/* 在 IT 块里 */

instruction does not set condition flags（指令不设置条件标记）

注解：这对于令 IT 对控制大量指令序列有用是必要的一例如，如果将它用来条件化（conditionalise）LDR Rn；ADD Rn，Rn，＃M；STR Rn 序列，那么如果所述 ADD 改变了由 STR 指令所看到的条件标记，则它将是无用的！

注解：这具有的必然结果是：具有 AL 条件（或 NV 条件）的 IT 指令可用于禁止由 1－4 个跟随指令进行此类标记设置。当前 Thumb 指令集的难处理方面之一是：晚插入的码(例如溢出/填充（spill/fill）码)可能具有破坏条件标记的副作用。能用单个额外的 16 位指令来处理这个问题是很有用的。

4、任何 PC-修改指令的成功（即没有对任何条件核对失败）执行将致使新的"执行状态"位变成：

对于异常返回指令（执行 SPSR→CPSR 复制的那些，以及在特许模式

里执行的 RFE）：

该值的位[26,25,15:10]被写入 CPSR。

否则:

全零。

注解：所述"否则"情况是使得 IT 只影响静态跟随指令的该基本规则。

这种 PC-修改指令的不成功执行，或非 PC-修改、非 IT 指令的任何执行（成功与否）都将以如下方式更新新的"执行状态"位：

当前状态							状态转换的条件	新状态								注解
26	25	15 14	13	12	11	10		26	25	15	14	13	12	11	10	
P4	1	condbase		P1	P2	P3	总是	1	0	Condbase			P2	P3	P4	
1	0	Condbase		P1	P2	P3	总是	0	0	Condbase			P2	P3	1	
0	0	Condbase		P1	P2	1	总是	0	0	Condbase			P2	1	0	
0	0	Condbase		P1	1	0	总是	0	0	0	0	0	0	0	0	A
0	0	任一			0	0	总是	0	0	0	0	0	0	0	0	B

A）终止 IT 执行-但没有要求特别动作。

B）IT 块之外

注解：

1、转移。除非提到，在 IT 块里没有指令被允许成为任何转移的目标，不管是作为转移指令的结果或一些其它改变所述 PC 的指令的结果。未能遵守这个限制能导致不可预测的行为（UNPREDICTABLE behavior）。

2、异常。在指令和它的 IT 块之间、或 IT 块内可能发生中断及其他异常。通常，它们将导致进入异常处理程序，带有适当的返回信息被置于适当模式下的 R14 和 SPSR 中。被设计用于异常返回的指令可按正常使用以从异常返回，并将正确地再继续 IT 块的执行。这是允许 PC-修改指令转移到 IT 块中指令的唯一方式。

3、IT 块。在 IT 块里的指令不允许是：

- IT 指令
- 条件转移指令

如果 IT 块里的指令是上述指令，那么所述结果是不可预测的

（UNPREDICTABLE）。

图 5 示意地示出了另一类断定指令。在这类断定指令里，存在与断定指令 30 有关联的两个相关联程序指令块 26,28。如果在所述断定指令 30 里指定的条件码被通过（pass），那么执行这些指令块中的第一个 26，并将其称作“If 块”。如果与断定指令 30 有关联的条件码失败，那么执行指令块的另一个 28，并且将其称之为"Else 块"。所述断定指令 30 指定这两个块 26,28 的大小，所述两个块在长度上可以各是 1-4 个指令不等。在图 5 的所述实例里，执行 If 块 26 或者 Else 块 28。这个示例将是：当执行断定指令 30 时，条件码实际上被评估一次，并且所述评估结果被应用来断定所有相关联程序指令。因此，将执行整个 If 块 26，或者作为选择，执行整个 Else 块 28。当上下文寄存器 22（MPC）指出已经到达所述 If 块的结尾，那么迫使转移到跟随 Else 块 28 中最后指令的指令。当到达在 Else 块 28 中的最后指令时，那么所述断定指令 30 的作用被移除。

图 6 示意地示出了根据图 5 示出的操作类型的系统的动作流程图。在步骤 32，系统等待识别断定指令（ITE 指令）。在步骤 34，所述系统识别现行程序状态寄存器内存储的上下文计数器值是否是非零的。这种非零的值表明正在从异常进行返回，且因此在步骤 36 接受该上下文值并将其存储在现行程序状态寄存器内，并且所述处理前进至步骤 38，在其中对断定指令内指定的条件码进行评估。可选择的，如果彻底检验步骤 34 产生否结果，那么在步骤 40 将上下文值设置为零。如果在步骤 38 确定条件码评估是所述条件码失败，那么这表明应该执行 Else 块 28，并因此在步骤 42 将程序计数器的值提前到 Else 块 28 内的第一个指令，然后执行所述 Else 块 28，而该断定指令在步骤 44 已经终止。可选择的，如果在步骤 38 评估的条件码被通过，那么在步骤 46 执行由 PC 值与上下文值的总和所指向的指令，并且使上下文值增加所述指令字节长度。

步骤 48 检查异常的发生，并且如果这样一个异常出现。那么步骤 50 用来保存包括上下文值（断定指令计数器值）的现行程序状态寄存器值，处理所述异常，执行所述异常处理例程，以及最后跟随所述异常而返回到由所述程序计数器值 PC 指示的指令，即断定指令。

步骤 52 检验所述转移的发生，以及如果这种转移发生，那么步骤 54

将所述程序计数器值修改为转移目标以及步骤 44 终止所述断定指令的动作。

步骤 56 检验以查看断定指令计数器值是否已经达到指示已到达 If 块 26 的结尾的 If 块大小。如果已经到达所述 If 块 26 的结尾，那么步骤 58 用来继续移动所述程序计数器值到超出所述 Else 块的结尾，以及在步骤 44 中所述断定指令的动作已经终止之后再继续正常处理。可选择的，处理返回到步骤 46，在那里执行 If 块内的下一个指令。

在下文中给出了在图 5 和 6 里示出的指令动作的另一描述：

ITE

If-Then-Else（ITE）指令允许指定：当条件被通过时将要执行的指令块的大小（这个块被定位为紧接在指令之后），以及当条件失败时将要执行的指令块的大小（这个块被定位为紧接在“if”块之后）。

多个 ITE 上下文位（context bits）应该被保留在 PSR 或其他的特定寄存器里；我们将这些位称作 PSR_uPC。所述 PSR_uPC 的精确数目确定"if"块的最大可能大小。

语法

ITE<cond>#if_size,#else_size

操作

```
if PSR_uPC !=0 then
   //从异常返回
   uPC=PSR_uPC
else
   uPC=0
   if ConditionPassed (cond) then
   PC = PC + if_size
   //转移到 else 部分并终止 ITE
loop
   Instr =FetchInstructionFrom (PC +uPC)
   Execute (Instr)
```

```
if Exception() then
    PSR_uPC=uPC
    PC = ExceptionHandler
    //处理异常

if Branch (Instr) then
    PSR_uPC=0
    PC = BranchTarget
    //转移并终止 ITE

uPC +=InstructionLength (Instr)

if uPC > = if _size then
    PSR_uPC =0
    PC = PC + if_size + else_size
    //在 else 部分之后转移并终止 ITE
endloop
```

注解：

1、转移。在 ITE 的“if”段里没有指令被允许成为任何转移的目标，不管是作为转移指令的结果或是一些其它改变所述 PC 的指令的结果。未能遵守这--限制可导致不可预测的行为（UNPREDICTABLE behavior）。

2、异常。在“if”块内的指令之间可发生中断及其他异常。通常，它们将导致进入一异常处理程序，带有适当的返回信息被置于特定异常模式的 R14 和 SPSR 中。被设计用于异常返回的指令可按正常使用以从异常返回，并将正确地再继续“if”块的执行。这是允许 PC-修改的指令转移到“if”块里的指令的唯一方式。

作为对利用图 5 和图 6 所举例说明的 If 块 26 和 Else 块 28 执行断定指令形式（即其中当执行断定指令 30 时评估所述条件码一次）的一种替代方案，一种当执行每个相关联程序指令时为受断定指令 30 影响的每一个相关联程序指令都评估该条件码的替代方案可以由下列行为代表。

ITE Cond, # THEN, # ELSE

(0-3) (0-3)

CPSR 持有 8 位: 4 cond [3: 0], 2 tlm [1: 0], 2 else [1: 0]

ITE docs

Cond, then, else = 来自指令的值

GENERK execution（执行）

IF CPSR_then = =0 THEN

EXECUTE INSTRUCTION NORMALLY

ELSE

EXECUTE INSTRUCTION WITH CONDITION CPSR_cond

/* 且并行*/

IF CPSR_then > 1 THEN

CPSR_then -= 1

ELSE

CPSR _then = CPSR_else

CPSR_else =0

CPSR_cond [0] =NOT (CPSR_cond [0])/*FLIP COND */

		CPSR_cond	_then	_else
例如：	ITE	EQ, #3, #1	X	0	x
	Instr 1	/*EQ*/	EQ	3	1
	Instr 2	/*EQ*/	EQ	2	1
	Instr 3	/*EQ*/	EQ	1	1
	Instr 4	/*NE*/	NE	1	0
	Instr 5	/*uncond*/	EQ	0	0

从以上可以看到，在所述实例里，断定指令指定包含三个相关联程序指令的 If 块，和包含一个相关联程序指令的 Else 块。对这些程序指令的每一个实际指定的条件码是由 CPSR_cond 值示出的。当 Else 块已经退出时，第五个指令是非断定的，并以与这种无条件指令相关联的正常方式无条件地执行。

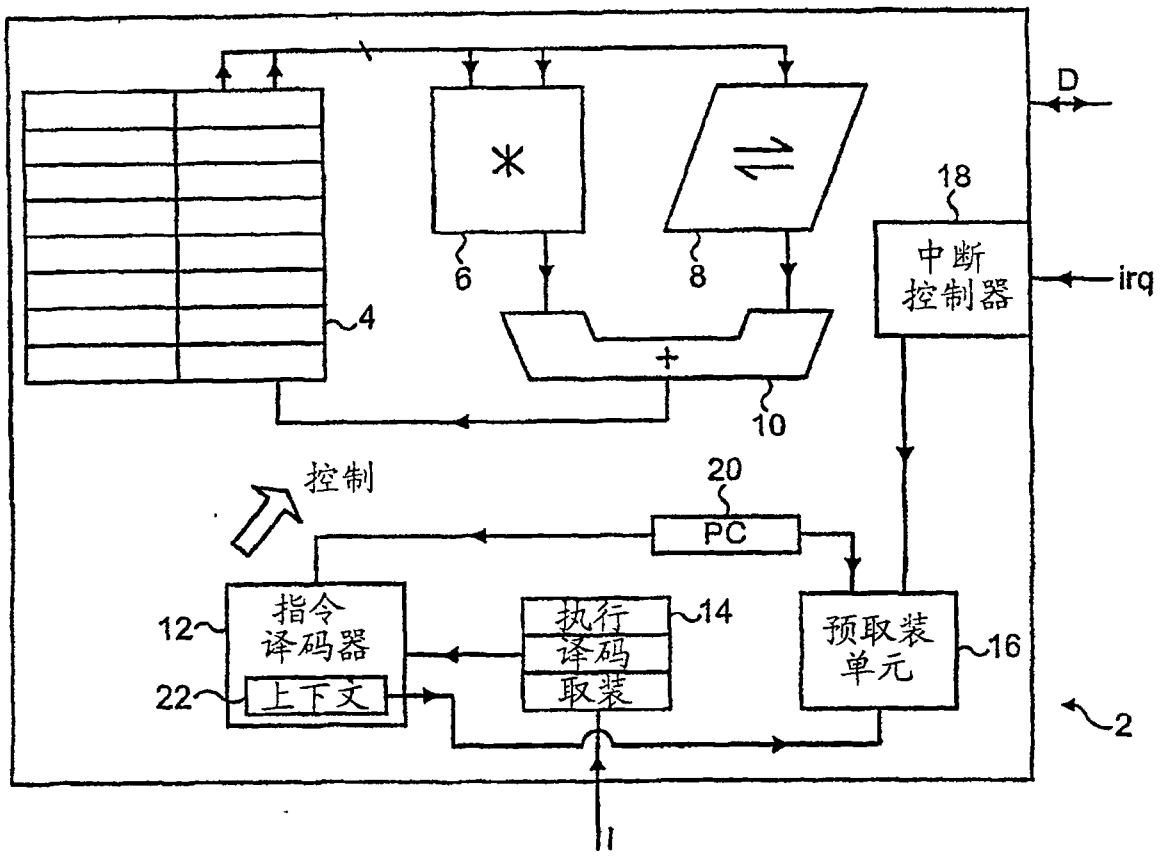

D
18
中断
控制器
irq
6
8
4
10
控制
20
PC
指令
译码器
12
22
上下文
执行
译码
取装
14
预取装
单元
16
2
I

图 1

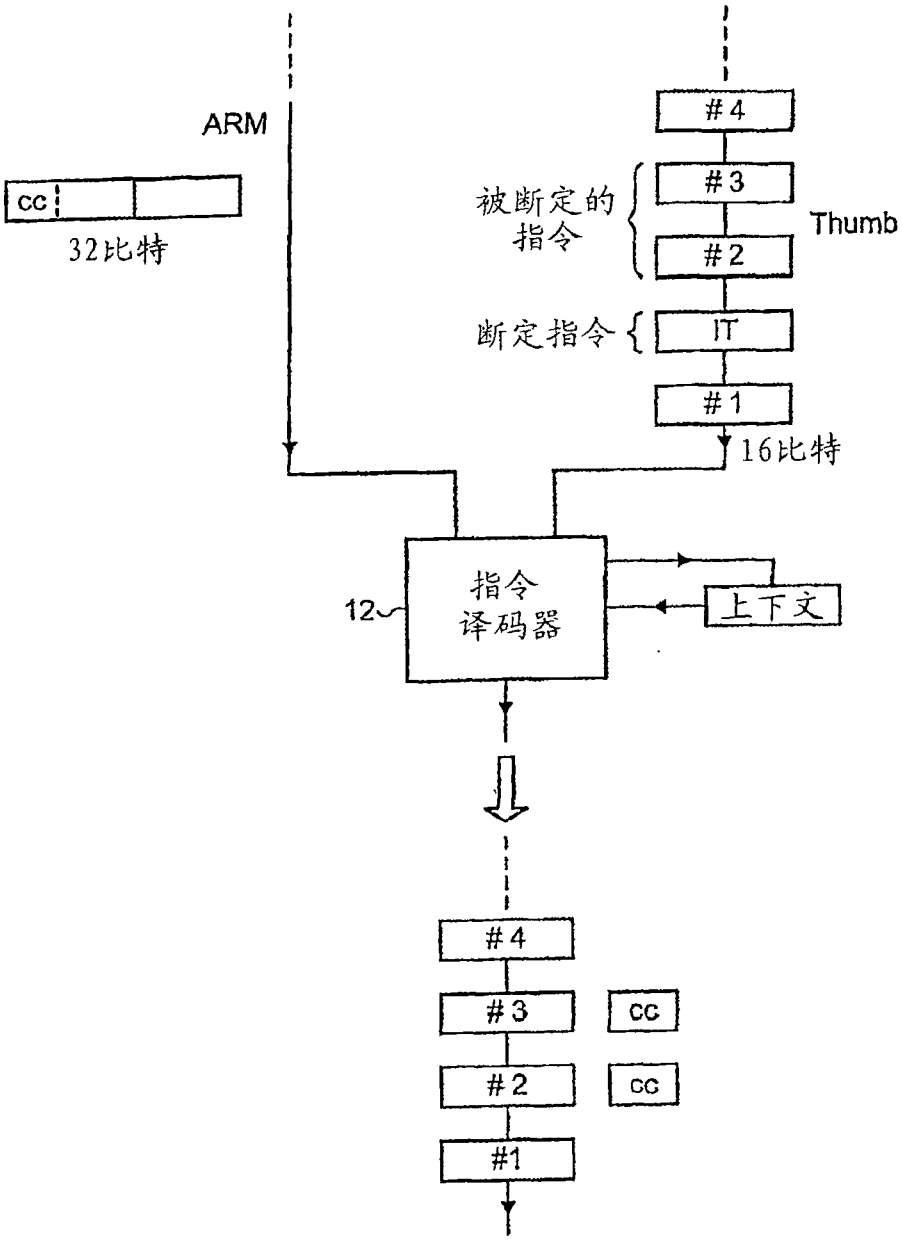

ARM
cc
32比特
#4
被断定的指令
#3
#2
Thumb
断定指令
IT
#1
16比特
12
指令译码器
上下文
#4
#3
cc
#2
cc
#1

图 2

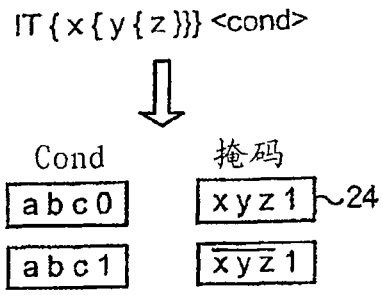

IT{x{y{z}}} <cond>
Cond
掩码
abc0
xyz1
24
abc1
xyz1

图 3

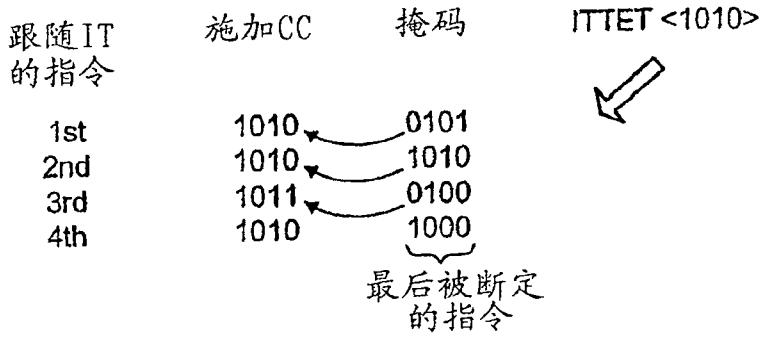

跟随IT的指令
施加CC
掩码
ITTET <1010>
1st
2nd
3rd
4th
1010
1010
1011
1010
0101
1010
0100
1000
最后被断定的指令

图 4

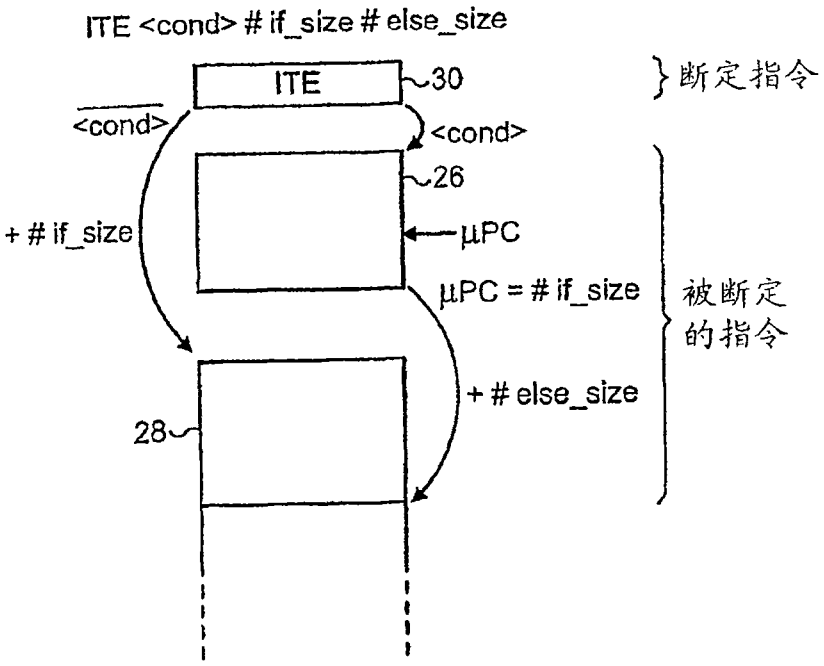

ITE <cond> # if_size # else_size
ITE
30
断定指令
<cond>
<cond>
26
+ # if_size
μPC
μPC = # if_size
被断定的指令
+ # else_size
28

图 5

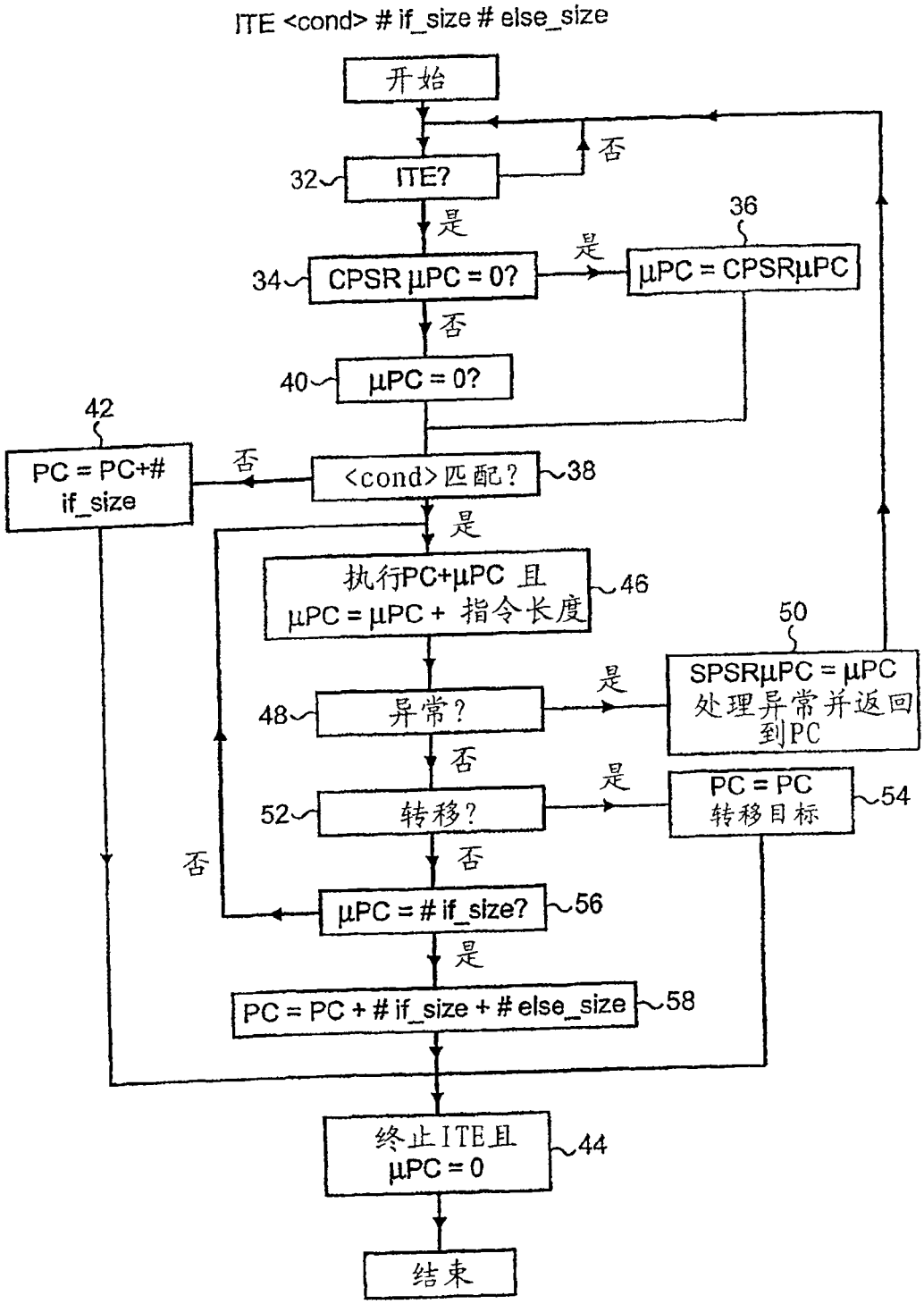

ITE <cond> # if_size # else_size
开始
32
ITE?
否
是
34
CPSR μPC = 0?
是
36
μPC = CPSRμPC
否
40
μPC = 0?
42
PC = PC+# if_size
否
<cond>匹配?
38
是
执行PC+μPC 且 μPC = μPC + 指令长度
46
50
SPSRμPC = μPC 处理异常并返回到PC
48
异常?
是
否
52
转移?
是
PC = PC 转移目标
54
否
否
μPC = # if_size?
56
是
PC = PC + # if_size + # else_size
58
终止ITE且 μPC = 0
44
结束

图 6

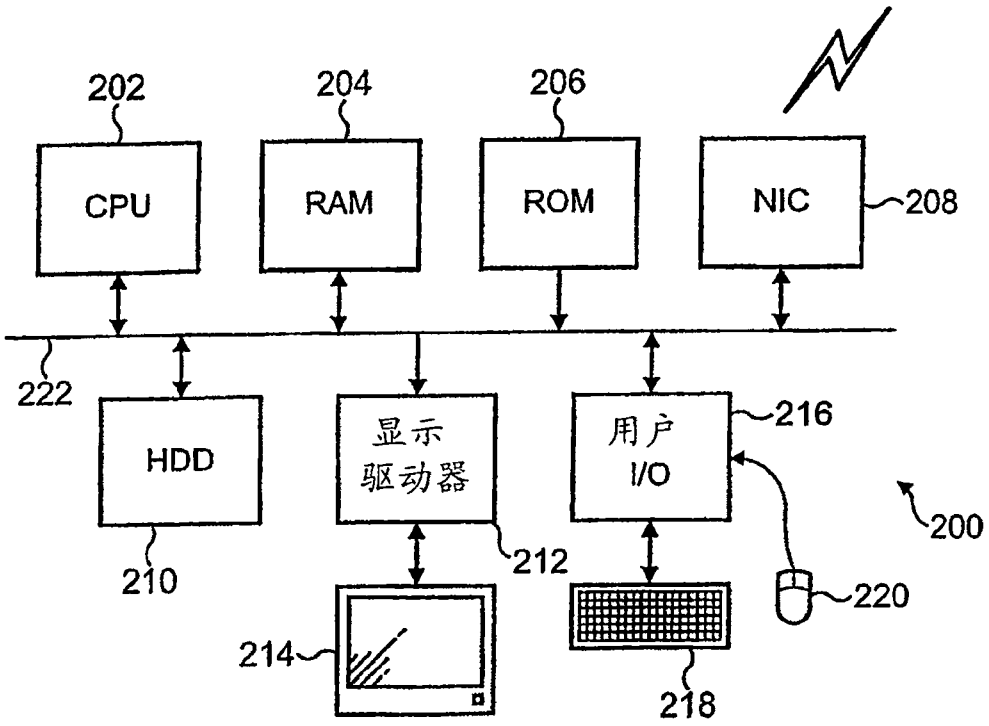

202
204
206
CPU
RAM
ROM
NIC
208
222
HDD
显示
驱动器
用户
I/O
216
200
210
212
220
214
218

图 7