[19] 中华人民共和国国家知识产权局

[51] Int. Cl[7]
G06F 9/38

[12] 发明专利说明书

[21] ZL 专利号 96100643.9

[43] 授权公告日 2003 年 5 月 28 日

[11] 授权公告号 CN 1109966C

[22] 申请日 1996.1.12 [21] 申请号 96100643.9

[30] 优先权

[32] 1995. 1.24 [33] US [31] 377563

[71] 专利权人 国际商业机器公司

地址 美国纽约

[72] 发明人 K·艾西格卢 G·M·西伯曼

审查员 陈 炜

[74] 专利代理机构 中国专利代理(香港)有限公司

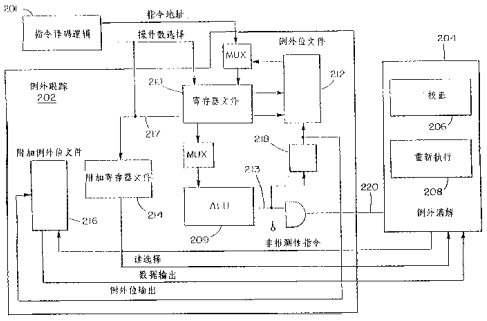
代理人 王 勇 邹光新

权利要求书 5 页 说明书 24 页 附图 4 页

[54] 发明名称 用于在并行处理中处理推测性指令例外的方法和装置

[57] 摘要

通过跟踪推测性例外(200)用于随后在例外消解过程(204)中进行处理，其中包括指向这些推测性指令的地址，并且通过校正(206)造成例外的原因并且重新执行(208)已知是在选定路径中的指令来消除(204)这些例外，CPU 的额外开销达到最小化。 跟踪推测性指令包括采用例外位的两个步骤，该例外位响应例外条件(213)被置位。



ISSN1008-4274

1. 一种在并行处理顺序代码时同时处理推测性例外的方法，包括以下步骤:

推测性地执行一个或多个指令;

跟踪由推测性指令产生的推测性例外;

随后通过校正例外条件和只重新执行该推测性指令和任何依赖于该推测性指令的推测性指令，来消除推测性例外。

2. 根据权利要求 1 所述的方法，其特征在于跟踪步骤包括跟踪原始推测性例外以及辅助推测性例外，后者在推测性指令试图采用相关的例外位已被置位的一个值为作操作数时发生，

3. 根据权利要求 1 所述的方法，其特征在于跟踪推测性例外的步骤包括以下步骤:

(1) 跟踪原始推测性例外，包括:

①响应所述例外条件，在与寄存器文件中第一目标寄存器相关的例外位文件中设置例外位;

②设置第一目标寄存器指向推测性指令的地址;

③在附加寄存器文件的附加寄存器中存储推测性指令的至少一个操作数（如果有的话），并且在附加例外位文件中存储例外位;

(2) 当推测性指令试图采用例外位已被设置的一个寄存器时，跟踪辅助推测性例外，包括以下步骤:

①响应所述例外条件，在与寄存器文件中第二目标寄存器相关的例外位文件中设置例外位;

②设置第二目标寄存器推向推测性指令的地址;

③将附加寄存器文件的附加寄存器的内容设置为推测性指令中操作数的寄存器序号，而不是寄存器的内容，并且设置附加例外位文件的例外位，以允许反向跟踪造成原始推测性例外的推测性指令，以达到对其消解的目的。

4. 根据权利要求 1 所述的方法，其特征在于执行该方法的初始条件

包括将所有指令都标记为非推测性的，但是只将移到条件分支上面的指令标为推测性的，所述条件分支确定该指令是否将在顺序码的选用路径中执行。

5. 在并行处理连续码的同时处理推测性例外的方法，包括以下步骤：

跟踪推测性例外，包括：

检测造成例外条件的推测性指令的发生；

响应例外条件，存储推测性指令的地址，同时存储例外位；

响应例外条件，存储推测性指令的至少一个操作数（如果有的话），并且存储相关的例外位；

当非推测性指令采用相关例外位已被设置的一个操作数，如所述推测性指令的目标寄存器，则通过校正所述的例外条件，以及仅重新执行由目标寄存器所指向的采用存储的至少一个操作数的推测性指令和任何与由目标寄存器所指向的采用所存储的至少一个操作数的推测性指令相关的推测性指令，来消除推测性例外。

6. 根据权利要求 5 所述的方法，其特征在于跟踪步骤包括跟踪原始推测性例外以及辅助推测性例外，后者在推测性指令试图采用相关的例外位已被置位的一个值作为操作数时发生。

7. 根据权利要求 5 所述的方法，其特征在于跟踪步骤包括以下步骤：

(1) 跟踪原始推测性例外包括：

①响应所述例外条件在与寄存器文件中第一目标寄存器相关的例外位文件中设置例外位；

②设置第一目标寄存器以指向推测性指令的地址；以及

③在附加寄存器文件的附加寄存器中存储推测性指令的至少一个操作数（如果有的话），在附加例外位文件中存储例外位；

(2) 当推测性指令试图采用例外位已被设置的一个寄存器时，跟踪辅助推测性例外，包括以下步骤：

①响应所述例外条件在与寄存器文件中第二目标寄存器相关的例外位文件中设置例外位；

②设置第二目标寄存器以指向推测性指令的地址；以及

③将附加寄存器文件的附加寄存器的内容设置为推测性指令中操作数的寄存器序号，而不是寄存器的内容，并且设置附加例外位文件的例外位，以允许反向跟踪造成原始推测性例外的推测性指令，以达到对其消解的目的。

8. 根据权利要求 5 所述的方法，其特征在于执行该方法的初始条件包括将所有指令都标记为非推测性的，但是只将移到条件分支上面的指令标为推测性的，所述条件分支确定该指令是否将在顺序码的选用路径中执行。

9. 一种在并行处理顺序代码时同时处理推测性例外的设备，包括：

提供造成例外条件的一个或多个推测性指令的装置；

响应所述例外条件跟踪由推测性指令产生的推测性例外的装置；

响应非推测性指令通过校正所述例外情况和只重新执行所述推测性指令来消除所述推测性例外的装置。

10. 根据权利要求 9 所述的设备，其特征在于所述跟踪装置包括跟踪原始推测性例外以及辅助推测性例外的装置，辅助推测例外在推测性指令试图采用相关的例外位已被置位的一个值作为操作数时发生。

11. 根据权利要求 9 所述的设备，其特征在于跟踪推测性例外的装置包括：

(1) 跟踪原始推测性例外的装置，包括：

①响应所述例外条件在与寄存器文件中第一目标寄存器相关的例外位文件中设置例外位的装置；

②设置第一目标寄存器以指向推测性指令的地址的装置；

③在附加寄存器文件的附加寄存器中存储推测性指令的至少一个操作数（如果有的话），以及在附加例外位文件中存储例外位的装置；

(2) 当推测性指令试图采用例外位已被设置的一个寄存器时，跟踪辅助推测性例外的装置，包括：

①响应所述例外条件在与寄存器文件中第二目标寄存器相关的例外位文件中设置例外位的装置；

②设置第二目标寄存器以指向推测性指令的地址的装置；

③将附加寄存器文件的附加寄存器的内容设置为推测性指令中操作数的寄存器序号，而不是寄存器的内容，并且设置附加例外位文件的例外位，以允许反向跟踪造成原始推测性例外的推测性指令，以达到对其消解的目的的装置。

12. 根据权利要求 9 所述的设备，其特征在于运行该设备的初始条件包括将所有指令都标记为非推测性的，但是只将移到条件分支上面的指令标为推测性的，所述条件分支确定该指令是否将在顺序码的选用路径中执行。

13. 一种在并行处理顺序码的同时处理推测性例外的设备，包括：

跟踪推测性例外的装置，包括：

检测造成例外条件的推测性指令的装置；

响应所述例外条件存储所述推测性指令的地址和例外位的装置；

响应所述例外条件存储所述推测性指令的一个或多个操作数并且存储相关的例外位的装置；

当非推测性指令采用相关例外位已被设置的一个操作数时，通过校正所述的例外条件，以及只重新执行采用所存储的至少一个操作数的所述推测性指令和任何与采用所存储的至少一个操作数的所述推测性指令相关的推测性指令，来消除推测性例外的装置。

14. 根据权利要求 13 所述的设备，其特征在于跟踪装置包括跟踪原始推测性例外以及辅助推测性例外的装置，辅助推测性例外在推测性指令试图采用相关的例外位已被置位的一个值为作操作数时发生。

15. 根据权利要求 13 所述的设备，其特征在于所述跟踪装置包括：

(1) 跟踪原始推测性例外的装置，包括：

①响应所述例外条件在与寄存器文件中第一目标寄存器相关的例外位文件中设置例外位的装置；

②设置第一目标寄存器以指向推测性指令的地址的装置；

③在附加寄存器文件的附加寄存器中存储推测性指令的至少一个操作数（如果有的话），以及在附加例外位文件中存储例外位的装置；

(2) 当推测性指令试图采用例外位已被设置的一个寄存器时，跟踪辅

助推测性例外的装置，包括:

①响应所述例外条件在与寄存器文件中第二目标寄存器相关的例外位文件中设置例外位的装置;

②设置第二目标寄存器以指向推测性指令的地址的装置;

③一种装置，用于将附加寄存器文件的附加寄存器的内容设置为推测性指令中操作数的寄存器序号，而不是寄存器的内容，并且设置附加例外位文件的例外位，以允许反向跟踪造成原始推测性例外的推测性指令，以达到对其消解的目的。

16. 根据权利要求 13 所述的设备，其特征在于运行该设备的初始条件包括将所有指令都标记为非推测性的，但是只将移到条件分支上面的指令标记为推测性的，所述条件分支确定该指令是否将中顺序码的选用路径中执行。

17. 根据权利要求 13 所述的设备，其特征在于所述顺序码包括按超长指令字格式安排的 RISC 指令。

用于在并行处理中处理推测性例外的方法和装置

本发明相关于计算机程序的并行处理，详细地说相关于推测性（speculative）指令的处理.

改进CPU运行的一种方法是改进循环时间．而改进循环时间的一个障碍则是分支指令，分支指令通常必须等待某些指令的结果得知之后才能确定采用哪条路径，即或者是“选择（taken）”（目标）路径，或者是“非选择”（顺序的）路径．选择路径包括一系列指令，这些指令可能包括分支，其中一些是选择分支，而另一些是非选择分支．某些通过流水线技术和/或采用多功能单元支持并行性的结构，允许有条件的执行指令，以增加应用程序中可用的并行性数量．这些“推测性指令”出现在条件还没有确定的条件分支指令之后，因此，可能在或可能不在由顺序程序执行所选定的实际路径中（即所谓的选择路径）．CPU猜测分支将采用哪条路径，一旦做出猜测，CPU即开始执行该路径．如果猜测正确，就不会有延迟，并以全速继续执行．如果猜测不正确，CPU就开始执行应选择的路径，执行中的延迟并不比不进行猜测更坏．由于它们的执行可能并不为程序所需要，因此直到能够确定它们是否在选择路径中之前，并不需要确定推测性指令的结果．非推测性指令处于选择路径中．

推测性指令的调度被认为是在全局范围内移动指令超出基本程序块范围之外的一个重要手段．

和非推测性指令的情况一样，在推测性指令执行过程中也可能出现诸如页面出错、算术溢出和无效寻址之类的例外．但是在后一种情况

下，在处理出现的每一种例外时，要求产生尽可能少的额外开销，因为推测性指令可能被证明是处在选择路径之外。另一方面，如果产生例外的推测性指令被发现是处在选择路径之中，需要适当地处理这些例外，并且仅仅重新执行与它们的结果相关的那些指令。进一步说，对例外自身是“致命的”情况来说（例如程序执行失败），CPU和存储器的内容应该是准确的。例如，如果储存的处理状态相关于一系列程序执行模式，（在该程序执行中一个指令在下一指令开始之前结束），那么中断是准确的，因此CPU和存储器的内容必须与同一程序串行执行所达到的状态是相同的。

1990年5月第17届计算机结构国际研讨会（西雅图）会刊第344至354页刊登的M.D.Smith，M.S.Lam和M.A.Horowitz的题为“Boosting Beyond Static Scheduling in a Superscalar Processor（提升到超级标量处理器的静态调度之外）"的论文中用提出的结构支持猜测性指令（称为“提升（boosted）”指令），通过采用寄存器文件的“阴影”复制和存储器存储缓冲来缓冲它们的副作用（对当前的推测性指令可能相关的每个条件分支需要一个这种阴影复制）。在该论文中没有明确指出处理例外的方案，但是提到它们仅仅在“……该提升指令试图执行”时出现。此时阴影结构无效，所有的提升指令现在作为非推测性指令被重新执行。最后，导致例外情况的指令将被重新执行，并在此时处理该例外。对于跟踪需要重新执行的提升指令的实际硬件以及存在的与提升指令相关的例外都没作任何描述。此外，重新执行所有提升指令，而不管是哪一个产生的例外，这显然是没有效率的。例如，考虑大量指令被提升并且在最后一条提升指令中出现例外的情况。M.D.Smith等人的方案是对所有的指令重新执行。即使重新执行一条单一指令就可能足够了。

第二种解决方案是采用软件结构来处理有限数量的推测性指令的特殊情况，即仅仅只是“全局”地址（通过采用指向已知的“全局”表格的基地址寄存器来标识）、“类似”地址（相同的基地址寄存器以及对一个“关闭”非推测性加载的类似位移）、以及“NIL指针”地址（可以证明包括零值，它对存储器中的第一页面寻址）的推测性装入指令。（参见D.Bernstein, M.Rodeh and M.Sagiv,"Proving Safety of Speculative Load Instructions at Compile Time（在编译时间证明推测性装入指令的安全性）” in Lecture Notes in Computer Science, Vol. 582, B. Krieg-Brueckner(Ed.), Proc. 4th Eueopean Symposium on Programming, Rennes, France, Feb. 1992, Springer-Verlag）。Bernstein等人提出的方案有赖于这些特殊情况，它通过对目标码或源代码的分析，并在连接程序和操作系统的支持下避免在推测性指令不在选定路径上的这些情况下的例外，来产生推测性装入。

在这个方案中，有三个问题没有阐述，即在选定路径中的推测性指令产生例外的可能性、对由于在连接程序中所引入的变化可能在顺序程序中产生的例外的（危险）屏蔽、以及引入可能不会出现在原始程序中的一个例外的可能性。在所有这三种情况下，从程序员的眼光看，带有推测性指令的程序的行为与原始（顺序）代码的行为是不一样的。

为了说明这三种情况，考虑下述指令序列:

```
# 1     LOAD        R3, 0 (R5)
# 2     LOAD        R4, 8 (R5)
# 3     COMPARE     R4, 0
# 4     BEQ         OUT
# 5     COMPARE     R3, 1000
```

```
# 6      BEQ          OUT
# 7      LOAD         R5, 16 (R4)
OUT :
```

根据Bernstein等人的解决方案，指令#7可以很安全地移到指令#3的上面，并且被转变为一个推测性加载。在第一种情况下，并且假定程序中有一个错误，如程序员在指令#2中指定位移为8而不是4，通过将16加到R4的（非零）内容中构成的地址将导致指向一个无效地址。执行指令#7可能产生一个例外----不管该指令是否被移动，但是它的移动会导致产生一个不明确的例外（在两种情况下机器的状态可能是不同的）。

更严重的是第二种情况，此时序列代码中存在的一个例外----代表程序中的一个错误----通过在数据段边界加入哑附加页面而屏蔽（如Bernstein等人所建议的那样），来防止“类似”加载的边界交叉。在上面的代码段中，假定R3由指令#1装入了某些不等于1000的值（如2000）。此外，假定指令#7的有效地址是在程序的数据段之外（例如位移可能是0而不是16）。由于连接程序所插入的哑页面，装入指令将不会产生可能造成错误结果的例外（不管它是否被移动），对该错误结果的原因，程序员 也不是显而易见的 .

第三种情况相关于引入一个例外，该例外由推测性装入所造成，它在原始代码中是不存在的。为了说明这个情况，现在假定R3由指令#1装入值1000. 另外，假定对R4的（非零）内容加上16所构成的地址是一个无效地址. 由于指令#5中的比较产生一个“相等”结果，因此指令#6的分支被采用. 因此，指令#7不会在原始代码中执行. 但是，如果它被移到指令#3的上面，那么它不仅要被执行，还会产生例外.

本发明的目的，包括使推测性指令引起的例外而导致的CPU额外开销最小化，以及有效地处理出现在选定路径中的推测性指令的例外。根据本发明的教导，推测性指令的例外只有并且仅仅只有当它出现在原始顺序程序中时才被处理。

本发明的公开包括处理由推测性指令产生的例外的硬件系统。推测性指令有一个附加位，称为推测位，该位是被置位的。推测性指令是被移到条件转移上面的一道指令，该条件转移确定该推测性指令是否在选定路径中。非推测性指令使推测位被复位。

本发明是在VLIW（超大型指令字）处理器结构的环境内进行描述的，但是类似的处理方法也可以用于超级标量和/或流水线式结构中。在VLIW的上下文环境中，一个单一指令包括若干条指令，所有这些指令是在相同时间发出的，并且其中一些可能是推测性的，因此，推测性操作变成推测性指令，但是其原理是一样的。因此，为了一般性起见，我们采用推测性指令的说法。这个机理是由并行化的编译器所支持的，该编译器跟踪推测性指令对寄存器的应用，以使得这些推测性指令在受到例外情况影响时能重复执行。本发明技术显著地降低了推测指令产生的例外情况导致的额外开销。

本发明一方面提出一种在并行处理顺序代码时同时处理推测性例外的方法和设备，用于：

推测性地执行一个或多个指令；

跟踪又推测性指令产生的推测性例外：

通过校正例外条件和只重新执行该推测性指令和任何与该推测性指令相关的推测性指令，来消除推测性例外。

本发明的另一方面是一种在并行处理连续码的同时处理推测性例外的方法和设备，用于：

检测造成例外条件的推测性指令；

在发生例外条件的情况下，连同一个例外位一起存储推测性指令的地址；

在发生例外条件的情况下，存储推测性指令的至少一个操作数，并存储相关的例外位；

当非推测性指令采用相关例外位被置位的操作数，如推测性指令的目标寄存器时，则通过校正例外条件，并仅仅重新执行由目标寄存器指向的采用所存储的至少一个操作数的推测性指令和任何依赖于由目标寄存指向的采用所存储的至少一个操作数的推测性指令的推测性指令，来消除推测性例外。

根据本发明，CPU额外开销通过以下方式来达到最小化：1）跟踪推测性例外，用于随后在消解（resolution）过程中处理，包括指向这些推测性指令的地址；2）通过校正例外的起因，并且重复执行已经得知是在选定路径中的指令来消解这些例外，跟踪推测性指令有两个组成部分（该组成部分采用一个例外位）。第一个组成部分是跟踪原始的推测性例外，当推测性指令----其操作数没有任何例外位组----遇到例外情况时发生的推测性例外。简而言之，这个跟踪包括：①设置寄存器文件中的目标寄存器，它指向推测性指令的地址；②在例外位文

件中设置一个例外位；③在一个附加寄存器中存储推测性指令的操作数；④在一个附加例外位文件中存贮所述例外位。

当推测性指令试图采用例外位已经按如上所述被设置的寄存器时，第二个组成部分被采用。在这种情况下，产生一个辅助推测性例外，并且例外跟踪包括：①设置目标寄存器指向如上所述的推测性指令地址；②设置目标寄存器的例外位；③将附加寄存器的内容设置为推测性指令中操作数的寄存器号，而不是寄存器的内容（并且设置附加例外文件的例外位）。这允许反向跟踪到造成原始推测性例外的推测性指令，以达到消解该例外的目的。例外跟踪也包括排除在选定路径之外的推测性例外的信息。当不在选定路径中的非推测性指令采用例外位已被设置的一个寄存器中的操作数时，触发推测性例外消解。推测性例外消解包括校正产生例外的例外条件，并且重复执行与产生推测性例外的指令的结果相关的指令。

这一解决方案的优点是只有那些与1）造成例外的指令和2）位于选定路径中的指令相关的推测性指令才被重复执行，而不是象现有技术中那样重复执行所有的推测性指令。这个解决方案节省了CPU的处理时间。

根据下面结合附图所作的描述将会一一了解本发明的其它目的、特征和优点。

图1a是VLIW编译器的方框图，该编译器有一个用于向VLIW数据处理器提供VLIW的输出端；

图1b示出了一行VLIW码的一部分；

图2是本发明的推测性指令处理机构的方框图；

图3的流程图示出了用于标识推测性指令的技术；

图4是根据本发明用于推测性例外跟踪的硬件方框图。

首先参看图1a和图1b，一个超长指令字（VLIW）编译器108运行时将用户程序104直接编译为VLIW 106，如它的名字的含义一样，VLIW 106是一个高度并行和水平的微代码（长度大概是500到2000位）。如图1b所示，VLIW 106包括若干精简指令集（或RISC）指令，也称为字段，每个控制VLIW数据处理器或机器102中的一个不同资源（例如算术逻辑单元（ALU）、分支单元）。因此在一个单一周期内，VLIW机器102可以执行许多RISC指令的等同指令。

每个RISC指令可以包括一个操作码字段、一个第一操作数字段、一个第二操作数字段以及一个目标寄存器序号字段。提供一个额外的一位字段----这里称之为推测位字段（或者更简单地称之为推测位）来区分推测性RISC指令和非推测性RISC指令，对此下面将作更详细的描述。

在操作过程中，并行化的编译器108采用常规的顺序汇编码（或来自高级语言编译器的中间码）作为输入，如果需要的话，将任何复杂的指令扩展为较简单的RISC指令，以提供顺序代码的RISC版本，并且采用高级编译技术（例如，一种适合的编译技术在T.Nakatani和K.Ebcioglu的论文“在基于压缩的并行编译器中采用前视窗”中做了描述，该论文发表在第23届微程序设计和微结构研讨会论文汇编，1990年11月，第57至68页，由IEEE计算机协会出版社出版）确定能够同时执行的RISC指令和分支的组109（在编码中它们可能是相隔很远的）。然后并行编译器108将每个这种组109放在VLIW机器102可以在一个单一周期中执行的VLIW106中。结果在不需要原代码变化的情况下，通过并行化编译器108中进行的再编译过程，用户程序104执行得更快。

值得注意的是，VLIW编译器108可以是VLIW机器102的一部分，也可以是一个单独的数据处理器。VLIW编译器108的结构和运行细节并不构成本发明的一部分，因此不作详细的描述。

为了处理由于推测性指令所产生的例外，VLIW机器102有若干寄存器，在图1a中一般性地示为寄存器103，所述寄存器有一个附加位（假定寄存器103通常是32位，那么该附加位是第33位。对此可参考K.Ebciogled "Some Design Ideas for a VLIW Architecture for Sequential--Natured Software," Proc. IFIPWG10.3 Working Conf. on Parallel Processing, N. Holland, 1988, pp.3--21）。

因此，在图1b中用于VLIW指令106的字段中的每个RISC指令有一个推测性版本和非推测性版本，也就是说，字段中的附加位（推测位）指示这个RISC指令是推测性还是非推测性的。

如果RISC指令的推测位被设置，那么在得知VLIW 106中的条件转移结果之前，该指令就在VLIW 102中执行，所述条件转移确定这个指令是否在原始顺序码中执行。另外，当寄存器103中的例外位被置位时，推测性指令已产生一个错误（例如溢出、从一个无效地址装入）。

处理推测性例外

图2是根据本发明的数据处理系统的方框图，CPU的额外开销通过以下步骤得到最下化：①跟踪推测性例外202，用于随后在例外消解过程204中进行处理，包括指向这些推测性指令的地址；②通过对引起例外的因素进行校正（206），并且重复执行已知是在选择路径中的指令（208）来消解这些例外。

跟踪推测性指令有采用例外位的两个组成部分，该例外位是响应例外条件213而设置的．例外条件可以以若干方式产生，包括从算术逻辑单元209的运算中产生。

首先是跟踪原始推测性例外，当推测性指令----其操作数的所有例外位都没有被设置----遇到例外条件时发生所述的原始推测性例外．简而言之，这一跟踪包括：①在寄存器文件210中对目标寄存器进行设置，以指向推测性指令的地址，该推测性指令是由多路复用器（MUX）提供的一个指令地址来给出的；②响应ALU209的例外条件213在例外文件212中设置与目标寄存器相关的一个例外位；③在附加寄存器文件214中的附加寄存器中存储推测性指令的操作数，并且在附加例外位文件216中存储例外位（对初始的推测性例外它没有被设置，而对辅助推测性例外它被设置，也就是说，它采用造成推测性例外的推测性指令的结果）．指令译码逻辑201的操作数选择线路217向寄存器文件210和附加寄存器文件214提供信号，方框218的逻辑提供原始的和辅助推测性例外之间的区别。

第二，当推测性指令试图采用已设置了例外位的寄存器时，如上面所述的原始推测性例外那样，这时产生辅助推测性例外，并且例外跟踪包括：①设置寄存器文件210中的目标寄存器指向如上所述的推测性指令地址；②响应例外条件213设置附加寄存器212的例外位；③通过寄存器选择线路217将附加寄存器文件214的附加寄存器的内容设置为推测性指令中操作数的寄存器序号，而不是寄存器的内容（并且设置附加例外位文件216的例外位），以允许反向跟踪产生原始推测性例外的推测性指令，达到对其消解204的目的。

当位于选择路径中的非推测性指令采用已设置了例外位的一个寄存器中的操作数时，推测性例外消解被触发．例外条件和非推测性指令

的存在向例外消解方框204产生一个例外信号220。推测性例外消解204包括响应附加寄存器的输出信号和附加例外位来校正204造成例外的例外条件，并且包括重新执行208与造成推测性例外的指令的结果相关的指令。

编译器支持

在图3 中并行编译器108能按如下所述利用推测性/非推测性位。首先，将用户程序104（图1）中的所有指令都标记为非推测性（302）。在并行化处理过程中，当指令被移到一条条件转移指令一该条件转移指令确定这个指令是否在原始码中执行一上面时，由编译器304将其标记为推测性。（如果一条指令没有明确地在条件分支的两条路径上执行，那么这个条件分支确定在连续码中该指令是否将被执行。）这就是以最简单的可能方式利用例外位机制所必需的一切。

如果要求在原始程序中产生的所有例外都在并行程序中被检测出来，那么并行编译器108（图1）必须保证每个推测性指令的结果最终直接或间接地被非推测性指令采用,必要时通过引入新的非推测性指令（通常这并非总是必须的，因为存储和条件转移不是推测性的）。

这里描述的发明也解决处理非必须的中断的问题。按照本发明所作的教导，如果并且仅仅如果在例外出现在原始顺序程序中时，例外在VLIW 106中被处理。

推测性例外硬件机制一跟踪

如前面所述，推测性例外与接收造成例外的指令的结果（称为目标）的寄存器相关。在这种情况下，“寄存器”一词的含义比通常所用的“通用”寄存器的意义更广，它指的是设置了例外位的任何寄存器。因此，如果编译器保证同一寄存器在同一时刻不会是一个以上推测性指令的目标，那么例外发生的记录可以与该寄存器相关。这就意

味着在寄存器的结果还没有被非推测性指令所使用之前，该寄存器不能被再次利用，或者因为它处于选择路径之外，因此是不需要的。

假定寄存器的大小是32位，并且有一个附加位（第33位）作为推测性例外的标志位。将寄存器扩展为较大（或较小）是在本发明范围之内的，即对n位的寄存器来说，n+1位起例外位的作用。

在本发明中，通过将造成例外的指令的地址保存在目标寄存器着中，并且设置其例外位来使推测性例外与目标寄存器产生关联。另一种方法是保存指令自身，可能是在译码之后进行。这会需要附加的存贮器来保持指令，但是如果指令是在选定路径中的话，它的执行将被加快。此外，这也可以避免超高速缓冲存贮器不中，因为包含该指令的线路不必进行第二次存取。

剩下的推测性指令处理包括两个组成部分：①例外跟踪，这全部在硬件中实施；②例外消解，主要由软件控制。

例外跟踪是在推测性例外出现时对其进行处理，并且对两种情况加以区分，第一种情况相关于原始推测性例外，第二种情况相关于推测性指令试图采用设置了例外位的一个值作为操作数，在此称之为辅助推测性例外。例外跟踪也负责排除有关信息，该信息与发现在选定路径之外的推测性例外相关。

当非推测性指令采用设置了例外位的一个寄存器作为操作数时，例外消解步骤被启动，我们说在这一点上设置了例外位的推测性例外已经实现。例外实现和消解是下一节的讨论主题，下一节主要是从软件支持方面来进行讨论。本节的剩余部分讨论对例外跟踪的硬件支持。

1）初始推测性指令跟踪

本节相关于产生规则例外的推测性指令，即该例外不是通过试图采用设置了例外位的寄存器值来产生的。如果在寄存器文件210中包括任

何指令操作数的话，那么它们的值被复制到额外寄存器文件214的相应的额外寄存器中，并且例外位被复位。如果一个额外寄存器没有被使用的话（相应的操作数不在寄存器中或者指令不需要它），那么它的例外位被复位，并且它的值没作限定。复制寄存器文件210的值是经由从寄存器文件端口到附加寄存器文件214的一条附加数据路径，直接从寄存器文件210进行的。

参看图4，对原始推测性例外作更详细的描述（发黑的圆圈是来自指令译码器201的输入，发黑的矩形是来自图2的例外消解204的输入/输出）。当推测性指令（它的操作数没有设置任何例外位）遇到例外条件时如溢出、页面出错等等，这种类型的例外发生。此时，例外线401被置位，使得或门402的输出变为真。由于造成例外的指令是推测性的，到与门404的下方的输入403为假，从而与门404中没有例外信号220输出，这意味着在这一时间没有发生对例外的直接处理。

为了保持对例外的跟踪，以便随后进行处理，如果需要的话，线路405将例外信号传输到例外位寄存器406“输入”端口，而406和417的“写选择”端口为造成例外的推测性指令选择对应于目标寄存器的位，正如由目标选择线407所确定的那样。在同一时间，线路405控制MUX 410，后者选择它的左输入端一指令地址线411一而不是结果线412。这使得指令地址411被存贮在目标寄存器文件417中，从而起到指向造成例外的指令的指针作用，以便在例外消解过程中使用。

注意，例外位寄存器406（一个宽度为一位的两个读出、一个写入文件）的“右输出”和“左输出”端口，各自按照右寄存器选择线408和左寄存器选择线409所作的选择为指令的右、左操作数提供例外位。由于假定操作数没有对它的例外位置位，这些输出都为假，而且因此线路415为假。将一个文件指定为“两个读出、一个写入”指的是这个

文件能够经由两个读出端口同时输出数据，并且也能够经由一个写入端口输入数据。

线路405也作为左额外寄存器文件413和左例外位文件414的写允许信号。对等同的右寄存器文件（没有示出），线路405起同样的作用。这里虽然是对每个指令假定最多两个操作数，但要扩展到更多的操作数是显而易见的。两个文件的“写选择”端口从目标选择线407接受输入，对造成推测性例外的指令来说，起到对存贮在寄存器文件417的目标寄存器中的信息的扩展作用。左额外寄存器文件413（一个1读出、1写入文件）的宽度由给定结构（例如32位）中操作数的长度确定，它存储由线路415（如上所述，415为假）控制的MUX 417c选定、并且由寄存器文件417的“左数据输出”端口产生的指令的左操作数。寄存器文件417是一个2读出、1写入文件，它的宽度由给定结构（例如32位）中的操作数所确定。这个输出是由左寄存器选择线409选择的。与此并行，线路415的状态被存储在左例外位文件414中（宽度为1位的1读出、1写入文件）。

某些操作数不是造成例外的推测性指令的必备部分。附加寄存器文件413的作用是存储足够多的与不是造成例外的推测性指令的必备部分的操作数有关的信息，以便当发现该指令在选定路径中时可以对其重新执行。换句话说，在例外消解过程中，作为指令一部分的操作数一包括在指令中的文字一可以从指令中抽取出来在例外消解过程中再次执行（下面将进行描述）。其它输入在寄存器文件417的底部馈入MUX 417a和417b，这些输入可能来自于指令中的文字（直接）字段。因此，一条指令可以有两个操作数，但其中只有一个（或者一个也没有）来自寄存器。另一个操作数可能是通过左右操作数选择线在指令译码器的控制下从指令中的某个字段中取得的。操作数的其它可

能的来源，包括数据处理器中的特殊用途寄存器，它与寄存器文件是分离的。

2）辅助推测性例外

当推测性指令试图采用寄存器文件417（它将406中相应的例外位置位）中的寄存器作为操作数时，就产生了“辅助推测性例外”。在下面的讨论中假定左操作数（至少）将其例外位置位。对应于推测性指令中的左操作数的例外位由左寄存器选择线409在例外位文件406中选择。这个例外位出现在例外位文件406的“左输出”端口处，并且馈入与门418，该与门也是由命名为“左操作数已采用”419的信号所控制的，当指令实际采用寄存器文件417的寄存器中的左操作数时，该信号为真。注意到在支持具有两个操作数的指令的结构中，某些指令可能只有一个或没有操作数。在这种情况下被假定为真的与门418的输出被馈入或门402，后者在线路405上产生一个真值，从而标明推测性例外。

在文件417的目标寄存器中保存指令地址以及在例外文件406中标记它的例外位，都与原始例外的情况下一样处理。但是在这种情况下，文件413、414的内容与原始推测性例外的情况下不一样。左附加寄存器文件413不是接收左操作数本身，而是接收打算保存该操作数的寄存器号。由于假定编译器保证每个目标寄存器至多有一个推测性例外，因此，当需要进行消解时，指定寄存器号能够允许反向跟踪造成原始推测性例外的指令。寄存器号是由左寄存器选择线409确定的，并且按照线路415（它有一个真值）的控制通过MUX 417c馈入。线路415也用于在左例外位文件414中设置例外位。记录寄存器号而不是它的内容，要求多路复用器417c位于两个附加寄存器文件的每一输入端口

处。多路复用器417c的输入来自目标寄存器文件（提供内容）以及指令译码逻辑（提供寄存器号）。

显然，指令地址包括它的位置可以被锁定并且被保存，包括在VLIW指令106中（图1）的位置，以便在推测性例外发生时它能复制到寄存器文件417中。一旦这个处理过程结束，对VLIW106的执行恢复正常。

3）排除推测性例外

由于在推测性指令执行过程中的一个例外，结果证明是在选定路径之外的推测性指令，也可能使得寄存器文件417的例外位仍然被置位。因此每次当通过非推测性指令或者不造成例外的推测性指令在寄存器中存入一个结果时，该寄存器的例外位被复位。通过保持对活动寄存器的跟踪，并行化编译器108（图1）保证推测性和非推测性值不被混合在一起，从而在用户程序104（图1）中不产生异常的例外。

注意采用这种技术可能存在以下问题。举例来说，原始（连续）程序可能采用没有被初始化的寄存器（由于错误的原因）。可以由推测性指令来对这个寄存器进行置位，这样它或者包括一个值，或者由于推测性例外它的例外位被置位。在任一种情况下，程序的行为都不会与原始版本相符合，为了防止这种现象，包含推测性值的寄存器的初始化校验，或者（至少推测位）显式复位可以插入在程序段的末尾，对所述的程序段来说，该寄存器是保持处于活动状态的（见下一节）。

考虑表1中所示的指令序列（主要是推测性的）。为了简化起见，指令是单独列出的（每行一条指令），但是在VLIW环境中，对每条VLIW指令，它们被分成几个组。

表1

100	LOAD	R4, 0 (R5)	S
101	LOAD	R6, 0 (R3)	S
102	LOAD	R1, 8 (R4)	S
103	LOAD	R2, 12 (R6)	S
104	ADD	R1, R2, R7	S
105	MULT	R1, R8, R9	S
106	ADD	R9, R9, R10	S
	.		
	:		
200	STORE	R7, 8 (R11)	N

表1包括一个指令样本序列，当指令为推测性时被标志为“S”，当它为非推测时标志为“N”（这可以由指令中代表推测位的一个值来表示）。LOAD采用右操作数中的地址来存取存储器，并且将该内容装入左（目标）寄存器（STORE 执行相反的功能）。ADD/MULT采用第一个和第二个（左和右）操作数作为输入，其结果被存储在第三（目标）寄存器中。

假定寄存器R5和R3包括不是驻留在存储器中的页面的地址，如2000和30000，并且它们在指令100和101中的应用会产生页面出错例外。其它相关的寄存器是R8和R11，假定这两个寄存器的例外位被复位（它们不是产生例外的推测性指令的目标），并且分别包括234和

10000（假定包括地址10000的页面驻留在存贮器中）。当执行过程到达指令200（一个非推测性指令）时，推测性例外成为事实。在例外被解决之前，相关寄存器的内容被列在表2中，表2中每一行示出了“常规”寄存器417以及相应的（“左”和“右”）附加寄存器413的内容，以及每一个的例外位406和414。I-nnn入口代表序列中指令nnn的地址。

表 2

寄存器	Regular+f	Extra_Left+f	Extra_Right+f	说明
R1	I-102+1	???+0	4+1	右操作数指向R4
R2	I-103+1	???+0	6+1	右操作数指向R6
R3	30000+0	???+0	???+0	设定
R4	I-100+1	???+0	2000+0	右操作数是(R5)
R5	2000+0	???+0	???+0	设定
R6	I-101+1	???+0	30000+0	右操作数是(R3)
R7	I-104+1	1+1	2+1	两个操作数表示例外(R1与R2)
R8	234+1	???+0	???+0	设定
R9	I-105+1	1+1	234+0	左操作数指向R1 右操作数指向(R8)
R10	I-106+1	9+1	9+1	两个操作数指向R9
R11	10000+0	???+0	???+0	设定

表2示出了常规和附加寄存器的内容和标志。中间三列对应于在文件417/406、413/414和413/414对右操作数的等同文件中的位置。状态反应了（常规）寄存器R1、R2、R4、R6、R7、R9、R10的推测性

例外。执行指令200将使R7上的例外得以实现，并且间接的使R1、R2、R4、R6上的例外得以实现。

推测性例外的实现和消解

当非推测性指令遇到例外位被置位的一个操作数（寄存器）时，就说设置例外位的推测性例外已实现了。控制被传送到表3中所示的推测性例外消解机制中，该机制试图1）消除例外的原因以及2）重新执行与该例外相关并且现在已经被确定是在选定路径中的任何推测性指令。消除例外的原因意味着让系统采取某些行动尽可能地校正任何造成推测性例外的因素。例如，如果例外是一页面出错，它是由对当前不驻留在存储器中的数据寻址而造成的，消除该原因可能造成将包含该数据的页面引入该存储器，并且更新表列等等。第二部分重新执行可能导致对指令的重复执行，该指令现在已经被得知是在选定路径中，这种重新执行多少要用到造成例外的操作结果。

表 3

```
resolve_exception: FUNCTION(register_index)  RECURSIVE RETURNS (register_value)
    /*  例外可能已被消解. 若是则跳过 */
    /*  处理并返回R [register_index ] 的内容 */
     IF     f (R [register_index] )  THEN

          /*  检查左操作数是否需消解. 若是, 则递归.    */
       IF     f (XL [register_index] )    THEN
              c(XL [register_index] ) = resolve_exception (c(XL [register_index] ) )
              f (XL [register_index] ) = 0
    /*    检查右操作数是否需消解. 若是, 则递归.    */
```

```
IF      f (XR [register_index] )    THEN
        c(XR [register_index] ) = resolve_exception (c(XR [register_index] ) )
        f (XR [register_index] ) = 0

      /*   操作数已准备好，在复位其第33位标志位时   */
      /*      执行指向并设置 R [register-index] 内容    */
      /*   在指令执行期间，动态处理例外，          */
      /*    就好象它们来自正常(非探测性)指令执行.  */
   EXECUTE (c (R [register_index] ),   /*   指令指针   */
            c (XL [register_index] ),  /*   左操作数寄存器  */
            c ( XR [register_index] )  /*   右操作数寄存器  */
  /*   f (R [register_index] ) = 0 是隐含的.

  /*   此时寄存器具有正确的内容.   */
     RETURN    (c ( R[register_index] ) )
END  resolve_exception.
```

表3示出了实施例外消解的一个递归函数的概要。假设R[i]是例外得以实现的寄存器。回想一下R[i]的内容指向应该被重新执行用以消解例外的（推测性）指令，并且对应于R[i]的附加寄存器XL[i]（左）和XR[i]（右）可以包括执行指令所需要的操作数。记号c（r）和f（r）被分别用于寄存器r的内容和例外位，其中r是常规（R）寄存器、附加左（XL）寄存器或者是附加右（XR）寄存器。因此，对非推测性指令中的每个操作数寄存器（其推测位已被设置）该register_exception函数被调用一次。

示例

再考虑表1中所示的指令序列和表2中给出的寄存器内容，并且考虑表4．执行指令200将引起表4中给定的调用序列（缩进表示递归的级别）其中vvv、www、xxx和yyy分别代表赋给R4（通过I-100），R1（通过I-102），R6（通过I-101）以及R2（通过I-103）的值。

表 4

```
Left _Op = resolve_exception (7)
        c (XL [7] ) = resolve_exception (1)
                c (XR [1] ) = resolve_exception (4)
                 EXECUTE  ( I- 100, ???, 2000)        /* f( R[4] )=0 是隐含的 */
            EXECUTE  ( I- 102, ???, vvv)         /* F( R[1] )=0 是隐含的 */

        c (XR [7] ) = resolve_exception (2)
                c (XR [2] )= resolve_exception (6)
                 EXECUTE  ( I- 101, ???, 30000)        /* f( R[6] )=0 是隐含的 */
              EXECUTE  ( I- 103, ???, xxx)          /* f( R[2] )=0 是隐含的 */

         EXECUTE  ( I- 104, www, yyy)          /* f( R[7] )=0 是隐含的 */
```

现在，当指令200完成后，如果某些非推测性指令采用R10作为它的右操作数，那么调用序列将如表5所示（其中zzz代表由指令I-105分配给R9的值）。

表 5

```
Right_Op = resolve_exception (10)
        c( XL[101]) = resolve_exception (9)
```

```
c( XL[9]) = resolve_exception (1)     /*  其内容自f(R[1]) = 0   */
        EXECUTE (I-105, www, 234 )    /*  f (R[9] = 0 是隐含的   */
c( XR[10]) = resolve_exception (9)    /*  其内容自f(R[9]) = 0  返回 */
EXECUTE (i-106, zzz, zzz )            /*  f (R[10] = 0 是隐含的   */
```

显然，需要硬件来支持对附加寄存器的内容和其标志位（部分示于图4）的读出和写入，以及通过推测性例外消解机制204或者通过在寄存器417中的非推测性指令写入来（隐含地）对寄存器417的例外位复位。

如果在推测性例外正被消解的过程中出现一个致命的例外，某些寄存器可能与它们在相同程序的顺序执行版本中的状态不一致。例如，在消解R7的推测性例外时试图重新执行指令102，如上所述，考虑一个致命的例外。在这种情况下，指令100（设置R4）已经完成，但是指令101（设置R6）等待被消解．因此，R6的内容与102就是造成致命例外的指令的事实不一致。不过由于R6的例外位仍然被置位，可以判别它包含具有未决推测性例外的一个指令的地址。此外，可以提供一列所有这种指令作为与致命例外相关的数据的一部分（用于诊断的目的），这只需要将例外位被置位的所有寄存器的内容简单地列出来。

编译器支持

在前面一节中已经提及，并行化编译器108（或者任何其它用于产生推测性指令的程序）必须保持对推测性指令所采用的寄存器413或417的跟踪，以便在程序中任何一点都没有一个以上的这种指令，将同一个寄存器313或417作为其结果的目标．换句话说，由推测性指令置位的寄存器413或417，在它的值被非推测性指令所采用之前，不应该

被再次使用或者是由于推测性指令在选定路径之外因而确定这个值不被需要。第一种情况，即非推测性指令采用由推测性指令设置的寄存器413或417的内容作为操作数，强制编译器108使寄存器413或417保持“活跃”，从而阻止了其它指令对它的设置。

为了解决第二种情况，并行化编译器108在推测路径的末尾引入一个哑指令，它采用由推测性指令置位并且不被非推测性指令用作操作数的所有寄存器413或417作为操作数。之所以将这些寄存器413、417保持“活跃”到能够确定它们是否在选定路径之外为止，原因是它们被用作推测性指令的操作数，从而它们可能包含一个推测性例外，当确定它们是选定路径的一部分时，可能需要消解该推测性例外。

采用上面的解决方案，如果寄存器413、417在原始程序中没有被初始化，那么在它被适当初始化之前，由推测性指令置位的寄存器413、417也可以被非推测性指令采用，这是有可能的。这可以造成一种新异的例外（如果寄存器413、417的例外位已经被设置的话），该例外实际上是没有对寄存器413、417的初始化的结果。为了避免这种情况，对寄存器413、417的推测位复位（并且任意地将其值设置为“非定义”）的一条指令，可以用于替代哑指令，该哑指令保持由推测性指令置位的寄存器413、417处于“活跃”。

另外要加以考虑的是推测性例外与对某个特定任务或任务段例外的允许/禁止之间的关系。简单的解决方案会忽略特定例外是否被允许或禁止的问题，直到该例外实现为止。此时，需要进行检验，看应该采取什么处理手段，就好象该例外是由非推测性指令所造成的那样。在大多数情况下，编译器能够保持对例外允许和禁止指令的跟踪，以便使得所产生的现象与顺序程序保持一致。在这种做法无法实现的情况下，编译器108会避免对指令序列产生推测性指令。总而言之，不管是

否被禁止，推测性指令都被记录下来．只有在消解时间才对被允许例外检查推测性例外．并行化编译器保证任何例外当它应该被禁止时都不会被采用．

虽然本发明是特别针对最佳实施例来进行描述的，但本领域的普通技术人员会理解，在不脱离本发明的范围和精神的前提下，可以做出形式和细节上的变化。

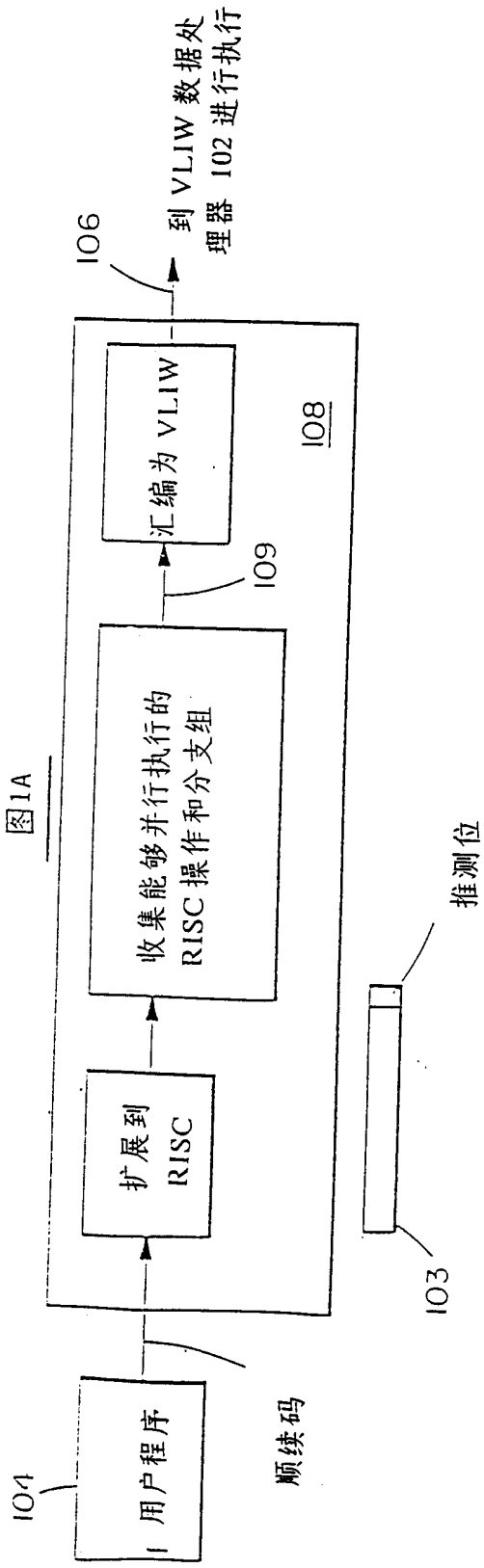


图1A
101
用户程序
顺续码
扩展到 RISC
收集能够并行执行的 RISC 操作和分支组
汇编为 VLIW
108
109
106
到 VLIW 数据处理器 102 进行执行
推测位
103

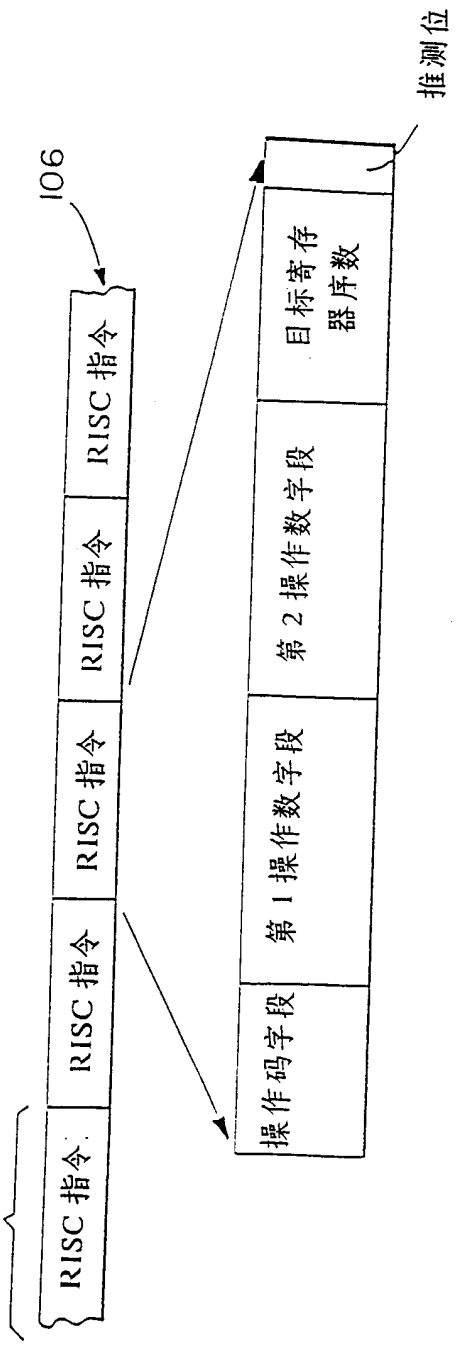


图1B
字段
RISC 指令
RISC 指令
RISC 指令
RISC 指令
RISC 指令
106
操作码字段
第 1 操作数字段
第 2 操作数字段
目标寄存器序数
推测位

图 2

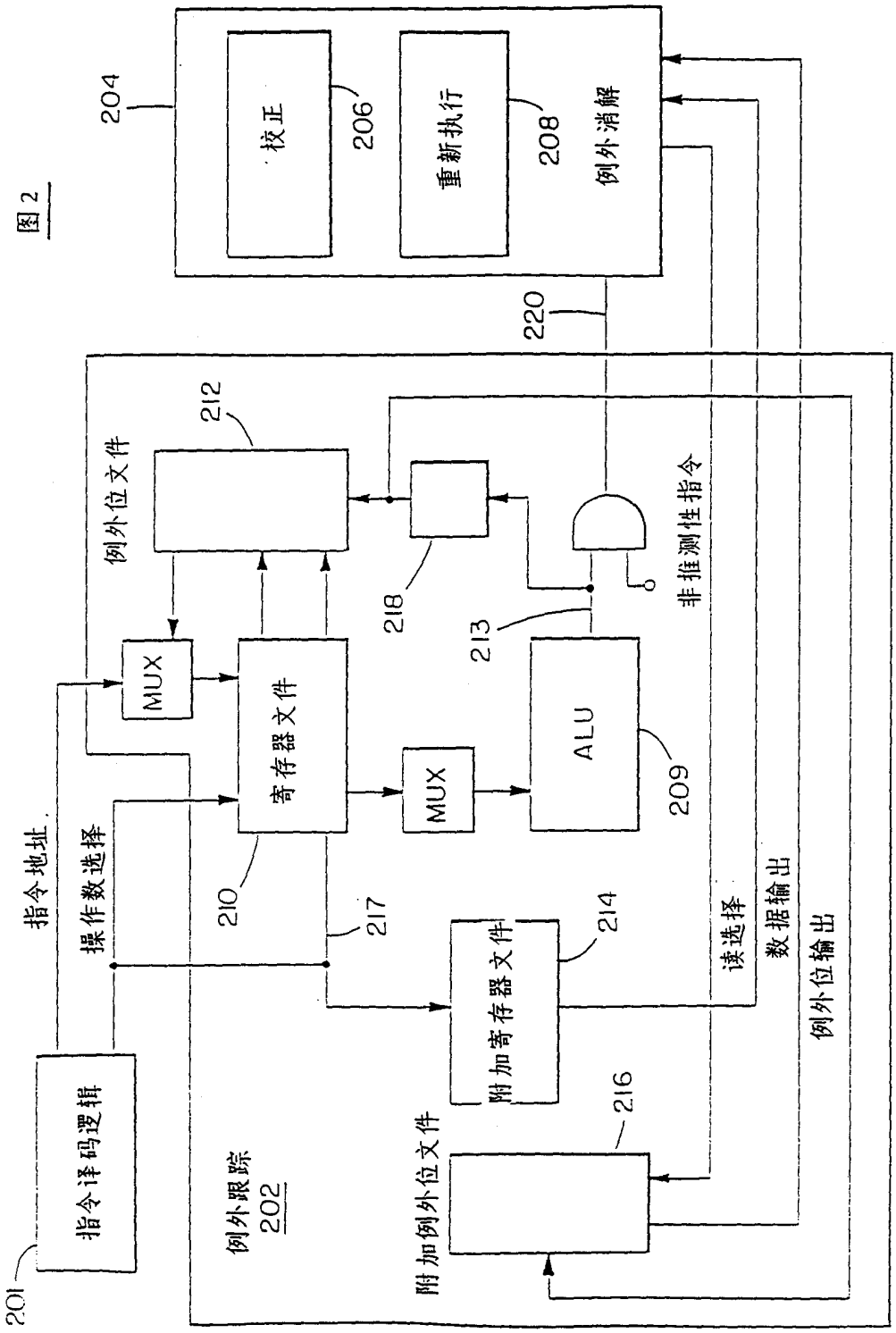
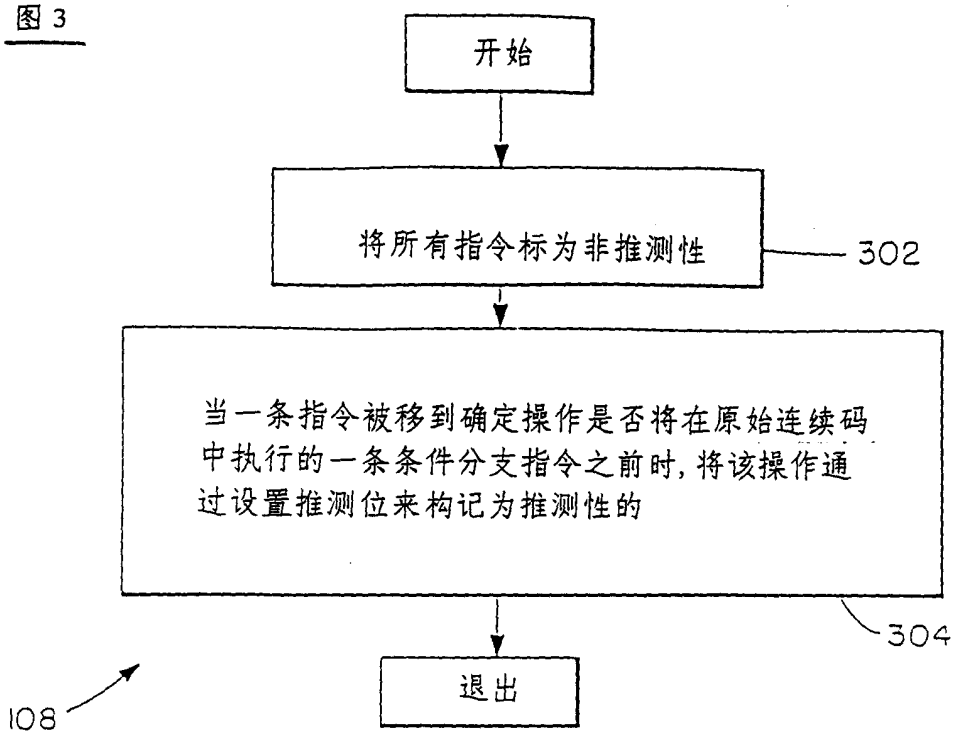

204
校正
206
重新执行
208
例外消解
220
例外位文件
212
MUX
寄存器文件
MUX
ALU
209
218
213
非推测性指令
指令地址
操作数选择
210
217
214
读选择
数据输出
例外位输出
附加寄存器文件
附加例外位文件
216
例外跟踪
202
指令译码逻辑
201

图 3


开始
将所有指令标为非推测性
302
当一条指令被移到确定操作是否将在原始连续码中执行的一条条件分支指令之前时，将该操作通过设置推测位来构记为推测性的
304
退出
108

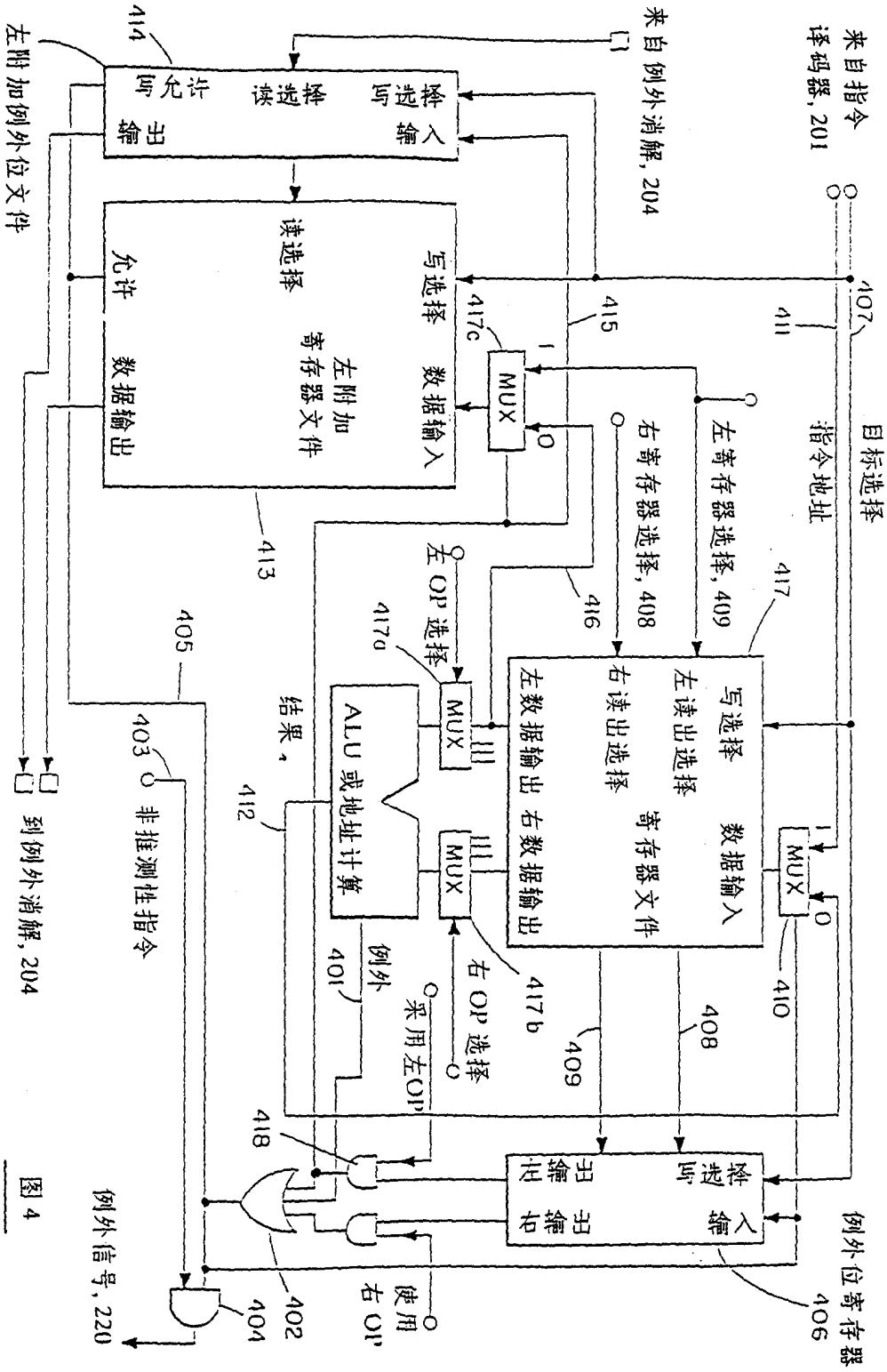

来自指令译码器, 201
来自例外消解, 204
左附加例外位文件
414
允许
写选择
读选择
数据输入
数据输出
左附加寄存器文件
413
415
417c
MUX
目标选择
指令地址
411
左寄存器选择, 409
右寄存器选择, 408
416
417
写选择
左读出选择
右读出选择
数据输入
寄存器文件
左数据输出
右数据输出
410
408
409
左 OP 选择
417a
417b
右 OP 选择
采用左OP
ALU 或地址计算
结果
412
405
例外
401
非推测性指令
403
到例外消解, 204
418
使用右 OP
402
404
例外信号, 220
例外位寄存器
406

图 4