



(12) 发明专利

(10) 授权公告号 CN 102707988 B

(45) 授权公告日 2015.09.09

(21) 申请号 201210103608.8

罗德尼 . E. 虎克

(22) 申请日 2012.04.09

(74) 专利代理机构 北京市柳沈律师事务所

11105

(30) 优先权数据

代理人 钱大勇

61/473,062 2011.04.07 US

(51) Int. Cl.

61/473,067 2011.04.07 US

G06F 9/455(2006.01)

61/473,069 2011.04.07 US

13/224,310 2011.09.01 US

(56) 对比文件

US 5854913 A, 1998.12.29,

61/537,473 2011.09.21 US

CN 101116053 A, 2008.01.30,

61/541,307 2011.09.30 US

US 2008/0104376 A1, 2008.05.01,

61/547,449 2011.10.14 US

CN 101689107 A, 2010.03.31,

61/555,023 2011.11.03 US

US 2008/0189519 A1, 2008.08.07,

13/333,520 2011.12.21 US

13/333,572 2011.12.21 US

审查员 章媛

13/333,631 2011.12.21 US

61/604,561 2012.02.29 US

13/413,300 2012.03.06 US

13/413,314 2012.03.06 US

(73) 专利权人 威盛电子股份有限公司

地址 中国台湾新北市

(72) 发明人 G. 葛兰 . 亨利 泰瑞 . 派克斯

权利要求书8页 说明书43页 附图22页

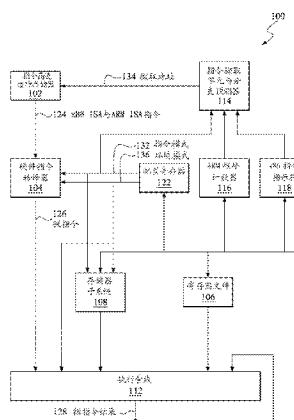
(54) 发明名称

微处理器及其操作方法

(57) 摘要

一种包含处理模式的微处理器，处理模式包含使用者模式与多个例外事件模式。执行单元在指定于程序指令的操作数上执行算数运算。第一存储元件组具有第一操作数子集，并供第一操作数子集给其耦接的执行单元。第二存储元件组关联于各处理模式，并具有第二操作数子集，且第二存储元件组无法直接提供第二操作数子集给执行单元。在自当前的模式进入新的模式时，逻辑单元将第一存储元件组中的第一操作数子集存储至关联于当前处理模式的第二存储元件组，并将关联于新处理模式的第二存储元件组中的第二操作数子集恢复至第一存储元件组。

CN 102707988 B



1. 一种微处理器,包含 :

多个处理模式,包含一使用者模式与多个例外事件模式;

至少一执行单元,用以在程序指令指定的操作数上执行多个算数运算;

一第一存储元件组,耦接于该执行单元,其中该第一存储元件组包含一第一操作数子集,并提供该第一操作数子集给该执行单元;

一第二存储元件组,关联于各处理模式,其中该第二存储元件组包含一第二操作数子集,其中该第二存储元件组无法直接提供该第二操作数子集给该执行单元;以及

一逻辑单元,其中,当从该些处理模式中的一当前处理模式进入一新处理模式时,该逻辑单元将该第一存储元件组中的该第一操作数子集存储至关联于该当前处理模式的第二存储元件组,并将关联于该新处理模式的该第二存储元件组中的该第二操作数子集恢复至该第一存储元件组。

2. 如权利要求 1 所述的微处理器,还包含 :

一第三存储元件组,耦接于该执行单元,其中该第三存储元件组包含一第三操作数子集,并提供该第三操作数子集至该执行单元;

其中该新处理模式是该些例外事件模式中的一第一例外事件模式;

一第四存储元件组,关联于该第一例外事件模式,其中该第四存储元件组包含一第四操作数子集,其中该第四存储元件组无法直接提供该第四操作数子集至该执行单元;以及

一第五存储元件组,全域性地关联于除了该第一例外事件模式之外的所有该些处理模式,其中该第五存储元件组包含一第五操作数子集,其中该第五存储元件组无法直接提供该第五操作数子集至该执行单元;

其中,当从该当前处理模式进入该新处理模式或该第一例外事件模式时,该逻辑单元额外将该第三存储单元组的该第三操作数子集存储至该第五存储单元,并将该第四存储元件组中的该第四操作数子集恢复至该第三存储元件组;

其中,当从该第一例外事件模式进入该些例外事件模式的一第二例外事件模式时,该逻辑单元将该第三存储单元组的该第四操作数子集存储至该第四存储单元,并将关联于该新处理模式的该第五存储元件组中的该第三操作数子集恢复至该第三存储元件组。

3. 如权利要求 2 所述的微处理器,

其中,该微处理器利用该第一存储元件组的一第一存储元件以保存 ARMISA 的一堆迭式指示符寄存器操作数,并利用该第一存储元件组的一第二存储元件以保存该执行单元执行该些算数运算的 ARM ISA 的一连结寄存器操作数;

其中,该第二存储元件组包含一第一存储元件以保存一 ARM ISA 堆迭式指示符寄存器操作数,以及一第二存储元件以保存用以关联处理模式的一 ARM ISA 连结寄存器操作数;

其中,该微处理器利用该第三存储元件组以保存该执行单元执行该些算数运算的 ARM ISA R8-R12 通用寄存器的操作数;

其中,该第四存储元件组包含多个用以保存 ARM ISA R8-R12 通用寄存器操作数的存储元件,以对应 ARM ISA FIQ 例外事件模式;

其中,该第五存储元件组包含多个用以保存 ARM ISA R8-R12 通用寄存器操作数的存储元件,以对应除了 ARM ISA FIQ 例外事件模式之外的全域性 ARM ISA 处理模式。

4. 如权利要求 1 所述的微处理器,其中该微处理器利用该第一存储元件组的一第一存

储元件以保存 ARM ISA 的一堆迭式指示符寄存器操作数,以及利用该第一存储元件组的第二存储元件以保存该执行单元执行该些算数运算的该 ARM ISA 的一连结寄存器操作数。

5. 如权利要求 1 所述的微处理器,其中该第一存储元件组包含多个硬件寄存器,其中该第二存储元件组包含一随机存取存储器 (RAM)。

6. 如权利要求 5 所述的微处理器,

其中该随机存取存储器可通过该微处理器的微码进行载入或写入;

其中该随机存取存储器不可通过 ISA 机器语言程序指令进行载入或写入。

7. 如权利要求 1 所述的微处理器,还包含:

一超纯量非循序执行管线,包含:

至少一执行单元;以及

一载入单元,耦接于该第一存储元件组,其中该第二存储元件组提供该第二操作数子集至该载入单元,其中该载入单元提供该第二操作数子集至该执行单元。

8. 如权利要求 1 所述的微处理器,其中该些例外事件模式包含 ARM ISA 例外事件模式。

9. 如权利要求 1 所述的微处理器,其中该逻辑单元包含该微处理器的微码。

10. 如权利要求 1 所述的微处理器,其中该逻辑单元包含一硬件组合逻辑单元。

11. 如权利要求 1 所述的微处理器,还包含:

一指令转译器,用以将 ARM ISA 机器语言的指令转译至多个微指令,其中至少一 ARM ISA 指令指示该微处理器自该当前处理模式进入至该新处理模式;以及

一执行管线,用以执行该些微指令以将该第一存储元件组的该第一操作数子集存储至关联于该当前处理模式的该第二存储元件组,以及恢复关联于该新处理模式的该第二存储元件组的该第二操作数子集至该第一存储元件组。

12. 如权利要求 11 所述的微处理器,其中该指令转译器还将 x86ISA 机器语言的指令转译为多个微指令,其中该些微指令是以不同于 x86ISA 指令集的指令编码方式进行编码,其中该执行管线还执行该些微指令以产生由 x86ISA 指令所定义的结果。

13. 一种操作一微处理器的方法,该微处理器包含多个处理模式,该些处理模式具有一使用者模式以及多个例外事件模式,其中该微处理器还包含至少一执行单元,该执行单元通过特定程序指令在操作数上执行多个算数运算,该方法包含:

当该微处理器在该些处理模式中的一当前处理模式运行时,自一第一存储元件组中提供一第一操作数子集至该执行单元以执行该些算数运算;

当自该当前处理模式进入该些处理模式的一新处理模式时,包含以下步骤:

将该第一存储元件组的该第一操作数子集存储至关联于该当前处理模式的一第二存储单元组;

将该关联于该新处理模式的一第三存储元件组的一第二操作数子集恢复至该第一存储元件组;以及

当该微处理器于该新处理模式中运行时,自该第一存储元件组提供该第二操作数子集至该执行单元以执行该些算数运算。

14. 如权利要求 13 所述的方法,还包含:

当该微处理器于该当前处理模式下运行时,自一第四存储元件组提供一第三操作数子集至该执行单元以执行该些算数运算;

其中当自该当前处理模式进入该新处理模式时,还包含 :

将该第四存储元件组的该第三操作数子集存储至关联于该新处理模式的一第五存储元件组;以及

将一第六存储元件组的一第四操作数子集恢复至该第四存储元件组,且该第六存储元件组全域性关联于除了该第一例外事件模式之外的该些处理模式;

当该微处理器于该新处理模式下运行时,自该第四存储元件组提供该第四操作数子集至该执行单元以执行该些算数运算;

当自该新处理模式进入该些处理模式的一第三处理模式时,包含:

自该第四存储元件组的第四操作数子集存储至该第六存储元件组;以及

将该第五存储元件组的该第三操作数子集恢复至该第四存储元件组;以及

当该微处理器于该第三处理模式下运行时,自该第四存储元件组提供该第三操作数子集至该执行单元以执行该些算数运算。

15. 如权利要求 14 所述的方法,其中该第一存储元件组的一第一存储元件具有一 ARM ISA 的堆迭式寄存器操作数以及该第一存储元件组的一第二存储元件具有一该执行单元执行该些算数运算的 ARM ISA 的连结寄存器操作数;

其中,该第二存储元件组与该第三存储元件组各包含具有一 ARM ISA 堆迭式指示符寄存器操作数的一第一存储元件以及一关联于该些处理模式并具有一 ARM ISA 连结寄存器操作数的一第二存储元件;

其中,该第四存储元件组具有该执行单元执行该些算数运算的 ARM ISAR8-R12 通用寄存器;

其中,该第五存储元件组具有 ARM ISA R8-R12 通用寄存器操作数的存储元件,以对应 ARM ISA FIQ 例外事件模式;

其中,该第六存储元件组包含具有 ARM ISA R8-R12 通用寄存器操作数的存储元件,以全域性地对应该除了 ARM ISA FIQ 例外事件模式外所有的 ARM ISA 处理模式。

16. 如权利要求 13 所述的方法,其中该第一存储元件组的一第一存储元件具有 ARM ISA 的一堆迭式指示符寄存器操作数,以及该第一存储元件组的一第二存储元件具有该执行单元执行该些算数运算的 ARM ISA 的一连结寄存器操作数。

17. 如权利要求 13 所述的方法,其中该些例外事件模式包含 ARM ISA 例外事件模式。

18. 如权利要求 13 所述的方法,还包含:

将 ARM ISA 机器指令语言的指令转译至多个微指令,其中至少一 ARMISA 的指令指示该微处理器自该当前处理模式进入该新处理模式;以及

执行该些微指令以将该第一存储元件组的该第一操作数子集存储至关联于该当前处理模式的该第二存储元件组,以及将关联于该新处理模式的该第二存储元件组的该第二操作数子集恢复至该第一存储元件组。

19. 如权利要求 18 所述的方法,还包含:

将 x86ISA 机器指令语言的指令转译至该些微指令,其中该些微指令是以不同于 x86ISA 指令集的指令的编码方式进行编码。

20. 一种微处理器,其支持一 ISA,该 ISA 指定多个处理模式以及指定多个架构寄存器,且该些架构寄存器关联于各处理模式,以及指定一载入多重指令,该载入多重指令指示该

微处理器自存储器内载入数据，并传入被该载入多重指令指定的一个或多个架构寄存器，该微处理器包含：

一直接存储器，具有关联于该些架构寄存器的第一部分的数据，并耦接于该处理器的至少一执行单元，以提供该数据给该执行单元；

一间接存储器，具有关联于该些架构寄存器的第二部分的数据，其中该间接存储器无法直接提供关联于该架构寄存器的该第二部分的数据至该执行单元；

其中，该些架构寄存器依据该些处理模式中的一当前处理模式，动态地分布于该架构寄存器的该第一部分与该架构寄存器的该第二部分；以及

其中，各架构寄存器被该载入多重指令指定：

若当该架构寄存器位于该第一部分，该微处理器自存储器内载入数据，并传入至该直接存储器；以及

若当该架构寄存器位于该第二部分，该微处理器自存储器内载入数据，并传入至该直接存储器，而后将该直接存储器的数据转至该间接存储器。

21. 如权利要求 20 所述的微处理器，其中该微处理器所支持的该 ISA 包含 ARM ISA，其中指定于 ISA 的该些处理模式包含 ARM ISA 使用者、系统、管理者、终止、未定、IRQ 以及 FIQ 处理模式，其中指定于该 ISA 的该架构寄存器包含关联于该使用者处理模式的 ARM ISA R0-R14 寄存器，以及关联于该管理者、终止、未定、IRQ 与 FIQ 处理模式的备份寄存器，其中指定于该 ISA 的该载入多重指令包含 ARM ISA 载入多重指令。

22. 如权利要求 20 所述的微处理器，还包含：

一指令转译器，用以将该载入多重指令转译为该微处理器可执行的多个微指令，其中各架构寄存器被该载入多重指令指定：

若当该架构寄存器位于该第一部分，该指令转译器发送一微指令，以自该存储器载入数据转入至该直接存储器；

若当该架构寄存器位于该第二部分，该指令转译器发送一第一微指令，以自该存储器载入数据转入至该直接存储器，并且发送一第二微指令以将该直接存储器的数据转至该间接存储器。

23. 如权利要求 22 所述的微处理器，其中该指令转译器包含：

一第一部分，发送该微指令以自存储器内载入数据，并送入该直接存储器，以及发送该第一微指令以自存储器内载入数据并送入该直接存储器，其中该第一部分包含一硬件状态机器；以及

一第二部分，发送该第二微指令以将该直接存储器的数据转至该间接存储器，其中该第二部分包含一微码。

24. 如权利要求 22 所述的微处理器，其中一硬件指令转译器将 x86ISA 机器语言指令 ARM ISA 机器语言指令的指令转译为该些微指令，其中该些微指令是以不同于 x86ISA 与 ARM ISA 指令集的指令编码方式进行编码，其中该微处理器还包含一执行管线，耦接于该硬件指令转译器，其中该执行管线执行该些微指令以产生由 x86ISA 与 ARM ISA 指令所定义的结果。

25. 如权利要求 21 所述的微处理器，

其中，若该当前处理模式为该 ARM ISA 使用者模式，则该直接存储器具有关联于该使用

者模式架构寄存器的数据,且该间接存储器具有关联于 ARM ISA 例外事件模式的 R13-R14 架构寄存器的数据以及关联于 ARM ISAFIQ 模式 R8-R12 架构寄存器的数据;

其中,若该当前处理模式为该 ARM ISA FIQ 模式,则该直接存储器具有关联于该 FIQ 架构寄存器的数据,且该间接存储器具有关联于该 ARM ISA 使用者模式与 non-FIQ 例外事件模式的 R13-R14 架构寄存器的数据,以及关联于早于该当前处理模式的该 ARM ISA 模式的 R8-R12 架构寄存器的数据;

其中,若该当前处理模式为一 ARM ISA non-FIQ 例外事件模式,则该直接存储器具有关联于该 non-FIQ 例外事件模式架构寄存器的数据,且该间接存储器具有关联于该 ARM ISA 使用者模式与非当前的例外事件模式的 R13-R14 架构寄存器的数据,以及关联于该 ARM ISA FIQ 模式的 R8-R12 架构寄存器的数据。

26. 一种用以操作一微处理器的方法,该处理器支持一 ISA,该 ISA 指定多个处理模式以及指定多个架构寄存器,且该些架构寄存器关联于各处理模式,以及指定一载入多重指令,该载入多重指令指示该微处理器自存储器内载入数据,并传入被该载入多重指令指定的一个或多个架构寄存器,该方法包含:

各架构寄存器被该载入多重指令指定:

若当该架构寄存器位于一第一部分,则自存储器内载入数据,并传入至该微处理器的直接存储器;以及

若当该架构寄存器位于一第二部分,则自存储器内载入数据并传入至该直接存储器,而后将该直接存储器的数据转至该间接存储器;

其中,该直接存储器具有关联于该架构寄存器的一第一部分的数据,并耦接于该微处理器的至少一执行单元以提供该数据至该执行单元;

其中,该间接存储器具有关联于该架构寄存器的一第二部分的数据,其中该间接存储器无法直接提供关联于该架构寄存器的该第二部分的数据至该执行单元;

其中,该些架构寄存器依据该些处理模式中的该当前处理模式,动态地分布于该架构寄存器的该第一部分与该架构寄存器的该第二部分。

27. 如权利要求 26 所述的方法,其中该微处理器所支持的该 ISA 包含 ARM ISA,其中指定于 ISA 的该些处理模式包含 ARM ISA 使用者、系统、管理者、终止、未定、IRQ 以及 FIQ 处理模式,其中指定于该 ISA 的该架构寄存器包含关联于该使用者处理模式的 ARM ISA R0-R14 寄存器,以及关联于该管理者、终止、未定、IRQ 与 FIQ 处理模式的备份寄存器,其中指定于该 ISA 的该载入多重指令包含 ARM ISA 载入多重指令。

28. 如权利要求 27 所述的方法,还包含:

利用该微处理器将该载入多重指令转译为该微处理器可执行的多个微指令,其中各架构寄存器被该载入多重指令指定:

若当该架构寄存器位于该第一部分,该指令转译器发送一微指令,以自该存储器载入数据转入至该直接存储器;

若当该架构寄存器位于该第二部分,该指令转译器发送一第一微指令,以自该存储器载入数据转入至该直接存储器,并且发送一第二微指令以将该直接存储器的数据转至该间接存储器。

29. 如权利要求 28 所述的方法,其中通过一硬件状态机器发送该微指令,以自存储器

内载入数据并送入该直接存储器，其中通过一微码发送该第二微指令，以将该直接存储器的数据转至该间接存储器。

30. 如权利要求 27 所述的方法，

其中，若该当前处理模式为该 ARM ISA 使用者模式，则该直接存储器具有关联于该使用者模式架构寄存器的数据，且该间接存储器具有关联于 ARM ISA 例外事件模式的 R13-R14 架构寄存器的数据以及关联于 ARM ISA FIQ 模式的 R8-R12 架构寄存器的数据；

其中，若该当前处理模式为该 ARM ISA FIQ 模式，则该直接存储器具有关联于该 FIQ 架构寄存器的数据，且该间接存储器具有关联于该 ARM ISA 使用者模式与 non-FIQ 例外事件模式的 R13-R14 架构寄存器的数据，以及关联于早于该当前处理模式的该 ARM ISA 模式的 R8-R12 架构寄存器的数据；

其中，若该当前处理模式为一 ARM ISA non-FIQ 例外事件模式，则该直接存储器具有关联于该 non-FIQ 例外事件模式架构寄存器的数据，且该间接存储器具有关联于该 ARM ISA 使用者模式与非当前的例外事件模式的 R13-R14 架构寄存器的数据，以及关联于该 ARM ISA FIQ 模式的 R8-R12 架构寄存器的数据。

31. 一种微处理器，其支持一 ISA，该 ISA 指定多个处理模式以及指定多个架构寄存器，且该些架构寄存器关联于各处理模式，以及指定一存储多重指令，该存储多重指令指令该微处理器将数据自被该存储多重指令指定的一个或多个架构寄存器中转存至一存储器，该微处理器包含：

一直接存储器，具有关联于该些架构寄存器的第一部分的数据，并耦接于该处理器的至少一执行单元，以提供该数据给该执行单元；

一间接存储器，具有关联于该些架构寄存器的第二部分的数据，其中该间接存储器无法直接提供关联于该架构寄存器的该第二部分的数据至该执行单元；

其中，该些架构寄存器依据该些处理模式中的该当前处理模式，动态地分布于该架构寄存器的第一部分与该架构寄存器的第二部分；以及

其中，各架构寄存器被该存储多重指令指定；

若当该架构寄存器位于该第一部分，该微处理器将数据自该直接存储器转存至存储器；以及

若当该架构寄存器位于该第二部分，该微处理器自该间接存储器内载入数据，并传入至该直接存储器，而后将数据自该直接存储器转存至存储器。

32. 如权利要求 31 所述的微处理器，其中该微处理器所支持的该 ISA 包含 ARM ISA，其中指定于 ISA 的该些处理模式包含 ARM ISA 使用者、系统、管理者、终止、未定、IRQ 以及 FIQ 处理模式，其中指定于该 ISA 的该架构寄存器包含关联于该使用者处理模式的 ARM ISA R0-R14 寄存器，以及关联于该管理者、终止、未定、IRQ 与 FIQ 处理模式的备份寄存器，其中指定于该 ISA 的该存储多重指令包含 ARM ISA 存储多重指令。

33. 如权利要求 32 所述的微处理器，还包含

一指令转译器，用以将该存储多重指令转译为该微处理器可执行的多个微指令，其中各架构寄存器被该存储多重指令指定；

若当该架构寄存器位于该第一部分，该指令转译器发送一微指令，以将数据自该直接存储器转存至存储器；

若当该架构寄存器位于该第二部分,该指令转译器发送一第一微指令,以自该间接存储器内载入数据并传入至该直接存储器,而后将数据自该直接存储器转存至存储器。

34. 如权利要求 32 所述的微处理器,其中,若该当前处理模式为该 ARMISA 使用者模式,则该直接存储器具有关联于该使用者模式架构寄存器的数据,且该间接存储器具有关联于 ARM ISA 例外事件模式的 R13-R14 架构寄存器的数据以及关联于 ARM ISA FIQ 模式的 R8-R12 架构寄存器的数据;其中,若该当前处理模式为该 ARM ISA FIQ 模式,则该直接存储器具有关联于该 FIQ 架构寄存器的数据,且该间接存储器具有关联于该 ARM ISA 使用者模式与 non-FIQ 例外事件模式的 R13-R14 架构寄存器的数据,以及关联于早于该当前处理模式的该 ARM ISA 模式的 R8-R12 架构寄存器的数据;其中,若该当前处理模式为一 ARM ISA non-FIQ 例外事件模式,则该直接存储器具有关联于该 non-FIQ 例外事件模式架构寄存器的数据,且该间接存储器具有关联于该 ARM ISA 使用者模式与非当前的例外事件模式的 R13-R14 架构寄存器的数据,以及关联于该 ARM ISA FIQ 模式的 R8-R12 架构寄存器的数据。

35. 一种用以操作一微处理器的方法,该处理器支持一 ISA,该 ISA 指定多个处理模式以及指定多个架构寄存器,且该些架构寄存器关联于各处理模式,以及指定一存储多重指令该存储多重指令指示该微处理器将数据自被该存储多重指令指定的一个或多个架构寄存器中转存至一存储器,该方法包含:

各架构寄存器被该存储多重指令指定:

若当该架构寄存器位于该第一部分,则将数据自该该微处理器的直接存储器转存至存储器;以及

若当该架构寄存器位于该第二部分,则自该间接存储器内载入数据并传入至该直接存储器,而后将数据自该直接存储器转存至存储器;

其中,该直接存储器具有关联于该架构寄存器的第一部分的数据,并耦接于该微处理器的至少一执行单元以提供该数据至该执行单元;

其中,该间接存储器具有关联于该架构寄存器的第二部分的数据,其中该间接存储器无法直接提供关联于该架构寄存器的该第二部分的数据至该执行单元;

其中,该些架构寄存器依据该些处理模式中的该当前处理模式,动态地分布于该架构寄存器的该第一部分与该架构寄存器的该第二部分。

36. 如权利要求 35 所述的方法,其中该微处理器所支持的该 ISA 包含 ARM ISA,其中指定于 ISA 的该些处理模式包含 ARM ISA 使用者、系统、管理者、终止、未定、IRQ 以及 FIQ 处理模式,其中指定于该 ISA 的该架构寄存器包含关联于该使用者处理模式的 ARM ISA R0-R14 寄存器,以及关联于该管理者、终止、未定、IRQ 与 FIQ 处理模式的备份寄存器,其中指定于该 ISA 的该存储多重指令包含 ARM ISA 存储多重指令。

37. 如权利要求 36 所述的方法,还包含

利用该微处理器将该存储多重指令转译为该微处理器可执行的多个微指令,其中各架构寄存器被该存储多重指令指定:

若当该架构寄存器位于该第一部分,该指令转译器发送一微指令,以将数据自该直接存储器转存至存储器;

若当该架构寄存器位于该第二部分,该指令转译器发送一第一微指令,以自该间接存

储器内载入数据，并传入至该直接存储器，而后将数据自该直接存储器转存至存储器。

38. 如权利要求 35 所述的方法，其中若该当前处理模式为该 ARM ISA 使用者模式，则该直接存储器具有关联于该使用者模式架构寄存器的数据，且该间接存储器具有关联于 ARM ISA 例外事件模式的 R13-R14 架构寄存器的数据以及关联于 ARM ISA FIQ 模式的 R8-R12 架构寄存器的数据；

其中，若该当前处理模式为该 ARM ISA FIQ 模式，则该直接存储器具有关联于该 FIQ 架构寄存器的数据，且该间接存储器具有关联于该 ARM ISA 使用者模式与 non-FIQ 例外事件模式的 R13-R14 架构寄存器的数据，以及关联于早于该当前处理模式的该 ARM ISA 模式的 R8-R12 架构寄存器的数据；

其中，若该当前处理模式为一 ARM ISA non-FIQ 例外事件模式，则该直接存储器具有关联于该 non-FIQ 例外事件模式架构寄存器的数据，且该间接存储器具有关联于该 ARM ISA 使用者模式与非当前的例外事件模式的 R13-R14 架构寄存器的数据，以及关联于该 ARM ISA FIQ 模式的 R8-R12 架构寄存器的数据。

微处理器及其操作方法

[0001] 相关申请的交叉引用

[0002] 本申请是同在申请中美国专利正式申请的部分连续申请,这些申请整体都纳入本申请参考:

[0003]

申请号	申请日
13/224, 310(CNTR. 2575)	09/01/2011
13/333, 520(CNTR. 2569)	12/21/2011
13/333, 572(CNTR. 2572)	12/21/2011
13/333, 631(CNTR. 2618)	12/21/2011

[0004] 本申请是引用于以下美国临时专利申请作优先权,每一申请整体都纳入本申请参考:

[0005]

申请号	申请日
61/473, 062(CNTR. 2547)	04/07/2011
61/473, 067(CNTR. 2552)	04/07/2011
61/473, 069(CNTR. 2556)	04/07/2011
61/537, 473(CNTR. 2569)	09/21/2011
61/541, 307(CNTR. 2585)	09/30/2011
61/547, 449(CNTR. 2573)	10/14/2011
61/555, 023(CNTR. 2564)	11/03/2011
61/604, 561(CNTR. 2552)	02/29/2012

[0006] 美国正式专利申请

[0007]

13/224, 310(CNTR. 2575)	09/01/2011
-------------------------	------------

[0008] 是引用下列美国临时申请的优先权:

[0009]

61/473, 062(CNTR. 2547)	04/07/2011
61/473, 067(CNTR. 2552)	04/07/2011
61/473, 069(CNTR. 2556)	04/07/2011

[0010] 以下三个本美国正式申请

[0011]

13/333, 520(CNTR. 2569)	12/21/2011
13/333, 572(CNTR. 2572)	12/21/2011
13/333, 631(CNTR. 2618)	12/21/2011

[0012] 都是以下美国正式申请的延续申请:

[0013]

13/224, 310(CNTR. 2575)	09/01/2011
-------------------------	------------

[0014] 并引用下列美国临时申请的优先权:

[0015]

61/473, 062(CNTR. 2547)	04/07/2011
61/473, 067(CNTR. 2552)	04/07/2011
61/473, 069(CNTR. 2556)	04/07/2011
61/537, 473(CNTR. 2569)	09/21/2011

[0016] 本申请是以下美国正式专利申请的相关申请：

[0017]

13/413, 258(CNTR. 2552)	03/06/2012
13/412, 888(CNTR. 2580)	03/06/2012
13/412, 904(CNTR. 2583)	03/06/2012
13/412, 914(CNTR. 2585)	03/06/2012
13/413, 346(CNTR. 2573)	03/06/2012
13/413, 300(CNTR. 2564)	03/06/2012
13/413, 314(CNTR. 2568)	03/06/2012

技术领域

[0018] 本发明是关于微处理器的技术领域，特别是关于在指令集中具有条件指令的微处理器。

背景技术

[0019] 由 Intel Corporation of Santa Clara, California 开发出来的 x86 处理器架构以及由 ARM Ltd. of Cambridge, UK 开发出来的先进精简指令集机器 (advanced risc machines, ARM) 架构是电脑领域中两种广为人知的处理器架构。许多使用 ARM 或 x86 处理器的电脑系统已经出现，并且，对于此电脑系统的需求正在快速增长。现今，ARM 架构处理核心是主宰低功耗、低价位的电脑市场，例如手机、手持式电子产品、平板电脑、网路路由器与集线器、机上盒等。举例来说，苹果 iPhone 与 iPad 主要的处理能力即是由 ARM 架构的处理核心提供。另一方面，x86 架构处理器则是主宰需要高效能的高价位市场，例如膝上电脑、桌上型电脑与服务器等。然而，随着 ARM 核心效能的提升，以及某些 x86 处理器在功耗与成本的改善，前述低价位与高价位市场的界线逐渐模糊。在移动运算市场，如智能型手机，这两种架构已经开始激烈竞争。在膝上电脑、桌上型电脑与服务器市场，可以预期这两种架构将会有更频繁的竞争。

[0020] 前述竞争态势使得电脑装置制造业者与消费者陷入两难，因无从判断哪一个架构将会主宰市场，更精确来说，无法判定哪一种架构的软件开发商将会开发更多软件。举例来说，一些每月或每年会定期购买大量电脑系统的消费个体，基于成本效率的考虑，例如大量采购的价格优惠与系统维修的简化等，会倾向于购买具有相同系统配置设定的电脑系统。然而，这些大型消费个体中的使用者群体，对于这些具有相同系统配置设定的电脑系统，往往有各种各样的运算需求。具体来说，部分使用者的需求是希望能够在 ARM 架构处理器上执行程序，其他部分使用者的需求是希望能够在 x86 架构处理器上执行程序，甚至有部分使用者希望能够同时在两种架构上执行程序。此外，新的、预期外的运算需求也可能出现而需要使用另一种架构。在这些情况下，这些大型个体所投入的部分资金就变成浪费。在另一个例子中，使用者具有一个重要的应用程序只能在 x86 架构上执行，因而他购买了 x86 架构的电脑系统(反之亦然)。不过，这个应用程序的后续版本改为针对 ARM 架构开发，并且优于原本的 x86 版本。使用者会希望转换架构来执行新版本的应用程序，但不幸地，他已经对于不倾向使用的架构投入相当成本。同样地，使用者原本投资于只能在 ARM 架构上执行的应用程序，但是后来也希望能够使用针对 x86 架构开发而未见于 ARM 架构的应用程序或是优于以 ARM 架构开发的应用程序，也会遭遇这样的问题，反之亦然。值得注意的是，虽然小

实体或是个人投入的金额较大实体为小,然而投资损失比例可能更高。其他类似的投资损失的例子可能出现在各种不同的运算市场中,例如由 x86 架构转换至 ARM 架构或是由 ARM 架构转换至 x86 架构的情况。最后,投资大量资源来开发新产品的运算装置制造业者,例如 OEM 厂商,也会陷入此架构选择的困境。若是制造业者基于 x86 或 ARM 架构研发制造大量产品,而使用者的需求突然改变,则会导致许多有价值的研发资源的浪费。

[0021] 对于运算装置的制造业者与消费者,能够保有其投资免于受到二种架构中何者胜出的影响是有帮助的,因而有必要提出一种解决方法让系统制造业者发展出可让使用者同时执行 x86 架构与 ARM 架构的程序的运算装置。

[0022] 使系统能够执行多个指令集程序的需求由来已久,这些需求主要是因为消费者会投入相当成本在旧硬件上执行的软件程序,而其指令集往往不兼容于新硬件。举例来说, IBM360 系统 Model30 即具有兼容于 IBM1401 系统的特征来缓和使用者由 1401 系统转换至较高效能与改良特征的 360 系统的痛苦。Model30 具有 360 系统与 1401 系统的只读存储控制 (Read Only Storage, ROS)), 使其在辅助存储空间预先存入所需信息的情况下能够使用于 1401 系统。此外,在软件程序以高阶语言开发的情况下,新的硬件开发商几乎没有办法控制为旧硬件所编译的软件程序,软件开发商也欠缺动力为新硬件重新编译 (re-compile) 源码,此情形尤其发生在软件开发商与硬件开发商是不同个体的情况。Sberman 与 Ebcio glu 于 Computer, June1993, No. 6 提出的文章 “An Architectural Framework for Supporting Heterogeneous Instruction-Set Architectures” 中公开一种利用执行于精简指令集 (RISC)、超纯量架构 (superscalar) 与超长指令字 (VLIW) 架构 (下称原生架构) 的系统来改善既存复杂指令集 (CISC) 架构 (例如 IBM S/390) 执行效率的技术,其所公开的系统包含有执行原生码的原生引擎 (native engine) 与执行目的码的迁移引擎 (migrant engine), 并可依据转译软件将目的码 (object code) 转译为原生码 (native code) 的转译效果,在这两种编码间视需要进行转换。请参照 2006 年 5 月 16 日公开的美国专利第 7,047,394 号专利案, Van Dyke 等公开一处理器,具有用以执行原生精简指令集 (Tapestry) 的程序指令的执行管线,并利用硬件转译与软件转译的结合,将 x86 程序指令转译为原生精简指令集的指令。Nakada 等提出具有 ARM 架构的前端管线与 Fujitsu FR-V (超长指令字) 架构的前端管线的异质多线程处理器 (heterogeneous SMT processor), ARM 架构前端管线是用于非规则软件程序 (如操作系统), 而 Fujitsu FR-V (超长指令字) 架构的前端管线是用于多介质应用程序以将一增加的超长指令字队列汇入 FR-V 超长指令字的后端管线以维持来自前端管线的指令。请参照 Buchty 与 Weib, eds, Universitatsverlag Karlsruhe 于 2008 年 11 月在 First International Workshop on New Frontiers in High-performance and Hardware-aware Computing (HipHaC' 08), Lake Como, Italy, (配合 MICRO-41) 发表的论文集 (ISBN 978-3-86644-298-6) 的文章 “OROCHI:A Multiple Instruction Set SMTProcessor”。文中提出的方法是用以降低整个系统在异质系统单晶片 (SOC) 装置 (如德州仪器 OMAP 应用处理器) 内所占据的空间。此异质系统单晶片装置具有一个 ARM 处理器核心加上一个或多个协同处理器 (co-processors) (例如 TMS320、多种数字信号处理器、或是多种图形处理单元 (GPUs))。这些协同处理器并不分享指令执行资源,只是整合于同一晶片上的不同处理核心。

[0023] 软件转译器 (software translator)、或称软件模拟器 (software

emulator, software simulator)、动态二进制码转译器等,也被用于支持将软件程序在与此软件程序架构不同的处理器上执行的能力。其中受欢迎的商用实例如搭配苹果麦金塔 (Macintosh) 电脑的 Motorola68K-to-PowerPC 模拟器,其可在具有 PowerPC 处理器的麦金塔电脑上执行 68K 程序,以及后续研发出来的 PowerPC-to-x86 模拟器其可在具有 x86 处理器的麦金塔电脑上执行 68K 程序。位于加州圣塔克拉拉 (Santa Clara, California) 的全美达公司,结合超长指令字 (VLIW) 的核心硬件与“纯粹软件指令的转译器(也即程序码转译软件 (Code Morphing Software))以动态地编译或模拟 (emulate) x86 程序码序列”以执行 x86 程序码,请参照 2011 年维基百科针对全美达 (Transmeta) 的说明 <<http://en.wikipedia.org/wiki/Transmeta>>。另外,参照 1998 年 11 月 3 日由 Kelly 等提出的美国专利第 5,832,205 号公告案。IBM 的 DAISY (Dynamic Architecture Instruction Set from Yorktown) 系统具有超长指令字 (VLIW) 机器与动态二进制软件转译,可提供 100% 的旧架构软件相容模拟。DAISY 具有位于只读存储器内的虚拟机器监视器 (Virtual Machine Monitor),以并行处理 (parallelize) 与存储超长指令字原始码 (VLIW primitives) 至未见于旧有系统架构的部分主要存储器内,期能避免这些旧有体系架构的程序码片段在后续程序被重新编译 (re-translation)。DAISY 具有高速编译器优化演算法 (fast compiler optimization algorithms) 以提升效能。QEMU 是具有软件动态转译器的机器模拟器 (machine emulator)。QEMU 可在多种主机 (host),如 x86、PowerPC、ARM、SPARC、Alpha 与 MIPS,模拟多种中央处理器,如 x86、PowerPC、ARM 与 SPARC。请参照 QEMU, a Fast and Portable Dynamic Translator, Fabrice Bellard, USENIX Association, FREENIX Track:2005USENIX Annual Technical Conference,如同其开发者所称“动态转译器对目标处理器指令执行时的转换(runtime conversion),将其转换至主机指令集,所 产生的二进制码存储于一转译高速缓冲存储器以利重复取用。QEMU [较之其他动态转译器] 远为简单,因为它只连接 GNC C 编译器于离线 (off line) 时所产生的机器码片段”。同时可参照 2009 年 6 月 19 日 Adelaide 大学 Lee WangHao 的学位论文“ARM Instruction Set Simulation on Multi-core x86Hardware”。虽然以软件转译为基础的解决方案所提供的处理效能可以满足多个运算需求的一部分,但是不大能够满足多个使用者的情况。

[0024] 静态 (static) 二进位制转译是另一种具有高效能潜力的技术。不过,二进位制转译技术的使用存在技术上的问题(例如自我修改程序码 (self-modifying code)、只在执行时 (run-time) 可知的间接分支 (indirect branches) 数值) 以及商业与法律上的障碍(例如:此技术可能需要硬件开发商配合开发发布新程序所需的管道;对原程序发布者存在潜在的授权或是著作权侵害的风险)。

发明内容

[0025] 本发明的一实施例提供一微处理器。该微处理器包含多个处理模式,该处理模式包含一使用者模式与多个例外事件模式。该微处理器还包含至少一执行单元,用以在程序指令指定的操作数上执行算数运算;该微处理器还包含一第一存储元件组,耦接于该执行单元,其中,该第一存储元件组包含第一操作数子集,并提供该第一操作数子集给该执行单元;该微处理器还包含一第二存储元件组,关联于各处理模式,其中,该第二存储元件组包含一第二操作数子集,其中,该第二存储元件组无法直接提供该第二操作数子集给该执行

单元；以及，该微处理器还包含一逻辑，其中，当从一当前处理模式进入至一新处理模式时，该逻辑将该第一存储元件组中的该第一操作数子集存储至关联于该当前处理模式的第二存储元件组，并将关联于该新处理模式的该第二存储元件组中的该第二操作数子集恢复至该第一存储元件组。

[0026] 本发明的另一实施例提供用于操作一种微处理器的方法，该微处理器包含多个处理模式，该些处理模式具有一使用者模式以及多个例外事件模式，其中该微处理器还包含至少一执行单元，该执行单元通过特定程序指令在操作数上执行算数运算，该方法包含：当该微处理器在该些处理模式中的一当前处理模式运行时，自一第一存储元件组中提供一第一操作数集至该执行单元以执行算数运算；而当自该当前的处理模式进入该些处理模式中的一新处理模式时，则包含以下步骤：将该第一存储元件组的该第一操作数集存储至 关联于该当前处理模式的一第二存储单元组；将该关联于该新处理模式的一第三存储元件组的一第二操作数集恢复至该第一存储元件组；以及当该微处理器于该新处理模式中运行时，自该第一存储元件组提供该第二操作数集至该执行单元以执行算数运算。

[0027] 本发明的又一实施例提供一种电脑程序产品。此电脑程序产品编码于至少一电脑可读取存储介质以使用于一运算装置。此电脑程序产品具有适用于前述介质的电脑可读取程序码，该电脑程序产品包括：适用于该介质的电脑可读取程序码，用以指定一微处理器，该电脑可读取程序码包含第一程序码，用以指定于多个处理模式，该些处理模式包含一使用者模式与多个例外事件模式；电脑可读取程序码还包含第二程序码，用以指定于至少一执行单元，该执行单元通过特定程序指令在操作数上执行算数运算；该电脑可读取程序码还包含第三程序码，用以指定于一第一存储元件组，该第一存储元件组耦接于该执行单元，其中该第一存储元件组具有一第一操作数子集，并提供该第一操作数子集至该执行单元；该电脑可读取程序码还包含第四程序码，用以指定关联于该些处理模式的一第二存储元件组。其中该第二存储元件组具有一第二操作数子集，其中该第二操作数不可直接提供该第二操作数子集至该执行单元；以及该电脑可读取程序码还包含第五程序码，用以指定一逻辑，其中当自依当前处理模式进入该些处理模式的一新处理模式时，该逻辑存储该第一存储元件组的该第一操作数子集至关联于该当前处理模式的该第二存储元件组，并恢复关联于该新处理模式的该第二存储元件组的该第二操作数子集至该第一存储元件组。

[0028] 本发明的一实施例提供一种微处理器，其支持一 ISA，该 ISA 指定于多个处理模式以及指定于多个架构寄存器，且该些架构寄存器关联于各处理模式，以及指定一载入多重指令，该载入多重指令指示该微处理器自存储器内载入数据，并传入指定于该载入多重指令的一个或多个架构寄存器，该微处理器包含：直接存储器，其具有关联于该些架构寄存器的一第一部分的数据，并耦接于该微处理器的至少一执行单元，以提供该数据给该执行单元；该微处理器还包含间接存储器，其具有关联于该些架构寄存器的一第二部分的数据，其中该间接存储器无法直接提供关联于该架构寄存器的该第二部分的数据至该执行单元；其中，该些架构寄存器依据该些处理模式中的该当前处理模式，动态地分布于该些架构寄存器的该第一部分与该些架构寄存器的该第二部分；以及，其中各架构寄存器指定于该载入多重指令：若当该架构寄存器位于该第一部分，该微处理器自存储器内载入数据，并传入至直接存储器；以及若当该架构寄存器位于该第二部分，该微处理器自存储器内载入数据，并传入至该直接存储器，而后将该直接存储器的数据转至该间接存储器。

[0029] 本发明的另一实施例提供用于操作一种微处理器的方法，其支持一 ISA，该 ISA 指定于多个处理模式、指定于关联于各处理模式的多个架构寄存器，以及指定于一载入多重指令，该载入多重指令指示该微处理器自存储器内载入数据，并传入指定于该载入多重指令的一个或多个架构寄存器，该方法包含，对于指定于载入多重指令的各架构寄存器，若该架构寄存器位于该第一部分，则自存储器内载入数据至该微处理器的直接存储器，而若该架构寄存器位于该第二部分，则自存储器内载入数据至该直接存储器，并接着将该直接存储器的数据存储至该间接存储器。该直接存储器具有关联于该些架构寄存器的第一部分的数据，并耦接于该处理器的至少一执行单元，以提供该数据给该执行单元；间接存储器具有关联于该些架构寄存器的第二部分的数据，其中该间接存储器无法直接提供关联于该架构寄存器的该第二部分的数据至该执行单元；其中，该些架构寄存器依据该些处理模式中的该当前处理模式，动态地分布于该架构寄存器的该第一部分与该架构寄存器的该第二部分。

[0030] 本发明的另一实施例提供一种微处理器，其支持一 ISA，该 ISA 指定多个处理模式以及指定多个架构寄存器，且该些架构寄存器关联于各处理模式，以及指定于一存储多重指令，该存储多重指令指示该微处理器将数据自指定于该存储多重指令的一个或多个架构寄存器中转存至该存储器，该微处理器包含直接存储器，具有关联于该些架构寄存器的第一部分的数据，并耦接于该微处理器的至少一执行单元，以提供该数据给该执行单元；该微处理器还包含间接存储器，具有关联于该些架构寄存器的第二部分的数据，其中该间接存储器无法直接提供关联于该架构寄存器的该第二部分的数据至该执行单元；其中，该些架构寄存器依据该些处理模式中的该当前处理模式，动态地分布于该架构寄存器的该第一部分与该架构寄存器的该第二部分；以及，其中，各架构寄存器指定于该存储多重指令：若当该架构寄存器位于该第一部分，该微处理器将数据自该直接存储器转存至存储器；以及若当该架构寄存器位于该第二部分，该微处理器自该间接存储器内载入数据，并传入至该直接存储器，而后将数据自该直接存储器转存至存储器。

[0031] 本发明的又一实施例提供一种用以操作一微处理器的方法，该微处理器支持一 ISA，该 ISA 指定多个处理模式以及指定多个架构寄存器，且该些架构寄存器关联于各处理模式，以及指定一存储多重指令，该存储多重指令指示该微处理器将数据自指定于该存储多重指令一个或多个架构寄存器中转存至该存储器，该方法包含：各架构寄存器指定于该存储多重指令：若当该架构寄存器位于该第一部分，则将数据自该微处理器的直接存储器转存至存储器；以及若当该架构寄存器位于该第二部分，则自该间接存储器内载入数据，并传入至该直接存储器，而后将数据自该直接存储器转存至存储器。其中，该直接存储器具有关联于该架构寄存器的第一部分的数据，并耦接于该微处理器的至少一执行单元以提供该数据至该执行单元；其中，该间接存储器具有关联于该架构寄存器的第二部分的数据。该间接存储器无法直接提供关联于该架构寄存器的该第二部分的数据至该执行单元；其中，该些架构寄存器依据该些处理模式中的该当前处理模式，动态地分布于该架构寄存器的该第一部分与该架构寄存器的该第二部分。

[0032] 本发明的又一实施例提供一种电脑程序产品，此电脑程序产品编码于至少一电脑可读取存储介质，以使用于一运算装置，该电脑程序产品包括：适用于该介质的电脑可读取程序码，用以指定一微处理器，该微处理器支持一 ISA，该 ISA 指定多个处理模式以及指定

多个架构寄存器，且该些架构寄存器关联于各处理模式，以及指定于一载入多重指令，该载入多重指令指令该微处理器自存储器内载入数据，并传入指定于该载入多重指令一个或多个架构寄存器，该电脑可读取程序码包含第一程序码，用以指定于直接存储器，该直接存储器具有关联于该架构寄存器的第一部分的数据，且并耦接于该微处理器的至少一执行单元，以提供该数据给该执行单元；该电脑可读取程序码还包含第二程序码，用以指定于间接存储器，该间接存储器具有关联于该些架构寄存器的第二部分的数据，其中该间接存储器无法直接提供关联于该架构寄存器的该第二部分的数据至该执行单元；其中，该些架构寄存器依据该些处理模式中的该当前处理模式，动态地分布于该架构寄存器的该第一部分与该架构寄存器的该第二部分；其中，各架构寄存器指定于该载入多重指令：若当该架构寄存器位于该第一部分，该微处理器自存储器内载入数据，并传入至该直接存储器；以及若当该架构寄存器位于该第二部分，则该微处理器自存储器内载入数据，并传入至该直接存储器，而后将该直接存储器的数据转至该间接存储器。

[0033] 关于本发明的优点与精神可以通过以下的发明详述及所附图式得到进一步的了解。

附图说明

[0034] 图 1 是本发明执行 x86 程序集架构与 ARM 程序集架构机器语言程序的微处理器一实施例的方块图；

[0035] 图 2 是一方块图，详细显示图 1 的硬件指令转译器；

[0036] 图 3 是一方块图，详细显示图 2 的指令格式化程序 (instructionformatter)；

[0037] 图 4 是一方块图，详细显示图 1 的执行管线；

[0038] 图 5 是一方块图，详细显示图 1 的寄存器文件；

[0039] 图 6 (包含图 6A 与图 6B) 是一流程图，显示图 1 的微处理器的操作步骤；

[0040] 图 7 是本发明一双核心微处理器的方块图；

[0041] 图 8 是本发明执行 x86ISA 与 ARM ISA 机器语言程序的微处理器另一实施例的方块图；

[0042] 图 9 是一已知硬件寄存器文件架构示意图；

[0043] 图 10 是本发明的系统方块图，详细显示图 1 的微处理器；

[0044] 图 11 (包含图 11A 与图 11B) 是显示在本发明中，如图 10 的微处理器 100 操作的流程图；

[0045] 图 12 是一流程图，显示图 10 的微处理器依据图 11，数据在直接存储器与间接存储器间流动；

[0046] 图 13 (包含图 13A 与图 13B) 是一流程图，显示在本发明中如图 1 的微处理器 100 执行一 LDM 指令的流程图；

[0047] 图 14 (包含图 14A 与图 14B) 是一流程图，显示在本发明中如图 1 的微处理器 100 执行一 LDM 指令的另一流程图；

[0048] 图 15 (包含图 15A 与图 15B) 是一流程图，显示在本发明中如图 1 的微处理器 100 执行一 STM 指令的流程图；以及

[0049] 图 16 (包含图 16A 与图 16B) 是一流程图，显示在本发明中如图 1 的微处理器 100

执行一 STM 指令的另一流程图。

- [0050] 【主要元件符号说明】
- [0051] 微处理器 (处理核心) 100
- [0052] 指令高速缓冲存储器 102
- [0053] 硬件指令转译器 104
- [0054] 寄存器文件 106
- [0055] 存储器子系统 108
- [0056] 执行管线 112
- [0057] 指令撷取单元与分支预测器 114
- [0058] ARM 程序计数器 (PC) 寄存器 116
- [0059] x86 指令指示符 (IP) 寄存器 118
- [0060] 配置寄存器 (configuration register) 122
- [0061] ISA 指令 124
- [0062] 微指令 126
- [0063] 结果 128
- [0064] 指令模式指示符 (instruction mode indicator) 132
- [0065] 撷取地址 134
- [0066] 环境模式指示符 (environment mode indicator) 136
- [0067] 指令格式化程序 202
- [0068] 简单指令转译器 (SIT) 204
- [0069] 复杂指令转译器 (CIT) 206
- [0070] 多工器 (mux) 212
- [0071] x86 简单指令转译器 222
- [0072] ARM 简单指令转译器 224
- [0073] 微程序计数器 (micro-program counter, micro-PC) 232
- [0074] 微码只读存储器 234
- [0075] 微序列器 (microsequencer) 236
- [0076] 指令间接寄存器 (instruction indirection register, IIR) 235
- [0077] 微转译器 (microtranslator) 237
- [0078] 格式化 ISA 指令 242
- [0079] 实行微指令 (implementing microinstructions) 244
- [0080] 实行微指令 246
- [0081] 选择输入 248
- [0082] 微码地址 252
- [0083] 只读存储器地址 254
- [0084] ISA 指令信息 255
- [0085] 预解码器 (pre-decoder) 302
- [0086] 指令比特组队列 (IBQ) 304
- [0087] 长度解码器 (length decoders) 与纹波逻辑 (ripple logic) 306

- [0088] 多工器队列 (mux queue, MQ) 308
- [0089] 多工器 312
- [0090] 格式化指令队列 (formatted instruction queue, FIQ) 314
- [0091] ARM 指令集状态 322
- [0092] 微指令队列 401
- [0093] 寄存器分配表 (register allocation table, RAT) 402
- [0094] 指令调度器 (instruction dispatcher) 404
- [0095] 保留站 (reservation station) 406
- [0096] 指令发送单元 (instruction issue unit) 408
- [0097] 整数 / 分支 (integer/branch) 单元 412
- [0098] 介质单元 (media unit) 414
- [0099] 载入 / 存储 (load/store) 单元 416
- [0100] 浮点 (floating point) 单元 418
- [0101] 重排缓冲器 (reorder buffer, ROB) 422
- [0102] 执行单元 424
- [0103] ARM 特定寄存器 502
- [0104] x86 特定寄存器 504
- [0105] 共享寄存器 506
- [0106] 双核心微处理器 700
- [0107] 微指令高速缓冲存储器 892
- [0108] 硬件寄存器文件 902
- [0109] 硬件多工逻辑 904
- [0110] 硬件寄存器 906
- [0111] 硬件多工逻辑 908
- [0112] 寄存器地址 912
- [0113] 处理模式 914
- [0114] 多工器 1004、1006、1008、1014、1016、1018

具体实施方式

- [0115] 名词定义
- [0116] 指令集, 定义二进位制编码值的集合(即机器语言指令)与微处理器所执行操作间的对应关系。机器语言程序基本上以二进位制进行编码, 不过也可使用其他进位制的系统, 如部分早期 IBM 电脑的机器语言程序, 虽然最终也是以电压高低呈现二进位值的物理信号来表现, 不过却是以十进位制进行编码。机器语言指令指示微处理器执行的操作如: 将寄存器 1 内的操作数与寄存器 2 内的操作数相加并将结果写入寄存器 3、将存储器地址 0x12345678 的操作数减掉指令所指定的立即操作数并将结果写入寄存器 5、依据寄存器 7 所指定的比特数移动寄存器 6 内的数值、若是零旗标被设定时, 分支到指令后方的 36 个比特组、将存储器地址 0xABCD0000 的数值载入寄存器 8。因此, 指令集定义各个机器语言指令使微处理器执行所要执行的操作的二进位编码值。需要了解的是, 指令集定义二进位值

与微处理器操作间的对应关系，并不意味着单一个二进位值就会对应至单一个微处理器操作。具体来说，在部分指令集中，多个二进位值可能会对应至同一个微处理器操作。

[0117] 指令集架构 (ISA)，从微处理器家族的脉络来看包含 (1) 指令集；(2) 指令集的指令所能存取的资源集(例如：存储器定址所需的寄存器与模式)；以及 (3) 微处理器回应指令集的指令执行所产生的例外事件集(例如：除以零、分页错误、存储器保护违反等)。因为程序撰写者，如组译器与编译器的撰写者，想要作出机器语言程序在一微处理器家族执行时，就需要此微处理器家族的 ISA 定义。所以微处理器家族的制造者通常会将 ISA 定义于操作者操作手册中。举例来说，2009 年 3 月公布的 Intel64 与 IA-32 架构软件开发者手册 (Intel64and IA-32Architectures Software Developer's Manual) 即定义 Intel64 与 IA-32 处理器架构的 ISA。此软件开发者手册包含有五个章节，第一章是基本架构；第二 A 章是指令集参考 A 至 M；第二 B 章是指令集参考 N 至 Z；第三 A 章是系统编程指南；第三 B 章是系统编程指南第二部分，此手册列为本申请的参考文件。此种处理器架构通常被称为 x86 架构，本文中则是以 x86、x86ISA、x86ISA 家族、x86 家族或是相似用语来说明。在另一个例子中，2010 年公布的 ARM 架构参考手册，ARM v7-A 与 ARM v7-R 版本 Erratamarkup，定义 ARM 处理器架构的 ISA。此参考手册列为参考文件。此 ARM 处理器架构的 ISA 在此也被称为 ARM、ARM ISA、ARM ISA 家族、ARM 家族或是相似用语。其他众所周知的 ISA 家族还有 IBM System/360/370/390 与 z/Architecture、DEC VAX、Motorola68k、MIPS、SPARC、PowerPC 与 DEC Alpha 等等。ISA 的定义会涵盖处理器家族，因为处理器家族的发展中，制造者会通过在指令集中增加新指令、以及 / 或在寄存器组中增加新的寄存器等方式来改进原始处理器的 ISA。举例来说，随着 x86 程序集架构的发展，其于 IntelPentium III 处理器家族导入一组 128 比特的多介质扩展指令集 (MMX) 寄存器作为单指令多重数据流扩展 (SSE) 指令集的一部分，而 x86ISA 机器语言程序已经开发来利用 XMM 寄存器以提升效能，虽然现存的 x86ISA 机器语言程序并不使用单指令多重数据流扩展指令集的 XMM 寄存器。此外，其他制造商也设计且制造出可执行 x86ISA 机器语言程序的微处理器。例如，超微半导体 (AMD) 与威盛电子 (VIA Technologies) 即在 x86ISA 增加新技术特征，如超微半导体的 3DNOW! 单指令多重数据流 (SIMD) 向量处理指令，以及威盛电子的 Padlock 安全引擎随机数生成器 (random number generator) 与先进译码引擎 (advanced cryptography engine) 的技术，前述技术都是采用 x86ISA 的机器语言程序，但却非由现有的 Intel 微处理器实现。以另一个实例来说明，ARMISA 原本定义 ARM 指令集状态具有 4 比特组的指令。然而，随着 ARM ISA 的发展而增加其他指令集状态，如具有 2 比特组指令以提升编码密度的 Thumb 指令集状态以及用以加速 Java 比特组码程序的 Jazelle 指令集状态，ARM ISA 机器语言程序已被发展来使用部分或所有其他 ARM ISA 指令集状态，即使现存的 ARM ISA 机器语言程序产生之初并非采用这些其他 ARM ISA 指令集状态。

[0118] 指令集架构 (ISA) 机器语言程序，包含 ISA 指令序列，即 ISA 指令集对应至程序撰写者要程序执行的操作序列的二进位编码值序列。因此，x86ISA 机器语言程序包含 x86ISA 指令序列，ARM ISA 机器语言程序则包含 ARM ISA 指令序列。机器语言程序指令存放于存储器内，且由微处理器撷取并执行。

[0119] 硬件指令转译器，包含多个电晶体的配置，用以接收 ISA 机器语言指令(例如 x86ISA 或是 ARM ISA 机器语言指令)作为输入，并对应地输出一个 或多个微指令至微处理

器的执行管线。执行管线执行微指令的执行结果由 ISA 指令所定义。因此，执行管线通过对这些微指令的集体执行来“实现”ISA 指令。也就是说，执行管线通过对于硬件指令转译器输出的实行微指令的集体执行，实现所输入 ISA 指令所指定的操作，以产生此 ISA 指令定义的结果。因此，硬件指令转译器可视为是将 ISA 指令“转译 (translate)”为一个或多个实行微指令。本实施例所描述的微处理器具有硬件指令转译器以将 x86ISA 指令与 ARM ISA 指令转译为微指令。不过，需要理解的是，硬件指令转译器并非必然可对 x86 使用者操作手册或是 ARM 使用者操作手册所定义的整个指令集进行转译，而往往只能转译这些指令中一个子集合，如同绝大多数 x86ISA 与 ARM ISA 处理器只支持其相对应的使用者操作手册所定义的一个指令子集合。具体来说，x86 使用者操作手册定义由硬件指令转译器转译的指令子集合，不必然就对应至所有既有的 x86ISA 处理器，ARM 使用者操作手册定义而由硬件指令转译器转译的指令子集合，不必然就对应至所有现存的 ARM ISA 处理器。

[0120] 执行管线，是一多层次级序列 (sequence of stages)。此多层次级序列的各个层级分别具有硬件逻辑与一硬件寄存器。硬件寄存器保持硬件逻辑的输出信号，并依据微处理器的时钟信号，将此输出信号提供至多层次级序列的下一层级。执行管线可以具有多个多层次级序列，例多重执行管线。执行管线接收微指令作为输入信号，并相应地执行微指令所指定的操作以输出执行结果。微指令所指定且由执行管线的硬件逻辑所执行的操作包括但不限于算数、逻辑、存储器载入 / 存储、比较、测试、与分支解析，对进行操作的数据格式包括但不限于整数、浮点数、字母、二进编码十进数 (BCD)、与压缩格式 (packed format)。执行管线执行微指令以实现 ISA 指令(如 x86 与 ARM)，藉以产生 ISA 指令所定义的结果。执行管线不同于硬件指令转译器。具体来说，硬件指令转译器产生实行微指令，执行管线则是执行这些指令，但不产生这些实行微指令。

[0121] 指令高速缓冲存储器，是微处理器内的一个随机存取装置，微处理器将 ISA 机器语言程序的指令(例如 x86ISA 与 ARM ISA 的机器语言指令)放置其中，这些指令撷取自系统存储器并由微处理器依据 ISA 机器语言程序的执行流程，来执行。具体来说，ISA 定义一指令地址寄存器以持有下一个待执行 ISA 指令的存储器地址(举例来说，在 x86ISA 定义为指令指示符 (IP) 而在 ARM ISA 定义为程序计数器 (PC))，而在微处理器执行机器语言程序以控制 程序流程时，微处理器会更新指令地址寄存器的内容。ISA 指令被高速缓冲存储器来供后续撷取之用。当该寄存器所包含的下一个机器语言程式的 ISA 指令位址位于目前的指令快取中，可依据指令寄存器的内容快速地指令高速缓冲存储器撷取 ISA 指令由系统存储器中取出该 ISA 指令。尤其是，此程序是基于指令地址寄存器(如指令指示符 (IP) 或是程序计数器 (PC))的存储器地址向指令高速缓冲存储器取得数据，而非特定运用一载入或存储指令所特定的存储器地址进行数据撷取。因此，将指令集架构的指令视为数据(例如采用软件转译的系统的硬件部分所呈现的数据)的专用数据高速缓冲存储器，特地运用一载入 / 存储地址，而非基于指令地址寄存器的数值做存取的，就不是此处所称的指令高速缓冲存储器。此外，可取得指令与数据的混合式高速缓冲存储器，基于指令地址寄存器的数值以及基于载入 / 存储地址，而非仅仅基于载入 / 存储地址，也被涵盖在本说明对指令高速缓冲存储器的定义内。在本说明内容中，载入指令是指将数据由存储器载入至微处理器的指令，存储指令是指将数据由微处理器写入存储器的指令。

[0122] 微指令集，是微处理器的执行管线能够执行的指令(微指令)的集合。

[0123] 实施例说明

[0124] 本发明实施例公开的微处理器可通过硬件将其对应的 x86ISA 与 ARMISA 指令转译为由微处理器执行管线直接执行的微指令,以达到可执行 x86ISA 与 ARM ISA 机器语言程序的目的。此微指令由不同于 x86 与 ARM ISA 的微处理器的微架构 (microarchitecture) 的微指令集所定义。由于本文所述的微处理器需要执行 x86 与 ARM 机器语言程序,微处理器的硬件指令转译器会将 x86 与 ARM 指令转译为微指令,并将这些微指令提供至微处理器的执行管线,由微处理器执行这些微指令以实现前述 x86 与 ARM 指令。由于这些实行微指令是直接由硬件指令转译器提供至执行管线来执行,而不同于采用软件转译器的系统需于执行管线执行指令前,将预先存储本机 (host) 指令至存储器,因此,前述微处理器具有潜力能够以较快的执行速度执行 x86 与 ARM 机器语言程序。

[0125] 图1是一方块图显示本发明能够执行 x86ISA 与 ARM ISA 机器语言程序的微处理器 100 的实施例。此微处理器 100 具有一指令高速缓冲存储器 102 ;一硬件指令转译器 104,用以由指令高速缓冲存储器 102 接收 x86ISA 指令与 ARM ISA 指令 124 并将其转译为微指令 126 ;一执行管线 112,执行由硬件 指令转译器 104 接收的微指令 126 以产生微指令结果 128,该结果是以操作数的型式回传至执行管线 112 ;一寄存器文件 106 与一存储器子系统 108,分别提供操作数至执行管线 112 并由执行管线 112 接收微指令结果 128 ;一指令撷取单元与分支预测器 114,提供一撷取地址 134 至指令高速缓冲存储器 102 ;一 ARM ISA 定义的程序计数器寄存器 116 与一 x86ISA 定义的指令指示符寄存器 118,依据微指令结果 128 进行更新,并且提供其内容至指令撷取单元与分支预测器 114 ;以及多个配置寄存器 122,提供一指令模式指示符 132 与一环境模式指示符 136 至硬件指令转译器 104 与指令撷取单元与分支预测器 114,并基于微指令结果 128 进行更新。

[0126] 由于微处理器 100 可执行 x86ISA 与 ARM ISA 机器语言指令,微处理器 100 依据程序流程由系统存储器(未图示)撷取指令至微处理器 100。微处理器 100 存取最近撷取的 x86ISA 与 ARM ISA 的机器语言指令至指令高速缓冲存储器 102。指令撷取单元 114 将依据由系统存储器撷取的 x86 或 ARM 指令比特组区段,产生一撷取地址 134。若是命中指令高速缓冲存储器 102,指令高速缓冲存储器 102 将位于撷取地址 134 的 x86 或 ARM 指令比特组区段提供至硬件指令转译器 104,否则由系统存储器中撷取指令集架构的指令 124。指令撷取单元 114 是基于 ARM 程序计数器 116 与 x86 指令指示符 118 的值产生撷取地址 134。具体来说,指令撷取单元 114 会在一撷取地址寄存器中维持一撷取地址。任何时候指令撷取单元 114 撷取到新的 ISA 指令比特组区段,它就会依据此区段的大小更新撷取地址,并依据既有方式依序进行,直到出现一控制流程事件。控制流程事件包含例外事件的产生、分支预测器 114 的预测显示撷取区段内有一将发生的分支 (taken branch)、以及执行管线 112 回应一非由分支预测器 114 所预测的将发生分支指令的执行结果,而对 ARM 程序计数器 116 与 x86 指令指示符 118 进行的更新。指令撷取单元 114 将撷取地址相应地更新为例外处理程序地址、预测目标地址或是执行目标地址以回应一控制流程事件。在一实施例中,指令高速缓冲存储器 102 是一混合高速缓冲存储器,以存取 ISA 指令 124 与数据。值得注意的是,在此混合高速缓冲存储器的实施例中,虽然混合高速缓冲存储器可基于一载入 / 存储地址将数据写入高速缓冲存储器或由高速缓冲存储器载入数据,在微处理器 100 由混合高速缓冲存储器撷取指令集架构的指令 124 的情况下,混合高速缓冲存储器是基于 ARM 程序计数器

116 与 x86 指令指示符 118 的数值来存取,而非基于载入 / 存储地址。指令高速缓冲存储器 102 可以是一随机存取存储器装置。

[0127] 指令模式指示符 132 是一状态指示微处理器 100 当前是否正在撷取、格式化 / 解码、以及将 x86ISA 或 ARM ISA 指令 124 转译为微指令 126。此外,执行管线 112 与存储器子系统 108 接收此指令模式指示符 132,此指令模式指示符 132 会影响微指令 126 的执行方式,尽管只是微指令集内的一个小集合受影响而已。x86 指令指示符寄存器 118 持有下一个待执行的 x86ISA 指令 124 的存储器地址,ARM 程序计数器寄存器 116 持有下一个待执行的 ARMISA 指令 124 的存储器地址。为了控制程序流程,微处理器 100 在其执行 x86 与 ARM 机器语言程序时,分别更新 x86 指令指示符寄存器 118 与 ARM 程序计数器寄存器 116,至下一个指令、分支指令的目标地址或是例外处理程序地址。在微处理器 100 执行 x86 与 ARM ISA 的机器语言程序的指令时,微处理器 100 由系统存储器撷取机器语言程序的指令集架构的指令,并将其置入指令高速缓冲存储器 102 以取代最近较不被撷取与执行的指令。此指令撷取单元 114 基于 x86 指令指示符寄存器 118 或是 ARM 程序计数器寄存器 116 的数值,并依据指令模式指示符 132 指示微处理器 100 正在撷取的 ISA 指令 124 是 x86 或是 ARM 模式来产生撷取地址 134。在一实施例中,x86 指令指示符寄存器 118 与 ARM 程序计数器寄存器 116 可实施为一共享的硬件指令地址寄存器,用以提供其内容至指令撷取单元与分支预测器 114 并由执行管线 112 依据指令模式指示符 132 指示的模式是 x86 或 ARM 与 x86 或 ARM 的语意 (semantics) 来进行更新。

[0128] 环境模式指示符 136 是一状态指示微处理器 100 是使用 x86 或 ARMISA 的语意于此微处理器 100 所操作的多种执行环境,例如虚拟存储器、例外事件、高速缓冲存储器控制、与全域执行时间保护。因此,指令模式指示符 132 与环境模式指示符 136 共同产生多个执行模式。在第一种模式中,指令模式指示符 132 与环境模式指示符 136 都指向 x86ISA,微处理器 100 是作为一般的 x86ISA 处理器。在第二种模式中,指令模式指示符 132 与环境模式指示符 136 都指向 ARM ISA,微处理器 100 是作为一般的 ARM ISA 处理器。在第三种模式中,指令模式指示符 132 指向 x86ISA,不过环境模式指示符 136 则是指向 ARM ISA,此模式有利于在 ARM 操作系统或是超管理器的控制下执行使用者模式 x86 机器语言程序;相反地,在第四种模式中,指令模式指示符 132 是指向 ARM ISA,不过环境模式指示符 136 则是指向 x86 ISA,此模式有利于在 x86 操作系统或超管理器的控制下执行使用者模式 ARM 机器语言程序。指令模式指示符 132 与环境模式指示符 136 的数值在重置 (reset) 之初就已确定。在一实施例中,此初始值被视为微码常数进行编码,不过可通过熔断配置熔丝与 / 或使用微码修补进行修改。在另一实施例中,此初始值则是由一外部输入提供至微处理器 100。在一实施例中,环境模式指示符 136 只在由一重置至 ARM (reset-to-ARM) 指令 124 或是一重置至 x86 (reset-to-x86) 指令 124 执行重置后才会改变(请参照下述图 6A 以及图 6B);也即,在微处理器 100 正常运行而未由一般重置、重置至 x86 或重置至 ARM 指令 124 执行重置时,环境模式指示符 136 并不会改变。

[0129] 硬件指令转译器 104 接收 x86 与 ARM ISA 的机器语言指令 124 作为输入,相应地提供一个或多个微指令 126 作为输出信号以实现 x86 或 ARM ISA 指令 124。执行管线 112 执行前述一个或多个微指令 126,其集体执行的结果实现 x86 或 ARM ISA 指令 124。也就是说,这些微指令 126 的集体执行可依据输入端所指定的 x86 或 ARM ISA 指令 124,来执行 x86

或是 ARM ISA 指令 124 所指定的操作,以产生 x86 或 ARM ISA 指令 124 所定义的结果。因此,硬件指令转译器 104 将 x86 或 ARM ISA 指令 124 转译为一个或多个微指令 126。硬件指令转译器 104 包含一组电晶体,以一预设方式进行配置来将 x86ISA 与 ARM ISA 的机器语言指令 124 转译为实行微指令 126。硬件指令转译器 104 并具有布林逻辑闸以产生实行微指令 126(如图 2 所示的简单指令转译器 204)。在一实施例中,硬件指令转译器 104 并具有一微码只读存储器(如图 2 中复杂指令转译器 206 的元件 234)。硬件指令转译器 104 利用此微码只读存储器,并依据复杂 ISA 指令 124 产生实行微指令 126,这部分将在图 2 的说明内容会有进一步的说明。就一较佳实施例而言,硬件指令转译器 104 不必然要能转译 x86 使用者操作手册或是 ARM 使用者操作手册所定义的整个 ISA 指令 124 集,而只要能够转译这些指令的一个子集合即可。具体来说,由 x86 使用者操作手册定义且由硬件指令转译器 104 转译的 ISA 指令 124 的子集合,并不必然对应至任何 Intel 开发的既有 x86ISA 处理器,而由 ARM 使用者操作手册定义且由硬件指令转译器 104 转译的 ISA 指令 124 的子集合并不必然对应至任何由 ARM Ltd. 开发的既有的 ISA 处理器。前述一个或多个用以实现 x86 或 ARM ISA 指令 124 的实行微指令 126,可由硬件指令转译器 104 一次全部提供至执行管线 112 或是依序提供。本实施例的优点在于,硬件指令转译器 104 可将实行微指令 126 直接提供至执行管线 112 执行,而不需要将这些微指令 126 存储于设置两者间的存储器。在图 1 的微处理器 100 的实施例中,当微处理器 100 执行 x86 或是 ARM 机器语言程序时,微处理器 100 每一次执行 x86 或是 ARM 指令 124 时,硬件指令转译器 104 就会将 x86 或 ARM 机器语言指令 124 转译为一个或多个微指令 126。不过,图 8 的实施例则是利用一微指令高速缓冲存储器以避免微处理器 100 每次执行 x86 或 ARM ISA 指令 124 所会遭遇到的重复转译的问题。硬件指令转译器 104 的实施例在图 2 会有更详细的说明。

[0130] 执行管线 112 执行由硬件指令转译器 104 提供的实行微指令 126。基本上,执行管线 112 是一通用高速微指令处理器。虽然本文所描述的功能由具有 x86/ARM 特定特征的执行管线 112 执行,但大多数 x86/ARM 特定功能其实是由此微处理器 100 的其他部分,如硬件指令转译器 104,来执行。在一实施例中,执行管线 112 执行由硬件指令转译器 104 接收到的实行微指令 126 的寄存器重命名、超纯量技术、与非循序执行。执行管线 112 在图 4 会有更详细的说明。

[0131] 微处理器 100 的微架构包含:(1) 微指令集;(2) 微指令集的微指令 126 所能取用的资源集,此资源集是 x86 与 ARM ISA 的资源的超集合 (superset);以及 (3) 微处理器 100 相应于微指令 126 的执行所定义的微例外事件 (micro-exception) 集,此微例外事件集是 x86ISA 与 ARM ISA 的例外事件的超集合。此微架构不同于 x86ISA 与 ARM ISA。具体来说,此微指令集在许多面向是不同于 x86ISA 与 ARM ISA 的指令集。首先,微指令集的微指令指示执行管线 112 执行的操作与 x86ISA 与 ARM ISA 的指令集的指令指示微处理器执行的操作并非一对一对。虽然其中许多操作相同,不过,仍有一些微指令集指定的操作并非 x86ISA 和 / 或 ARM ISA 指令集所指定。相反地,有一些 x86ISA 和 / 或 ARM ISA 指令集特定的操作并非微指令集所指定。其次,微指令集的微指令是以不同于 x86ISA 与 ARM ISA 指令集的指令的编码方式进行编码。亦即,虽然有许多相同的操作(如:相加、偏移、载入、返回)在微指令集以及 x86 与 ARM ISA 指令集中都有指定,微指令集与 x86 或 ARM ISA 指令集的二进制操作码值对应表并没有一对一对应。微指令集与 x86 或 ARM ISA 指令集的二进制操作码值

对应表相同通常也是巧合，其间仍不具有一对一的对应关系。第三，微指令集的微指令的比特栏与 x86 或是 ARM ISA 指令集的指令比特栏也不是一对一对应。

[0132] 整体而言，微处理器 100 可执行 x86ISA 与 ARM ISA 机器语言程序指令。然而，执行管线 112 本身无法执行 x86 或 ARM ISA 机器语言指令；而是执行由 x86ISA 与 ARM ISA 指令转译成的微处理器 100 微架构的微指令集的实行微指令 126。然而，虽然此微架构与 x86ISA 以及 ARM ISA 不同，本发明也提出其他实施例将微指令集与其他微架构特定的资源系开放给使用者。在这些实施例中，此微架构可有效地作为在 x86ISA 与 ARM ISA 外的一个具有微处理器所能执行的机器语言程序的第三 ISA。

[0133] 下表(表一)描述本发明微处理器 100 的一实施例的微指令集的微指令 126 的一些比特栏。

[0134]

<u>比特栏</u>	<u>描述</u>
Opcode	将要执行的操作（参照下列指令）
Destination	指定微指令结果的目的寄存器
source 1	指定第一输入操作数的来源（例如通用寄存器、浮点寄存器、架构特定寄存器、条件旗标寄存器、立即数据、移位数据、可用常数、下一个序列的指令指示符的数值）
source 2	指定第二输入寄存器的来源
source 3	指定第三输入寄存器的来源（不能是 GPR 或 FPR）
condition code	条件，此条件满足时将会执行操作，不满足时就不执行操作
operand size	微指令使用的操作数的比特的编码数
address size	微指令产生的地址的比特的编码数
top of x87 FP register stack	需要 x87 型式的浮点指令

[0135] 表一

[0136] 下表(表二)描述本发明微处理器 100 的一实施例的微指令集的一些微指令。

[0137]

<u>指令</u>	<u>描述</u>
ALU-type	例如：加、减、转动、移位、布林、乘、除、浮点

[0138]

	ALU、介质类型 ALU (如 packed 操作)
load/store	从存储器载入寄存器/从寄存器存储至存储器
conditional jump	在条件满足时跳到目标地址, 此条件如: 零、大于、不等于; 由 ISA 旗标或微架构特定(而非 ISA 可见)条件旗标
Move	将数值从来源寄存器移动至目的寄存器
conditional move	在条件满足时, 将数值从来源寄存器移动至目的寄存器
move to control register	将数值从通用寄存器移动至控制寄存器
move from control register	将数值从控制寄存器移动至通用寄存器
Gprefetch	经保证的高速缓冲存储器线预撷取指令 (也即, 并非暗示, 除非出现特定例外事件, 否则就会预撷取。)
Grabline	于处理器汇流排执行零拍读取无效周期(zero beat read-invalidate cycle), 以取得高速缓冲存储器线的排他所有权, 而不需从系统存储器读取数据 (因为整个高速缓冲存储器线将会被写入)
load pram	(未见于 ISA 的私有微架构特定 RAM, 这在下文会有进一步描述)载入寄存器
store pram	存储至 PRAM
jump condition on/off	在“静态”条件满足时 (在相关时间表、程序员保证不会有更旧的、未引退出的微指令来改变此“静态”条件), 跳跃至目标地址; 利用复杂的指令转译器, 而非执行管线来解决, 执行速度较快
Call	呼叫副程序
Return	从副程序返回
set bit on/off	在寄存器中设定/清除比特

[0139]

copy bit	从来源寄存器复制比特数值至目的寄存器
branch to next sequential instruction pointer	在 x86 或 ARM ISM 指令转译为微指令后，分支到下一个 x86 或 ARM 的 ISA 指令序列
Fence	等待到所有微指令都从执行管线被清除，并且执行此微指令后的微指令
indirect jump	依据一寄存器数值进行非条件跳跃

[0140] 表二

[0141] 微处理器 100 也包含一些微架构特定的资源,如微架构特定的通用寄存器、介质寄存器与区段寄存器(如用于重命名的寄存器或由微码所使用的寄存器)以及未见于 x86 或 ARM ISA 的控制寄存器,以及一私有随机存取存储器 (PRAM)。此外,此微架构可产生例外事件,也即前述的微例外事件。这些例外事件未见于 x86 或 ARM ISA 或是由它们所指定,而通常是微指令 126 与相关微指令 126 的重新执行。举例来说,这些情形包含:载入错过 (load miss) 的情况,其是执行管线 112 假设载入动作并于错过时重新执行此载入微指令 126;错过转译后备缓冲区 (TLB),在查表 (page table walk) 与转译后备缓冲区填满后,重新执行微指令 126;浮点微指令 126 接收一异常操作数 (denormal operand) 但此操作数被评估为正常,需在执行管线 112 正常化此操作数后重新执行微指令 126;一载入微指令 126 执行后侦测到一个更早的存储微指令 126 与其地址冲突 (address-colliding) 需要重新执行此载入微指令 126。需理解的是,本文表一所列的比特栏,表二所列的微指令,以及微架构特定的资源与微架构特定的例外事件,只是作为例示说明本发明的微架构,而非穷尽本发明的所有可能实施例。

[0142] 寄存器文件 106 包含微指令 126 所使用的硬件寄存器,以持有资源与 / 或目的操作数。执行管线 112 将其结果 128 写入寄存器文件 106,并由寄存器文件 106 为微指令 126 接收操作数。硬件寄存器是引用 (instantiate) x86ISA 定义与 ARM ISA 定义通用寄存器是共享寄存器文件 106 中的一些寄存器。举例来说,在一实施例中,寄存器文件 106 是引用十五个 32 比特的寄存器,由 ARM ISA 寄存器 R0 至 R14 以及 x86ISA 累积寄存器 (EAX register) 至 R14D 寄存器所共享。因此,若是一第一微指令 126 将一数值写入 ARM R2 寄存器,随后一后续的第二微指令 126 读取 x86 累积寄存器将会接收到与第一微指令 126 写入相同的数值,反之亦然。此技术特征有利于使 x86ISA 与 ARM ISA 的机器语言程序得以快速通过寄存器进行沟通。举例来说假设在 ARM 机器语言操作系统执行的 ARM 机器语言程序能够使指令模式 132 改变为 x86ISA,并将控制权转换至一 x86 机器语言程序以执行特定功能,因为 x86ISA 可支援一些指令,其执行操作的速度快于 ARM ISA,在这种情形下将有利于执行速度的提升。ARM 程序可通过寄存器文件 106 的共享寄存器提供需要的数据给 x86 执行程序。反之,x86 执行程序可将执行结果提供至寄存器文件 106 的共享寄存器内,以使 ARM 程序在 x86 执行程序回复后可见到此执行结果。相似地,在 x86 机器语言操作系统执行的 x86 机器

语言程序可使指令模式 132 改变为 ARM ISA 并将控制权转换至 ARM 机器语言程序；此 x86 程序通过寄存器文件 106 的共享寄存器提供所需的数据给 ARM 执行程序，而此 ARM 执行程序可通过寄存器文件 106 的共享寄存器提供执行结果，以使 x86 程序在 ARM 执行程序回复后可见到此执行结果。因为 ARM R15 寄存器是一独立引用的 ARM 程序计数器寄存器 116，因此，引用 x86R15D 寄存器的第十六 32 比特寄存器并不分享给 ARM R15 寄存器。此外，在一实施例中，x86 的十六个 128 比特 XMM0 至 XMM15 寄存器与十六个 128 比特先进单指令多重数据扩展 (Advanced SIMD (“Neon”)) 寄存器的 32 比特区段是分享给三十二个 32 比特 ARM VFPv3 浮点寄存器。寄存器文件 106 也引用旗标寄存器 (即 x86EFLAGS 寄存器与 ARM 条件旗标寄存器)，以及 x86ISA 与 ARM ISA 所定义的多种控制权与状态寄存器，这些架构控制与状态寄存器包括 x86 架构的特定模型寄存器 (model specific registers, MSRs) 与保留给 ARM 架构的协同处理器 (8-15) 寄存器。此寄存器文件 106 也引用非架构寄存器，如用于寄存器重命名或是由微码 234 所使用的非架构通用寄存器，以及非架构 x86 特定模型寄存器与实作定义的或是由制造商指定的 ARM 协同处理器寄存器。寄存器文件 106 在图 5 会有更进一步的说明。

[0143] 存储器子系统 108 包含一由高速缓冲存储器存储器构成的高速缓冲存储器存储器阶层架构(在一实施例中包含第 1 层 (level-1) 指令高速缓冲存储器 102、第 1 层 (level-1) 数据高速缓冲存储器与第 2 层混合高速缓冲存储器)。此存储器子系统 108 并包含多种存储器请求队列，如载入、存储、填入、窥探、合并写入归并缓冲区。存储器子系统也包含一存储器管理单元 (MMU)。存储器管理单元具有转译后备缓冲区 (TLBs)，尤以独立的指令与数据转译后备缓冲区为佳。存储器子系统还包含一查表引擎 (table walk engine) 以获得虚拟与实体地址间的转译，来回应转译后备缓冲区的错失。虽然在图 1 中指令高速缓冲存储器 102 与存储器子系统 108 是显示为各自独立，不过，在逻辑上，指令高速缓冲存储器 102 也是存储器子系统 108 的一部分。存储器子系统 108 是设定来使 x86 与 ARM 机器语言程序分享一共同的记忆空间，以使 x86 与 ARM 机器语言程序容易通过存储器互相沟通。

[0144] 存储器子系统 108 得知指令模式 132 与环境模式 136，使其能够在适当 ISA 内容中执行多种操作。举例来说，存储器子系统 108 依据指令模式指示符 132 指示为 x86 或 ARM ISA，来执行特定存储器存取违规的检验(例如过限检验 (limit violation check))。在一实施例中，回应环境模式指示符 136 的改变，存储器子系统 108 会更新 (flush) 转译后备缓冲区；不过在指令模式指示符 132 改变时，存储器子系统 108 并不相应地更新转译后备缓冲区，以在前述指令模式指示符 132 与环境模式指示符 136 分指 x86 与 ARM 的第三与第四模式中提供较佳的效能。在一实施例中，回应一转译后备缓冲区错失 (TKB miss)，查表引擎依据环境模式指示符 136 指示为 x86 或 ARM ISA，从而决定利用 x86 分页表或 ARM 分页表执行一分页查表动作以取出转译后备缓冲区。在一实施例中，若是环境状态指示符 136 指示为 x86ISA，存储器次系统 108 检查会影响高速缓冲存储器策略的 x86ISA 控制寄存器(如 CROCD 与 NW 比特)的架构状态；若是环境模式指示符 136 指示为 ARM ISA，则检查相关的 ARM ISA 控制寄存器(如 SCTRLR I 与 C 比特)的架构模式。在一实施例中，若是状态指示符 136 指示为 x86ISA，存储器子系统 108 检查会影响存储器管理的 x86ISA 控制寄存器(如 CROPG 比特)的架构状态；若是环境模式指示符 136 指示为 ARM ISA，则检查相关的 ARM ISA 控制寄存器(如 SCTRLR M 比特)的架构模式。在一实施例中，若是状态指示符 136 指

示为 x86ISA, 存储器次系统 108 检查会影响对准检测的 x86ISA 控制寄存器(如 CROAM 比特)的架构状态, 若是环境模式指示符 136 指示为 ARM ISA, 则检查相关的 ARM ISA 控制寄存器(如 SCTRLR A 比特)的架构模式。在另一实施例中, 若是状态指示符 136 指示为 x86ISA, 存储器子系统 108 (以及用于特权指令的硬件指令转译器 104) 检查当前所指定特权级 (CPL) 的 x86ISA 控制寄存器的架构状态;若是环境模式指示符 136 指示为 ARM ISA, 则检查指示使用者或特权模式的相关 ARM ISA 控制寄存器的架构模式。不过, 在一实施例中, x86ISA 与 ARM ISA 是分享微处理器 100 中具有相似功能的控制比特组 / 寄存器, 微处理器 100 并不对各个指令集架构引用独立的控制比特组 / 寄存器。

[0145] 虽然配置寄存器 122 与寄存器文件 106 在图示中是各自独立, 不过配置寄存器 122 可被理解为寄存器文件 106 的一部分。配置寄存器 122 具有一全域配置寄存器, 用以控制微处理器 100 在 x86ISA 与 ARM ISA 各种不同面向的操作, 例如使多种特征生效或失效的功能。全域配置寄存器可使微处理器 100 执行 ARM ISA 机器语言程序的能力失效, 即让微处理器 100 成为一个仅能执行 x86 指令的微处理器 100, 并可使其他相关且专属于 ARM 的能力(如启动 x86(launch-x86) 与重置至 x86(reset-to-x86) 的指令 124 与本文所称的实作定义(implementation-defined) 协同处理器寄存器)失效。全域配置寄存器也可使微处理器 100 执行 x86ISA 机器语言程序的能力失效, 也即让微处理器 100 成为一个仅能执行 ARM 指令的微处理器 100, 并可使其他相关的能力(如启动 ARM 与重置至 ARM 的指令 124 与本文所称的新的非架构特定模型寄存器)失效。在一实施例中, 微处理器 100 在制造时具有预设的配置设定, 如微码 234 中的硬式编码值, 此微码 234 在启动时利用此硬式编码值来设定微处理器 100 的配置, 例如写入编码寄存器 122。不过, 部分编码寄存器 122 是以硬件而非以微码 234 进行设定。此外, 微处理器 100 具有多个熔丝, 可由微码 234 进行读取。这些熔丝可被熔断以修改预设配置值。在一实施例中, 微码 234 读取熔丝值, 对预设值与熔丝值执行一互斥或操作, 并将操作结果写入配置寄存器 122。此外, 对于熔丝值修改的效果可利用一微码 234 修补而恢复。在微处理器 100 能够执行 x86 与 ARM 程序的情况下, 全域配置寄存器可用于确认微处理器 100 (或如图 7 所示处理器的一多核心部分的一特定核心 100) 在重置或如图 6A 或图 6B 所示在回应 x86 形式的 INIT 指令时, 会以 x86 微处理器的形态还是以 ARM 微处理器的形态进行开机。全域配置寄存器并具有一些比特提供起始预设值给特定的架构控制寄存器, 如 ARMISA SCTLT 与 CPACR 寄存器。图 7 所示的多核心的实施例中仅具有一个全域配置寄存器, 即使各核心的配置可分别设定, 如在指令模式指示符 132 与环境模式指示符 136 都设定为 x86 或 ARM 时, 选择以 x86 核心或是 ARM 核心 开机。此外, 启动 ARM 指令 126 与启动 x86 指令 126 可用以在 x86 与 ARM 指令模式 132 间动态切换。在一实施例中, 全域配置寄存器可通过一 x86RDMSR 指令对一新的非架构特定模型寄存器进行读取, 并且其中部分的控制比特可通过 x86WRMSR 指令对前述新的非架构特定模型寄存器的写入来进行写入操作。全域配置寄存器还可通过 ARM MCR/MCRR 指令对一对应至前述新的非架构特定模型寄存器的 ARM 协处理器寄存器进行读取, 而其中部分的控制比特可通过 ARM MRC/MMRC 指令对应至此新的非架构特定模型寄存器的 ARM 协处理器寄存器的写入来进行写入操作。

[0146] 配置寄存器 122 并包含多种不同的控制寄存器从不同面向控制微处理器 100 的操作。这些非 x86 (non-x86)/ARM 的控制寄存器包括本文所称的全域控制寄存器、非指令集

架构控制寄存器、非 x86/ARM 控制寄存器、通用控制寄存器、以及其他类似的寄存器。在一实施例中，这些控制寄存器可利用 x86RDMSR/WRMSR 指令至非架构特定模型寄存器 (MSRs) 进行存取、以及利用 ARM MCR/MRC(或 MCRR/MRRC) 指令至新实作定义的协同处理器寄存器进行存取。举例来说，微处理器 100 包含非专属于 x86/ARM 的控制寄存器，以确认微型 (fine-grained) 高速缓冲存储器控制，此微型高速缓冲存储器控制小于 x86ISA 与 ARM ISA 控制寄存器所能提供者。

[0147] 在一实施例中，微处理器 100 提供 ARM ISA 机器语言程序通过实作定义 ARM ISA 协同处理器寄存器存取 x86ISA 特定模型寄存器，这些实作定义 ARM ISA 协同处理器寄存器是直接对应于相对应的 x86 特定模型寄存器。此特定模型寄存器的地址指定于 ARM ISA R1 寄存器。此数据由 MRC/MRRC/MCR/MCRR 指令所指定的 ARM ISA 寄存器读出或写入。在一实施例中，特定模型寄存器的一子集合是以密码保护，也即指令在尝试存取特定模型寄存器时必须使用密码。在此实施例中，密码指定于 ARM R7:R6 寄存器。若是此存取动作导致 x86 通用保护错误，微处理器 100 随即产生一 ARM ISA 未定义指令中止模式 (UND) 例外事件。在一实施例中，ARM 协同处理器 4 (地址为 :0, 7, 15, 0) 存取相对应的 x86 特定模型寄存器。

[0148] 微处理器 100 并包含一个耦接至执行管线 112 的中断控制器 (未图示)。在一实施例中，此中断控制器是一 x86 型式的先进可程序化中断控制器 (APIC)。中断控制器将 x86ISA 中断事件对应至 ARM ISA 中断事件。在一实施例中，x86INTR 是对应至 ARM IRQ 中断事件；x86NMI 是对应至 ARM FIQ 中断事件；x86INIT 在微处理器 100 启动时引发起动重置循序过程 (INIT-reset sequence)，无论那一个指令集架构 (x86 或 ARM) 原本是由硬件重置启动的；x86SMI 是对应至 ARM FIQ 中断事件；以及 x86STPCLK、A20、Thermal、PREQ、与 Rebranch 则不对应至 ARM 中断事件。ARM 机器语言能通过新的实作定义的 ARM 协同处理器寄存器存取先进可程序化中断控制器的功能。在一实施例中，APIC 寄存器地址指定于 ARM R0 寄存器，此 APIC 寄存器的地址与 x86 的地址相同。在一实施例中，ARM 协同处理器 6 是通常用于操作系统通常需执行的特权模式功能 (privileged mode functions)。此 ARM 协同处理器 6 的地址为 :0, 7, nn, 0；其中 nn 是 15 时可存取先进可程序化中断控制器；nn 是 12-14 以存取汇流排介面单元 (BIU) 藉以在处理器汇流排上执行 8 比特、16 比特与 32 比特输入 / 输出循环。微处理器 100 并包含一汇流排介面单元 (未图示)，此汇流排介面单元耦接至存储器子系统 108 与执行管线 112，作为微处理器 100 与处理器汇流排的介面。在一实施例中，处理器汇流排符合一个 Intel Pentium 微处理器家族的微处理器汇流排的规格。ARM 机器语言程序可通过新的实作定义的 ARM 协同处理器寄存器存取汇流排介面单元的功能可以在处理器汇流排上产生输入 / 输出循环，即由输入输出汇流排传送至输入输出空间的一特定地址，藉以与系统晶片组沟通。举例来说，ARM 机器语言程序可产生一 SMI 认可的特定循环或是关于 C 状态转换的输入输出循环。在一实施例中，输入输出地址指定于 ARM R0 寄存器。在一实施例中，微处理器 100 具有电力管理能力，如已知的 P-state 与 C-state 管理。ARM 机器语言程序可通过新的实作定义 ARM 同协处理器寄存器执行电力管理。在一实施例中，微处理器 100 包含一加密单元 (未图示)，此加密单元位于执行管线 112 内。在一实施例中，此加密单元实质上类似于具有 Padlock 安全科技功能的 VIA 微处理器的加密单元。ARM 机器语言程序能通过新的实作定义的 ARM 协同处理器寄存器取得加密单元的功能，如加密指令。在一实施例中，ARM 协同处理器 5 系用于通常由使用者模式应用程序执行的使

用者模式功能,例如那些使用加密单元的技术特征所产生的功能。

[0149] 在微处理器 100 执行 x86ISA 与 ARM ISA 机器语言程序时,每一次微处理器 100 执行 x86 或是 ARM ISA 指令 124,硬件指令转译器 104 就会执行硬件转译。反之,采用软件转译的系统则能在多个事件中重复使用同一个转译,而非对之前已转译过的机器语言指令重复转译,因而有助于改善效能。此外,图 8 的实施例使用微指令高速缓冲存储器以避免微处理器每一次执行 x86 或 ARM ISA 指令 124 时可能发生的重复转译动作。本发明的前述各个实施例所描述的方式配合不同的程序的特征及其执行环境,因此确实有助于改善效能。

[0150] 分支预测器 114 存取之前执行过的 x86 与 ARM 分支指令的历史数据。分支预测器 114 依据之前的高速缓冲存储器历史数据,来分析由指令高速缓冲存储器 102 所取得高速缓冲存储器线是否存在 x86 与 ARM 分支指令以及其目标地址。在一实施例中,高速缓冲存储器历史数据包含分支指令 124 的存储器地址、分支目标地址、一个方向指示符、分支指令的种类、分支指令在高速缓冲存储器线的起始比特组、以及一个显示是否横跨多个高速缓冲存储器管线的指标。在一实施例中,如 2011 年 4 月 7 日提出的美国第 61/473,067 号临时申请案“APPARATUS AND METHOD FOR USING BRANCHPREDICTION TO EFFICIENTLY EXECUTE CONDITIONAL NON-BRANCHINSTRUCTIONS”,其提供改善分支预测器 114 的效能以使其能预测 ARM ISA 条件非分支指令方向的方法。在一实施例中,硬件指令转译器 104 并包含一静态分支预测器,可依据执行码、条件码的类型、向后 (backward) 或向前 (forward) 等等数据,预测 x86 与 ARM 分支指令的方向与分支目标地址。

[0151] 本发明也涉及多种不同的实施例以实现 x86ISA 与 ARM ISA 定义的不同特征的组合。举例来说,在一实施例中,微处理器 100 实现 ARM、Thumb、ThumbEE 与 Jazelle 指令集状态,但对 Jazelle 扩充指令集则是提供无意义的实现 (trivial implementation);微处理器 100 并实现下述扩充指令集,包含:Thumb-2、VFPv3-D32、先进单指令多重数据 (Advanced SIMD (Neon))、多重处理、与 VMSA;但不实现下述扩充指令集,包含:安全性扩充、快速内容切换扩充、ARM 除错 (ARM 程序可通过 ARM MCR/MRC 指令至新的实作定义协处理器寄存器取得 x86 除错功能)、效能侦测计数器 (ARM 程序可通过新的实作定义协处理器寄存器取得 x86 效能计数器)。举例来说,在一实施例中,微处理器 100 将 ARM SETEND 指令视为一无操作指令 (NOP) 并且只支持 Little-endian 数据格式。在另一实施例中,微处理器 100 并不实现 x86SSE4.2 的功能。

[0152] 本发明考量多个实施例的微处理器 100 的改良,例如对台湾台北的威盛电子股份有限公司所生产的商用微处理器 VIA NanoTM进行改良。此 Nano 微处理器能够执行 x86ISA 机器语言程序,但无法执行 ARM ISA 机器语言程序。Nano 微处理器包含高效能寄存器重命名、超纯量指令技术、非循序执行管线与一硬件转译器以将 x86ISA 指令转译为微指令供执行管线执行。本发明对于 Nano 硬件指令转译器的改良,使其除了可转译 x86 机器语言指令外,还可将 ARM ISA 机器语言指令转译为微指令供执行管线执行。硬件指令转译器的改良包含简单指令转译器的改良与复杂指令转译器的改良,也包含微码在内。此外,微指令集可加入新的微指令以支持 ARM ISA 机器语言指令与微指令间的转译,并可改善执行管线使能执行新的微指令。此外, Nano 寄存器文件与存储器子系统也可经改善使其能支持 ARM ISA,也包含特定寄存器的共享。分支预测单元可通过改善使其在 x86 分支预测外,也能适用于 ARM 分支指令预测。此实施例的优点在于,因为在很大的程度上于 ISA 无关 (largely

ISA-agnostic) 的限制,因而只需对于 Nano 微处理器的执行管线进行轻微的修改,即可适用于 ARM ISA 指令。对于执行管线的改良包含条件码旗标的产生与使用方式、用以更新与回报指令指示符寄存器的语意、存取特权保护方法、以及多种存储器管理相关的功能,如存取违规检测、分页与转译后备缓冲区 (TLB) 的使用、与高速缓冲存储器策略等。前述内容仅为示意,而非限定本案发明,其中部分特征在后续内容会有进一步的说明。最后,如前述,x86ISA 与 ARM ISA 定义的部分特征可能无法为前述对 Nano 微处理器进行改良的实施例所支持,这些特征如 x86SSE4.2 与 ARM 安全性扩充、快速内容切换扩充、除错与效能计数器,其中部分特征在后续内容会有更进一步的说明。此外,前述通过对于 Nano 处理器的改良以支持 ARM ISA 机器语言程序,为一整合使用设计、测试与制造资源以完成能够执行 x86 与 ARM 机器语言程序的单积体电路产品的实施例,此单积体电路产品涵盖市场绝大多数既存的机器语言程序,而符合现今市场潮流。本文所述的微处理器 100 的实施例实质上可被配置为 x86 微处理器、ARM 微处理器、或是可同时执行 x86ISA 与 ARM ISA 机器语言程序微处理器。此微处理器可通过在单一微处理器 100 (或是图 7 的核心 100) 上的 x86 与 ARM 指令模式 132 间的动态切换以取得同时执行 x86ISA 与 ARM ISA 机器语言程序的能力,也可通过将多核心微处理 100 (对应于图 7 所示) 的一个或多个核心配置为 ARM 核心而一个或多个核心配置为 x86 核心,也即通过在多核心 100 的每一个核心上进行 x86 与 ARM 指令间的动态切换,以取得同时执行 x86ISA 与 ARM ISA 机器语言程序的能力。此外,传统上,ARM ISA 核心被设计作为知识产权核心,而被各个第三 者协力厂商纳入其应用,如系统晶片与 / 或嵌入式应用。因此,ARM ISA 并不具有一特定的标准处理器汇流排,作为 ARM 核心与系统的其他部分(如晶片组或其他周边设备)间的介面。有利的是,Nano 处理器已具有一高速 x86 型式处理器汇流排作为连接至存储器与周边设备的介面,以及一存储器一致性结构可协同微处理器 100 在 x86 电脑系统环境下支持 ARM ISA 机器语言程序的执行。

[0153] 请参照图 2,图中是以方块图详细显示图 1 的硬件指令转译器 104。此硬件指令转译器 104 包含硬件,更具体来说,就是电晶体的集合。硬件指令转译器 104 包含一指令格式化程序 202,由图 1 的指令高速缓冲存储器 102 接收指令模式指示符 132 以及 x86ISA 与 ARM ISA 指令比特组 124 的区块,并输出格式化的 x86ISA 与 ARM ISA 指令 242;一简单指令转译器 (SIT) 204 接收指令模式指示符 132 与环境模式指示符 136,并输出实行微指令 244 与一微码地址 252;一复杂指令转译器 (CIT) 206 (也称为一微码单元),接收微码地址 252 与环境模式指示符 136,并提供实行微指令 246;以及一多工器 212,其一输入端由简单指令转译器 204 接收微指令 244,另一输入端由复杂指令转译器 206 接收微指令 246,并提供实行微指令 126 至图 1 的执行管线 112。指令格式化程序 202 在图 3 会有更详细的说明。简单指令转译器 204 包含一 x86 简单指令转译器 222 与一 ARM 简单指令转译器 224。复杂指令转译器 206 包含一接收微码地址 252 的微程序计数器 232,一由微程序计数器 232 接收只读存储器地址 254 的微码只读存储器 234,一用以更新微程序计数器的微序列器 236、一指令间接寄存器 (instruction indirection register, IIR) 235、以及一用以产生复杂指令转译器所输出的实行微指令 246 的微转译器 (microtranslator) 237。由简单指令转译器 204 所产生的实行微指令 244 与由复杂指令转译器 206 所产生的实行微指令 246 都属于微处理器 100 的微架构的微指令集的微指令 126,并且都可直接由执行管线 112 执行。

[0154] 多工器 212 受到一选择输入 248 所控制。一般的时候,多工器 212 会选择来自简

单指令转译器 204 的微指令 ;然而,当简单指令转译器 204 遭遇一复杂 x86 或 ARM ISA 指令 242,而将控制权转移、或遭遇陷阱 (traps)、以转移至复杂指令转译器 206 时,简单指令转译器 204 控制选择输入 248 让多工器 212 选择来自复杂指令转译器的微指令 246。当寄存器分配表 (RAT) 402 (请参照图 4) 遭遇到一个微指令 126 具有一特定比特组指出其为实现复杂 ISA 指令 242 序列的最后一个微指令 126 时,寄存器分配表 402 随即控制选择输入 248 使多工器 212 恢复至选择来自简单指令转译器 204 的微指令 244。此外,当重排缓冲器 422 (请参照图 4) 准备要使微指令 126 引退且该指令的状态指出需要选择来自复杂指令器的微指令时,重排缓冲器 422 控制选择输入 248 使多工器 212 选择来自复杂指令转译器 206 的微指令 246。前述需引退微指令 126 的情形如 :微指令 126 已经导致一例外条件产生。

[0155] 简单指令转译器 204 接收 ISA 指令 242,并且在指令模式指示符 132 指示为 x86 时,将这些指令视为 x86ISA 指令进行解码,而在指令模式指示符 132 指示为 ARM 时,将这些指令视为 ARM ISA 指令进行解码。简单指令转译器 204 并确认此 ISA 指令 242 为简单或是复杂 ISA 指令。简单指令转译器 204 能够为简单 ISA 指令 242,输出所有用以实现此 ISA 指令 242 的实行微指令 126 ;也就是说,复杂指令转译器 206 并不提供任何实行微指令 126 给简单 ISA 指令 124。反之,复杂 ISA 指令 124 要求复杂指令转译器 206 提供至少部分(若非全部)的实行微指令 126。在一实施例中,对 ARM 与 x86ISA 指令集的指令 124 的子集合而言,简单指令转译器 204 输出部分实现 x86/ARMISA 指令 126 的微指令 244,随后将控制权转移至复杂指令转译器 206,由复杂指令转译器 206 接续输出剩下的微指令 246 来实现 x86/ARM ISA 指令 126。多工器 212 受到控制,首先提供来自简单指令转译器 204 的实行微指令 244 作为提供至执行管线 112 的微指令 126,随后提供来自复杂指令转译器 206 的实行微指令 246 作为提供至执行管线 112 的微指令 126。简单指令转译器 204 知道由硬件指令转译器 104 执行,以针对多个不同复杂 ISA 指令 124 产生实行微指令 126 的多个微码程序中的起始微码只读存储器 234 的地址,并且,当简单指令转译器 204 对一复杂 ISA 指令 242 进行解码时,简单指令转译器 204 会提供相对应的微码程序地址 252 至复杂指令转译器 206 的微程序计数器 232。简单指令转译器 204 输出实现 ARM 与 x86ISA 指令集中相当大比例的指令 124 所需的微指令 244,尤其是对于需要由 x86ISA 与 ARM ISA 机器语言程序来说较常执行的 ISA 指令 124,而只有相对少数的指令 124 需要由复杂指令转译器 206 提供实行微指令 246。依据一实施例,主要由复杂指令转译器 206 实现的 x86 指令如 RDMSR/WRMSR、CPUID、复杂运算指令(如 FSQRT 与超越指令 (transcendental instruction))、以及 IRET 指令 ;主要由复杂指令转译器 206 实现的 ARM 指令如 MCR、MRC、MSR、MRS、SRS、与 RFE 指令。前述列出的指令并非限定本案发明,仅例示指出本案复杂指令转译器 206 所能实现的 ISA 指令的种类。

[0156] 当指令模式指示符 132 指示为 x86,x86 简单指令转译器 222 对于 x86ISA 指令 242 进行解码,并且将其转译为实行微指令 244 ;当指令模式指示符 132 指示为 ARM,ARM 简单指令转译器 224 对于 ARM ISA 指令 242 进行解码,并将其转译为实行微指令 244。在一实施例中,简单指令转译器 204 是一可由已知合成工具合成的布林逻辑方块。在一实施例中, x86 简单指令转译器 222 与 ARM 简单指令转译器 224 是独立的布林逻辑方块 ;不过,在另一实施例中, x86 简单指令转译器 222 与 ARM 简单指令转译器 224 位于同一个布林逻辑方块。在一实施例中,简单指令转译器 204 在单一脉冲周期中转译最多三个 ISA 指令 242 并提供最

多六个实行微指令 244 至执行管线 112。在一实施例中，简单指令转译器 204 包含三个次转译器(未图示)，各个次转译器转译单一个格式化的 ISA 指令 242，其中，第一个转译器能够转译需要不多于三个实行微指令 126 的格式化 ISA 指令 242；第二个转译器能够转译需要不多于两个实行微指令 126 的格式化 ISA 指令 242；第三次转译器能后转译需要不多于一个实行微指令 126 的格式化 ISA 指令 242。在一实施例中，简单指令转译器 204 包含一硬件状态机器使其能够在多个时脉周期输出多个微指令 244 以实现一个 ISA 指令 242。

[0157] 在一实施例中，简单指令转译器 204 并依据指令模式指示符 132 与 / 或环境模式指示符 136，执行多个不同的例外事件检测。举例来说，若是指令模式指示符 132 指示为 x86 且 x86 简单指令转译器 222 对一个就 x86ISA 而言是无效的 ISA 指令 124 进行解码，简单指令转译器 204 随即产生一个 x86 无效操作码例外事件；相似地，若是指令模式指示符 132 指示为 ARM 且 ARM 简单指令转译器 224 对一个就 ARM ISA 而言是无效的 ISA 指令 124 进行解码，简单指令转译器 204 随即产生一个 ARM 未定义指令例外事件。在另一实施例中，若是环境模式指示符 136 指示为 x86ISA，简单指令转译器 204 随即检测是否其所遭遇的每个 x86ISA 指令 242 需要一特别特权级 (particular privilege level)，若是，检测当前特权级 (CPL) 是否满足此 x86ISA 指令 242 所需的特别特权级，并于不满足时产生一例外事件；相似地，若是环境模式指示符 136 指示为 ARM ISA，简单指令转译器 204 随即检测是否每个格式化 ARM ISA 指令 242 需要一特权模式指令，若是，检测当前的模式是否为特权 模式，并于现在模式为使用者模式时，产生一例外事件。复杂指令转译器 206 对于特定复杂 ISA 指令 242 也执行类似的功能。

[0158] 复杂指令转译器 206 输出一系列实行微指令 246 至多工器 212。微码只读存储器 234 存储微码程序的只读存储器指令 247。微码只读存储器 234 输出只读存储器指令 247 以回应由微码只读存储器 234 取得的下一个只读存储器指令 247 的地址，并由微程序计数器 232 所持有。一般来说，微程序计数器 232 由简单指令转译器 204 接收其起始值 252，以回应简单指令转译器 204 对于一复杂 ISA 指令 242 的解码动作。在其他情形，例如回应一重置或例外事件，微程序计数器 232 分别接收重置微码程序地址或适当的微码例外事件处理地址。微序列器 236 通常依据只读存储器指令 247 的大小，将微程序计数器 232 更新为微码程序的序列以及选择性地更新为执行管线 112 回应控制型微指令 126(如分支指令)执行所产生的目标地址，以使指向微码只读存储器 234 内的非程序地址的分支生效。微码只读存储器 234 是制造于微处理器 100 的半导体晶片内。

[0159] 除了用来实现简单 ISA 指令 124 或部分复杂 ISA 指令 124 的微指令 244 外，简单指令转译器 204 也产生 ISA 指令信息 255 以写入指令间接寄存器 235。存储于指令间接寄存器 (IIR) 235 的 ISA 指令信息 255 包含关于被转译的 ISA 指令 124 的信息，例如，确认由 ISA 指令所特定的来源与目的寄存器的信息以及 ISA 指令 124 的格式，如 ISA 指令 124 是在存储器的一操作数上或是在微处理器 100 的一架构寄存器 106 内执行。这样可藉此使微码程序能够变为通用，也即不需对于各个不同的来源与 / 或目的架构寄存器 106 使用不同的微码程序。尤其是，简单指令转译器 204 知道寄存器文件 106 的内容，包含哪些寄存器是共享寄存器 504，而能将 x86ISA 与 ARM ISA 指令 124 内提供的寄存器信息，通过 ISA 指令信息 255 的使用，转译至寄存器文件 106 内的适当的寄存器。ISA 指令信息 255 包含一移位栏、一立即栏、一常数栏、各个来源操作数与微指令 126 本身的重命名信息、用以实现 ISA 指令

124的一系列微指令 126 中指示第一个与最后一个微指令 126 的信息、以及存储由硬件指令转译器 104 对 ISA 指令 124 转译时所搜集到的有用信息的其他位元。

[0160] 微转译器 237 由微码只读存储器 234 与间接指令寄存器 235 的内容接收只读存储器指令 247，并相应地产生实行微指令 246。微转译器 237 依据由间接指令寄存器 235 接收的信息，如依据 ISA 指令 124 的格式以及由其所指定 的来源与 / 或目的架构寄存器 106 组合，来将特定只读存储器指令 247 转译为不同的微指令 246 系列。在一些实施例中，许多 ISA 指令信息 255 与只读存储器指令 247 合并以产生实行微指令 246。在一实施例中，各个只读存储器指令 247 大约有 40 比特宽，并且各个微指令 246 大约有 200 比特宽。在一实施例中，微转译器 237 最多能够由一个微读存储器指令 247 产生三个微指令 246。微转译器 237 包含多个布林逻辑以产生实行微指令 246。

[0161] 使用微转译器 237 的优点在于，由于简单指令转译器 204 本身就会产生 ISA 指令信息 255，微码只读存储器 234 不需要存储间接指令寄存器 235 提供的 ISA 指令信息 255，因此可以减少其大小。此外，因为微码只读存储器 234 不需要为了各个不同的 ISA 指令格式、以及各个来源与 / 或目的架构寄存器 106 的组合，提供一独立的程序，微码只读存储器 234 程序可包含较少的条件分支指令。举例来说，若是复杂 ISA 指令 124 是存储器格式，简单指令转译器 204 会产生微指令 244 的逻辑编程，包含将源操作数由存储器载入一暂时寄存器 106 的微指令 244，并且微转译器 237 会产生微指令 246 将结果由暂时寄存器 106 存储至存储器。然而，若是复杂 ISA 指令 124 是寄存器格式 (register form)，此逻辑编程会将源操作数由 ISA 指令 124 所特定的来源寄存器移动至暂时寄存器，并且微转译器 237 会产生微指令 246 用以将结果由暂时寄存器移动至由间接指令寄存器 235 所指定的架构目的寄存器 106。在一实施例中，微转译器 237 的许多面向是类似于 2010 年 4 月 23 日提出的美国专利第 12/766,244 号申请案，在此系列为参考数据。不过，本案的微转译器 237 除了 x86ISA 指令 124 外，也经改良以转译 ARM ISA 指令 124。

[0162] 值得注意的是，微程序计数器 232 不同于 ARM 程序计数器 116 与 x86 指令指示符 118，也即，微程序计数器 232 并不持有 ISA 指令 124 的地址，微程序计数器 232 所持有的地址也不落于系统存储器地址空间内。此外，更值得注意的是，微指令 246 由硬件指令转译器 104 所产生，并且直接提供给执行管线 112 执行，而非作为执行管线 112 的执行结果 128。

[0163] 请参照图 3，图中是以方块图详述图 2 的指令格式化程序 202。指令格式化程序 202 由图 1 的指令高速缓冲存储器 102 接收 x86ISA 与 ARM ISA 指令比特组 124 区块。凭借 x86ISA 指令长度可变的特性，x86 指令 124 可以由指令比特组 124 区块的任何比特组开始。由于 x86ISA 容许首码比特组的长度会受到当前地址长度与操作数长度预设值的影响，因此确认高速缓冲存储 器区块内的 x86ISA 指令的长度与位置的任务会更为复杂。此外，依据当前 ARM 指令集状态 322 与 ARM ISA 指令 124 的操作码，ARM ISA 指令的长度不是 2 比特组就是 4 比特组，因而不是 2 比特组对齐就是 4 比特组对齐。因此，指令格式化程序 202 由指令比特组 124 串 (stream) 撷取不同的 x86ISA 与 ARM ISA 指令，此指令比特组 124 串由指令高速缓冲存储器 102 接收的区块所构成。也就是说，指令格式化程序 202 格式化 x86ISA 与 ARM ISA 指令比特组串，因而大幅简化图 2 的简单指令转译器对 ISA 指令 124 进行解码与转译的困难任务。

[0164] 指令格式化程序 202 包含一预解码器 302，在指令模式指示符 132 指示为 x86 时，

预解码器 302 预先将指令比特组 124 视为 x86 指令比特组进行解码以产生预解码信息，在指令模式指示符 132 指示为 ARM 时，预解码器 302 预先将指令比特组 124 视为 ARM 指令比特组进行解码以产生预解码信息。指令比特组队列 (IBQ) 304 接收 ISA 指令比特组 124 区块以及由预解码器 302 产生的相关预解码信息。

[0165] 一个由长度解码器与纹波逻辑 306 构成的阵列接收指令比特组队列 304 的底部项目 (bottom entry) 的内容，也即 ISA 指令比特组 124 区块与相关的预解码信息。此长度解码器与纹波逻辑 306 也接收指令模式指示符 132 与 ARMISA 指令集状态 322。在一实施例中，ARM ISA 指令集状态 322 包含 ARM ISACPSR 寄存器的 J 与 T 比特。为了回应其输入信息，此长度解码器与纹波逻辑 306 产生解码信息，此解码信息包含 ISA 指令比特组 124 区块内的 x86 与 ARM 指令的长度、x86 首码信息、以及关于各个 ISA 指令比特组 124 的指示符，此指示符指出此比特组是否为 ISA 指令 124 的起始比特组、终止比特组、以及 / 或一有效比特组。一多工器队列 308 接收 ISA 指令比特组 126 区块、由预解码器 302 产生的相关预解码信息、以及由长度解码器与纹波逻辑 306 产生的相关解码信息。

[0166] 控制逻辑(未图示)检验多工器队列 (MQ) 308 底部项目的内容，并控制多工器 312 撷取不同的或格式化的 ISA 指令与相关的预解码与解码信息，所撷取的信息提供至一格式化指令队列 (FIQ) 314。格式化指令队列 (FIQ) 314 在格式化 ISA 指令 242 与提供至图 2 的简单指令转译器 204 的相关信息间作为缓冲。在一实施例中，多工器 312 在每一个时钟周期内撷取至多三个格式化 ISA 指令与相关的信息。

[0167] 在一实施例中，指令格式化程序 202 在许多方面类似于 2009 年 10 月 1 日提出的美国专利第 12/571,997 号、第 12/572,002 号、第 12/572,045 号、第 12/572,024 号、第 12/572,052 号与第 12/572,058 号申请案共同公开的 XIBQ、指令格式化程序、与 FIQ，这些申请案在此系列为参考数据。然而，前述专利申请案所揭示的 XIBQ、指令格式化程序、与 FIQ 通过修改，使其能在格式化 x86ISA 指令 124 外，还能格式化 ARM ISA 指令 124。长度解码器 306 被修改，使能对 ARM ISA 指令 124 进行解码以产生长度以及起点、终点与有效性的比特组指示符。尤其，若是指令模式指示符 132 指示为 ARM ISA，长度解码器 306 检测当前 ARM 指令集状态 322 与 ARM ISA 指令 124 的操作码，以确认 ARM 指令 124 是一个 2 比特组长度或是 4 比特组长度的指令。在一实施例中，长度解码器 306 包含多个独立的长度解码器分别用以产生 x86ISA 指令 124 的长度数据以及 ARM ISA 指令 124 的长度数据，这些独立的长度解码器的再以连线或 (wire-ORed) 耦接在一起，以提供输出至纹波逻辑 306。在一实施例中，此格式化指令队列 314 包含独立的队列以持有格式化指令 242 的多个互相分离的部分。在一实施例中，指令格式化程序 202 在单一脉冲周期内，提供简单指令转译器 204 至多三个格式化 ISA 指令 242。

[0168] 请参照图 4，图中是以方块图详细显示图 1 的执行管线 112，此执行管线 112 耦接至硬件指令转译器 104 以直接接收来自图 2 的硬件指令转译器 104 的实行微指令。执行管线 112 包含一微指令队列 401，以接收微指令 126；一寄存器分配表 402，由微指令队列 401 接收微指令；一指令调度器 404，耦接至寄存器分配表 402；多个保留站 406，耦接至指令调度器 404；一指令发送单元 408，耦接至保留站 406；一重排缓冲器 422，耦接至寄存器分配表 402、指令调度器 404 与保留站 406；以及执行单元 424 耦接至保留站 406、指令发送单元 408 与重排缓冲器 422。寄存器分配表 402 与执行单元 424 接收指令模式指示符 132。

[0169] 在硬件指令转译器 104 产生实行微指令 126 的速率不同于执行管线 112 执行微指令 126 的情况下,微指令队列 401 是作为一缓冲器。在一实施例中,微指令队列 401 包含一个 M 至 N 可压缩微指令队列。此可压缩微指令队列使执行管线 112 能够在一给定的时脉周期内,从硬件指令转译器 104 接收至多 M 个(在一实施例中,M 是六)微指令 126,并且随后将接收到的微指令 126 存储至宽度为 N(在一实施例中,N 是三)的队列结构,以在每个时钟周期 提供至多 N 个微指令 126 至寄存器分配表 402,此寄存器分配表 402 能够在每个时脉周期处理最多 N 个微指令 126。微指令队列 401 是可压缩的,因它不论接收到微指令 126 的特定时脉周期为何,都会依序将由硬件指令转译器 104 所传送的微指令 126 时填满队列的空项目,因而不会在队列项目中留下空洞。此方法的优点为能够充分利用执行单元 424(请参 照图 4),因为它可对不可压缩宽度 M 或宽度 N 的指令队列提供较高指令储存效能。具体来说,不可压缩宽度 N 的队列会需要硬件指令转译器 104,尤其是简单指令转译器 204,在之后的时脉周期内会重复转译一个或多个已经在之前的时脉周期内已经被转译过的 ISA 指令 124。会这样做的原因是,不可压缩宽度 N 的队列无法在同一个时脉周期接收多于 N 个微指 令 126,而重复转译将导致电力耗损。不过,不可压缩宽度 M 的队列虽然不需要简单指令转译器 204 重复转译,但却会在队列项目中产生空洞而导致浪费,因而需要更多列项目以及一个较大且更耗能的队列来提供相当的缓冲能力。

[0170] 寄存器分配表 402 由微指令队列 401 接收微指令 126 并产生与微处理器 100 内进行中的微指令 126 的附属信息,寄存器分配表 402 并执行寄存器重命名动作增加微指令平行处理,以利于执行管线 112 的超纯量、非循序执行能力。若是 ISA 指令 124 指示为 x86,寄存器分配表 402 会对应于微处理器 100 的 x86ISA 寄存器 106,产生附属信息且执行相对应的寄存器重命名动作;反之,若是 ISA 指令 124 指示为 ARM,寄存器分配表 402 就会对应于微处理器 100 的 ARM ISA 寄存器 106,产生附属信息且执行相对应的寄存器重命名动作;不过,如前述,部分寄存器 106 可能是由 x86ISA 与 ARM ISA 所共享。寄存器分配表 402 也在重排缓冲器 422 中依据程序顺序配置一项目给各个微指令 126,因此重排缓冲器 422 可使微指令 126 以及其相关的 x86ISA 与 ARM ISA 指令 124 依据程序顺序进行引退,即使微指令 126 的执行对应于其所欲实现的 x86ISA 与 ARM ISA 指令 124 而言是以非循序的方式进行的。重排缓冲器 422 包含一环形队列,此环形队列的各个项目用以存储关于进行中的微指 令 126 的信息,此信息除了其他事项,还包含微指令 126 执行状态、一个确认微指令 126 是由 x86 或是 ARM ISA 指令 124 所转译的标签、以及用以存储微指令 126 的结果的存储空间。

[0171] 指令调度器 404 由寄存器分配表 402 接收寄存器重命名微指令 126 与附属信息,并依据指令的种类以及执行单元 424 的可利用性,将微指令 126 及 其附属信息分派至关联于适当的执行单元 424 的保留站 406。此执行单元 424 将会执行微指令 126。

[0172] 对各个在保留站 406 中等待的微指令 126 而言,指令发布单元 408 测得相关执行单元 424 可被运用且其附属信息被满足(如来源操作数可被运用)时,即发布微指令 126 至执行单元 424 供执行。如前述,指令发布单元 408 所发布的微指令 126,可以非循序执行于程序次序外以及以超纯量方式执行。

[0173] 在一实施例中,执行单元 424 包含整数 / 分支 (integer/branch) 单元 412、介质单 元 414、载入 / 存储单元 416、以及浮点单元 418。执行单元 424 执行微指令 126 以产生结果 128 并提供至重排缓冲器 422。虽然执行单元 424 并不大受到其所执行的微指令 126 由 x86

或是 ARM ISA 指令 124 转译而来的影响, 执行单元 424 仍会使用指令模式指示符 132 与环境模式指示符 136 以执行相对较小的微指令 126 子集。举例来说, 执行管线 112 管理旗标的产生, 其管理会依据指令模式指示符 132 指示为 x86ISA 或是 ARM ISA 而有些微不同, 并且, 执行管线 112 依据指令模式指示符 132 指示为 x86ISA 或是 ARM ISA, 对 x86EFLAGS 寄存器或是程序状态寄存器 (PSR) 内的 ARM 条件码旗标进行更新。在另一实例中, 执行管线 112 对指令模式指示符 132 进行取样以决定去更新 x86 指令指示符 (IP) 118 或 ARM 程序计数器 (PC) 116, 还是更新共通的指令地址寄存器。此外, 执行管线 122 也藉此来决定使用 x86 或是 ARM 语意 (semantics) 执行前述动作。一旦微指令 126 变成微处理器 100 中最旧的已完成微指令 126 (也即, 在重排缓冲器 422 队列的排头且呈现已完成的状态) 且其他用以实现相关的 ISA 指令 124 的所有微指令 126 均已完成, 重排缓冲器 422 就会引退 ISA 指令 124 并释放与实行微指令 126 相关的项目。在一实施例中, 微处理器 100 可在一时脉周期内引退至多三个 ISA 指令 124。此处理方法的优点在于, 执行管线 112 是一高效能、通用执行引擎, 其可执行支持 x86ISA 与 ARM ISA 指令 124 的微处理器 100 微架构的微指令 126。

[0174] 请参照图 5, 图中是以方块图详述图 1 的寄存器文件 106。就一较佳实施例而言, 寄存器文件 106 为独立的寄存器区块实体。在一实施例中, 通用寄存器由一具有多个读出埠与写入埠的寄存器文件实体来实现; 其他寄存器可在实体上独立于此通用寄存器文件以及其他会存取这些寄存器但具有较少的读取写入埠的邻近功能方块。在一实施例中, 部分非通用寄存器, 尤其是那些不直接控制微处理器 100 的硬件而仅存储微码 234 会使用到的数值的寄存器 (如部分 x86MSR 或是 ARM 协同处理器寄存器), 则是在一个微码 234 可存取的私有随机存取存储器 (PRAM) 内实现。不过, x86ISA 与 ARM ISA 编程者无法见到此私有随机存取存储器, 也即此存储器并不在 ISA 系统存储器地址空间内。

[0175] 总的来说, 如图 5 所示, 寄存器文件 106 在逻辑上是区分为三种, 也即 ARM 特定的寄存器 502、x86 特定的寄存器 504、以及共享寄存器 506。在一实施例中, 共享寄存器 506 包含十五个 32 比特寄存器, 由 ARM ISA 寄存器 R0 至 R14 以及 x86ISA EAX 至 R14D 寄存器所共享, 另外有十六个 128 比特寄存器由 x86ISA XMM0 至 XMM15 寄存器以及 ARM ISA 先进单指令多重数据扩展 (Neon) 寄存器所共享, 这些寄存器的部分是重迭于三十二个 32 比特 ARM VFPv3 浮点寄存器。如前文图 1 所述, 通用寄存器的共享意指由 x86ISA 指令 124 写入一共享寄存器的数值, 会被 ARM ISA 指令 124 在随后读取此共享寄存器时见到, 反之亦然。此方式的优点在于, 能够使 x86ISA 与 ARM ISA 程序通过寄存器互相沟通。此外, 如前述, x86ISA 与 ARM ISA 的架构控制寄存器的特定比特也可被引用为共享寄存器 506。如前述, 在一实施例中, x86 特定模型寄存器可被 ARM ISA 指令 124 通过实作定义协处理器寄存器存取, 因而是由 x86ISA 与 ARM ISA 所共享。此共享寄存器 506 可包含非架构寄存器, 例如条件旗标的非架构同等物, 这些非架构寄存器同样由寄存器分配表 402 重命名。硬件指令转译器 104 知道哪一个寄存器由 x86ISA 与 ARMISA 所共享, 因而会产生实行微指令 126 来存取正确的寄存器。

[0176] ARM 特定的寄存器 502 包含 ARM ISA 所定义但未被包含于共享寄存器 506 的其他寄存器, 而 x86 特定的寄存器 502 包含 x86ISA 所定义但未被包含于共享寄存器 506 的其他寄存器。举例来说, ARM 特定的寄存器 502 包含 ARM 程序计数器 116、CPSR、SCTRL、FPSCR、CPACR、协处理器寄存器、多种例外事件模式的备用 (banked) 通用寄存器与程序状态保存

寄存器 (saved program status registers, SPSRs) 等等。前文列出的 ARM 特定寄存器 502 并非为限定本案发明, 仅为例示以说明本发明。另外, 举例来说, x86 特定的寄存器 504 包含 x86 指令指示符 (EIP 或 IP) 118、EFLAGS、R15D、64 比特的 R0 至 R15 寄存器的上面 32 比特 (也即未落于共享寄存器 506 的部分)、区段寄存器 (SS, CS, DS, ES, FS, GS)、x87FPU 寄存器、MMX 寄存器、控制寄存器 (如 CR0-CR3, CR8) 等。前文列出的 x86 特定寄存器 504 并非为限定 本案发明, 而仅为例示以说明本发明。

[0177] 在一实施例中, 微处理器 100 包含新的实作定义 ARM 协同处理器寄存器, 在指令模式指示符 132 指示为 ARM ISA 时, 此实作定义协同处理器寄存器可被存取以执行 x86ISA 相关的操作。这些操作包含但不限于: 将微处理器 100 重置为一 x86ISA 处理器 (重置至 x86 指令) 的能力; 将微处理器 100 初始化为 x86 特定的状态, 将指令模式指示符 132 切换至 x86, 并开始在一特定 x86 目标地址撷取 x86 指令 124 (启动至 x86 指令) 的能力; 存取前述全域配置寄存器的能力; 存取 x86 特定寄存器 (如 EFLAGS) 的能力, 此 x86 寄存器是指定在 ARM R0 寄存器中, 存取电力管理 (如 P 状态与 C 状态的转换), 存取处理器汇流排功能 (如输入 / 输出循环)、中断控制器的存取、以及加密加速功能的存取。此外, 在一实施例中, 微处理器 100 包含新的 x86 非架构特定模型寄存器, 在指令模式指示符 132 指示为 x86ISA 时, 此非架构特定模型寄存器可被存取以执行 ARM ISA 相关的操作。这些操作包含但不限于: 将微处理器 100 重置为一 ARM ISA 处理器 (重置至 ARM 指令) 的能力; 将微处理器 100 初始化为 ARM 特定的状态, 将指令模式指示符 132 切换至 ARM, 且开始在一特定 ARM 目标地址撷取 ARM 指令 124 (启动至 ARM 指令) 的能力; 存取前述全域配置寄存器的能力; 存取 ARM 特定寄存器 (如 CPSR) 的能力, 此 ARM 寄存器是指定在 EAX 寄存器内。

[0178] 请参照第 6A 与 6B 图, 图中显示一流程说明图 1 的微处理器 100 的操作程序。此流程始于步骤 602。

[0179] 如步骤 602 所示, 微处理器 100 被重置。可向微处理器 100 的重置输入端发出信号来进行此重置动作。此外, 在一实施例中, 此微处理器汇流排是一 x86 型式的处理器汇流排, 此重置动作可由 x86 型式的 INIT 命令进行。回应此重置动作, 微码 234 的重置程序是被调用来执行。此重置微码的动作包含: (1) 将 x86 特定的状态 504 初始化为 x86ISA 所指定的预设数值; (2) 将 ARM 特定的状态 502 初始化为 ARM ISA 所指定的预设数值; (3) 将微处理器 100 的非 ISA 特定的状态初始化为微处理器 100 制造商所指定的预设数值; (4) 将共享 ISA 状态 506, 如 GPRs, 初始化为 x86ISA 所指定的预设数值; 以及 (5) 将指令模式指示符 132 与环境模式指示符 136 设定为指示 x86ISA。在另一实施例中, 不同于前述动作 (4) 与 (5), 此重置微码将共享 ISA 状态 506 初始化为 ARM ISA 特定的预设数值, 并将指令模式指示符 132 与环境模式指示符 136 设定为指示 ARM ISA。在此实施例中, 步骤 638 与 642 的动作不需要被执行, 并且, 在步骤 614 之前, 此重置微码会将共享 ISA 状态 506 初始化为 x86ISA 所指定的预设数值, 并将指令模式指示符 132 与环境模式指示符 136 设定为指示 x86ISA。接下来进入步骤 604。

[0180] 在步骤 604, 重置微码确认微处理器 100 是配置为一个 x86 处理器或是一个 ARM 处理器来进行开机。在一实施例中, 如前述, 预设 ISA 开机模式是硬式编码于微码, 不过可通过熔断配置熔丝的方式, 或利用一微码修补来修改。在一实施例中, 此预设 ISA 开机模式作为一外部输入提供至微处理器 100, 例如一外部输入接脚。接下来进入步骤 606。在步骤

606 中,若是预设 ISA 开机模式为 x86,就会进入步骤 614 ;反之,若是预设开机模式为 ARM,就会进入步骤 638。

[0181] 在步骤 614 中,重置微码使微处理器 100 开始由 x86ISA 指定的重置向量地址撷取 x86 指令 124。接下来进入步骤 616。

[0182] 在步骤 616 中, x86 系统软件(如 BIOS)是配置微处理器 100 来使用如 x86ISA RDMSR 与 WRMSR 指令 124。接下来进入步骤 618。

[0183] 在步骤 618 中,x86 系统软件执行一重置至 ARM 的指令 124。此重置至 ARM 的指令使微处理器 100 重置并以一 ARM 处理器的状态离开重置程序。然而,因为 x86 特定状态 504 以及非 ISA 特定配置状态不会因为重置至 ARM 的指令 126 而改变,此方式有利于使 x86 系统韧体执行微处理器 100 的初步设定并使微处理器 100 随后以 ARM 处理器的状态重开机,而同时还能使 x86 系统软件执行的微处理器 100 的非 ARM 配置配置维持完好。藉此,此方法能够使用“小型的”微开机码来执行 ARM 操作系统的开机程序,而不需要使用微开机码来解决如何配置微处理器 100 的复杂问题。在一实施例中,此重置至 ARM 指令是一 x86WRMSR 指令至一新的非架构特定模型寄存器。接下来进入步骤 622。

[0184] 在步骤 622,简单指令转译器 204 进入陷阱至重置微码,以回应复杂重置至 ARM(complex reset-to-ARM) 指令 124。此重置微码使 ARM 特定状态 502 初始化至由 ARM ISA 指定的预设数值。不过,重置微码并不修改微处理器 100 的非 ISA 特定状态,因而有利于保存步骤 616 执行所需的配置设定。此外,重置微码使共享 ISA 状态 506 初始化至 ARM ISA 指定的预设数值。最后,重置微码设定指令模式指示符 132 与环境模式指示符 136 以指示 ARM ISA。接下来进入步骤 624。

[0185] 在步骤 624 中,重置微码使微处理器 100 开始在 x86ISA EDX:EAX 寄存器指定的地址撷取 ARM 指令 124。此流程结束于步骤 624。

[0186] 在步骤 638 中,重置微码将共享 ISA 状态 506,如 GPRs,初始化至 ARMISA 指定的预设数值。接下来进入步骤 642。

[0187] 在步骤 642 中,重置微码设定指令模式指示符 132 与环境模式指示符 136 以指示 ARM ISA。接下来进入步骤 644。

[0188] 在步骤 644 中,重置微码使微处理器 100 开始在 ARM ISA 指定的重置向量地址撷取 ARM 指令 124。此 ARM ISA 定义两个重置向量地址,并可由一输入来选择。在一实施例中,微处理器 100 包含一外部输入,以在两个 ARMISA 定义的重置向量地址间进行选择。在另一实施例中,微码 234 包含在两个 ARM ISA 定义的重置向量地址间的一预设选择,此预设选则可通过熔断熔丝以及 / 或是微码修补来修改。接下来进入步骤 646。

[0189] 在步骤 646 中,ARM 系统软件设定微处理器 100 来使用特定指令,如 ARM ISA MCR 与 MRC 指令 124。接下来进入步骤 648。

[0190] 在步骤 648 中,ARM 系统软件执行一重置至 x86 的指令 124,来使微处理器 100 重置并以一 x86 处理器的状态离开重置程序。然而,因为 ARM 特定状态 502 以及非 ISA 特定配置状态不会因为重置至 x86 的指令 126 而改变,此方式有利于使 ARM 系统韧体执行微处理器 100 的初步设定并使微处理器 100 随后以 x86 处理器的状态重开机,而同时还能使由 ARM 系统软件执行的微处理器 100 的非 x86 配置配置维持完好。藉此,此方法能够使用“小型的”微开机码来执行 x86 操作系统的开机程序,而不需要使用微开机码来解决如何配置微

处理器 100 的复杂问题。在一实施例中,此重置至 x86 指令是一 ARMMRC/MRCC 指令至一新的实作定义协同处理器寄存器。接下来进入步骤 652。

[0191] 在步骤 652 中,简单指令转译器 204 进入陷阱至重置微码,以回应复杂重置至 x86 指令 124。重置微码使 x86 特定状态 504 初始化至 x86ISA 所特定的预设数值。不过,重置微码并不修改微处理器 100 的非 ISA 特定状态,此处理有利于保存步骤 646 所执行的配置设定。此外,重置微码使共享 ISA 状态 506 初始化至 x86ISA 所指定的预设数值。最后,重置微码设定指令模式指示符 132 与环境模式指示符 136 以指示 x86ISA。接下来进入步骤 654。

[0192] 在步骤 654 中,重置微码使微处理器 100 开始在 ARM ISA R1:R0 寄存器 所指定的地址撷取 ARM 指令 124。此流程终止于步骤 654。

[0193] 请参照图 7,图中是以一方块图说明本发明的一双核心微处理器 700。此双核心微处理器 700 包含两个处理核心 100,各个核心 100 包含有图 1 的微处理器 100 所具有的元件,藉此,各个核心均可执行 x86ISA 与 ARM ISA 机器语言程序。这些核心 100 可被设定为两个核心 100 都执行 x86ISA 程序、两个核心 100 都执行 ARM ISA 程序、或是一个核心 100 执行 x86ISA 程序而另一个核心 100 则是执行 ARM ISA 程序。在微处理器 700 的操作过程中,前述三种设定方式可混合且动态改变。如图 6A 和图 6B 的说明内容所述,各个核心 100 对于其指令模式指示符 132 与环境模式指示符 136 均具有一预设数值,此预设数值可利用熔丝或微码修补做修改,藉此,各个核心 100 可以独立地通过重置改变为 x86 或是 ARM 处理器。虽然图 7 的实施例仅具有二个核心 100,在其他实施例中,微处理器 700 可具有多于二个核心 100,而各个核心均可执行 x86ISA 与 ARM ISA 机器语言程序。

[0194] 请参照图 8,图中是以一方块图说明本发明另一实施例的可执行 x86ISA 与 ARM ISA 机器语言程序的微处理器 100。图 8 的微处理器 100 是类似于图 1 的微处理器 100,其中的元件也相似。然而,图 8 的微处理器 100 也包含一微指令高速缓冲存储器 892,此微指令高速缓冲存储器 892 存取由硬件指令转译器 104 产生且直接提供给执行管线 112 的微指令 126。微指令高速缓冲存储器 892 由指令撷取单元 114 所产生的撷取地址做索引。若是撷取地址 134 命中微指令高速缓冲存储器 892,执行管线 112 内的多工器(未图示)就选择来自微指令高速缓冲存储器 892 的微指令 126,而非来自硬件指令转译器 104 的微指令 126;反之,多工器则是选择直接由硬件指令转译器 104 提供的微指令 126。微指令高速缓冲存储器的操作,通常也称为追踪高速缓冲存储器,是微处理器设计的技术领域所已知的技术。微指令高速缓冲存储器 892 所带来的优点在于,由微指令高速缓冲存储器 892 撷取微指令 126 所需的时间通常会少于由指令高速缓冲存储器 102 撷取指令 124 并且利用硬件指令转译器将其转译为微指令 126 的时间。在图 8 的实施例中,微处理器 100 在执行 x86 或是 ARM ISA 机器语言程序时,硬件指令转译器 104 不需要在每次执行 x86 或 ARM ISA 指令 124 时都执行硬件转译,也即当实行微指令 126 已经存在于微指令高速缓冲存储器 892,就不需要执行硬件转译。

[0195] 在此所述的微处理器的实施例的优点在于,其通过内建的硬件指令转译 器来将 x86ISA 与 ARM ISA 指令转译为微指令集的微指令,而能执行 x86ISA 与 ARM ISA 机器语言程序,此微指令集是不同于 x86ISA 与 ARM ISA 指令集,且微指令可利用微处理器的共用的执行管线来执行以提供实行微指令。在此所述的微处理器的实施例的优点在于,通过协同利

用大量与 ISA 的执行管线来执行由 x86ISA 与 ARM ISA 指令硬件转译来的微指令, 微处理器的设计与制造所需的资源会少于两个独立设计制造的微处理器(也即一个能够执行 x86ISA 机器语言程序, 一个能够执行 ARM ISA 机器语言程序)所需的资源。此外, 这些微处理器的实施例中, 尤其是那些使用超纯量非循序执行管线的微处理器, 具有潜力能提供相较于既有 ARM ISA 处理器更高的效能。此外, 这些微处理器的实施例, 相较于采用软件转译器的系统, 也在 x86 与 ARM 的执行上可更具潜力地提供更高的效能。最后, 由于微处理器可执行 x86ISA 与 ARM ISA 机器语言程序, 此微处理器有利于建构一个能够高效地同时执行 x86 与 ARM 机器语言程序的系统。

[0196] 备份寄存器模拟该 ARM ISA 包含一备份寄存器的特征, 如表三所示, 该表是摘录自 ARM 使用者手册 (ARM programmer's manual) 中第 B1-9 页的图 B1-1。在第 B1 章节里描述了一 ARM ISA 核心的该系统层级程序开发者模型, 其包含了详细的 ARM 核心寄存器与备份寄存器的布局 (scheme)。如 ARM 程序开发者手册第 B1.3.2 章节所述:

[0197] 如第 A2-11 页 ARM 核心寄存器所述的该 ARM 寄存器文件的应用层级架构。此一架构提供了 16 个 ARM 核心寄存器, 即 R0-R15, 其包含了堆迭指示符器 (Stack Pointer, SP)、连结寄存器 (Link Register, LR) 以及程序记数器 (Program Counter, PC)。该些寄存器是选自总数 31 或 33 个的寄存器, 其依据是否实现了安全性扩充而定。如第 B1-1 图所示, 当前的执行模式决定寄存器的组别选择, 其显示寄存器的设置依据当前执行模式的选择, 而重制某些寄存器的内容。此一设置称之为寄存器备份, 而该重制部分的寄存器是称为备份寄存器。

[0198]

System level views								
Application level view	Privileged modes							
	Exception modes							
User mode	System mode	Supervisor mode	Monitor mode [‡]	Abort mode	Undefined mode	IRQ mode	FIQ mode	
R0	R0_usr							
R1	R1_usr							
R2	R2_usr							
R3	R3_usr							
R4	R4_usr							
R5	R5_usr							
R6	R6_usr							
R7	R7_usr							
R8	R8_usr							R8_fiq
R9	R9_usr							R9_fiq
R10	R10_usr							R10_fiq
R11	R11_usr							R11_fiq
R12	R12_usr							R12_fiq
SP	SP_usr	SP_svc	SP_mon [‡]	SP_abt	SP_und	SP_irq	SP_fiq	
LR	LR_usr	LR_svc	LR_mon [‡]	LR_abt	LR_und	LR_irq	LR_fiq	
PC	PC							
APSR	CPSR							
		SPSR_svc	SPSR_mon [‡]	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq	

[‡] Monitor mode, and the associated banked registers, are implemented only as part of the Security Extensions

[0199] 表三

[0200] 因此如表三所示,一个 ARM ISA 的核心可能执行八种不同执行模式之一。执行模式也可称为处理模式或操作方法。应用层级程序是执行于使用者模式,且不能存取受保护的系统资源,而且除非有例外事件发生外否则不能切换执行模式。相比之下,其他七个模式则统称为特权模式,其具有存取系统资源,并可随意更改核心的处理模式。特权模式中的六者,被称为例外事件模式,其是在有例外事件时进入该些模式,而特权模式的第七种也即系统模式,进入此一模式并非因为例外事件的发生,其通常是因为一指令的执行而进入。

[0201] 从前述表三中可知,ARM ISA 包含了 16 种通用核心寄存器 R0-R15,用以供应用层级程序在使用者模式中执行。R13-R15 寄存器各具有专属用途:R13 为堆迭寄存器 (SP);R14 为连结寄存器 (LR);以及 R15 为程序记数器 (PC)。该 16 个相同的通用寄存器 R0-R15 同样可于系统模式中被操作系统所取用。

[0202] 在六个例外事件模式中,如表三所示,每一模式都由相关于 SP 与 LR 寄存器的备份版本,以避免在使用时遭遇例外事件而导致 SP 与 LR 寄存器的损毁。也就是说,当遭遇例外事件时,核心是存取相关于例外事件模式的 SP 与 LR 寄存器,而非使用者模式下的 SP 与 LR 寄存器(或是另一例外事件模式下的 SP 与 LR 寄存器)。更具体地说,当遭遇例外事件,核心存储一特定于例外事件的例外事件回传地址于 LR 寄存器中,其中该 LR 寄存器相关于所遭遇例外事件的例外事件模式(如 LR_abt),而非存储该回传地址于使用者模式下的 LR 寄存器(LR_usr)。此外,当例外事件管理程序的指令存取 SP 或 LR 寄存器时,核心是存取相关

于例外事件的 SP 或 LR 寄存器的备份版本(除非该指令另有明确指定),而非使用者模式的 SP 与 LR 寄存器(或是另一例外事件模式下的 SP 与 LR 寄存器)。举例而言,在一管理者模式下执行的包含连结指令的 Branch,将会将次一指令的地址放置于 LR_svc 寄存器,而非 LR_usr 寄存器。在另一例子中,在 IRQ 模式下执行的一 Push 或 Pop 指令将会使用 SP_irq 寄存器而非 SP_usr 寄存器,以存储器中存取相关于该 IRQ 例外事件模式的一堆迭(a stack),而非存取使用者模式的堆迭(假设该 SP_irq 寄存器已依据操作系统的初始化,而得以存取一不同的存储器堆迭而非使用者模式堆迭)。

[0203] 此外,FIQ 模式具有 R8-R12 寄存器的备份版本,其可使 FIQ 中断管理程序避免必须自存储器中保存与恢复 R8-R12 寄存器,因此 FIQ 中断管理程序的执行速度会比其他例外事件处理程序来的快。当 FIQ 例外事件处理程序的指令存取 R8-R12 寄存器时,该核心存取 R8-R12 的 FIQ 备份版本(如标注于表三的 R8_q 至 R12_q)(除非该指令另由明确指定),而非使用者模式的 R8-R12 寄存器。举例而言,执行于 FIQ 模式下的存取 R10 寄存器的 Add 指令,其是存取 R10_q 寄存器而非 R10_user 寄存器。因此,实行了安全性扩充的 ISA ARM 核心,将从一组共 33 个寄存器选出相关监视模式备份寄存器,包含:16 种使用者模式寄存器、相关于各六种例外事件模式的 SP 与 LR 的寄存器备份版本,以及相关于 FIQ 模式的 R8-R12 的备份版本;相反的,未实行安全性扩充的 ARM ISA 核心将从一组共 31 个寄存器中选出相关监视模式备份寄存器,也就是说并不包含 LR_mon 与 SP_mon 寄存器。

[0204] 最后,指定于当前程序状态处理器 (Current program status register, CPSR) 的 ARM ISA 包含条件码旗标,执行状态比特、例外事件遮罩比特、以及定义当前处理模式的比特。CPSR 应用层级程序的模组被称为应用程序状态寄存器 (Application Program Status Register, APSR),并且仅提供存取条件码旗标。每一例外事件具有其自身的 CPSR 备份版本,如前述表三所示。当遭遇例外事件时,该 CPSR 数值的副本将写入相关于该所进入例外事件的 SPSR 中。如此可令该例外事件管理程序自该例外事件中恢复时,复原该 CPSR 至遭遇例外事件前的数值,以便检视例外事件发生时该 CPSR 的数值。

[0205] 图 9 为一传统的实施例,其 ARM ISA 通用寄存器是实施为如同硬件寄存器 906,硬件寄存器 906 位于包含该 ARM ISA 例外事件模式备份寄存器的硬件寄存器文件 902 中(电脑于图中未示)。如图 9 所示,寄存器文件 902 包含硬件多工逻辑 908,依据当前处理模式 914,以选择 R8 至 R12 寄存器的合适版本以及 R13 与 R14 寄存器的合适版本。额外的硬件多工逻辑 904 是基于指令所指定的寄存器地址 912,以选择指定于执行指令的寄存器。(一般的寄存器文件是实施为如同多埠寄存器文件,其包含二载入埠与一写入埠,是以一指令可指定二来源操作数以及一目的操作数,因此该硬件多工逻辑 908 与 904 可重复设置三次,各对应于一埠)实施例可合并操作模式硬件多工逻辑 908 与寄存器地址硬件多工逻辑 904;然而,这样的做法在处理模式 914 中需要额外的复杂度、电晶体、与电源组件,方能进行寄存器的选择。

[0206] 典型地,处理器在一所提供的处理模式下执行许多指令(有时达到上千种),而当一例外事件模式发生或者是执行模式切换指令以切换至新处理模式时,接着许多指令执行于一新处理模式,而后又发生新的模式切换等等。几乎(即使不是全部)所有被执行的指令存取包含备份版本的 R8-R14 的通用寄存器 902。依据该传统实施例,每一存取至该通用寄存器文件通过如第 9 图所示的硬件多工逻辑 908,以选择合适的备份寄存器(R8-R14 寄存

器),其增加每一存取连接至供应暂存文件 902 的延迟。此一现象同样发生于相对不那么频繁的处理模式切换上,以及相较于使用者模式寄存器的存取不那么频繁的备份寄存器的存取上。或者说,即使该选择输入 914 至该硬件多工逻辑 908 的更新的较不频繁,每一指令的执行而存取该寄存器文件 902 的操作也将导致硬件多工处理器 908 的延迟。基本上,存取寄存器文件 902 对处理器而言是一关键的硬件时序路程,其可能需要降低核心时脉,抑或将存取较高比例较高的部分被切割为较低频率分格窗口 (bin)。因此,需要一种可避免硬件多工逻辑 908 延迟的解决方案。

[0207] 在本实施例所提供的微处理器,其提供改良 ARM ISA 通用寄存器文件(其余部分均相同),由于本微处理器简化了在硬件多工逻辑中,并基于处理模式输入以不同方式选择合适寄存器,故相较传统的通用寄存文件有更加的存取性能。替代的,本实施例所描述的 R8-R14 的备份版本是模拟的,而非实际存在于寄存器文件(此寄存器文件会直接提供操作数给微处理器的执行单元),故而只有一个单独的实体寄存器 R8-R14 存在于寄存器文件中。更具体的说,该微处理器包含间接存储器以放置该模拟文件。在另一实施例中,该间接存储器是一私有随机存取存储器,其包含于该微处理器的存储器子系统中。为了因应处理模式的切换,硬件寄存器 R13-R14 的数值(或是 R8-R14,若切换至 FIQ 模式)系先存储于间接存储器中的相关于该旧处理模式的位置,且硬件寄存器 R13-R14 (或是 R8-R14,若切换至 FIQ 模式)接着在新处理模式中自间接存储器中的相联位置而恢复。此外,在切换至 FIQ 模式的情况下,R8-R12 的内容存储至全域间接存储器,而在从 FIQ 模式做切换的情况下,其内容系由全域间接存储器中恢复。应注意的是,该存储与恢复的操作系利用该微处理器的微码进行。因此,随后的执行单元自直接寄存器文件 (direct register file) 中的 R8-R14 单一副本,存取相关于新处理模式的数值。是以,从概念上来说,本实施例的优点在于,相对较不频繁的处理模式切换通过一个虚拟多工器执行,而非频繁通过一实体多工器执行每一寄存器的存取。本实施例的另一优点在于,由于处理模式的切换相对来说较不频繁,因此用相关于模式切换所造成额外的延迟做代价以获得其他益处,譬如在缺乏相关硬件多工逻辑且基于处理模式输入的自多个寄存器中选择的情况下,能更快的寄存器文件存取等等。

[0208] 请参阅图 10,图 10 是本发明的系统方块图,详细显示图 1 的微处理器。如先前所述,在实施例中微处理器 100 的微架构在许多方面系类似于由威盛电子所制造的 VIA Nano™ 处理器,但其已修改为支持 ARM ISA,更具体的说,可模拟 ARM ISA 的备份寄存器模式。

[0209] 微处理器 100 包含:如图 1 的寄存器文件 106,在图 10 中系标示为直接存储器 106;多工器 1014、1016 与 1018 耦接于直接存储器 106,以接收直接存储器 106 的输出;多工器 1004、1006 与 1008 耦接于多工器 1014、1016 与 1018 以接收多工器 1014、1016 与 1018 的输出;载入单元 416、存储单元 416,以及如图 4 的整数 / 分支单元、介质单元与浮点单元 412/414/418 (在图 10 中称为 ALU 单元 412/414/418),系分别耦接于多工器 1004、1006 与 1008,以接收多工器 1004、1006 与 1008 的输出;如图 4 的重排缓冲器 (ROB) 422 耦接于载入单元 416、存储单元 416 以及 ALU 单元 412/414/418,以接收载入单元 416、存储单元 416 以及 ALU 单元 412/414/418 的结果 128;以及间接存储器 1002,间接存储器 1002 耦接于重排缓冲器 422 与多工器 1008,用以自重排缓冲器 422 接收微指令 126 的结果 128,以及将其输出作为一输入传至多工器 1008。

[0210] 重排缓冲器 422 保留微指令 126 的结果 128 于其的重新命名寄存器 (rename registers)，直到结果 128 引退至架构寄存器。每一多工器 1014/1016/1018 基于相关于微指令 126 所指定的寄存器地址，以自直接存储器 106 中选择一操作数。每一多工器 1004/1006/1008 基于指定于微指令 126 的操作数类型，以自其输入来源中选择一操作数。虽然在各执行单元中仅显示一组操作数多工器对，1014 对应 1004、1016 对 1006 与 1018 对 1008，需了解的是，一多工器对系存在于每一源操作数与每一执行单元之间。此外，除了分别与多工器 1014/1016/1018 的输出耦接之外，多工器 1004、1006、1008 还耦接每一执行单元以自各执行单元接收结果 128，以及存储于重排缓冲器 422 中的结果 128。此外，载入单元 416 也自多工器 1008 接收间接存储器 1002 的输出。本发明的优点在于，当处理模式切换时，为了模拟 ARM ISA 备份寄存器，微处理器 100 可利用微码 234 在直接存储器 106 或间接存储器 1002 间存储或恢多个值，下面将会对其工作方式作进一步描述。

[0211] 如图 10 所示，直接存储器 106 包含多个寄存器以存储数据或操作数，以供 ARM R0-R14 通用寄存器的运用。虽然驻留在实体寄存器文件的通用寄存器与 CPSR(以及自 PC)不同，但直接存储器 106 仍包含一用以存储 CPSR 的寄存器。在一实施例中，一硬件寄存器文件包含直接存储器 106。

[0212] 间接存储器 1002 包含 R13、R14 与 SPSR 存储器，其系关连于每一 ARMISA 的处理模式，也即使用者 (User)、管理者 (SVC)、终止 (ABT)、未定 (UND)、IRQ 以及 FIQ 处理模式。此外，间接存储器 1002 包含关连于 FIQ 处理模式的 R8-R12 存储器。最后，除了 FIQ 模式的外，间接存储器 1002 包含关连于全域 (GLOBAL) 的全部处理模式。这些包含在间接存储器 1002 中不同存储器地址的运用将描述于后。

[0213] 在一实施例中，间接存储器 1002 包含属于存储器子系统 108 的一私有随机存取存储器 (PRAM)，如先前所述，该 PRAM 系利用如图 2 的微码 234 加以定址，但此一操作对于 x86ISA 与 ARM ISA 程序员不可见的，也就是说并不存在于 ISA 系统存储器地址空间。在 2010 年 2 月 11 日所发布的美国专利第 7,827,390 中所描述的 PRAM 实施例中，在此，将其列入参考。特别是，间接存储器 1002 系仅可利用载入单元 416 来加以载入，以及仅可利用存储单元 416 来加以存储。更具体的说，间接存储器 1002 仅可由载入单元 416 所执行的间接存储器 1002 的载入微指令 126 (在此其系对应于 load_PRAM 微指令)、以及由存储单元 416 所执行的间接存储器 1002 的存储微指令 126 (在此其系对应于 store_PRM 微指令) 予以定址。故此，其他的执行单元 412/414/418 不可载入或写入间接存储器 1002。该 load_PRAM 微指令指示载入单元 416 自间接存储器 1002 中的一指定地址载入数据至寄存器文件 106 的一特定寄存器，该特定寄存器可为如图 10 所示的一架构寄存器或是可由微码 234 存取的一非架构寄存器 (也可称为一临时寄存器)。相反的，该 store_PRAM 微指令指令存储单元 416 自寄存器文件 106 中的一指定寄存器，存储数据至间接存储器 1002 中的一指定地址中。

[0214] 请参阅至图 11A 及图 11B，图 11A 及图 11B 是显示在本发明图 10 的微处理器 100 的操作流程图，该流程系始于步骤 1102。

[0215] 如步骤 1102，硬件指令转译器 104 侦测一自当前处理模式切换至新处理模式的要求，并回应地将进入陷阱而前往如图 2 微码 234 中的适当的程序，其是设定用以管理处理模式切换的要求。指令转译器 104 可通过不同的方式以侦测切换处理模式要求，但不设限于以下所述的方法。首先，指令转译器 104 可能遭遇一明确要求切换处理模式的 ISA 指令

124,例如一ARM ISA 切换处理状态指令 (CPS)、管理者呼叫 (SVC) 指令、安全监视呼叫 (SMC) 指令或者是移动至特殊寄存器 (MSR)。其次,指令转译器 104 可能遭遇一隐含处理模式切换要求的 ISA 指令 124,如一 ARM ISA 自例外事件返回 (RFE) 指令、载入多重(自例外事件返回)、SUBS PC、LR 或断点 (BKPT) 指令。第三,指令转译器 104 可能遭遇一由未定义指令例外事件 (Undefined instruction exception) 所导致的未定义 ISA 指令 124。第四,指令转译器 104 可能接收到另一单元的微处理器 100 所发出一遭遇例外事件的信号。举例而言,指令转译器 104 可能自微处理器 100 的存储器子系统(未显示)中收到一信号,该信号表示有一指令要求一不在存取权限中的存取动作,譬如当微处理器并非处于一特权模式却请求存取一仅供特权存取的存储器区块时,则建立一数据终止例外事件条件 (Data Abort exception condition);或者是当一指令被提取 (fetched) 以及要求执行一无效指令时,指令转译器 104 可能接收到一发生存储器终止的指示,并建立一预先提取的终止例外事件条件 (Prefetch Abort exception condition);或者是指令转译器 104 可能自微处理器 100 的汇流排界面单元接受到一信号,该信号指示要求一中断操作 (IRQ 或 FIQ)。第五,该指令转译器 104 可能遭遇一用以存取如先前所述的全域配置寄存器 122 的 x86RDMSR/WRMSR 指令 124,或者 x86Launch-ARM,或者如先前所述的 reset-to-ARM 指令 124。微码 234 基于特定模式转换的种类执行多个动作,譬如准备更新中断遮罩比特、条件旗标或是在 CPSR 中的其他比特。此外,在步骤 1114 更新直接存储器 106CPSR 之前,微码 234 可存储当前的直接存储器 106CPSR 数值至 SPSR 在间接寄存器 1002 中关连于新处理模式的位置。更进一步的说,ARM SIT224 可在进入陷阱至微码 234 前执行其他动作。举例来说在 ARM ISA LDM(自例外事件恢复) 指令 124 的情况下,ARM SIT224 可发送载入微指令 126 以要求自存储器内载入特定的寄存器,接下来进入步骤 1104。

[0216] 在步骤 1104 中,微码 234 判断在步骤 1102 中所要求的新处理模式是否与当前处理模式相同,假如相同则流程结束;若不相同则进入步骤 1106。

[0217] 在步骤 1106,微码 234 将直接存储器 106 的寄存器 R13 与 R14 的数值存储在间接存储器 1002 中相关于当前处理模式所对应的位置。举例而言,假若该当前处理模式系管理者模式,微码 234 将直接存储器 106 的 R13/R14 的数值存储至间接存储器 1002 的 SVC 部分中 R13/R14 的位置,如图 12 的箭头 (1) 所示。而在另一例子中,假若当前处理模式系 FIQ 模式,微码 234 将直接存储器 106 的 R13/R14 数值存储至间接存储器 1002 的 FIQ 部分的 R13/R14 的位置,如图 12 的箭头 (5) 所示。有利的是,该微码可包含一 store_PRAM 微指令 126 的序列,以自直接存储器 106 中存储数值至间接存储器 1002。接着进入步骤 1108。

[0218] 在步骤 1108,微码 234 判断当前处理模式是否为 FIQ 处理模式,若是,则进入步骤 1112;否则进入步骤 1114。

[0219] 在步骤 1112,微码 234 将直接存储器 106 的寄存器 R8-R12 的数值,存储至间接存储器 1002 中的关连于 FIQ 模式所对应的位置,如图 12 的箭头 (6) 所示。此外,微码 234 将间接存储器 1002 中相关于全域的非 FIQ 模式的数值恢复至直接存储器 106 的寄存器 R8-R12 中,如图 12 的箭头 (7) 所示。有利的是,执行回复的微码可包含一 load_PRAM 微指令 126 的序列,以自间接存储器 1002 中载入数值至直接存储器 106。接着进入步骤 1114。

[0220] 在步骤 1114 中,微码 234 将步骤 1102 中所要求的新处理模式来更新 CPSR106 的模式 (Mode) 比特。CPSR106 的写入还包含对 CPSR106 在模式比特旁的其他比特的更新。接

着进入步骤 1116。

[0221] 在步骤 1116 中,微码 234 将间接存储器 1002 中关连于新处理模式的相对应位置的数值,恢复至直接存储器 106 的 R13 与 R14 寄存器中。举例而言,若新处理模式为 FIQ 模式,微码 234 将间接存储器 1002 的 FIQ 部分的 R13/R14 位置的数值恢复至直接存储器 106 的 R13/R14,如图 12 的箭头 (2) 所示。举例而言,假若新处理模式系 UND 模式,微码 234 将间接存储器的 UND 部分的 R13/R14 位置的数值恢复至直接存储器 106 的 R13/R14,如图 12 的箭头 (8) 所示。接着进入步骤 1118。

[0222] 在步骤 1118,微码 234 判断该新处理模式是否为 FIQ 模式,若是则进入步骤 1122 ;若否则进入步骤 1124。

[0223] 在步骤 1122,微码 234 将直接存储器 106 的 R8-R12 寄存器的数值,存储至间接存储器 1002 中关连于全域的非 FIQ 模式所对应的位置,如图 12 的箭头 (3) 所示。此外,微码 234 自间接存储器 1002 中关连于 FIQ 模式的相对应位置数值,恢复至直接存储器 106 的 R8-R12 寄存器,如图 12 的箭头 (4) 所示,接着进入步骤 1124。

[0224] 在步骤 1124 中,微码 234 执行更多基于特定模式切换类型的动作,举例来说,若在一例外事件发生后,微码 234 将一更新过的数值填入直接存储器 106 的 R14 寄存器(也就是 LR 寄存器)中,其中该更新过的数值依据 ARM 手册中第 B1-34 与第 B1-35 页中的表 B1-4,随后再跳跃至典型的例外事件管理程序,也即将控制权还给 ARM ISA 程序。流程结束于步骤 1124。

[0225] 可从图 12 中观察到,即使执行一自第一处理模式(譬如 SVC)至 FIQ 模式的切换,接着在没有立即恢复至该第一处理模式即切换至第三处理模式(譬如 UND)的情况下,使用该全域间接存储器 1002 的位置的优点在于仍可使微处理器 100 在直接存储器 106 的 R8-R12 寄存器中保持正确的数值,从而模拟 ARM ISA 备份寄存器。

[0226] 从前述内容可知,一种用以将 ARM ISA 备份寄存器的模拟的构思蓝图如下所述。当微处理器 100 系切换至一个新处理模式时,微处理器 100 将正确的数值存放至直接存储器 106 中,其为已知 ARM 处理器中新处理模式的备份寄存器。举例而言,在切换至 FIQ 模式后,直接存储器 106 的 R0-R14 寄存器具有在已知 ARM 处理器的 R0_usr-R7_usr 以及 R8_fiq-R14_fiq 的内容。因此,FIQ 处理模式的操作数可直接由直接存储器 106 中供给 ALU 单元 412/414/418,以在 FIQ 模式下的微处理器 100 (也即 FIQ 例外事件管理程序指令)可执行转译自 ARM ISA 数据处理指令 124 的微指令 126。在另一个例子中,在切换至 UND 模式后,直接存储器 106 的 R0-R14 寄存器将具有已知 ARM 处理器的 R0_usr-R12_usr 以及 R13_und-R14_und 的内容,因此,UND 处理模式的操作数可直接由直接存储器 106 中供给 ALU 单元 412/414/418,以使在 UND 模式下的微处理器 100 可执行转译自 ARM ISA 数据处理指令 124 的微指令 126。为了达成此一功效,该微码将直接存储器中合适的现有数值存储至间接存储器 1002 中所规划的位置(以便在随后的模式切换时可将其恢复),并且自间接存储器 1002 中其他规划的位置中恢复先前所存储的数值至直接存储器 106。一般而言,可通过存储直接存储器 106 的 R13 与 R14 中当前或是旧的数值至间接存储器 1002 中旧的处理模式的位置,并且自间接存储器 1002 中新处理模式的位置恢复至直接存储器 106 的 R13 与 R14。然而,在切换至 FIQ 模式或是自 FIQ 模式中切换出来时,有着更多处理要求。当自模式 X 切换至 FIQ 模式并随后自 FIQ 模式切换至模式 Y 时,直接存储器 106 的 R8-R12 寄存器中的数

值必须与在模式 Y 中时相同,虽然他们均系切换自模式 X,且该模式 X 可能与模式 Y 有着不同数值。因此在此一情况下,间接存储器 1002 中的全域位置系有利于自直接存储器 106 的 R8-R16 寄存器中存储与恢多个值。

[0227] 从前述内容更可得知,在此所描述的模拟备份寄存器可能在切换处理模式时,由于系利用微码将数值在直接存储器 106 与间接存储器 1002 间存储与恢复,相较传统的设计将导致些微的额外负荷。然而,此潜在额外负担所产生的潜在优势,可使直接存储器 106 相较传统的设计有着更快的存取。这系因为在此所述的实施例中,可避免在传统设计中由于硬件多工器必须考虑处理模式对于相对较不频繁备份寄存器的使用,所导致的额外传送延迟。由于 硬件多工器系典型的位于重要的时序路程中,使得硬件多工器加速运行将有利于时脉的提升,因此这系十分重要的。此外,在直接存储器 106 中的寄存器数量将少于传统设计,其可减少存取直接存储器 106 的时间。此外,实施例中的另一优点为,其仅需对现存的微架构进行相对些微的修改即可支持 ARM ISA 备份寄存器。更进一步的,实施例的另一优点在于,其可减轻 RAT106 中的相依检查器 (dependency checker) 在区分处理模式间的不同时的负担。另一优点在于,其也可减少寄存器更名表的大小,或者是避免在部分处理模式切换中要求序列化(如更新管线,即 RAT 与 ROB)。最后,实施例的另一优点在于,因为切换系利用微码实施而非硬件上的切换,故可增加切换一架构至 ARM ISA 备份寄存器等事件的弹性。总结地说,在此所描述的实施例,虽然可能增加处理模式切换所需的时间,但相对而言模式切换系较不频繁的,并可据此换取在一般情况下提供更高的效能表现,其中,在此所述的一般情况是指为获得 ALU 操作数的主要寄存器的存取。

[0228] 虽然前述的实施例中,微码系执行将数值在直接存储器与间接存储器的间的存储与恢复功能,在其他实施例中系利用具有硬件组合逻辑的微处理器,以执行回应处理模式切换所需的将数值在直接存储器与间接存储器之间的存储与恢复功能,而非利用微码执行的。更进一步的说,虽然在前述的实施例中,用以存储旧的处理模式数值的间接存储器为 PRAM,在其他实施例中系利用硬件寄存器作为间接存储器,但其不可由 ALU 单元直接存取。再更进一步的说,虽然前述的实施例系关于 ARM ISA,在其他实施例中系拟应用于其他关连于不同处理模式的特定备份寄存器的 ISA。

[0229] 载入多重 / 存储多重 ARM ISA 指令

[0230] 另一个关于 ARM ISA 的特征系载入多重 (LDM) 与存储多重 (STM) 指令。载入多重指令自存储器中载入每一指定于指令的通用寄存器,如 ARM 手册中第 A8-110 至 A8-116 页中所述。相反地,STM 指令自每一指定于指令的通用寄存器中存储至存储器,如 ARM 手册中第 A8-374 页至 A8-381 页中所述。在此所述的实施例,其系如前述的微处理器 100 的超纯量非循序执行微架构上实行 LDM 指令与 STM 指令。更具体而言,ARM ISA 指定自例外事件模式存取架构使用者模式寄存器的 LDM 指令与 STM 指令的版本(也即当微处理器 100 并非处于使用者模式下时)。这些指令版本系对应 LDM (使用者寄存器) 指令与 STM (使用者寄存器) 指令,如 ARM 手册中第 B6-7 页至第 B6-8 页以及第 B6-22 页至第 B6-23 中所述。在此所述的实施例中,其系在微处理器 100 的微架构上实行 LDM (使用者寄存器) 指令与 STM (使用者寄存器) 指令,微处理器 100 的微架构包含用以模拟前述的备份寄存器的间接存储器 1002。

[0231] 请参阅至图 13A 及图 13B,其是显示在本发明图 1 的微处理器 100 执行一 LDM 指令

的流程图,该流程系始于步骤 1302。

[0232] 在步骤 1302 中,如图 2 的软件指令转译器 204 接收到一 LDM 指令 124。尤其是,指令模式指示符 132 指示 ARM ISA 以及图 2 的 ARM SIT224 对 LDM 指令 124 解码。LDM 指令 124 指定一组用以载入数据的通用寄存器,以及将被载入数据的连续存储器地址。此外,LDM 指令 124 指定该指令是否为 LDM (使用者寄存器) 指令 124。接着进入步骤 1304。

[0233] 在步骤 1304 中,ARM SIT224 考虑指定于 LDM 指令 124 的次一(或第一)寄存器,接着进入步骤 1306。

[0234] 在步骤 1306 中,ARM SIT224 判断指令 124 是否为 LDM (使用者寄存器) 指令 124,若是,则进入步骤 1312;若否,则进入步骤 1308。

[0235] 在步骤 1308,ARM SIT224 发出载入微指令 126 以自指定于 LDM 指令 124 的存储器内中的次一(或第一)位置载入数据,并送至如图 10 的直接存储器 106 的特定寄存器(其系于步骤 1306 被考虑)。载入微指令 126 将会送往执行管线 112 而被载入单元 416 所执行。接着进入步骤 1318。

[0236] 在步骤 1312,ARM SIT224 判断寄存器是否为 R8-R12 其中的一种,以及当前处理模式是否为 FIQ 模式。若是,进入步骤 1314;若否,则进入步骤 1316。

[0237] 在步骤 1314,ARM SIT224 发出载入微指令 126 以自指定于 LDM 指令 124 的存储器内中的次一(或第一)位置载入数据,并送至非架构的、或是临时的直接存储器 106 的寄存器。载入微指令 126 将送往执行管线 112 而被载入单元 416 所执行,如后述的步骤 1324,数据将会持续的自临时寄存器中,存储至间接存储器 1002。接着进入步骤 1318。

[0238] 在步骤 1316,ARM SIT224 判断寄存器是否为 R13 或是 R14 寄存器,若是,则进入步骤 1314;若否,则进入步骤 1308。

[0239] 在步骤 1318,ARM SIT224 判断是否有其他特定于 LDM 指令 124 的寄存器尚未考虑,也即硬件指令转译器 104 尚未发送的相关微指令 126。若有其他寄存器时,则回到步骤 1304 以考虑特定于 LDM 指令 124 的次一寄存器;若否,则进入步骤 1322。

[0240] 在步骤 1322,ARM SIT224 判断在步骤 1314 中是否有发送任何微指令 126 以载入数据至临时寄存器 106,若有,则进入步骤 1324;若否,则流程结束。

[0241] 在步骤 1324,SIT204 转移控制权至如图 2 的复杂指令转译器 (CIT) 206,CIT206 是基于微码 234 以产生 store_PRAM 微指令 126,用以于将步骤 1314 中所载入的数据,自临时寄存器 106 中存储至间接存储器 1002 中的合适位置。更具体而言,在间接存储器 1002 中的合适位置是指关连于使用者模式的 R13 与 R14 位置,以及全域性关连于非 FIQ 处理模式的 R8-R12 位置。存储微指令 126 送往执行管线 112 被存储单元 416 所执行,且流程结束于步骤 1324。

[0242] 请参阅至图 14A 及图 14B,其显示在本发明图 1 的微处理器 100 执行一 LDM 指令的另一流程图,在图 14A 及图 14B 中许多步骤类似于图 13A 及图 13B 的步骤,且具有相同的标号。然而,在图 14A 及图 14B 中,若于步骤 1318 中,ARM SIT224 判断并无其他寄存器待考虑,则流程结束;故此,在图 14A 及图 14B 的流程中系不具有步骤 1322 与步骤 1324。此外,系有自步骤 1314 进入的新步骤 1424,以及自新步骤 1424 进入步骤 1318 的流程。

[0243] 在步骤 1424 中,ARM SIT224 发送 store_PRAM 微指令 126,以将在步骤 1314 中所载入的数据自临时寄存器 106 存储至间接存储器 1002 中的合适位置。

[0244] 从图 14A 及图 14B 中可知,本实施例中的优点在于 LDM (使用者寄存器) 微指令的执行并不需要将控制权转移至微码 234。而缺点则在于其增加了 ARM SIT224 的复杂度。具体而言,在 ARM SIT224 必须发送 store_PRAM 微指令 126 的前提之下,ARM SIT224 必须知道关于间接存储器 1002 中的合适位置,以及数据必须被存储至该处的相关信息,因此与图 13A 及图 13B 的实施例有所不同。

[0245] 请参阅图 15A 及图 15B,其是显示本发明中图 1 的微处理器 100 执行一 STM 指令的流程图,该流程系始于步骤 1502。

[0246] 在步骤 1302 中,如图 2 的软件指令转译器 204 接收到一 STM 指令 124。尤其是,指令模式指示符 132 指示 ARM ISA 而图 2 的 ARM SIT224 对 STM 指令 124 进行解码。STM 指令 124 指定用以存储的通用寄存器以及将存储数据的连续存储器地址。此外,STM 指令 124 指定指令为 STM (使用者寄存器) 指令 124。接着进入步骤 1504。

[0247] 在步骤 1504 中,ARM SIT224 考虑指定于 STM 指令 124 的次一(或第一)寄存器,接着进入步骤 1506。

[0248] 在步骤 1506 中,ARM SIT224 判断指令 124 是否为 SDM (使用者寄存器) 指令 124,若是,则进入步骤 1512 ;若否,则进入步骤 1508。

[0249] 在步骤 1508,ARM SIT224 发出存储微指令 126 以将数据自如图 10 的直接存储器 106 的指定寄存器中,存储至特定于 STM 指令 124 的存储器中的次一(或第一)位置(其系已考虑于步骤 1504 中)。存储微指令 126 将会送往执行管线 112 并被存储单元 416 所执行。接着进入步骤 1518。

[0250] 在步骤 1512,ARM SIT224 判断寄存器是否为 R8-R12 其中的一种,以及当前处理模式是否为 FIQ 模式。若是,进入步骤 1514 ;若否,则进入步骤 1516。

[0251] 在步骤 1514,ARM SIT224 略过此特定寄存器,并随后于步骤 1524 中的微码 234 处理,接着进入步骤 1518。

[0252] 在步骤 1516,ARM SIT224 判断寄存器是否为 R13 或是 R14 寄存器,若是,则进入步骤 1514 ;若否,则进入步骤 1508。

[0253] 在步骤 1518,ARM SIT224 判断是否有其他指定于 STM 指令 124 的寄存器尚未考虑,也即硬件指令转译器 104 尚未发送关连的微指令 126 (或于步骤 1514 中略过)。若尚有其他寄存器时,则回到步骤 1504 以考虑指定于 STM 指令 124 的次一寄存器 ;若否,则进入步骤 1522。

[0254] 在步骤 1522,ARM SIT224 判断是否有任何寄存器于步骤 1514 中被略过,若有,则进入步骤 1524 ;若否,则流程结束。

[0255] 步骤 1524,SIT204 转移控制权至图 2 的复杂指令转译器 (CIT) 206,CIT206 是基于微码 234 以对步骤 1514 所略过的每一寄存器产生一微指令 126 对(pair)。具体而言,微指令 126 对包含伴随一存储微指令 126 的载入微指令 load_PRAM 微指令 126,load_PRAM 微指令 126 自间接存储器 1002 中的合适位置载入数据,至一非架构,或临时的直接存储器 106 的寄存器。存储微指令 126 将临时寄存器的数据存储至存储器内指定于 STM 指令 124 的位置。更具体而言,在间接存储器 1002 中的合适位置是指关连于使用者模式的 R13 与 R14 位置,以及关连于全域性非 FIQ 处理模式的 R8-R12 位置。load_PRAM 与存储微指令 126 将会送往执行管线 112 而分别被存储单元 416 与载入单元 416 所执行,且流程结束于步骤 1524。

[0256] 请参阅至图 16A 及图 16B, 其是显示本发明中图 1 的微处理器 100 执行一 STM 指令的另一流程图, 在图 16A 及图 16B 中许多步骤类似于图 15A 及图 15B 的步骤, 且具有相同的标号。然而, 在图 16A 及图 16B 中, 若于步骤 1518 中, ARM SIT224 判断并无其他寄存器待考虑则流程结束; 故此, 在图 16A 及图 16B 的流程中系不具有步骤 1522 与步骤 1524。此外, 在步骤 1512 与 1516 判断为“是”的流程后具有新的流程, 其系进入新步骤 1624, 以及自新步骤 1624 至新步骤 1614, 与自新步骤 1624 至步骤 1518。

[0257] 在步骤 1624 中, ARM SIT224 发送 load_PRAM 微指令 126, 以自间接存储器 1002 中的合适位置载入数据至临时寄存器 106, 接着进入步骤 1614。

[0258] 在步骤 1614 中, ARM SIT224 发送存储微指令 126, 以将在临时寄存器 106 中, 已于步骤 1624 时所载入的数据, 存储至存储器内特定于 STM 指令 124 的次一(或是第一)位置, 接着进入步骤 1518。

[0259] 从图 16A 及图 16B 中可知, 在本实施例中的优点在于, STM(使用者寄存器)微指令的执行并不需要将控制转移至微码 234, 而缺点则在于其增加了 ARM SIT224 的复杂度。具体而言, 在 ARM SIT224 必须发送载入微指令 126/store_PRAM 微指令 126 的前提之下, ARM SIT224 必须知道关于间接存储器 1002 中的合适位置, 以及数据必须被存储至该处的相关信息, 因此与图 15A 及图 15B 的实施例有所不同。

[0260] 如先前所述, ARM SIT224 其优点在于包含一状态机器, 以在多重的时脉周期中, 发送多重微指令 126 来实行 ISA 指令 124。

[0261] ARM ISA 也包含一存储恢复状态(SRS)指令 124, SRS 指令 124 将当前处理模式的 LR 与 SPSR 寄存器存储于由 SRS 指令 124 所指定的目标处理模式的存储器堆迭中, 其中 SRS 指令 124 可与当前处理模式不同。因此, SRS 指令 124 需要微处理器 100 以载入目标处理模式的架构 SP 寄存器的数值, 以便存取其存储器堆迭。在一实施例中, 当 ARM SIT224 解码一 ARM ISA SRS 指令 124, 其产生一 load_PRAM 微指令 126, 以自间接存储器 1002 的目标模式部分的 R13 位置中载入目标模式的 SP 数值, 并送至直接存储器 106 的临时寄存器, 以存取目标模式的存储器堆迭。

[0262] 然而各种有关于本发明的实施例已在本文详述, 应可充分了解如何实施 并且不限于这些实施方式。举凡所属技术领域中具有通常知识者当可依据本发明的上述实施例说明而作其它种种的改良及变化。举例来说, 软件可以启动如功能、制造、模型、模拟、描述及 / 或测试本文所述的装置及方法。可以通过一般程序语言(如 C 及 C++)、硬件描述语言(Hardware Description Languages; HDL)或其他可用程序的使用来达成, 其中硬件描述语言(Hardware Description Languages; HDL)包含 Verilog HDL、VHDL 等硬件描述语言。这样的软件能在任何所知的计算机可用介质中处理执行, 例如磁带、半导体、磁碟或光碟(如 CD-ROM 及 DVD-ROM 等)、网路、有线电缆、无线网路或其他通信介质。本文所述的装置及方法的实施例中, 可包含在智能型核心半导体内, 并且转换为积体电路产品的硬件, 其中智能型核心半导体如微处理器核心(如硬件描述语言内的实施或设定)。此外, 本文所述的装置及方法可由硬件及软件的结合来实施。因此, 本发明并不局限于任何本发明所述的实施例, 但系根据下述的专利范围及等效的专利范围而定义。具体来说, 本发明能在普遍使用的微处理器装置里执行实施。最后, 熟练于本技术领域的应能体会他们能很快地以本文所公开的观念及具体的实施例为基础, 并且在没有背离本发明所述的附属项范围下, 来设计或修正

其他结构而实行与本发明的同样目的。

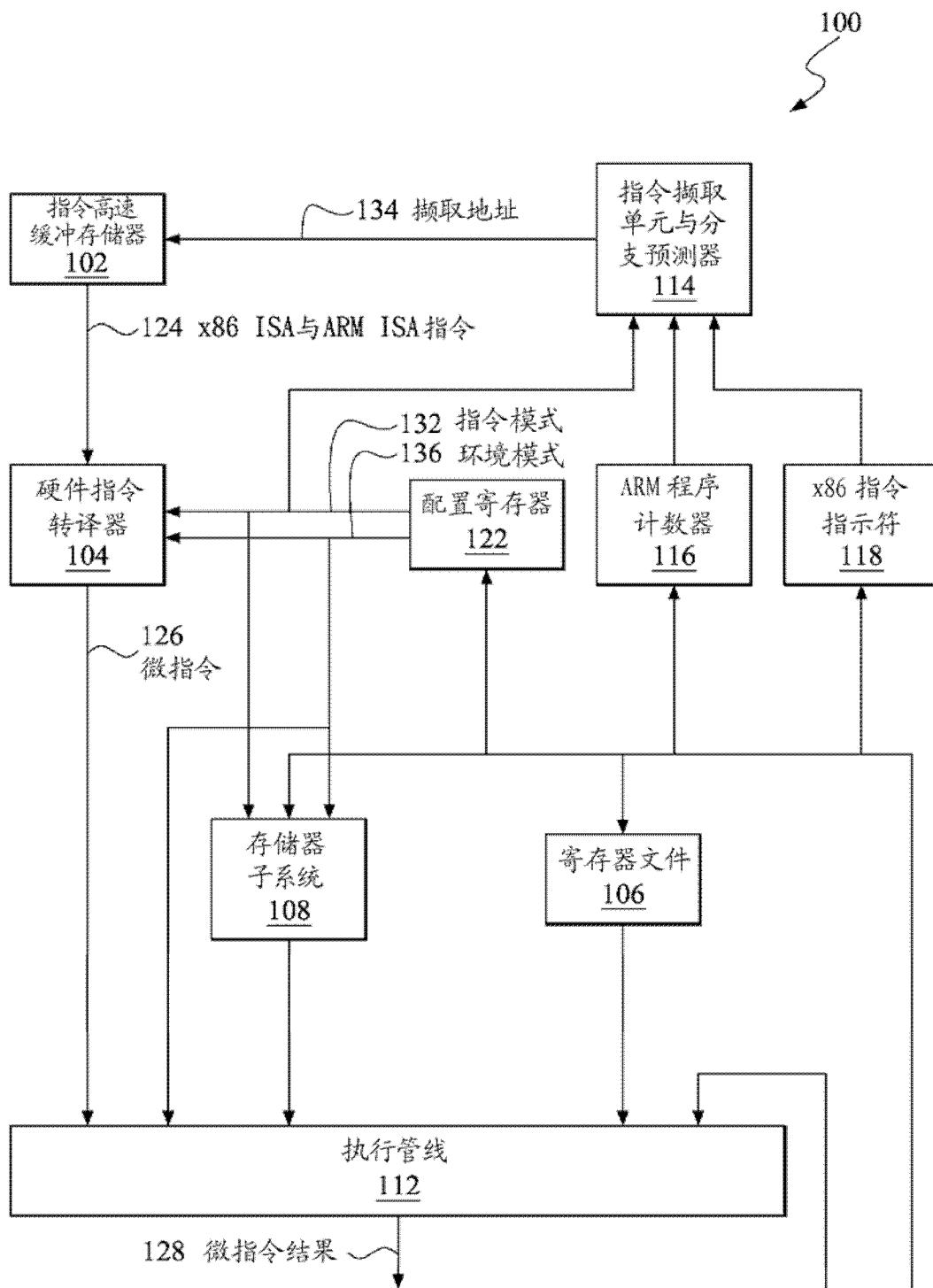


图 1

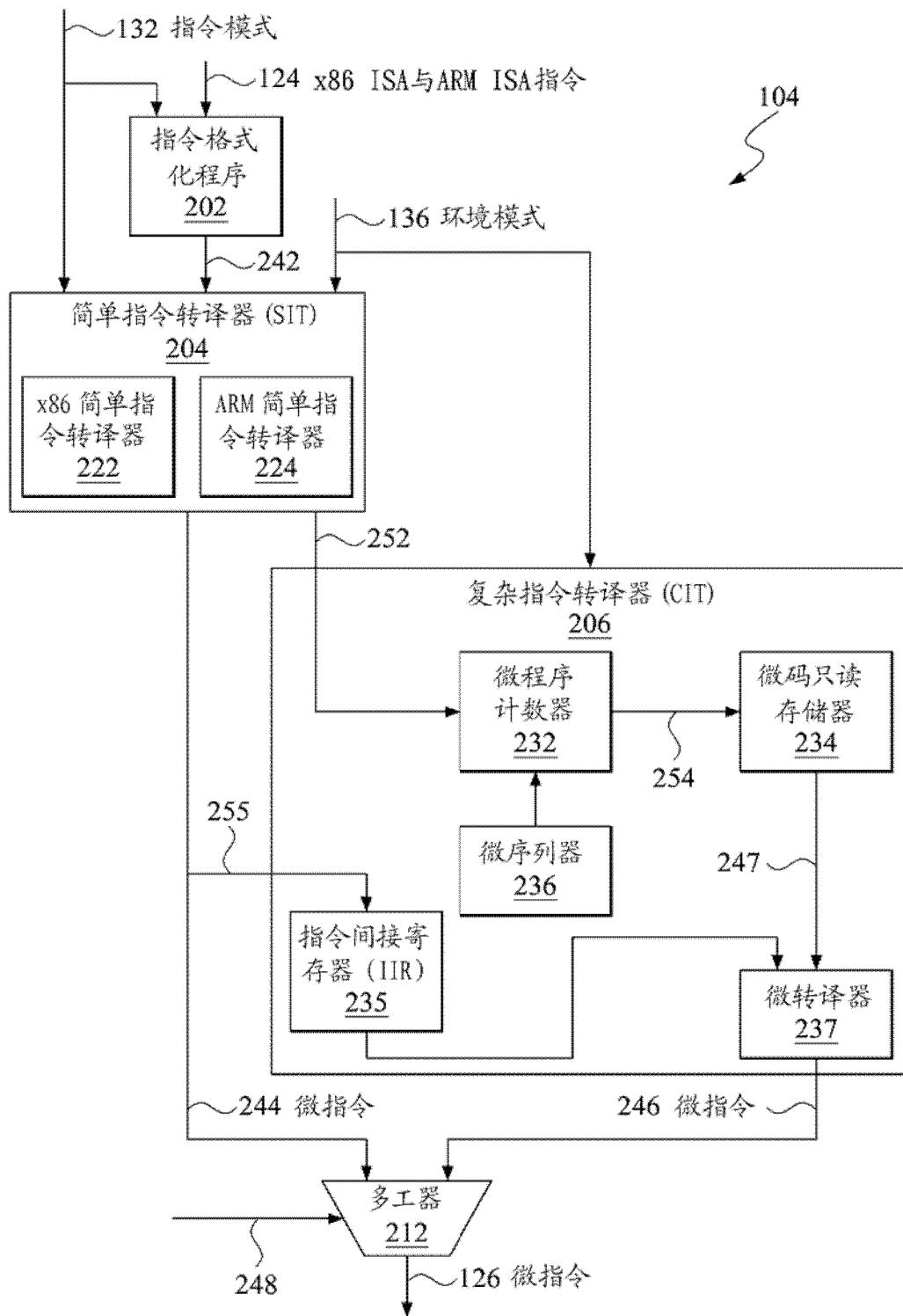


图 2

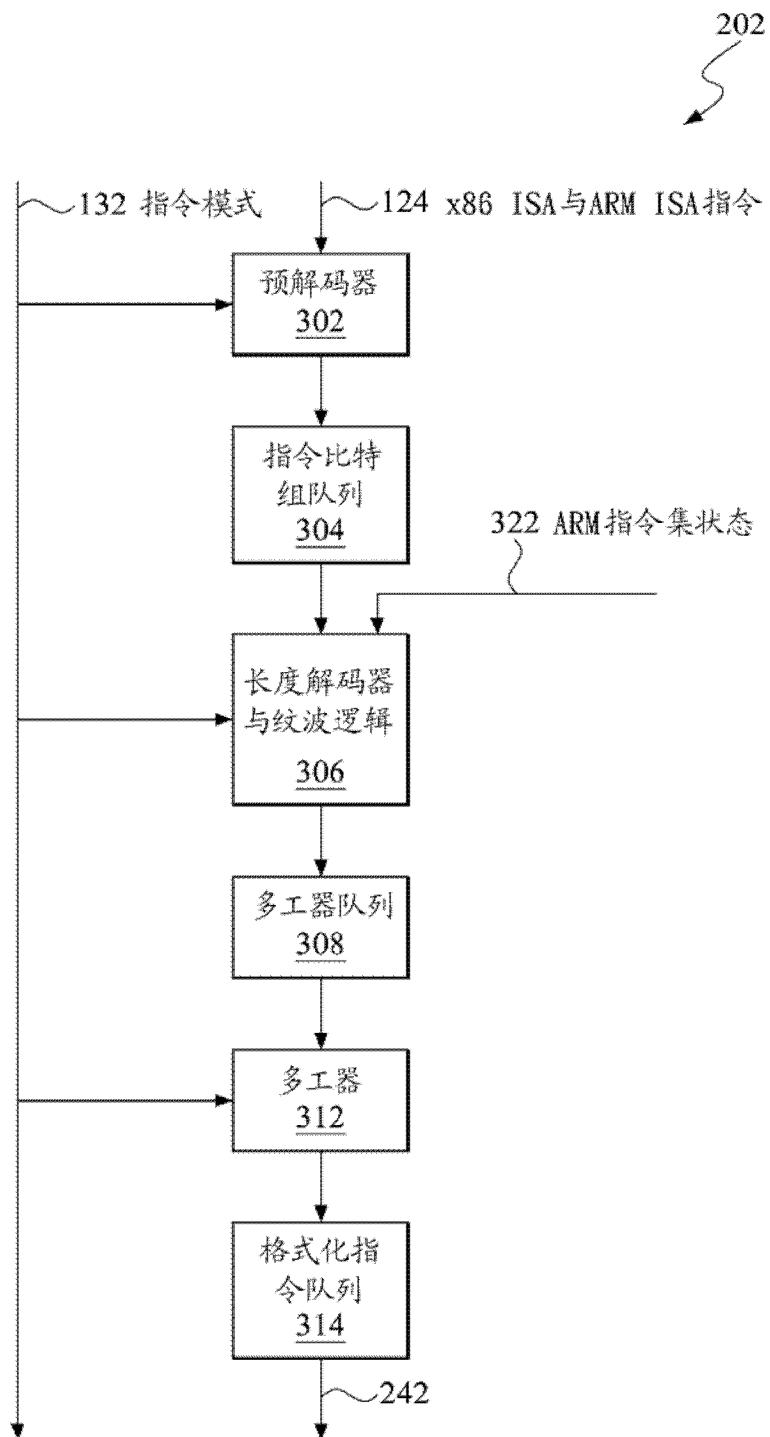


图 3

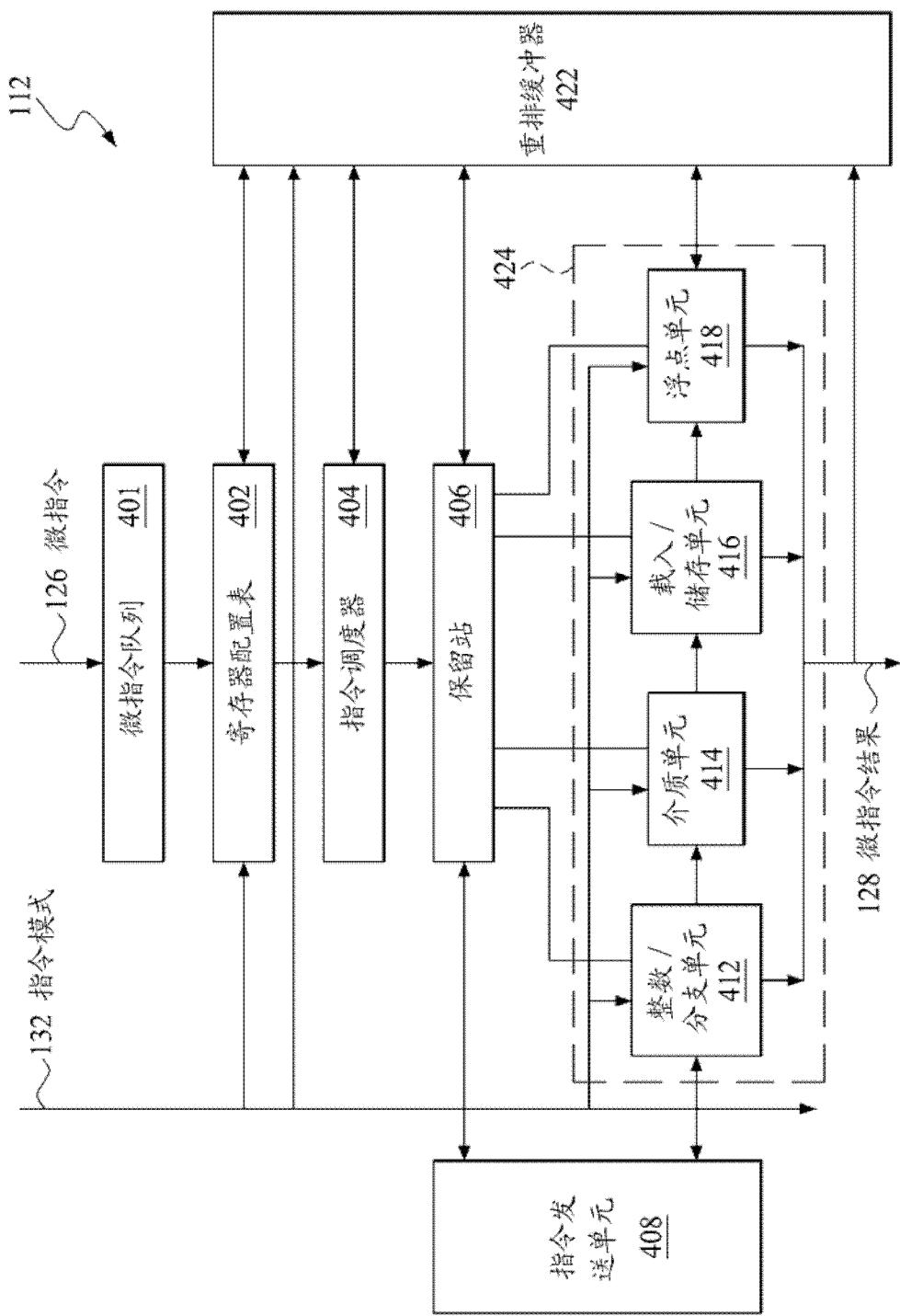


图 4

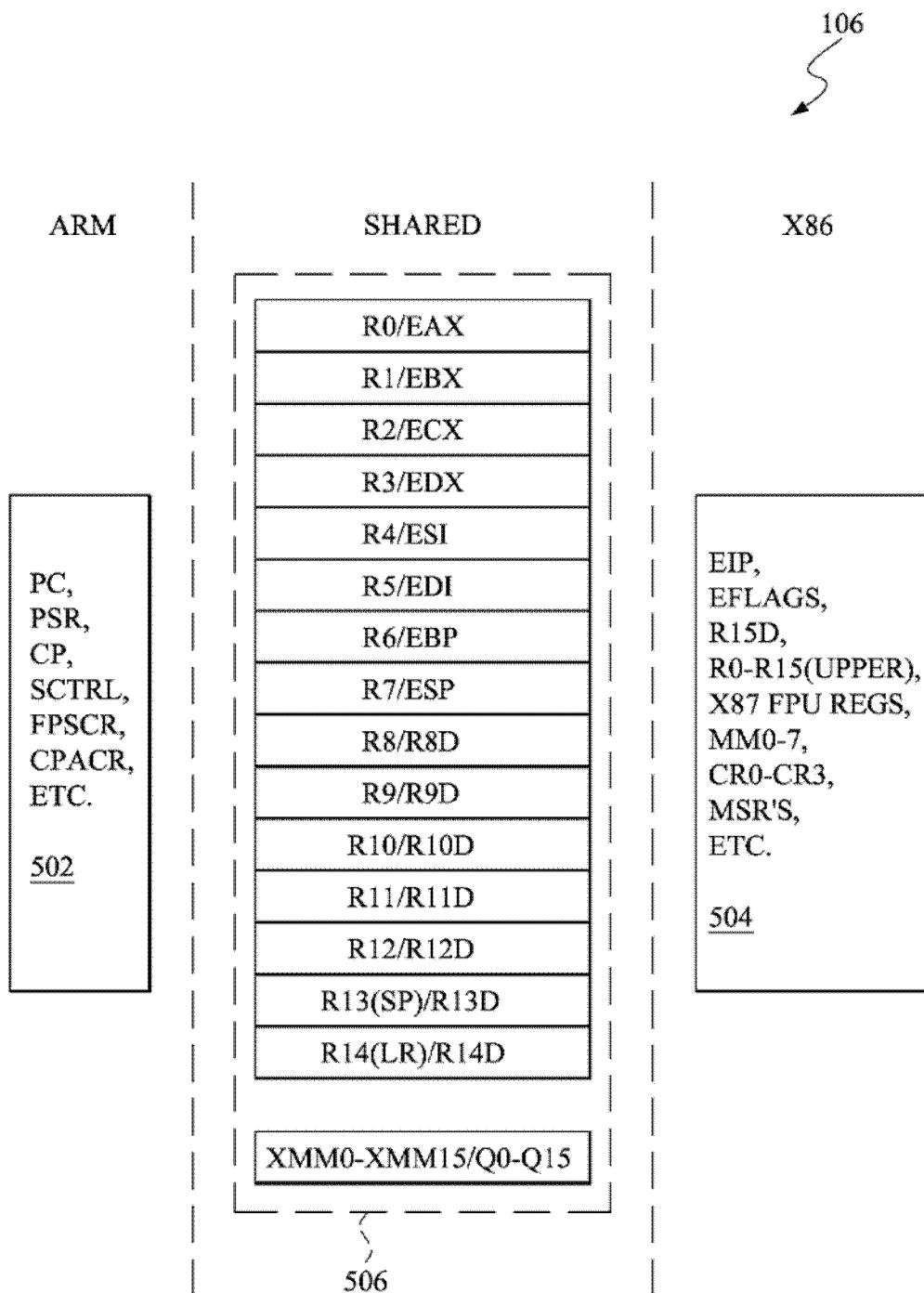


图 5

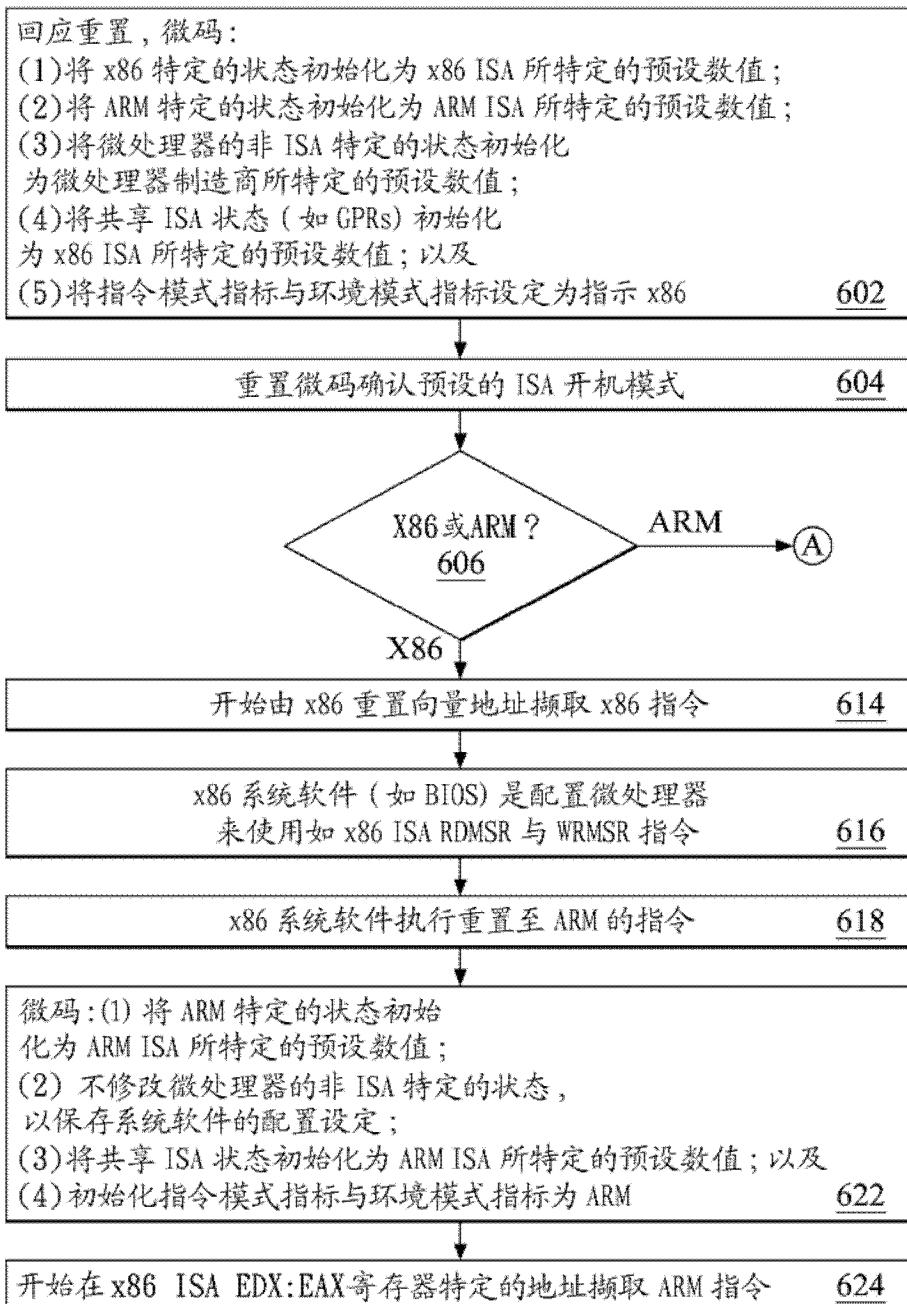


图 6A

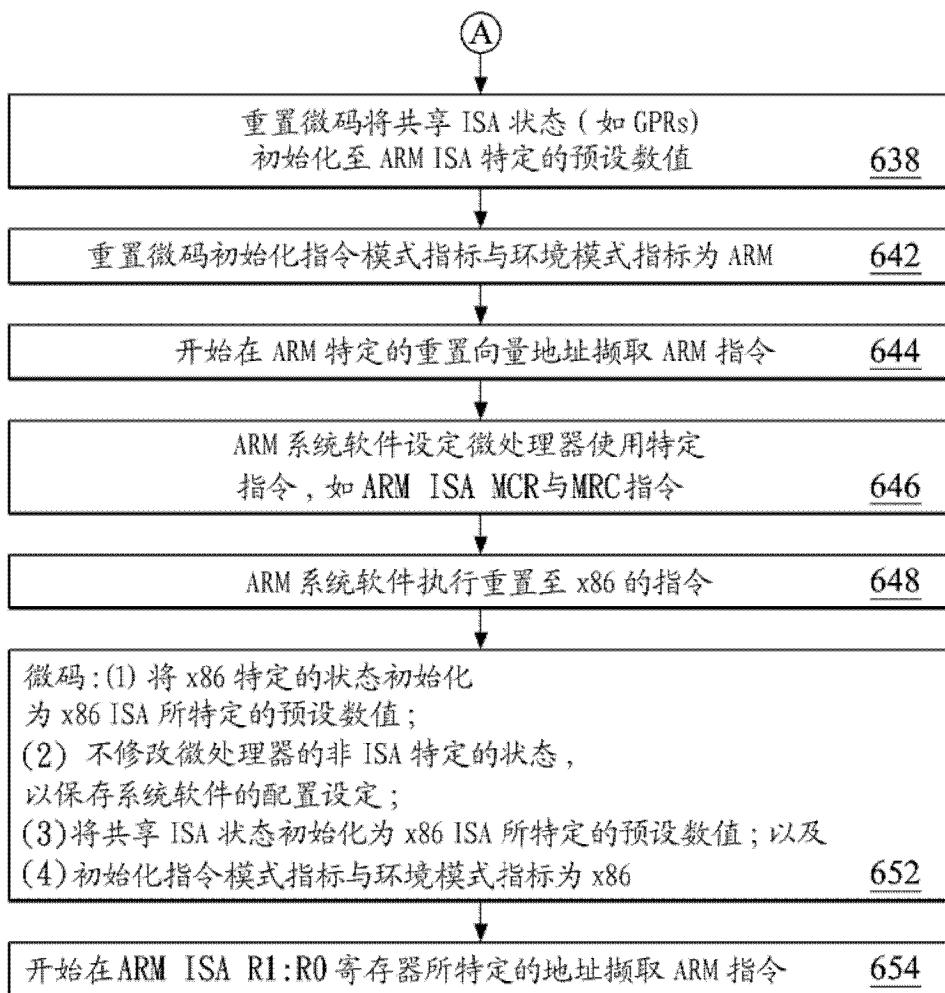


图 6B

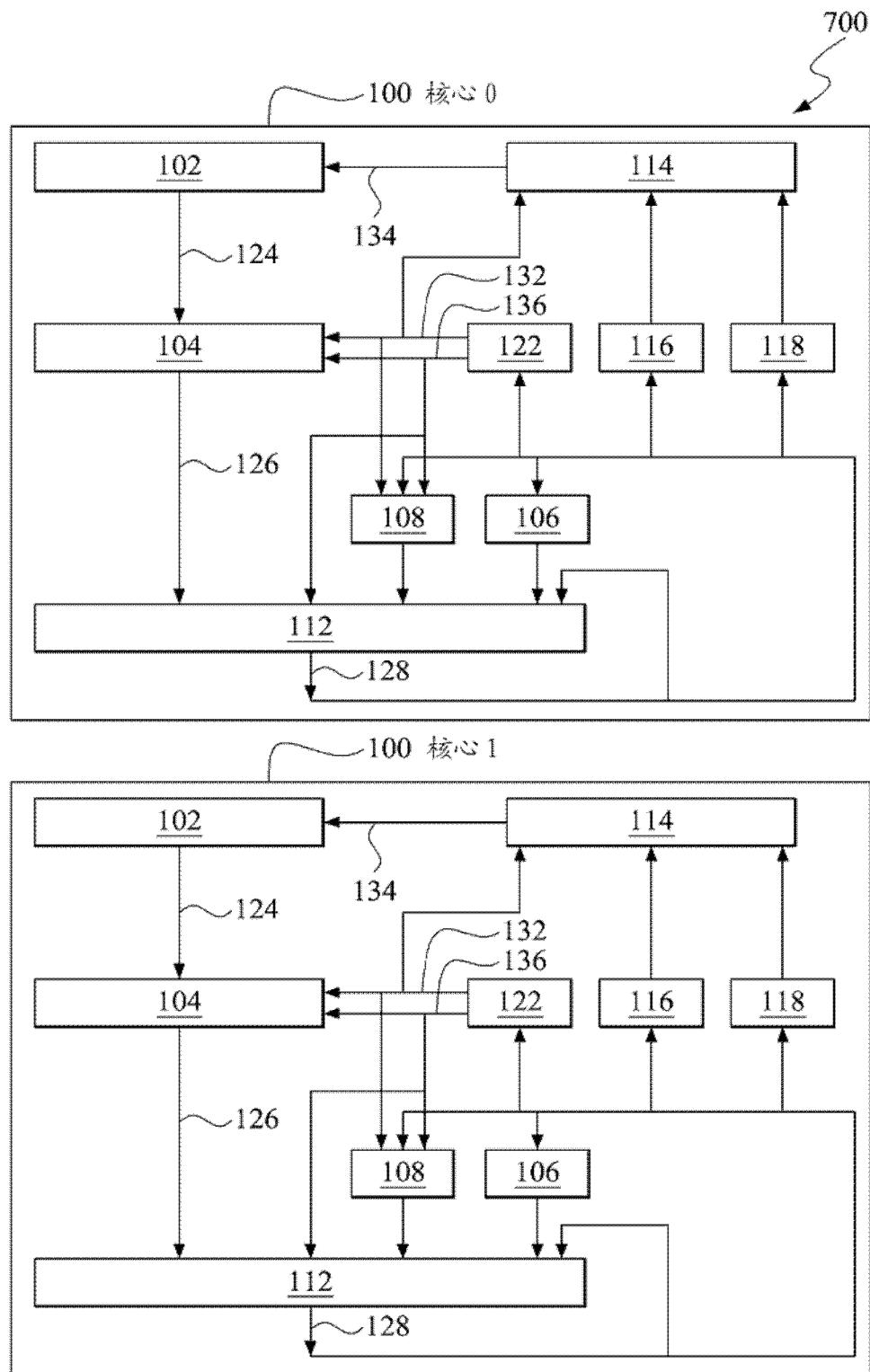


图 7

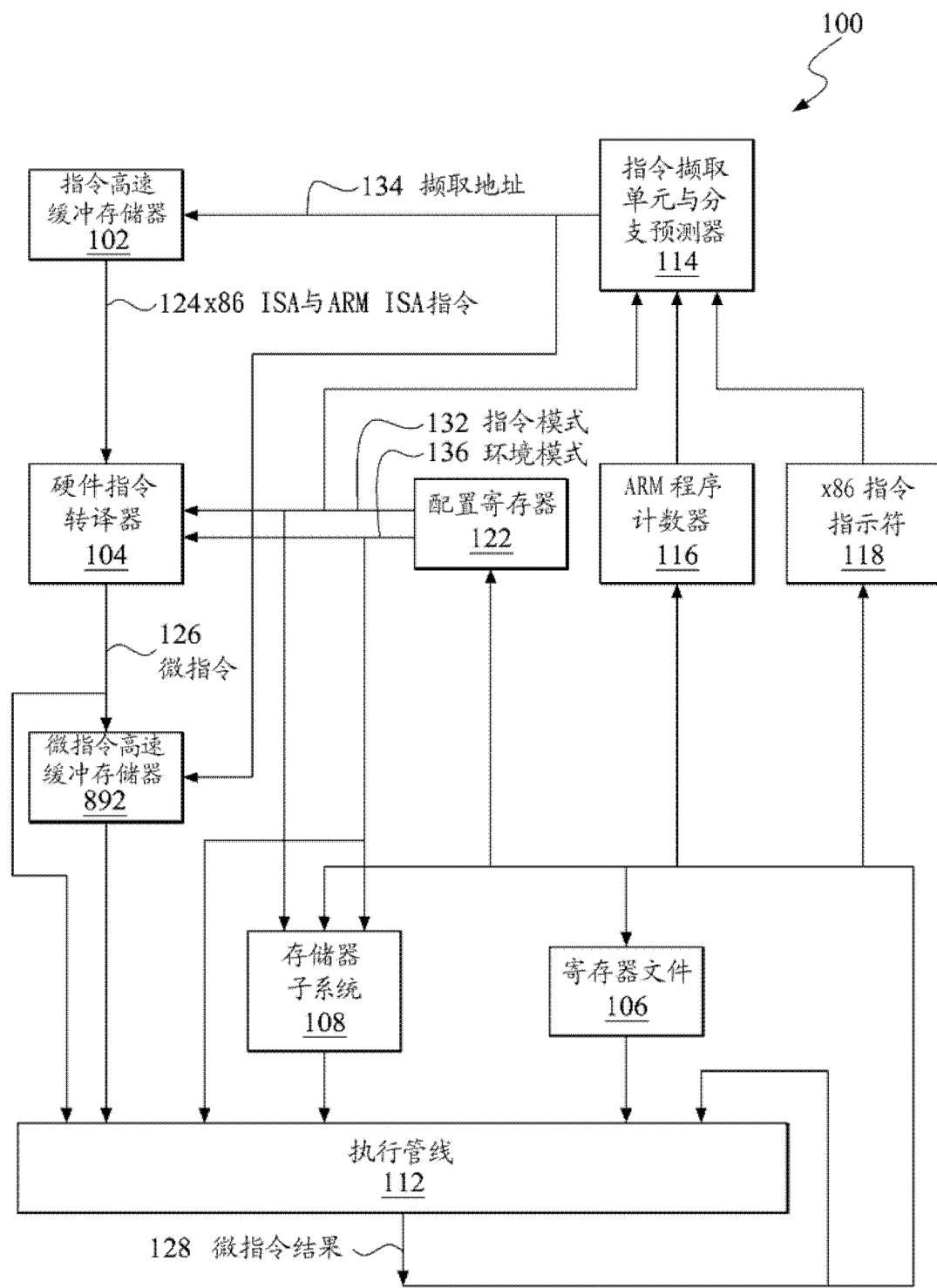


图 8

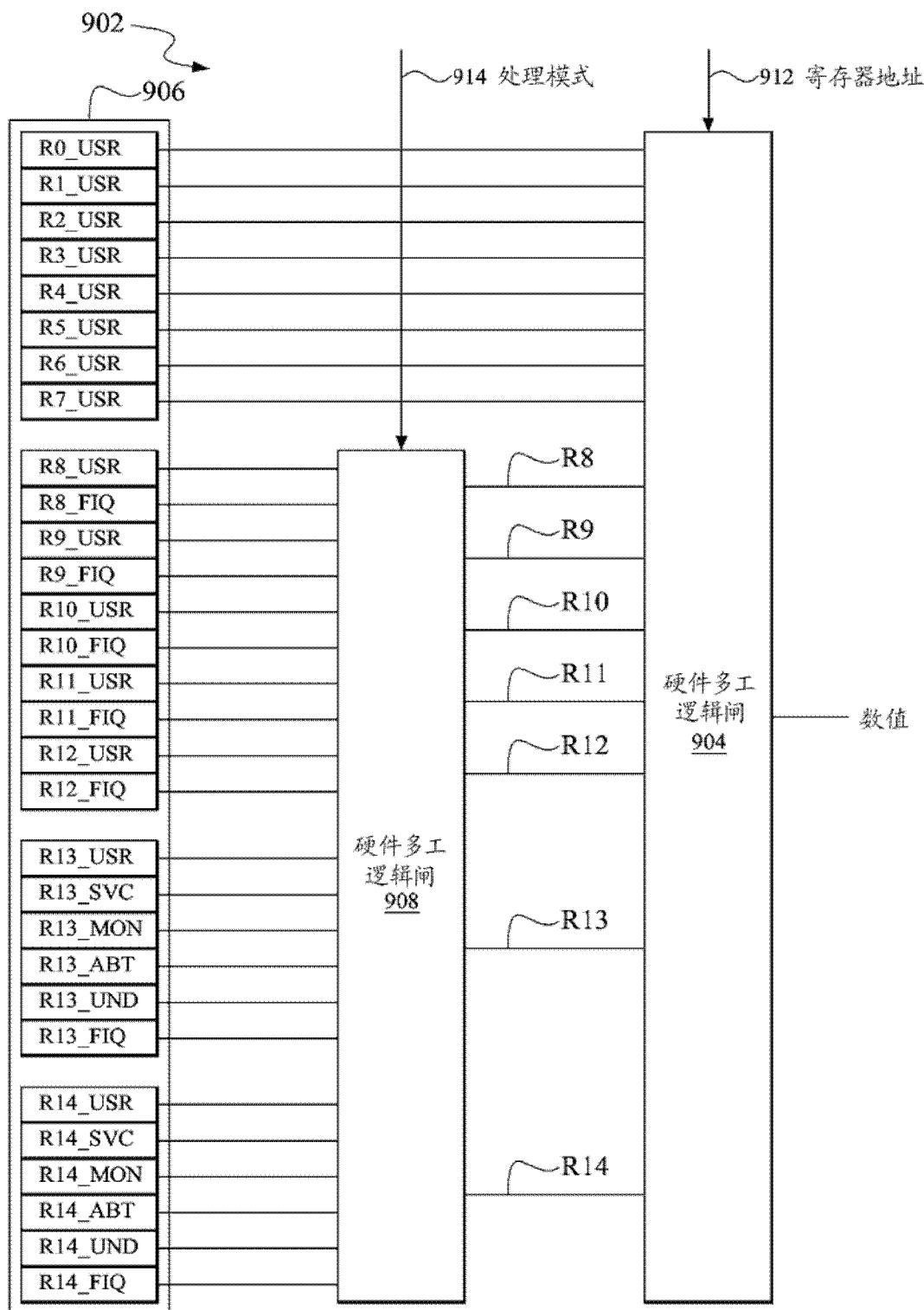


图 9

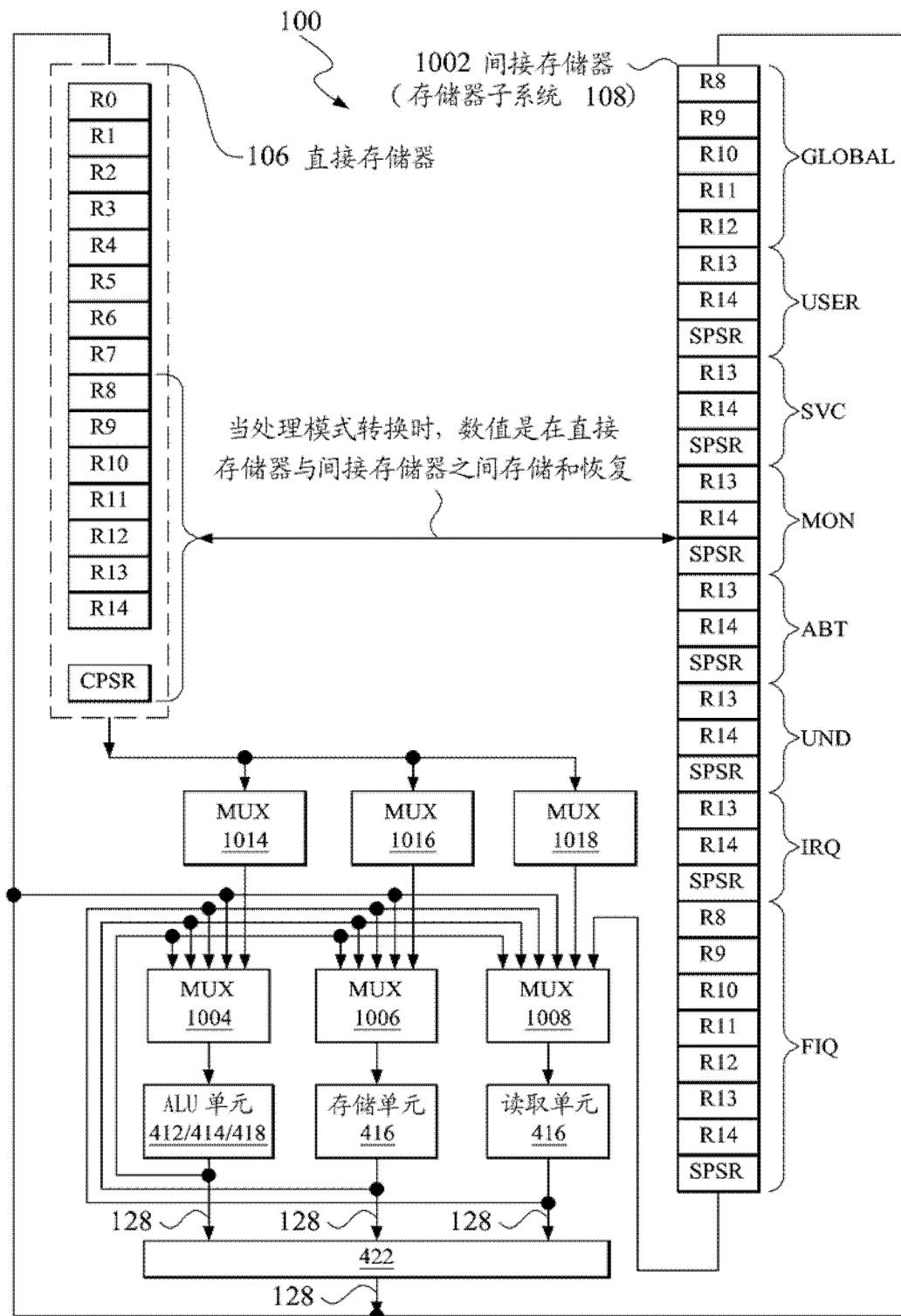


图 10

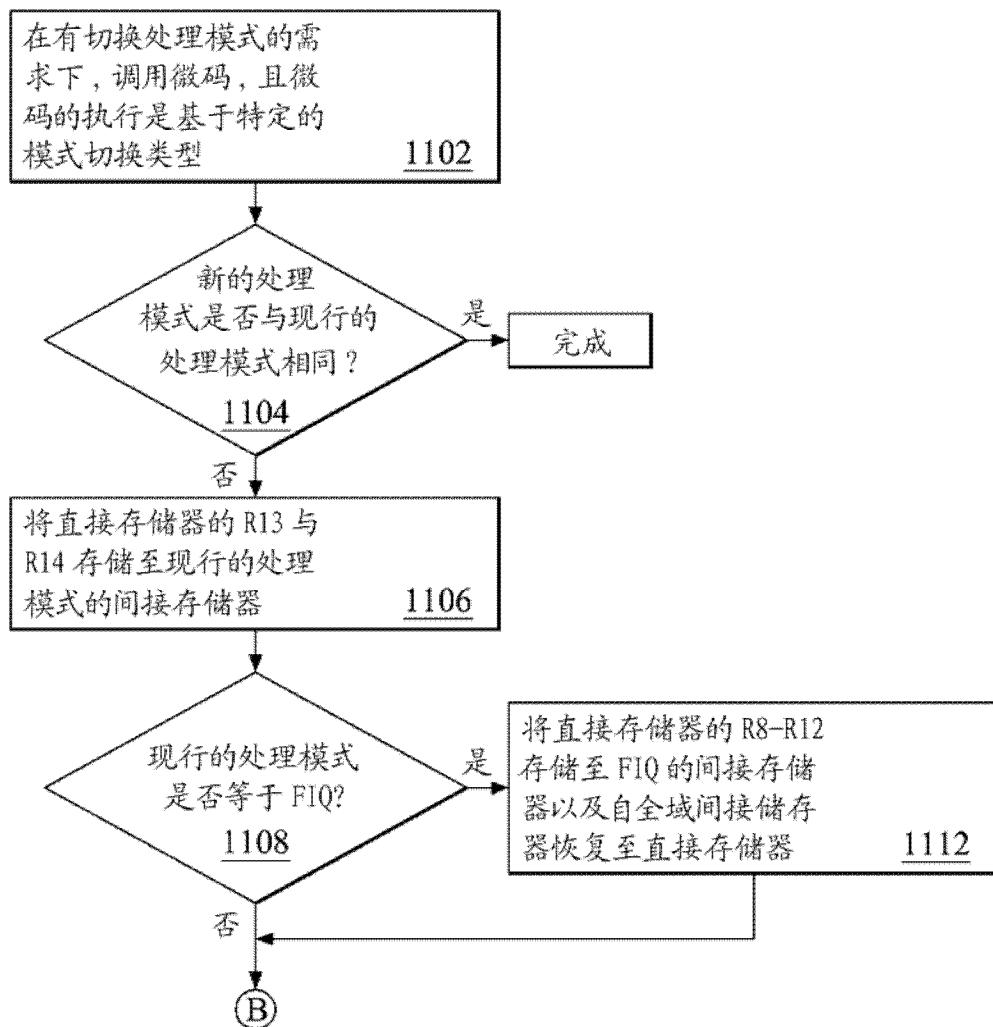


图 11A

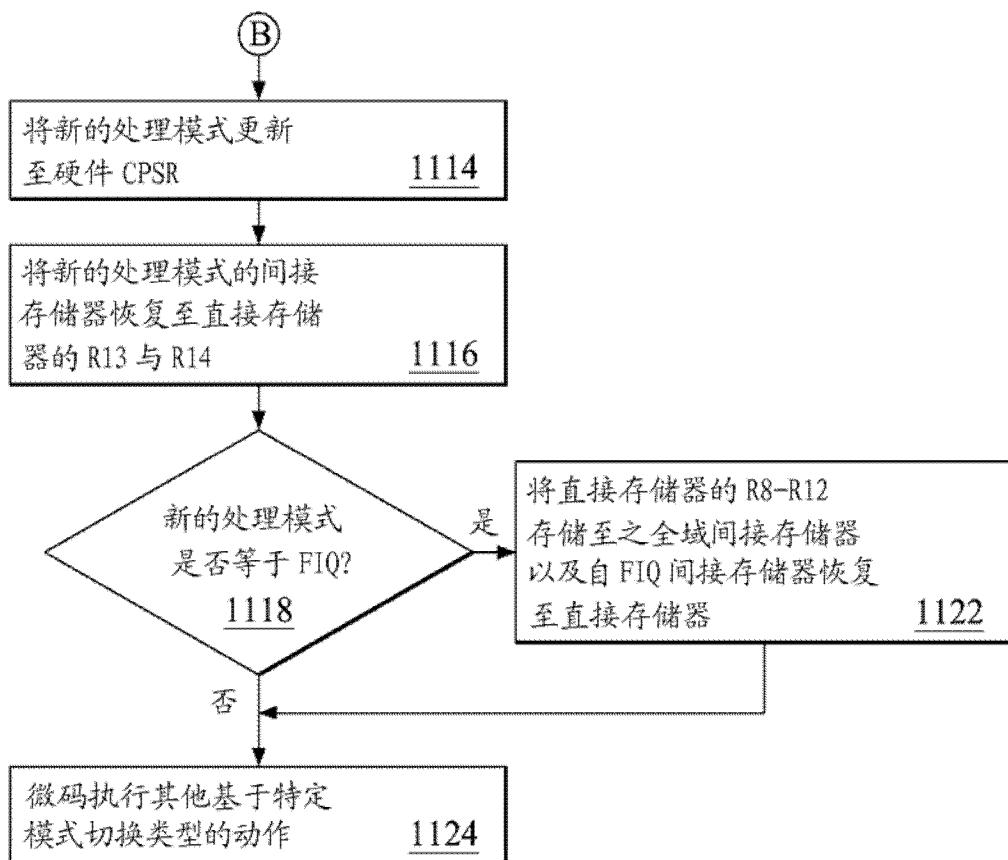


图 11B

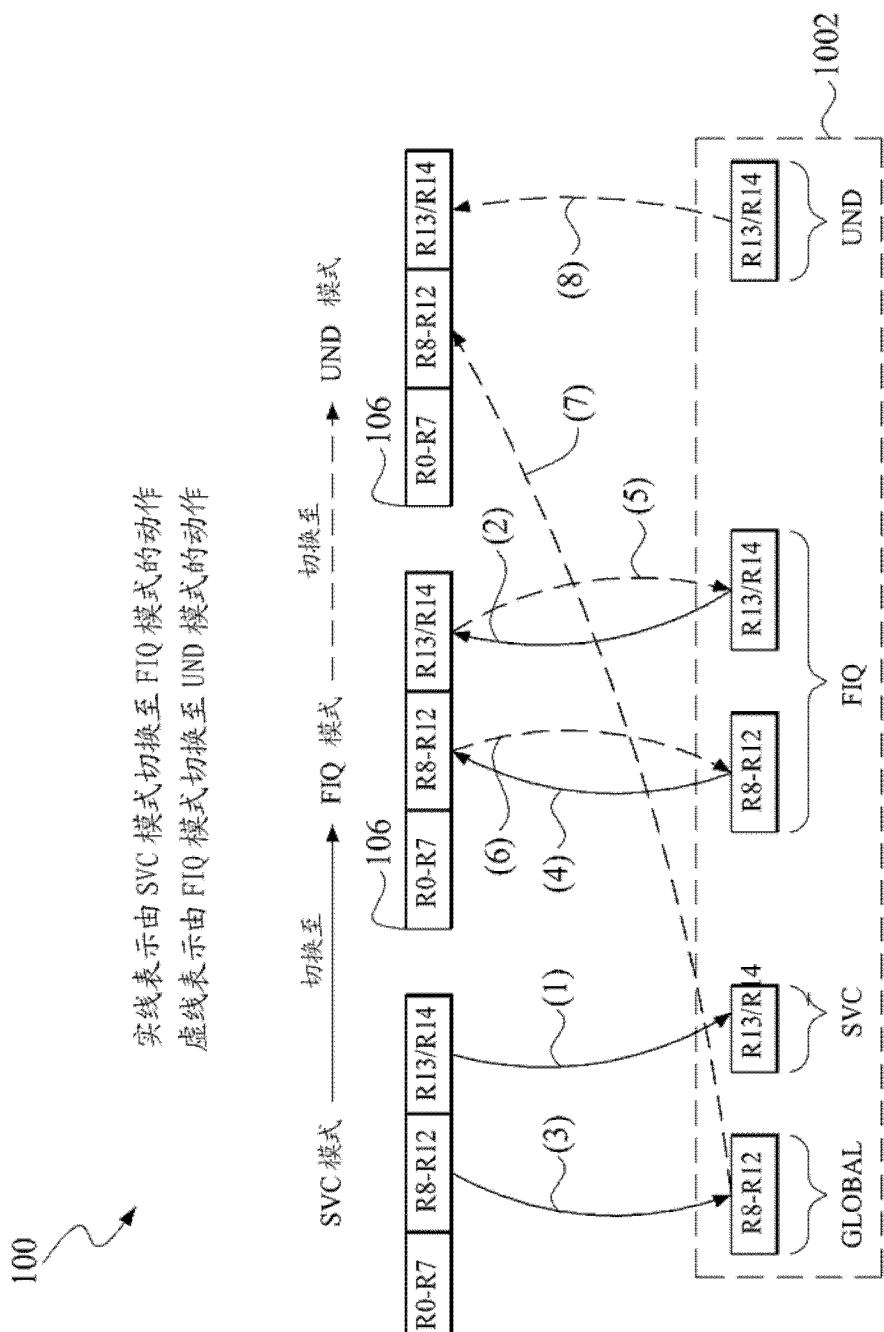


图 12

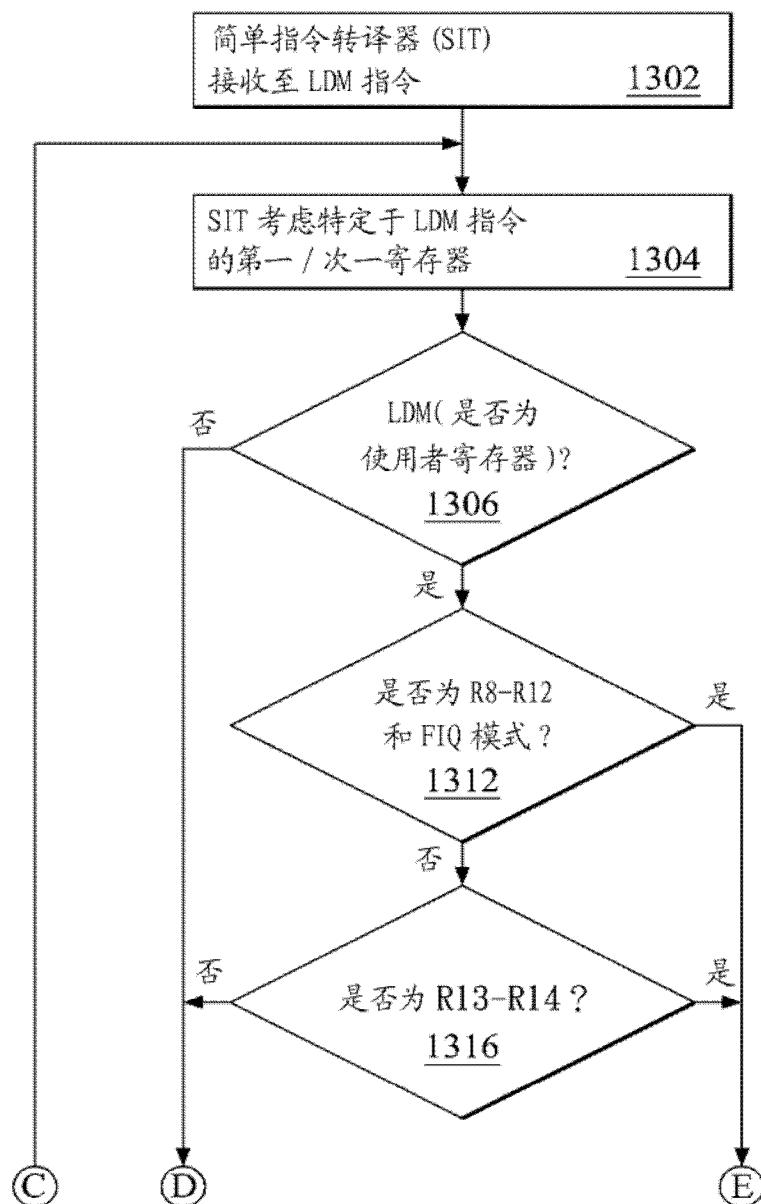


图 13A

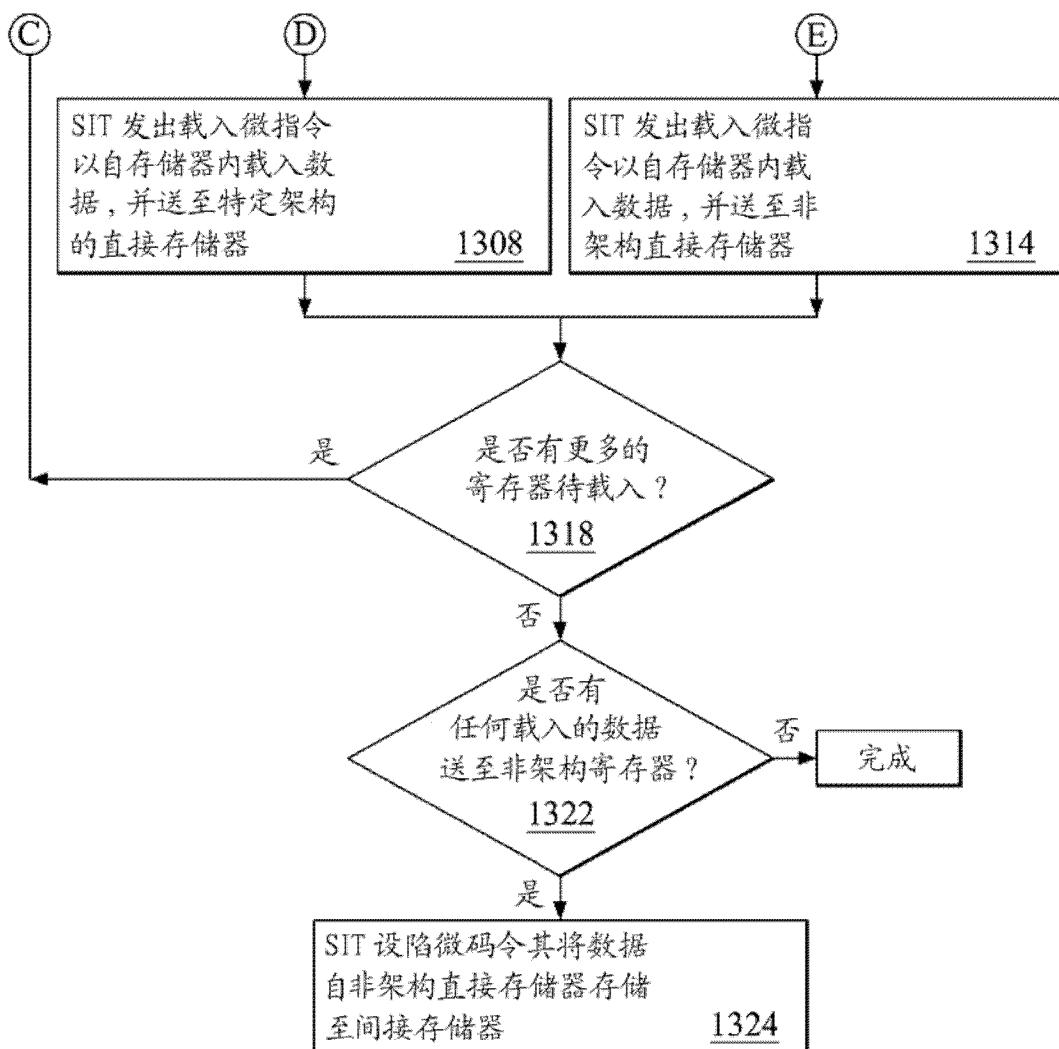


图 13B

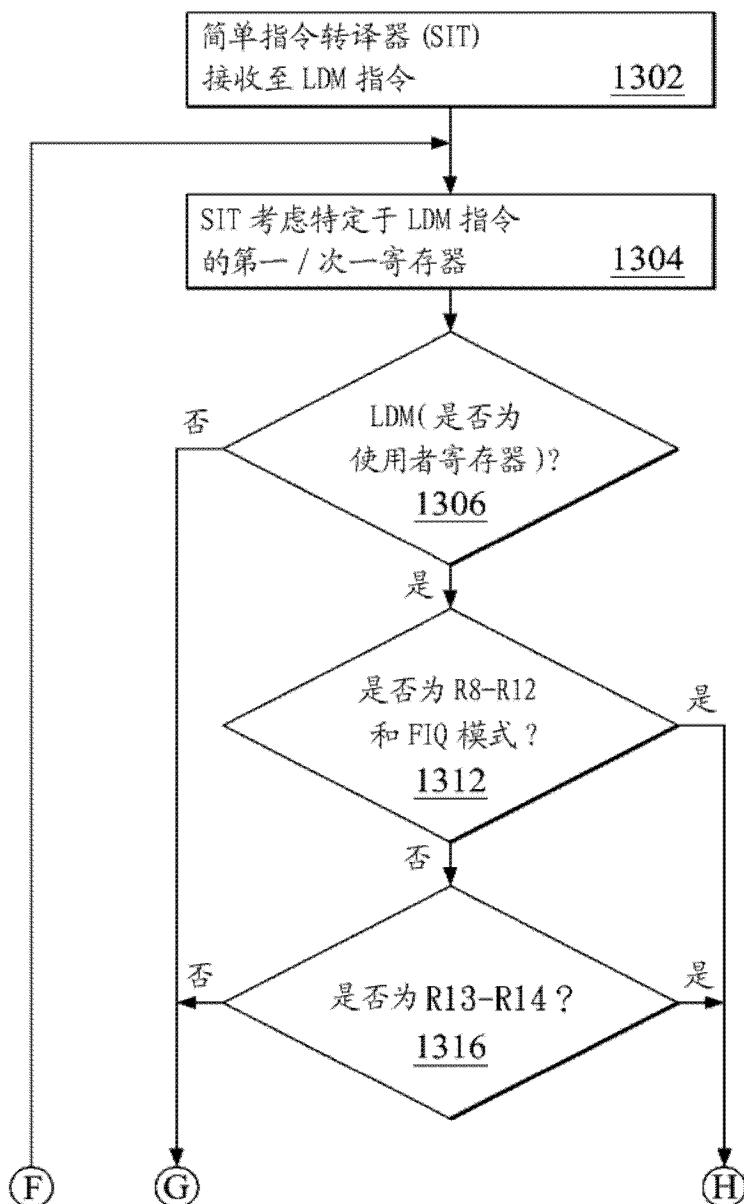


图 14A

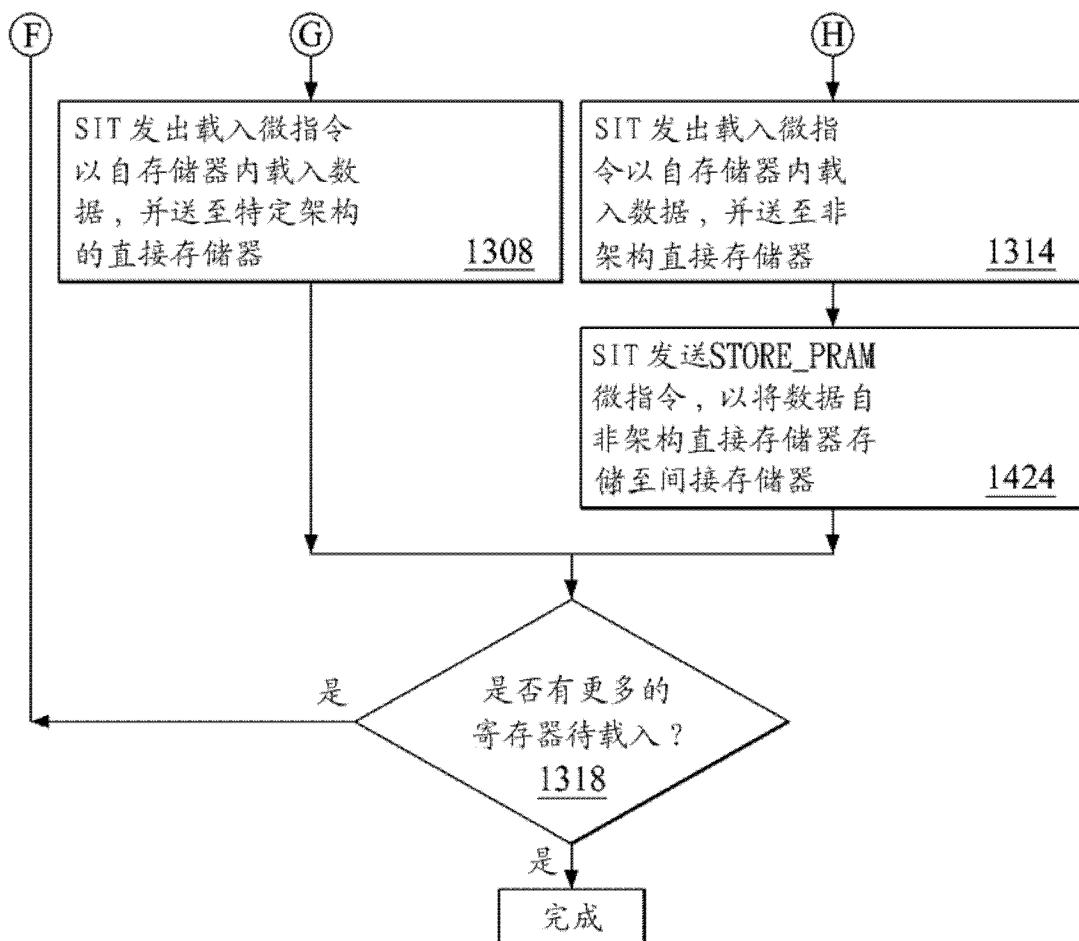


图 14B

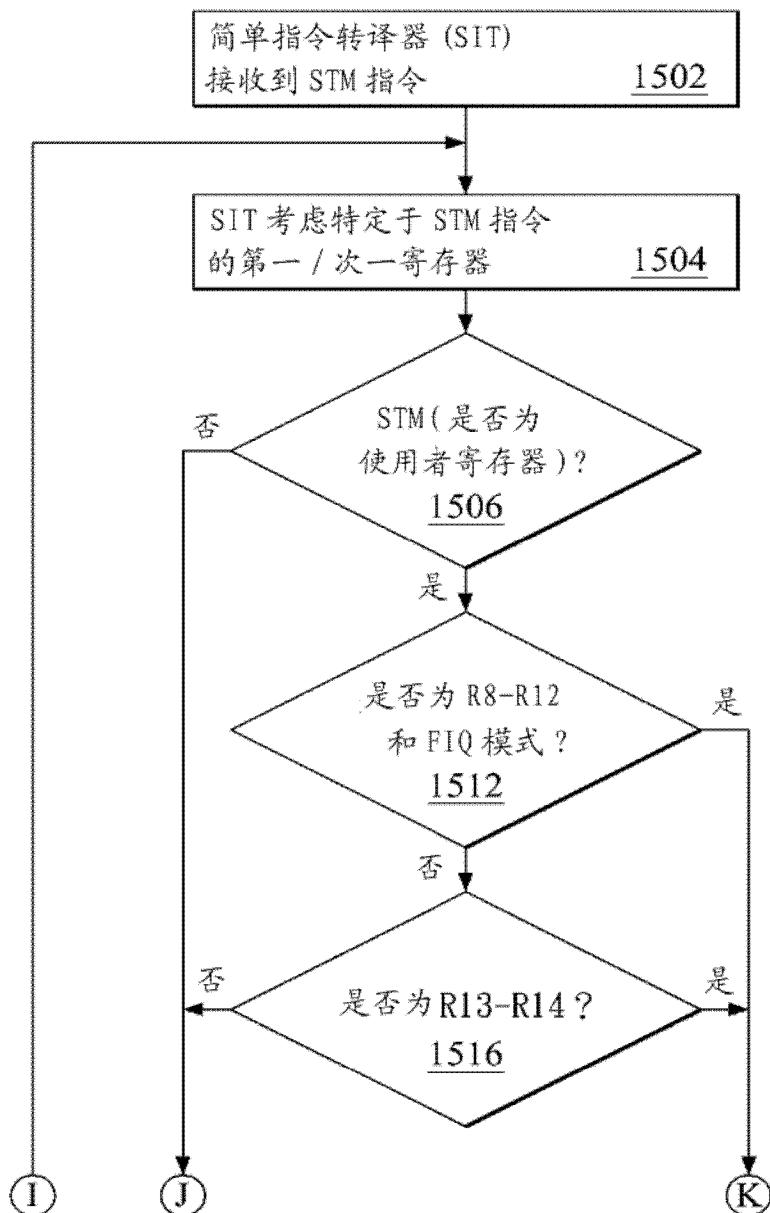


图 15A

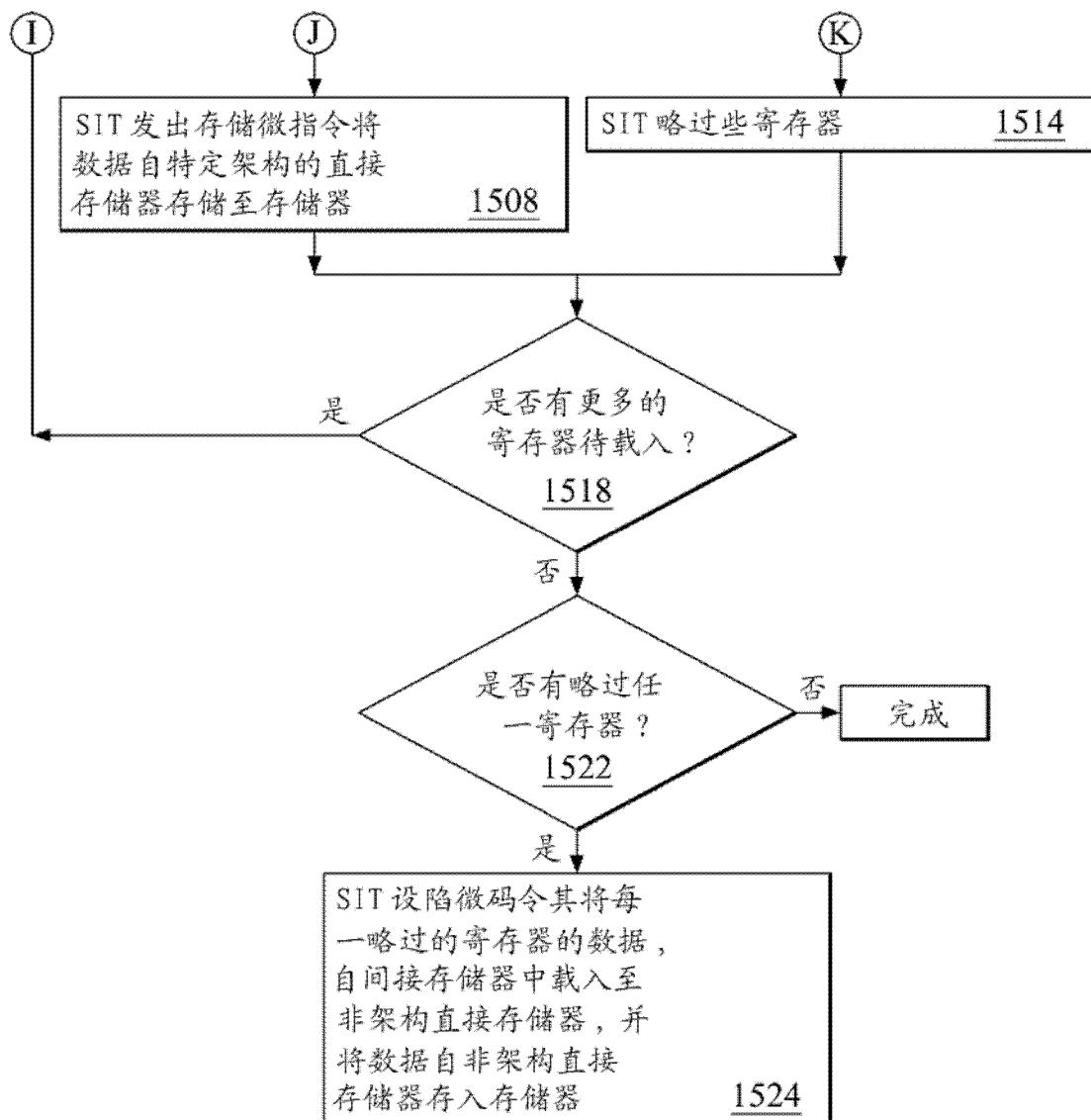


图 15B

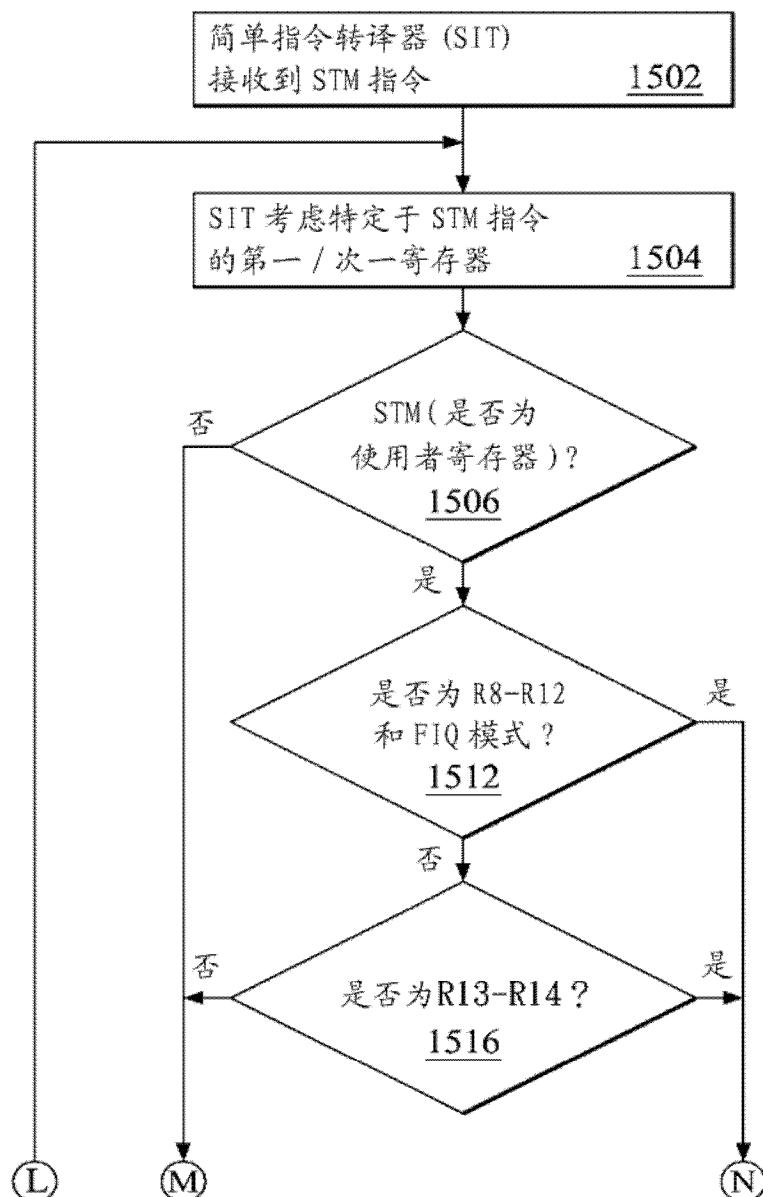


图 16A

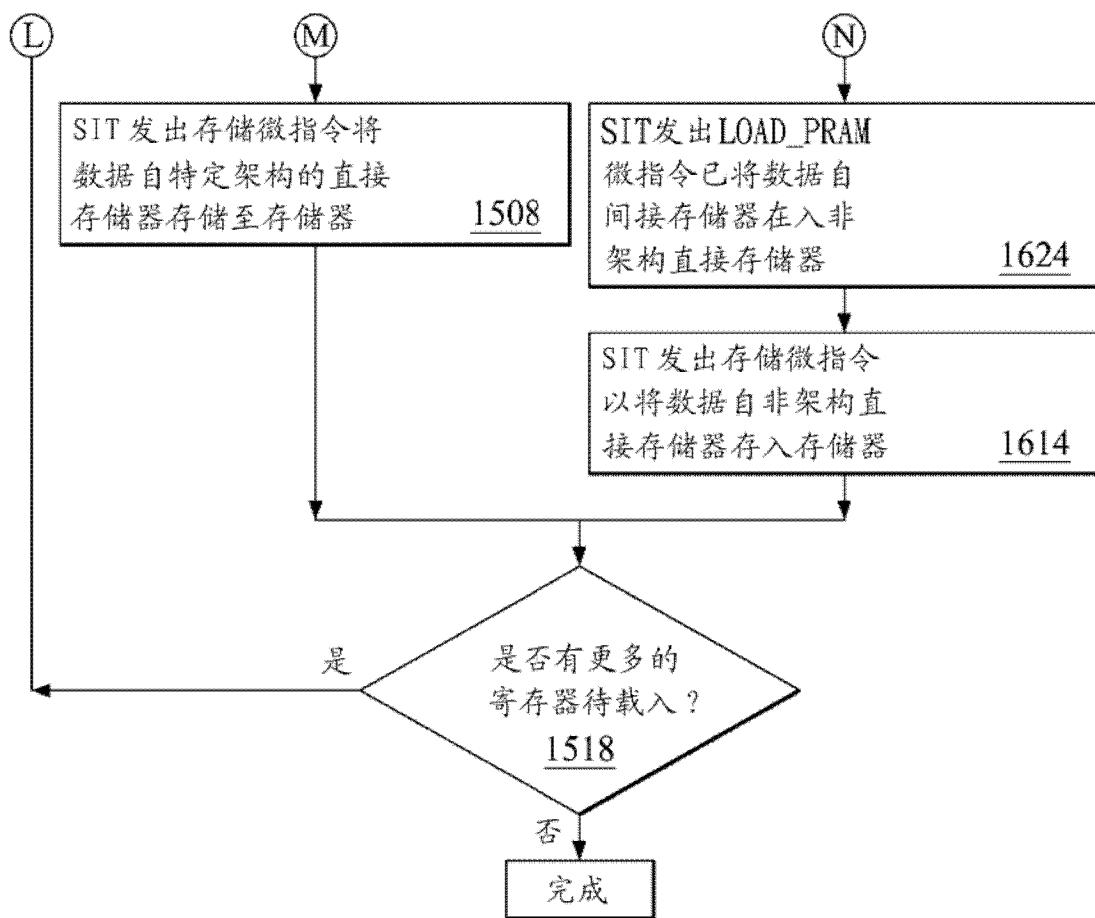


图 16B