



## (12) 发明专利

(10) 授权公告号 CN 102937889 B

(45) 授权公告日 2016. 05. 11

(21) 申请号 201210103501. 3

13/412, 914 2012. 03. 06 US

(22) 申请日 2012. 04. 09

13/416, 879 2012. 03. 09 US

## (30) 优先权数据

61/473, 062 2011. 04. 07 US

(73) 专利权人 威盛电子股份有限公司

61/473, 067 2011. 04. 07 US

地址 中国台湾新北市

61/473, 069 2011. 04. 07 US

(72) 发明人 G. 葛兰. 亨利 泰瑞. 派克斯  
罗德尼. E. 虎克

13/224, 310 2011. 09. 01 US

(74) 专利代理机构 北京市柳沈律师事务所  
11105

61/537, 473 2011. 09. 21 US

代理人 钱大勇

61/541, 307 2011. 09. 30 US

(51) Int. Cl.

61/547, 449 2011. 10. 14 US

G06F 9/30(2006. 01)

61/555, 023 2011. 11. 03 US

(56) 对比文件

13/333, 520 2011. 12. 21 US

US 5854913 A, 1998. 12. 29, 全文 .

13/333, 572 2011. 12. 21 US

US 6076155 A, 2000. 06. 13, 全文 .

13/333, 631 2011. 12. 21 US

CN 101261577 A, 2008. 09. 10, 全文 .

61/604, 561 2012. 02. 29 US

审查员 梁岩

13/412, 904 2012. 03. 06 US

权利要求书4页 说明书37页 附图12页

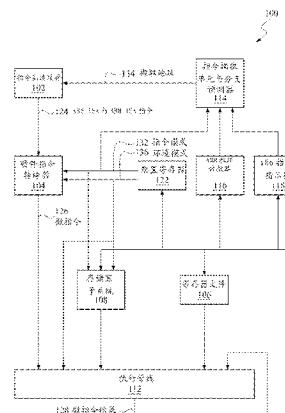
13/413, 346 2012. 03. 06 US

## (54) 发明名称

控制寄存器对应于异质指令集架构处理器

## (57) 摘要

一种可执行 x86 指令集架构 (ISA) 机器语言程序以及高级精简指令集机器 (ARM) ISA 机器语言程序的微处理器。该微处理器包含一指令模式，指示该微处理器撷取一 x86 ISA 或一 (ARM) ISA 机器语言程序指令，该微处理器更包含复数个特定模型寄存器，控制该微处理器操作方式。当该指令模式指示该微处理器撷取 x86 ISA 机器语言程序指令时，每一特定模型寄存器可经由一 x86 ISA RDMSR/WRMSR 指令而存取，且该 x86 ISAMRRC/MCRR 指令是指定该特定模型寄存器的地址。当该指令模式指示该微处理器撷取 ARM ISA 机器语言程序指令时，每一特定模型寄存器可经由一 ARM ISA MRRC/MCRR 指令而存取，且该 ARM ISA MRRC/MCRR 指令是指定该特定模型寄存器的地址。



1. 一种微处理器, 可执行x86指令集架构ISA机器语言程序以及高级精简指令集机器ARM ISA机器语言程序, 该微处理器包含:

一配置寄存器, 提供一指令模式, 指示该微处理器撷取一x86 ISA或一ARM ISA机器语言程序指令; 以及

复数个特定模型寄存器MSRs, 控制该微处理器的操作方式;

其中, 当该指令模式指示该微处理器撷取x86 ISA机器语言程序指令时, 每一特定模型寄存器可经由一x86 ISA RDMSR/WRMSR指令而存取, 且该x86 ISA RDMSR/WRMSR指令是指定该特定模型寄存器的地址;

其中, 当该指令模式指示该微处理器撷取ARM ISA机器语言程序指令时, 每一特定模型寄存器经由一ARM ISA MRRC/MCRR指令而存取, 且该ARM ISA MRRC/MCRR指令是指定该特定模型寄存器的地址,

其中所述的微处理器, 更包含:

一硬件指令转译器, 将该x86 ISA RDMSR/WRMSR指令转译成一第一序列的复数个第一程序微指令, 藉以存取于被指定地址的该特定模型寄存器, 且该硬件指令转译器将该ARM ISA MRRC/MCRR指令转译成一第二序列的复数个第二程序微指令, 藉以存取该特定模型寄存器; 以及

一执行管线, 执行该第一序列的该些第一程序微指令以执行该x86 ISA RDMSR/WRMSR指令, 并且执行该第二程序微指令该第二序列的该些第二程序微指令以执行该ARM ISA MRRC/MCRR指令,

其中, 该些第一程序微指令及该些第二程序微指令被该微处理器的一微指令集所定义, 其中, 该第一及第二程序微指令以不同于该x86 ISA及ARM ISA的该指令集所定义的编码方式的编码方式加以编码。

2. 如权利要求1所述的微处理器, 其中该ARM ISA MRRC/MCRR指令系在ARM ISA的复数个通用寄存器之一预设通用寄存器中, 指定该特定模型寄存器的该地址。

3. 如权利要求2所述的微处理器, 更包含:

一寄存器文件, 具有复数个硬件通用寄存器, 其中x86 ISA ECX寄存器以及该ARM ISA的预设通用寄存器分享该些硬件通用寄存器中相同之一者。

4. 如权利要求2所述的微处理器, 其中, 该ARM ISA的预设通用寄存器为通用寄存器R1。

5. 如权利要求1所述的微处理器, 其中, 该ARM ISA MRRC/MCRR指令系于该ARM ISA的一协同处理器空间中, 指定一预设协同处理器。

6. 如权利要求5所述的微处理器, 其中, 该ARM ISA的该协同处理器空间中的该预设协同处理器为编号被指定的协同处理器。

7. 如权利要求5所述的微处理器, 其中, 该ARM ISA MRRC/MCRR指令系于该预设协同处理器的一寄存器空间中, 指定一预设寄存器。

8. 如权利要求7所述的微处理器, 其中, 该预设协同处理器的该预设寄存器被一协同处理器寄存器地址(0, 7, 15, 0)所指定。

9. 如权利要求1所述的微处理器, 其中, 该ARM ISA MRRC/MCRR指令系在ARM ISA的复数个通用寄存器之二个预设通用寄存器中, 指定自该特定模型寄存器中读取数据到该二个预设通用寄存器, 或将数据从该二个预设通用寄存器写入该特定模型寄存器。

10. 如权利要求9所述的微处理器,更包含:

一寄存器文件,具有复数个硬件通用寄存器,其中x86 ISA EDX和EAX寄存器以及该ARM ISA的该二个预设通用寄存器分享该些硬件通用寄存器中相同之二者。

11. 如权利要求9所述的微处理器,其中该ARM ISA的该二个预设通用寄存器为通用寄存器R2和R0。

12. 如权利要求1所述的微处理器,更包含:

一简单指令转译器和一复杂指令转译器,且该简单指令转译器输出部分实现x86/ARM ISA指令的微指令,随后将控制权转移至复杂指令转译器。

13. 如权利要求1所述的微处理器,其中,由该些特定模型寄存器所控制的该微处理器的操作方式,非特定于x86 ISA及ARM ISA。

14. 一种存取复数个特定模型寄存器MSRs的方法,控制一微处理器的操作方式,而可以执行x86指令集架构ISA机器语言程序以及高级精简指令集机器ARM ISA机器语言程序,该微处理器具有一配置寄存器,提供一指令模式,该指令模式指示该微处理器撷取一x86 ISA或一ARM ISA机器语言程序指令,该存取复数个特定模型寄存器方法包含:

当该指令模式指示该微处理器撷取该x86 ISA机器语言程序时,经由一x86 ISA RDMSR/WRMSR指令来存取该些特定模型寄存器中之一者,而该x86 ISA RDMSR/WRMSR指令是用以指定该特定模型寄存器的地址;

当该指令模式指示该微处理器撷取该ARM ISA机器语言程序指令时,经由一ARM ISA MRRC/MCRR指令来存取该些特定模型寄存器中之一者,而该ARM ISA MRRC/MCRR指令是用以指定该特定模型寄存器的地址,

该方法更包含:

当该指令模式指示该微处理器撷取x86 ISA机器语言程序指令时,将该x86 ISA RDMSR/WRMSR指令转译成一第一序列的复数个第一程序微指令,藉以存取于被指定地址的该特定模型寄存器;

当该指令模式指示该微处理器撷取ARM ISA机器语言程序指令时,将该ARM ISA MRRC/MCRR指令转译成一第二序列的复数个第二程序微指令,藉以存取于被指定地址的该特定模型寄存器;以及

执行该第一序列的该些第一程序微指令以执行该x86 ISA RDMSR/WRMSR指令;以及

执行该第二序列的该些第二程序微指令以执行该ARM ISA MRRC/MCRR指令,

其中,上述转译系藉由该微处理器的一硬件指令转译器所执行。

15. 如权利要求14所述的存取复数个特定模型寄存器的方法,其中,该第一及第二程序微指令以不同于该x86 ISA及ARM ISA的该指令集所定义的编码方式的编码方式加以编码。

16. 如权利要求14所述的存取复数个特定模型寄存器的方法,其中,该微处理器具有一简单指令转译器和一复杂指令转译器,其中,该方法包含:

该简单指令转译器输出部分实现x86/ARM ISA指令的微指令,随后将控制权转移至复杂指令转译器。

17. 如权利要求14所述的存取复数个特定模型寄存器的方法,其中,该微处理器操作方式由该些特定模型寄存器所控制,且该微处理器操作方式非指定于x86 ISA及ARM ISA。

18. 一种微处理器,包含:

复数个处理核心,该些处理核心的每一处理核心包含:

一硬件指令转译器,将x86指令集架构ISA机器语言程序以及高级精简指令集机器ARM ISA机器语言程序转译成复数个微指令,且该些微指令由该微处理器的一微指令集所定义;其中,该些微指令以不同于该x86 ISA及ARM ISA的该指令集所定义的编码方式的编码方式加以编码;以及

一执行管线,耦合于该硬件指令转译器,其中,该执行管线执行该微指令来产生该x86 ISA及该ARM ISA指令所定义的结果,

其中,该些处理核心中的每一核心更包含:一指令模式,指示该硬件指令转译器将x86 ISA或ARM ISA指令转译成该些微指令。

19. 如权利要求18所述的微处理器,更包含:

一设定寄存器,储存与该些处理核心的相关指示符,其中,当该些处理核心的每一处理核心重置时,其利用该相关指示符来确定启动而作为一x86 ISA核心或是一ARM ISA核心。

20. 如权利要求19所述的微处理器,其中,该相关指示符的一预设值在该微处理器的一微码内编码为常数。

21. 如权利要求20所述的微处理器,其中,该微码可修正该相关指示符的该预设值。

22. 如权利要求20所述的微处理器,其中,该相关指示符的该预设值可藉由熔断微处理器的熔丝而修正。

23. 如权利要求19所述的微处理器,其中,该设定寄存器是可设定来指示该些处理核心中的至少一者将启动而作为一x86 ISA核心,并且指示该些处理核心中的其他至少一者将启动而作为一ARM ISA核心。

24. 如权利要求18所述的微处理器,其中,

该些处理核心的该指令模式于该微处理器运作时,在该x86 ISA及该ARM ISA间动态切换,藉以同步执行x86 ISA以及ARM ISA机器语言程序。

25. 如权利要求18所述的微处理器,其中,该微处理器可设定为同步执行x86 ISA以及ARM ISA机器语言程序,其藉由设定一或多个该些处理核心作为一ARM ISA核心,以及设定一或多个该些处理核心作为一x86 ISA核心。

26. 如权利要求18所述的微处理器,其中,该硬件指令转译器直接提供该些微指令至该执行管线来执行,藉以产生该x86 ISA及该ARM ISA指令所定义的结果。

27. 如权利要求18所述的微处理器,其中,该硬件指令转译器所提供到该执行管线的该些微指令,系有别于该执行管线执行该些微指令所产生的结果。

28. 如权利要求18所述的微处理器,其中,该硬件指令转译器所提供到该执行管线的该些微指令用以执行,而不是以该些微指令作为该执行管线的结果。

29. 如权利要求18所述的微处理器,其中,该执行管线不具有直接执行该x86 ISA以及ARM ISA指令的功能。

30. 一种运作一包含复数个处理核心的微处理器的方法,该些处理核心的每一核心包含一指示符,该指示符用以指示一x86指令集架构ISA机器语言程序或一高级精简指令集机器ARM ISA机器,该方法包含:

对该些处理核心的每一核心来说:

确定该指示符指示该x86 ISA或该ARM ISA;

当该指示符指示该x86 ISA时,根据该x86 ISA将机器语言指令转译成复数个微指令,而当该指示符指示该ARM ISA时,根据该ARM ISA将机器语言指令转译成复数个微指令,其中,该些微指令以不同于该x86 ISA及ARM ISA的该指令集所定义的编码方式的编码方式加以编码,其中,上述转译由该微处理器的一硬件指令转译器所执行;以及

当该指示符指示该x86 ISA时,执行该些微指令以产生该x86 ISA所定义的微指令,且当该指示符指示该ARM ISA时,执行该些微指令以产生该ARM ISA所定义的微指令,其中,上述执行由该微处理器的一执行管线所处理,且该执行管线耦合于该硬件指令转译器。

31. 如权利要求30所述的方法,更包含:

修补该微处理器的微码,藉以修正该指示符的预设值。

32. 如权利要求30所述的方法,更包含:

藉由熔断该微处理器的熔丝,以修正该指示符的预设值。

33. 如权利要求30所述的方法,更包含:

当重置时,该些处理核心的每一核心检查一设定寄存器,以确定启动而作为一x86 ISA核心或是一ARM ISA核心。

34. 如权利要求33所述的方法,其中,该设定寄存器可设定来指示该些处理核心中的至少一者将启动作为一x86 ISA核心,以及设定该些处理核心中的其他至少一者将启动作为ARM ISA核心。

35. 如权利要求30所述的方法,更包含:

于该微处理器运作时,在该x86 ISA及该ARM ISA间动态切换该些处理核心的指示符,藉以同步执行x86 ISA以及ARM ISA机器语言程序。

## 控制寄存器对应于异质指令集架构处理器

### 技术领域

[0001] 本发明关于微处理器的技术领域,特别是关于微处理器多重指令集架构的支援。

### 背景技术

[0002] 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开发出来的高级精简指令集机器(advanced risc machines, ARM)架构是计算机领域中两种广为人知的处理器架构。许多使用ARM或x86处理器的计算机系统已经出现,并且,对于此计算机系统的需求正在快速成长。现今,ARM架构处理器核心为主宰低功耗、低价位的计算机市场,例如手机、手持式电子产品、平板计算机、网络路由器与集线器、机顶盒等。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,例如膝上计算机、桌上型计算机与服务器等。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。在行动运算市场,如智能型手机,这两种架构已经开始激烈竞争。在膝上计算机、桌上型计算机与服务器市场,可以预期这两种架构将会有更频繁的竞争。

[0003] 前述竞争态势使得计算机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计算机系统的消费个体,基于成本效率的考量,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计算机系统。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计算机系统,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架构的处理器上执行程序,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架构的计算机系统(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,亦会遭遇这样的问题,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。

[0004] 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。

[0005] 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,而其指令集往往不相容于新硬件。举例来说,IBM 360系统Model 30即具有相容于IBM 1401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。Model 30具有360系统与1401系统的只读储存控制(Read Only Storage,ROS)),使其在辅助储存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。Siberman与Ebcio glu于Computer, June 1993, No. 6提出的文章“An Architectural Framework for Supporting Heterogeneous Instruction-Set Architectures”中揭露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,其所揭露的系统包含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日公告的美国专利第7,047,394号专利案, Van Dyke et al. 揭露一处理器,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 et al. 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架构前端管线系用于非规则(irregular)软件程序(如作业系统),而Fujitsu FR-V(超长指令字)架构的前端管线系用于多介质应用程序,其将一增加的超长指令字队列提供予FR-V超长指令字之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)发表之论文集(ISBN 978-3-86644-298-6)的文章“OROCHI:A Multiple Instruction Set SMT Processor”。文中提出的方法用以降低整个系统在异质系统单芯片(SOC)装置(如德州仪器OMAP应用处理器)内所占据的空间,此异质系统单芯片装置具有一个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,而只是整合于同一芯片上的不同处理核心。

[0006] 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,亦被用于支援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)计算机的Motorola 68K-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处理器的麦金塔计算机上执行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(亦即程序码转译软件(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 et al.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观测器(Virtual Machine Monitor),以并行处理(parallelize)与储存超长指令字原始码(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系一具有软件动态转译器的机器模拟器(machine emulator)。QEMU可在多种主系统(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),将其转换至主系统指令集。产生的二进制码储存于一转译高速缓存以利重复取用。...QEMU[较的其他动态转译器]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 Wang Hao 的学位论文“ARM Instruction Set Simulation on Multi-core x86 Hardware”。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。

[0007] 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散布新程序所需的管道;对原程序散布者存在潜在的授权或是著作权侵害的风险)。

## 发明内容

[0008] 本发明的一实施例提供一种微处理器,该微处理器可执行x86指令集架构(instruction set architecture;ISA)机器语言程序以及高级精简指令集机器(Advanced RISC Machines;ARM)ISA机器语言程序。该微处理器包含一指令模式,指示该微处理器撷取一x86 ISA或一ARM ISA机器语言程序指令。该微处理器更包含复数个特定模型寄存器(MSRs),用以控制微处理器操作方式。当该指令模式指示该微处理器撷取x86 ISA机器语言程序指令时,每一特定模型寄存器可经由一x86 ISA RDMSR/WRMSR53指令而存取,且该x86 ISA MRRC/MCRR指令是指定该特定模型寄存器的地址。当该指令模式指示该微处理器撷取ARM ISA机器语言程序指令时,每一特定模型寄存器可经由一ARM ISA MRRC/MCRR指令而存取,且该ARM ISA MRRC/MCRR指令是指定该特定模型寄存器的地址。

[0009] 本发明的另一实施例提供一种存取复数个特定模型寄存器(MSRs)方法,该些特定模型寄存器控制一微处理器操作方式,并可执行x86指令集架构(ISA)机器语言程序以及高级精简指令集机器(ARM)ISA机器语言程序,该微处理器具有一指令模式,该指令模式指示

该微处理器撷取一x86 ISA或一ARM ISA机器语言程序指令。该方法包含当该指令模式指示该微处理器撷取该x86 ISA机器语言程序时,经由一x86 ISA RDMSR/WRMSR指令来存取该些特定模型寄存器中之一者,而该x86 ISA RDMSR/WRMSR指令用以指定该特定模型寄存器的一地址。该方法更包含当该指令模式指示该微处理器撷取该ARM ISA机器语言程序指令时,经由一ARM ISA MRRC/MCRR指令来存取该些特定模型寄存器中之一者,而该ARM ISA MRRC/MCRR指令用以指定该特定模型寄存器的一地址。

[0010] 本发明的又一实施例提供一种计算机程序产品,其编码于至少一计算机可读取储存介质,以使用于一运算装置,该计算机程序产品包含一计算机可读程序码,应用于该计算机可读取储存介质,藉以指定一微处理器可执行x86指令集架构(ISA)机器语言程序以及高级精简指令集机器(ARM)ISA机器语言程序。该计算机可读程序码包含一第一程序码,用以指定一指令模式,该指令模式指示该微处理器撷取一x86 ISA或一ARMISA机器语言程序指令。该计算机可读程序码更包含一第二程序码,用以指定复数个特定模型寄存器(MSRs),且该些特定模型寄存器控制微处理器操作方式。当该指令模式指示该微处理器撷取x86 ISA机器语言程序指令时,每一特定模型寄存器可经由一x86 ISA RDMSR/WRMSR53指令而存取,且该x86 ISA MRRC/MCRR指令是指定该特定模型寄存器的地址。当该指令模式指示该微处理器撷取ARM ISA机器语言程序指令时,每一特定模型寄存器可经由一ARM ISA MRRC/MCRR指令而存取,且该ARM ISA MRRC/MCRR指令是指定该特定模型寄存器的地址。

[0011] 本发明的又一实施例提供一种微处理器,其执行x86指令集架构(ISA)机器语言程序以及高级精简指令集机器(ARM)ISA机器语言程序。该微处理器包含一指令模式及复数个硬件寄存器,该指令模式指示该微处理器撷取一x86 ISA或一ARM ISA机器语言程序指令。当该指令模式指示该微处理器撷取x86 ISA机器语言程序指令时,该些硬件寄存器储存x86 ISA架构状态。当该指令模式指示该微处理器撷取ARM ISA机器语言程序指令时,该些硬件寄存器储存ARM ISA架构状态。

[0012] 本发明的另一实施例提供一种运作一微处理器可执行x86指令集架构(ISA)机器语言程序以及高级精简指令集机器(ARM)ISA机器语言程序的方法。该方法包含设定一指令模式以指示该微处理器撷取一x86 ISA或ARM ISA机器语言程序指令。该方法更包含当该指令模式指示该微处理器撷取该x86 ISA机器语言程序指令时,在该微处理器的复数个硬件寄存器内储存x86 ISA架构状态。该方法更包含当该指令模式指示该微处理器撷取该ARM ISA机器语言程序指令时,在该微处理器的复数个硬件寄存器内储存ARM ISA架构状态。

[0013] 本发明的另一实施例提供一种计算机程序产品,其编码于至少一计算机可读取储存介质,以使用于一运算装置。该计算机程序产品包含一计算机可读程序码,其应用于该计算机可读取储存介质,藉以指定一微处理器可执行x86指令集架构(ISA)机器语言程序以及高级精简指令集机器(ARM)ISA机器语言程序。该计算机可读程序码包含一第一程序码,用以指定一指令模式,该指令模式指示该微处理器撷取一x86 ISA或一ARMISA机器语言程序指令。该计算机可读程序码更包含一第二程序码,用以指定复数个硬件寄存器。当该指令模式指示该微处理器撷取x86 ISA机器语言程序指令时,该些硬件寄存器储存x86 ISA架构状态。当该指令模式指示该微处理器撷取ARM ISA机器语言程序指令时,该些硬件寄存器储存x86 ISA架构状态。

[0014] 本文所揭露的本发明的实施例的必要条件需要一多核心处理器设计,且该多核心

处理器可执行x86指令集架构(ISA)机器语言程序以及高级精简指令集(RISC)机器(ARM)ISA机器语言程序。

[0015] 本发明的另一实施例提供一种微处理器。该微处理器包含复数个处理核心。该些处理核心的每一处理核心包含一硬件指令转译器,将x86指令集架构(ISA)机器语言程序以及高级精简指令集机器(ARM)ISA机器语言程序转译成复数个微指令,且该些微指令由该微处理器的一微指令集所定义。该些微指令以一独特编码方式加以编码,且该独特编码方式,被该x86 ISA及ARM ISA的该指令集所定义的指令所编码。该些处理核心的每一处理核心更包含一执行管线,耦合于该硬件指令转译器。该执行管线执行该些微指令来产生该x86 ISA及该ARM ISA指令所定义的结果。

[0016] 本发明的另一实施例提供一种运作一包含复数个处理核心的微处理器方法,该些处理核心的每一核心具有一指令,其用以指示一x86指令集架构(ISA)机器语言程序或一高级精简指令集机器(ARM)ISA机器。对该些处理核心的每一核心来说,该方法包含确定该指令指示该x86 ISA或该ARMISA,以及当该指令指示该x86 ISA时根据该x86 ISA将机器语言指令转译成复数个微指令,而当该指令指示该ARM ISA时,根据该ARM ISA将机器语言指令转译成复数个微指令。该些微指令由该微处理器的一微指令集所定义,而该些微指令以一独特编码方式加以编码,且该独特编码方式,被该x86 ISA及ARM ISA的该指令集所定义的指令所编码。转译由该微处理器的一硬件指令转译器所执行。该方法更包含当该指令指示该x86 ISA时,执行该些微指令以产生该x86 ISA所定义的微指令,且当该指令指示该ARM ISA时,执行该些微指令以产生该ARM ISA所定义的微指令。执行由该微处理器的一执行管线所处理,且该执行管线耦合于该硬件指令转译器。

[0017] 本发明的另一实施例提供一种计算机程序产品,编码于至少一计算机可读取储存介质,以使用于一运算装置。该计算机程序产品包含一计算机可读程序码,应用于该计算机可读取储存介质,藉以指定一微处理器。该计算机可读程序码包含程序码,用以指定复数个处理核心。该些处理核心的每一核心包含一硬件指令转译器,其将x86指令集架构(ISA)机器语言程序指令以及高级精简指令集机器(ARM)ISA机器语言程序指令转译成复数个微指令。该些微指令以一独特编码方式加以编码,且该独特编码方式,被该x86 ISA及ARM ISA的该指令集所定义的指令所编码。该些处理核心的每一核心更包含一执行管线,其耦合于该硬件指令转译器。该执行管线执行该些微指令来产生该x86 ISA及该ARM ISA指令所定义的结果。

[0018] 本发明的另一实施例提供一种微处理器,其可运作为一x86指令集架构(ISA)微处理器以及一高级精简指令集机器(ARM)ISA微处理器。该微处理器包含一第一储存器、一第二储存器及一第三储存器,该第一储存器储存该微处理器的x86 ISA特定状态,该第二储存器储存该微处理器的ARMISA特定状态,该第三储存器储存该微处理器的非ISA特定状态。响应于重置时,该微处理器将该第一储存器初始化至该x86 ISA所指定的预设值,并将该第二储存器初始化至该ARM ISA所指定的预设值,然后将该第三储存器初始化至预设值,接着开始撷取一第一ISA指令。该第一ISA指令系该x86 ISA或该ARM ISA,且一第二ISA为该另一ISA。该微处理器响应于一或多个该第一ISA指令,而更新该第三储存器的至少一部分。响应于该第一ISA指令中随后的指令,其指示该微处理器重置该第二ISA,应于该第一ISA指令中随后的指令,其指示该微处理器重置该第二ISA避免修正储存在该第三储存器的该非ISA特

定状态，然后开始撷取该第二ISA指令。

[0019] 本发明的另一实施例提供一种可使一微处理器运作为一x86指令集架构(ISA)微处理器以及一高级精简指令集机器(ARM)ISA微处理器的方法，该方法包含响应于该微处理器的一重置时，将一第一储存器初始化至该x86ISA所指定的预设值，然后将一第二储存器初始化至该ARM ISA所指定的预设值，接着将一第三储存器初始化至预设值，之后撷取一第一ISA指令。该第一储存器储存该微处理器的x86 ISA特定状态。该第二储存器储存该微处理器的ARM ISA特定状态。该第三储存器储存该微处理器的非ISA特定状态。该第一ISA指令系该x86 ISA或该ARM ISA，且一第二ISA为该另一ISA。该方法更包含响应于该第一ISA指令中随后的指令时，而指示该微处理器重置该第二ISA，然后避免修正储存在该第三储存器的该非ISA特定状态，以及撷取该第二ISA指令。

[0020] 本发明的另一实施例提供一种计算机程序产品，其编码于至少一计算机可读取储存介质，以使用于一运算装置。该计算机程序产品包含一计算机可读程序码，其应用于该计算机可读取储存介质，藉以指定一微处理器可运作为一x86指令集架构(ISA)微处理器以及一高级精简指令集机器(ARM)ISA微处理器。该计算机可读程序码包含一第一程序码，其用以指定一第一储存器储存该微处理器的x86 ISA特定状态。该计算机可读程序码更包含一第二程序码，其用以指定一第二储存器储存该微处理器的ARM ISA特定状态。该计算机可读程序码更包含一第三程序码，其用以指定一第三储存器储存该微处理器的非ISA特定状态。响应于一重置时，该微处理器将该第一储存器初始化至该x86 ISA所指定的预设值，然后将该第二储存器初始化至该ARM ISA所指定的预设值，接着将该第三储存器初始化至预设值，以及开始撷取一第一ISA指令。该第一ISA指令系该x86 ISA或该ARM ISA，且一第二ISA为该另一ISA。该微处理器响应于一或多个该第一ISA指令，而更新该第三储存器的至少一部分。应于该第一ISA指令中随后的指令时，而指示该微处理器重置该第二ISA，且该微处理器避免修正储存在该第三储存器的该非ISA特定状态，以及开始撷取该第二ISA指令。

## 附图说明

[0021] 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。

[0022] 图2是一方块图，详细显示第1图的硬件指令转译器。

[0023] 图3是一方块图，详细显示第2图的指令格式化程序(instruction formatter)。

[0024] 图4是一方块图，详细显示第1图的执行管线。

[0025] 图5是一方块图，详细显示第1图的寄存器文件。

[0026] 图6A是一流程图，显示第1图的微处理器的操作步骤。

[0027] 图6B是一流程图，显示第1图的微处理器的操作步骤。

[0028] 图7是本发明一双核心微处理器的方块图。

[0029] 图8是本发明执行x86 ISA与ARM ISA机器语言程序的微处理器另一实施例的方块图。

[0030] 图9是一方块图，详细显示微处理器藉由启动x86ISA及ARM ISA程序来存取第1图的微处理器的特定模型寄存器。

[0031] 图10是一流程图，显示第1图的微处理器执行存取特定模型寄存器的指令。

- [0032] 图11是微码的虚拟代码处理存取特定模型寄存器的指令示意图。
- [0033] 【主要元件符号说明】
- [0034] 微处理器(处理核心)100
- [0035] 指令高速缓存102
- [0036] 硬件指令转译器104
- [0037] 寄存器文件106
- [0038] 存储器子系统108
- [0039] 执行管线112
- [0040] 指令撷取单元与分支预测器114
- [0041] ARM程序计数器(PC)寄存器116
- [0042] x86指令指示符(IP)寄存器118
- [0043] 配置寄存器(configuration register)122
- [0044] ISA指令124
- [0045] 微指令126
- [0046] 结果128
- [0047] 指令模式指示符(instruction mode indicator)132
- [0048] 撷取地址134
- [0049] 环境模式指示符(environment mode indicator)136
- [0050] 指令格式化程序202
- [0051] 简单指令转译器(SIT)204
- [0052] 复杂指令转译器(CIT)206
- [0053] 多路复用器(mux)212
- [0054] x86简单指令转译器222
- [0055] ARM简单指令转译器224
- [0056] 微程序计数器(micro-program counter,micro-PC)232
- [0057] 微码只读存储器234
- [0058] 微序列器(microsequencer)236
- [0059] 指令间接寄存器(instruction indirection register,IIR)235
- [0060] 微转译器(microtranslator)237
- [0061] 格式化ISA指令242
- [0062] 实行微指令(implementing microinstructions)244
- [0063] 实行微指令246
- [0064] 选择输入248
- [0065] 微码地址252
- [0066] 只读存储器地址254
- [0067] ISA指令信息255
- [0068] 预解码器(pre-decoder)302
- [0069] 指令比特组队列(IBQ)304
- [0070] 长度解码器(length decoders)与纹波逻辑(ripple logic)306

- [0071] 多路复用器队列(mux queue, MQ)308
- [0072] 多路复用器312
- [0073] 格式化指令队列(formatted instruction queue, FIQ)314
- [0074] ARM指令集状态322
- [0075] 微指令队列401
- [0076] 寄存器分配表(register allocation table, RAT)402
- [0077] 指令调度器(instruction dispatcher)404
- [0078] 保留站(reservation station)406
- [0079] 指令发出单元(instruction issue unit)408
- [0080] 整数/分支(integer/branch)单元412
- [0081] 介质单元(media unit)414
- [0082] 载入/储存(load/store)单元416
- [0083] 浮点(floating point)单元418
- [0084] 重排缓冲器(reorder buffer, ROB)422
- [0085] 执行单元424
- [0086] ARM特定寄存器502
- [0087] x86特定寄存器504
- [0088] 共享寄存器506

### 具体实施方式

- [0089] 名词定义
- [0090] 指令集,是定义二进位制编码值的集合(即机器语言指令)与微处理器所执行的操作间的对应关系。机器语言程序基本上以二进位制进行编码,不过亦可使用其他进位制的系统,如部分早期IBM计算机的机器语言程序,虽然最终亦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、若是零标记被设定时,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欲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。
- [0091] 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定义于程序员操作手册中。举例来说,2009年3月公布的Intel64与IA-32架构软件开发者手册(Intel 64 and IA-32 Architectures Software Developer's Manual)即定义Intel 64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,此手册列为本案的参考文件。此种处理器架构通常被称为x86架构,本文中则是以x86、x86 ISA、x86 ISA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Errata markup,定义ARM处理器架构的ISA。此参考手册系列为参考文件。此ARM处理器架构的ISA在此亦被称为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、Motorola 68k、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,制造者会透过在指令集中增加新指令、和/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Intel Pentium 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x86 ISA机器语言程序已经开发来利用XMM寄存器以提升效能,虽然现有的x86 ISA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,其他制造商亦设计且制造出可执行x86 ISA机器语言程序的微处理器。例如,超微半导体与威盛电子(VIA Technologies)即在x86 ISA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安全引擎随机数产生器(random number generator)与先进译码引擎(advanced cryptography engine)的技术,前述技术都是采用x86 ISA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,ARM ISA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,而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,即使现有的ARM ISA机器语言程序并非采用这些ARM ISA指令集状态。

[0092] 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,x86 ISA机器语言程序包含x86 ISA指令序列,ARM ISA机器语言程序则包含ARM ISA指令序列。机器语言程序指令系存放于存储器内,且由微处理器撷取并执行。

[0093] 硬件指令转译器,包含多个晶体管的配置,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,执行管线透过对这些微指令的集体执行来“实现”ISA指令。也就是说,执行管线透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处理器只支援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既有的x86 ISA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有的ARM ISA处理器。

[0094] 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。硬件寄存器系保持硬件逻辑的输出信号,并依据微处理器的时脉信号,将此输出信号提供至多层次级序列的下一层。执行管线可以具有复数个多层次级序列,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/储存、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、字元、二进编码十进数(BCD)、与紧缩格式(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。

[0095] 指令高速缓存,系微处理器内的一个随机存取记忆装置,微处理器将ISA机器语言程序的指令(例如x86 ISA与ARM ISA的机器语言指令)放置其中,这些指令系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x86 ISA定义为指令指示符(,IP)而在ARMISA定义为程序计数器(program counter,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指令被高速缓存来供后续撷取之用。当该暂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从高速缓存撷取ISA指令由系统存储器存取。尤其是,此程序系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存取得数据,而非特地运用一载入或储存指令所指定的存储器地址来进行数据撷取。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)的专用数据高速缓存,特地运用一载入/储存地址,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存。此外,可取得指令与数据的混合式高速缓存,系基于指令地址寄存器的数值以及基于载入/储存地址,而非仅仅基于载入/储存地址,亦被涵盖在本说明对指令高速缓存的定义内。在本说明内容中,载入指令系指将数据由存储器读取至微处理器的指令,储存指令系指将数据由微处理器写入存储器的指令。

[0096] 微指令集,系微处理器的执行管线能够执行的指令(微指令)的集合。

[0097] 实施例说明

[0098] 本发明的实施例揭露的微处理器可透过硬件将其相对应的X86 ISA与ARM ISA指令转译为由微处理器的执行管线直接执行的微指令,以达到可执行x86 ISA与ARM ISA机器语言程序的目的。此微指令由微处理器微架构(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,预先储存本机(host)指令至存储器,因此,前揭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。

[0099] 第1图是一方块图显示本发明执行x86 ISA与ARM ISA机器语言程序的微处理器

100的一实施例。此微处理器100具有一指令高速缓存102;一硬件指令转译器104,用以由指令高速缓存102接收x86 ISA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至指令高速缓存102;一ARM ISA定义的程序计数器寄存器116与一x86 ISA定义的指令指示符寄存器118,其依据微指令结果128进行更新,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。

[0100] 由于微处理器100可执行x86 ISA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x86 ISA与ARM ISA的机器语言指令至指令高速缓存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存102,指令高速缓存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系基于ARM程序计数器116与x86指令指示符(IP)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,指令高速缓存102为一混合高速缓存,以存取ISA指令124与数据。值得注意的是,在此混合高速缓存的实施例中,虽然混合高速缓存可基于一载入/储存地址将数据写入高速缓存或由高速缓存读取数据,在微处理器100由混合高速缓存撷取指令集架构的指令124的情况下,混合高速缓存基于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/储存地址。指令高速缓存102可以是一随机存取存储器装置。

[0101] 指令模式指示符132为一状态指示微处理器100当前是否正在撷取、格式化(formatting)/解码、以及将x86 ISA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X86 ISA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ARM ISA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存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。

[0102] 环境模式指示符136系一状态指示微处理器100是使用x86或ARM ISA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、高速缓存控制、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x86 ISA，微处理器100是作为一般的x86 ISA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系作为一般的ARM ISA处理器。在第三种模式中，指令模式指示符132指向x86 ISA，不过环境模式指示符136则是指向ARM ISA，此模式有利于在ARM作业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系指向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作业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，不过可透过熔断配置熔丝与/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(请参照下述第6A图及第6B图)；亦即，在微处理器100正常运作而未由一般重置、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。

[0103] 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执行前揭一个或多个微指令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包含一组晶体管，以一预设方式进行配置来将x86 ISA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并具有布尔逻辑闸以产生实行微指令126(如第2图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(如第2图中复杂指令转译器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，这部分将在第2图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开发之既有x86 ISA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.开发之既有的ISA处理器。前揭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储存于设置两者间的存储器。在第1图的微处理器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，第8图的实施例则是利用一微指令高速缓存以避免微处理器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的实施例在第2图会有更详细的说明。

[0104] 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为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、超纯量发布、与非循序执行。执行管线112在第4图会有更详细的说明。

[0105] 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，此资源集为x86与ARM ISA的资源的超集合(superset)；以及(3)微处理器100响应于微指令126的执行所定义的微例外事件(micro-exception)集，此微例外事件集系x86 ISA与ARM ISA之例外事件的超集合。此微架构不同于x86 ISA与ARM ISA。具体来说，此微指令集在许多面向不同于x86 ISA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x86 ISA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，仍有一些微指令集特定的操作并非x86 ISA和/或ARM ISA指令集所指定。相反地，有一些x86 ISA和/或ARM ISA指令集特定的操作并非微指令集所指定。其次，微指令集的微指令以不同于x86 ISA与ARM ISA指令集的指令的编码方式进行编码。也就是说，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。微指令集与x86或ARM ISA指令集的二进制操作码值对应表通常也是巧合，其间仍不具有一对一的对应关系。第三，微指令集的微指令字段与x86或是ARM ISA指令集的指令字段也不是一对一对。

[0106] 整体而言，微处理器100可执行x86 ISA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x86 ISA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x86 ISA以及ARM ISA不同，本发明亦提出其他实施例将微指令集与其他微架构特定的资源开放给使用者。在这些实施例中，此微架构可有效地作为在x86 ISA与ARM ISA外之一个具有微处理器所能执行的机器语言程序的第三ISA。

[0107] 下表(表1)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的一些字段。

[0108]

比特字段	描述
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

[0109]

	点寄存器、架构特定寄存器、条件标记寄存器、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

[0110] 表1

[0111] 下表(表2)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。

[0112]

指令	描述
ALU-type	例如：加、减、转动、移位、布尔、乘、除、浮点 ALU、介质类型 ALU (如 packed 操作)
load/store	从存储器载入寄存器/从寄存器储存至存储器
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 标记或微架构特定(即非 ISA 可见)条件标记所指定
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	经保证的高速缓存线预撷取指令(亦即，并非暗示，除非出现特定例外事件，否则就会预撷取。)
Grabline	于处理器总线执行零拍读取无效周期(zero beat read-invalidate cycle)，以取得高速缓存线的排他所有权，而不需从系统存储器读取数据(因为整个高速缓存线将会被写入)
load pram	从 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	储存至 PRAM

[0113]

jump condition on/off	在“静态”条件满足时（在相关时间表、使用者保证不会有更旧的、未引退的微指令来改变此“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
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

[0114] 表2

[0115] 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,以及一私人随机存取存储器(PRAM)。此外,此微架构可产生例外事件,亦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,通常是微指令126与相关微指令126的重新执行(replay)。举例来说,这些情形包含:载入错过(load miss)的情况,其执行管线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,重新执行此微指令126;在浮点微指令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此微指令126;一载入微指令126执行后侦测到一个更早的储存(store)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1所列的字段,表2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。

[0116] 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。硬件寄存器系引用(instantiate)x86 ISA定义与ARM ISA定义的通用寄存器系共享寄存器文件106中的一些寄存器。举例来说,在一实施例中,寄存器文件106引用十五个32比特的寄存器,由ARM ISA寄存器R0至R14以及x86 ISA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x86 ISA与ARM ISA的机器语言程序得以快速透过寄存器进行沟通。举例来说,假设在ARM机器语言作业系统执行的ARM机器语言程序能使指令模式132改变为x86 ISA,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x86 ISA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结果。相似地,在x86机器语言作业系统执行的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程序可通过寄存器文件106的共享寄存器提供所需的数据给ARM执行程序,因此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寄存器系一独立引用的ARM程序计数器寄存器116,因此,引用x86 R15D寄存器的第十六个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比特高级单指令多重数据扩展(Advanced SIMD("Neon"))寄存器的32比特区段分享给三十二个32比特ARM VFPv3浮点寄存器。寄存器文件106亦引用标记寄存器(即x86EFLAGS寄存器与ARM条件标记寄存器),以及x86 ISA与ARM ISA所定义的多种控制与状态寄存器,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)与保留在ARM架构的协同处理器(8-15)寄存器。此寄存器文件106亦引用非架构寄存器,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在第5图会有更进一步的说明。

[0117] 存储器次系统108包含一由高速缓存存储器构成的高速缓存存储器阶层架构(在一实施例中包含第1层(level-1)指令高速缓存102、第1层(level-1)数据高速缓存与第2层混合高速缓存)。此存储器次系统108包含多种存储器请求队列,如载入、储存、填入、窥探、合并写入归并缓冲区。存储器次系统亦包含一存储器管理单元。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。存储器次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。虽然在第1图中指令高速缓存102与存储器次系统108系显示为各自独立,不过,在逻辑上,指令高速缓存102亦是存储器次系统108的一部分。存储器次系统108系设定使x86与ARM机器语言程序分享一共同的记忆空间,以使x86与ARM机器语言程序容易透过存储器互相沟通。

[0118] 存储器次系统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,存储器次系统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,存储器次系统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,存储器次系统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x86 ISA,存储器次系统108检查会影响高速缓存策略的X86 ISA控制寄存器(如CR0 CD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SCTLR 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x86 ISA,存储

器次系统108检查会影响存储器管理的X86 ISA控制寄存器(如CR0 PG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SCTLR M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x86 ISA,存储器次系统108检查会影响对准检测的X86ISA控制寄存器(如CR0AM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SCTLR 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x86 ISA,存储器次系统108(以及用于特权指令的硬件指令转译器104)检查特定当前特权级(,CPL)的X86 ISA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,x86 ISA与ARM ISA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。

[0119] 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x86 ISA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机器语言程序之能力失效,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。全域配置寄存器亦可使微处理器100执行x86 ISA机器语言程序的能力失效,亦即让微处理器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在启动时系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异或操作,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修补而回复。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(或如第7图所示处理器的多核心部分的一特定核心100)在重置或如第6A图及第6B图所示在回应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ARM ISA SCTLT与CPACR寄存器。第7图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,全域配置寄存器可透过一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透过x86 WRMSR指令对前揭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透过ARM MCR/MCRR指令对一对应至前揭新的非架构特定模型寄存器的ARM协同处理器寄存器进行读取,而其中部分的控制比特可透过ARM MRC/MRRC指令对应至此新的非架构特定模型寄存器的ARM协同处理器寄存器的写入来进行写入操作。

[0120] 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。

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)指令至非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x86/ARM的控制寄存器，以确认微型(fine-grained)高速缓存控制，此微型高速缓存控制系小于x86 ISA与ARM ISA控制寄存器所能提供者。

[0121] 在一实施例中，微处理器100提供ARM ISA机器语言程序透过实作定义ARM ISA协同处理器寄存器存取x86 ISA特定模型寄存器，这些实作定义ARM ISA协同处理器寄存器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是指定于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以密码保护，亦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，密码是指定于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。

[0122] 微处理器100并包含一个耦接至执行管线112之中断控制器(未图示)。在一实施例中，此中断控制器系一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，x86 INTR对应至ARM IRQ中断事件；x86 NMI对应至ARM IRQ中断事件；x86 INIT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；x86 SMI对应至ARM FIQ中断事件；以及x86 STPCLK、A20、Thermal、PREQ、与Rebranch则不对应至ARM中断事件。ARM机器语言能透过新的实作定义的ARM协同处理器寄存器存取先进可程序化中断控制器(APIC)的功能。在一实施例中，APIC寄存器地址是指定于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系通常用于作业系统执行的特权模式功能，此ARM协同处理器6的地址为:0,7,nn,0；其中nn为15时可存取先进可程序化中断控制器；nn系12-14以存取总线介面单元，藉以在处理器总线上执行8比特、16比特与32比特输入/输出循环。微处理器100并包含一总线介面单元(未图示)，此总线介面单元耦接至存储器次系统108与执行管线112，作为微处理器100与处理器总线的介面。在一实施例中，处理器总线符合一个Intel Pentium微处理器家族的微处理器的总线的规格。ARM机器语言程序可透过新的实作定义的ARM协同处理器寄存器存取总线介面单元的功能可以在处理器总线上产生输入/输出循环，即由输入输出总线传送至输入输出空间的一特定地址，藉以与系统芯片组沟通，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，输入输出地址是指定于ARM R0寄存器。在一实施例中，微处理器100具有电力管理能力，如习知的P-state与C-state管理。ARM机器语言程序可透过新的实作定义ARM协同处理器寄存器执行电力管理。在一实施例中，微处理器100并包含一加密单元(未图示)，此加密单元系位于执行管线112内。在一实施例中，此加密单元实质上系类似于具有Padlock安全科技功能的VIA微处理器的加密单元。ARM机器语言程序能够透过新的实作定义的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。

[0123] 在微处理器100执行x86 ISA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,第8图的实施例使用微指令高速缓存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是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。

[0124] 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依据之前的高速缓存历史数据,来分析由指令高速缓存102所取得高速缓存线是否存在x86与ARM分支指令以及其目标地址。在一实施例中,高速缓存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、分支指令在高速缓存线的起始比特组、以及一个显示是否横跨多个高速缓存线的指示符。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 BRANCH PREDICTION TO EFFICIENTLY EXECUTE CONDITIONAL NON-BRANCH INSTRUCTIONS”,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。

[0125] 本发明也考虑多种不同的实施例以实现x86 ISA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、高级单指令多重数据(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程序可透过ARM MCR/MRC指令至新的实作定义协同处理器寄存器取得x86除错功能)、效能侦测计数器(ARM程序可透过新的实作定义协同处理器寄存器取得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)并且只支援Little-endian数据格式。在另一实施例中,微处理器100并不实现x86 SSE 4.2的功能。

[0126] 本发明考虑多个实施例的微处理器100的改进,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 Nano<sup>TM</sup>进行改良。此Nano微处理器能够执行x86 ISA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x86 ISA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(亦包含微码在内)。此外,微指令集可加入新的微指令以支援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次系统亦可经改善使其能支援ARM ISA,亦包含特定寄存器的共享。分支预测单元可透过改善使其在x86分支预测外,亦能适用于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与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标记之产生与使用方式、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、与高速缓存策略等。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，x86 ISA与ARM ISA定义的部分特征可能无法为前揭对Nano微处理器进行改良的实施例所支援，这些特征如x86 SSE4.2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，前揭透过对于Nano处理器的改良以支援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机器语言程序的单集成电路产品的实施例，此单集成电路产品系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x86 ISA与ARM ISA机器语言程序微处理器。此微处理器可透过在单一微处理器100(或是第7图的核心100)上的X86与ARM指令模式132间的动态切换以取得同时执行x86 ISA与ARM ISA机器语言程序的能力，亦可透过将多核心微处理100(对应于第7图所示)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，即透过在多核心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x86 ISA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，如系统芯片与/或嵌入式应用。因此，ARM ISA并不具有一特定的标准处理器总线，作为ARM核心与系统的其他部分(如芯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型式处理器总线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计算机系统环境下支援ARM ISA机器语言程序的执行。

[0127] 请参照第2图，图中以方块图详细显示第1图的硬件指令转译器104。此硬件指令转译器104包含硬件，更具体来说，就是晶体管的集合。硬件指令转译器104包含一指令格式化程序202，由第1图的指令高速缓存102接收指令模式指示符132以及x86 ISA与ARM ISA指令比特组124的区块，并输出格式化的x86 ISA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(亦称为一微码单元)，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；以及一多路复用器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至第1图的执行管线112。指令格式化程序202在第3图会有更详细的说明。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(micro-PC)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。

[0128] 多路复用器212系受到一选择输入248所控制。一般的时候，多路复用器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而将控制权注意、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让多路复用器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(请参照第4图)遭遇到一个微指令126具有一特定比特指出其为实现复杂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使多路复用器212回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(请参照第4图)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使多路复用器212选择来自复杂指令转译器206的微指令246。前揭需引退微指令126的情形如:微指令126已经导致一例外条件产生。

[0129] 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x86 ISA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x86 ISA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。多路复用器212系受到控制,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的多个微码程序中之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x86 ISA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x86 ISA与ARM ISA机器语言程序来说是较常执行的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IMCR、MRC、MSR、MRS、SRS、与RFE指令。前揭列出的指令并非限定本案发明,而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。

[0130] 当指令模式指示符132指示为x86,x86简单指令转译器222对于x86 ISA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是一可由习知合成工具合成的布尔逻辑方块。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是独立的布尔逻辑方块;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是位于同一个布尔逻辑方块。在一实施例中,简单指令转译器204在单一时钟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之格式化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之格式化ISA指令242；第三个转译器能后转译需要不多于一个实行微指令126之格式化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钟周期输出多个微指令244以实现一个ISA指令242。

[0131] 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x86 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x86 ISA，简单指令转译器204随即检测是否其所遭遇的每个x86 ISA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x86 ISA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随即检测是否各个格式化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亦执行类似的功能。

[0132] 复杂指令转译器206输出一系列实行微指令246至多路复用器212。微码只读存储器234储存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的尺寸，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系制造于微处理器100的半导体芯片内。

[0133] 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。储存于指令间接寄存器(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指令所指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系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，亦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x86 ISA与ARM ISA指令124内提供的寄存器信息，透过ISA指令信息255的使用，转译至寄存器文件106内之适当的寄存器。ISA指令信息255包含一移位字段、一立即字段、一常数字段、各个源操作数与微指令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它比特。

[0134] 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系与只读存储器指令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系大约有40比特宽,并且各个微指令246系大约有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包含多个布尔逻辑以产生实行微指令246。

[0135] 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不需要储存间接指令寄存器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系存储器格式,简单指令转译器204会产生微指令244的逻辑编程,其包含将来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用以将结果由暂时寄存器106储存至存储器;然而,如果复杂ISA指令124是寄存器格式,此逻辑编程会将来源操作数由ISA指令124所指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的许多面向系类似于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x86 ISA指令124外,亦经改良以转译ARM ISA指令124。

[0136] 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,亦即,微程序计数器232并不持有ISA指令124的地址,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。

[0137] 请参照第3图,图中以方块图详述第2图的指令格式化器202。指令格式化器202由第1图的指令高速缓存102接收x86 ISA与ARM ISA指令比特组124区块。凭借x86 ISA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x86 ISA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,因此确认高速缓存区块内的X86 ISA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,指令格式化器202由指令比特组124串(stream)撷取不同的x86 ISA与ARM ISA指令,此指令比特组124串由指令高速缓存102接收的区块所构成。也就是说,指令格式化器202格式化x86 ISA与ARM ISA指令比特组串,因而大幅简化第2图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。

[0138] 指令格式化器202包含一预解码器(pre-decoder)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。

[0139] 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,亦即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亦接收指令模式指示符132与ARM ISA指令集状态322。在一实施例中,ARM ISA指令集状态322包含ARM ISA CPSR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。一多路复用器队列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。

[0140] 控制逻辑(未图示)检验多路复用器队列(MQ)308的底部项目的内容,并控制多路复用器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314在格式化ISA指令242与提供至第2图的简单指令转译器204之相关信息间作为缓冲。在一实施例中,多路复用器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。

[0141] 在一实施例中,指令格式化器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号申请案共同揭露的XIBQ、指令格式化器、与FIQ,这些申请案在此列为参考数据。然而,前述专利申请案所揭示的XIBQ、指令格式化器、与FIQ透过修改,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输出再以连线或(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(FIQ)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在单一时钟周期内,提供简单指令转译器204至多三个格式化ISA指令242。

[0142] 请参照第4图,图中以方块图详细显示第1图的执行管线112,此执行管线112系耦接至硬件指令转译器104以直接接收来自第2图的硬件指令转译器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;一指令发出单元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、指令发出单元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。

[0143] 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能够在一给定的时钟周期内,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储存至宽度为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能够在每个时钟周期处理最多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的特定时钟周期为何,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(请参照第4图),因为它可比不可压缩宽度M或宽度M的指令队列提供较高的指令储存效能。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,在之后的时钟周期内会重复转译一个或多个已经在之前的时钟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钟周期接收多于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。

[0144] 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来增加微指令平行处理的能力,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就会对应于微处理器100的X86 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;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x86 ISA与ARM ISA所共享。寄存器分配表402亦在重排缓冲器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x86 ISA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X86 ISA与ARM ISA指令124而言是以非循序方式进行的。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储存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、以及用以储存微指令126的结果的储存空间。

[0145] 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(dispatch)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。

[0146] 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,可以非循序以及以超纯量方式来执行。

[0147] 在一实施例中,执行单元424包含整数/分支单元412、介质单元414、载入/储存单元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管理标记的产生,其管理会依据指令模式指示符132指示为x86 ISA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x86 ISA或是ARM ISA,对x86EFLAGS寄存器或是程序状态寄存器(PSR)内的ARM条件码标记进行更新。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亦藉此来决定使用x86或是ARM语意执行前述动作。一旦微指令126变成微处理器100中最旧的已完成微指令126(亦即,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可在一时钟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系一高效能、通用执行引擎,其可执行支援x86 ISA与ARM ISA指令124的微处理器100微架构的微指令126。

[0148] 请参照第5图,图中以方块图详述第1图的寄存器文件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端口与写入端口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端口的邻近功能方块。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的硬件而仅储存微码234会使用到的数值的寄存器(如部分x86 MSR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,x86 ISA与ARM ISA程序者无法见到此私有随机存取存储器,亦即此存储器并不在ISA系统存储器地址空间内。

[0149] 总的来说,如第5图所示,寄存器文件106在逻辑上系区分为三种,亦即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x86 ISA XMM0至XMM15寄存器以及ARM ISA高级单指令多重数据扩展(Neon)寄存器所共享,这些寄存器的部分系重迭于三十二个32比特ARM VFPv3浮点寄存器。如前文第1图所述,通用寄存器的共享意指由x86 ISA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x86 ISA与ARM ISA程序透过寄存器互相沟通。此外,如前述,x86 ISA与ARM ISA的架构控制寄存器的特定比特亦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透过实作定义协同处理器寄存器存取,因而是由x86 ISA与ARM ISA所共享。此共享寄存器506可包含非架构寄存器,例如条件标记的非架构同等物,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ARM ISA所共享,因而会产生实行微指令126来存取正确的寄存器。

[0150] 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x86 ISA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、协同处理器寄存器、多种例外事件模式的备用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(亦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。

[0151] 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x86 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x86 ISA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寄存器系指定在ARM R0寄存器中,存取电力管理(如P状态与C状态的转换),存取处理器总线功能(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x86 ISA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寄存器系指定在EAX寄存器中。

[0152] 请参照第6A与6B图,图中显示一流程说明第1图的微处理器100的操作程序。此流程始于步骤602。

[0153] 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,此微处理器总线系一x86型式的处理器总线,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x86 ISA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x86 ISA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x86 ISA。在另一实施例中,不同于前揭动作(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x86 ISA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x86 ISA。接下来进入步骤604。

[0154] 在步骤604,重置微码确认微处理器100系配置为一个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开机模式系硬式编码于微码,不过可透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。

[0155] 在步骤614中,重置微码使微处理器100开始由x86 ISA指定的重置向量地址撷取x86指令124。接下来进入步骤616。

[0156] 在步骤616中,x86系统软件(如BIOS)系配置微处理器100来使用如x86 ISA RDMSR与WRMSR指令124。接下来进入步骤618。

[0157] 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系统固件执行微处理器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作业系统的开机程序,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指令系一x86 WRMSR指令至一新的非架构特定模型寄存器。接下来进入步骤622。

[0158] 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特定的状态,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ARMISA。接下来进入步骤624。

[0159] 在步骤624中,重置微码使微处理器100开始在x86 ISA 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。

[0160] 在步骤638中,重置微码将共享ISA状态506,如GPRs,初始化至ARM ISA指定的预设数值。接下来进入步骤642。

[0161] 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。

[0162] 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ARM ISA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,此预设选则可透过熔断熔丝以及/或是微码修补来修改。接下来进入步骤646。

[0163] 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。

[0164] 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系统固件执行微处理器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作业系统的开机程序,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指令系一ARM MRC/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。

[0165] 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x86 ISA所指定的预设数值。不过,重置微码并不修改微处理器100的非ISA特定状态,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x86 ISA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x86 ISA。接下来进入步骤654。

[0166] 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。

[0167] 请参照第7图，图中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包含第1图的微处理器100所具有的元件，藉此，各个核心均可执行x86 ISA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x86 ISA程序、两个核心100都执行ARM ISA程序、或是一个核心100执行x86 ISA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。如第6A图及第6B图之说明内容所述，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可以独立地透过重置改变为x86或是ARM处理器。虽然第7图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x86 ISA与ARM ISA机器语言程序。

[0168] 请参照第8图，图中以一方块图说明本发明另一实施例的可执行x86 ISA与ARM ISA机器语言程序的微处理器100。第8图的微处理器100类似于第1图的微处理器100，其中的元件编号亦相似。然而，第8图的微处理器100亦包含一微指令高速缓存892，此微指令高速缓存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。微指令高速缓存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命中微指令高速缓存892，执行管线112内的多路复用器(未图示)就选择来自微指令高速缓存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，多路复用器则是选择直接由硬件指令转译器104提供的微指令126。微指令高速缓存的操作，通常亦称为追踪高速缓存，系微处理器设计的技术领域所习知的技术。微指令高速缓存892所带来的优点在于，由微指令高速缓存892撷取微指令126所需的时间通常会少于由指令高速缓存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。在第8图的实施例中，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，亦即当实行微指令126已经存在于微指令高速缓存892，就不需要执行硬件转译。

[0169] 在此所述的微处理器的实施例的优点在于，其透过内建的硬件指令转译器来将x86 ISA与ARM ISA指令转译为微指令集的微指令，而能执行x86 ISA与ARM ISA机器语言程序，此微指令集为不同于x86 ISA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，透过协同利用大量与ISA无关的执行管线来执行由x86 ISA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(亦即一个能够执行x86 ISA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，亦在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x86 ISA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。

[0170] 控制与状态寄存器对应

[0171] 如上所述，第1图的配置寄存器122是以不同方式控制微处理器100的操作。本文所

述的配置寄存器122亦为控制及状态寄存器122。典型但不完全地,控制及状态寄存器122由系统固件(如BIOS)及系统软件(如操作系统)所读写,藉以配置所需要的微处理器100。

[0172] x86 ISA提供一通用机制来存取控制及状态寄存器,在x86 ISA中,许多控制及状态寄存器被称为特定模型寄存器,其可分别经由读取特定模型寄存器(Read MSR; RDMSR)以及写入特定模型寄存器(Write MSR; WRMSR)指令而读写。具体来说,RDMSR指令将64比特特定模型寄存器的内容读取到EDX:EAX寄存器,且64比特特定模型寄存器的地址是在ECX寄存器内所指定;相反地,WRMSR指令将EDX:EAX寄存器的内容写入64比特特定模型寄存器,且64比特特定模型寄存器的地址是在ECX寄存器内所指定。特定模型寄存器地址是由微处理器制造商所定义。

[0173] 有利的是,本发明实施例提供一种让ARM ISA程序存取第1图微处理器100的X86特定模型寄存器122的机制。具体来说,微处理器100采用ARM ISA协同处理器寄存器机制来存取x86特定模型寄存器122。

[0174] 从协同处理器移至ARM寄存器(Move to ARM Register from Coprocessor; MRC)指令以及从协同处理器移至两个ARM寄存器(Move to two ARM Registers from Coprocessor; MRRC)指令中,其系分别将协同处理器(coprocessor; CP)的内容移至一或两个32比特通用寄存器。从ARM寄存器移至协同处理器(Move to Coprocessor from ARM Register; MCR)指令,以及从两个ARM寄存器移至协同处理器(Move to Coprocessor from two ARM Registers; MCRR)指令,其系分别将一或两32个比特通用寄存器的内容移至协同处理器(coprocessor; CP)。协同处理器是由一协同处理器编号所识别。有利的是,当一MCR/MCRR/MRC/MRRC指令124指定一预设执行定义的(implementation-defined)ARM ISA协同处理器寄存器空间的协同处理器寄存器时,微处理器100即知道指令124指示它来存取(如读写)特定模型寄存器122。在一实施例中,特定模型寄存器122地址系在预设的ARM ISA通用寄存器中所指定。如上所述以及本文所揭露的微处理器100的特定模型寄存器122由x86 ISA及ARM ISA所分享的方式,在后面会有更详细的描述。

[0175] 包含藉由特定模型寄存器122控制微处理器100操作方式的实施例,包含但不限于:存储器排序缓冲器控制及状态、分页错误编码、清除分页目录高速缓存存储器及后备缓冲区入口、控制微处理器100的高速缓存存储器层内不同的高速缓存存储器,例如使部分或所有高速缓存失效、从部分或所有高速缓存移除电源、以及使高速缓存标签无效;微码修补机制控制;除错控制、处理器总线控制;硬件数据及指令预取控制;电源管理控制,例如休眠及唤醒控制、P状态及C状态转换,以及使对各种功能方块的时脉或电源失效;合并指令的控制及状态、错误更正编码存储器错误状态;总线校验错误状态;热管理控制及状态;服务处理器控制及状态;核心间通讯;芯片间通讯;与微处理器100的熔丝相关功能;稳压器模组电压识别符号(voltage identifier; VID)控制;锁相回路控制;高速缓存窥探控制、合并写入缓冲器控制及状态;超频功能控制;中断控制器控制及状态;温度感应器控制及状态;使多种功能启动或失效,例如加密/解密、特定模型寄存器保护密码、对L2高速缓存及处理器总线提出并行要求(making parallel request);个别分支预测功能、指令合并、微指令超时、执行计数器、储存转发(store forwarding),以及预测性查表(speculative tablewalks);载入队列大小;高速缓存存储器大小;控制如何存取至已处理的未定义特定模型存储器;以及多核心配置。

[0176] 这些方式是通用于微处理器100的操作,例如它们对x86 ISA及ARM ISA来说是非特定的。也就是说,尽管是指令模式指示符132所指示的特别ISA,通用的微处理器的操作方式还是会影响指令的处理。举例来说,控制寄存器内的比特将确定高速缓存存储器的配置,像是取消选择在高速缓存存储器内比特单元(bitcells)的损坏行,并且用比特单元的冗余行来取代它。对所有ISA来说,这样的高速缓存存储器配置会影响微处理器100的操作,也因此微处理器的操作方式是通用的。其他实施例如通用的微处理器100的操作方式是微处理器100的锁相回路工作周期及/或时脉比、以及是设定电压识别符号接脚,而设定电压识别符号接脚是对微处理器100控制电压源。一般来说,ARMISA指令124所存取的是通用特定模型寄存器122藉由,而不是x86指定的特定模型寄存器122。

[0177] 如上所述,在一实施例中,微处理器100是商用微处理器的增强型,微处理器100可执行x86 ISA程序,且更特别的是,其可执行x86 ISA RDMSR/WRMSR指令来存取特定模型寄存器122。商用微处理器是根据本文实施例所提供特定模型寄存器122存取至ARM ISA程序而获得增强。在一实施例中,第2图的复杂指令转译器206使用经由微码只读存储器234所输出的只读存储器指令247,藉以产生微指令126来执行RDMSR/WRMSR指令。这样的实施例的优点在于增加ARM ISA MRC/MRRC/MCR/MCRR指令来存取特定模型寄存器通用控制及状态寄存器的功能时,只需要在现有提供x86 ISA RDMSR/WRMSR指令存取上述特定模型寄存器通用控制及状态寄存器功能的微码234增加相对较小数量的微码234即可。

[0178] 请参阅第9图,其是一方块图,用以详细描述微处理器100藉由启动x86 ISA及ARM ISA程序来存取第1图的微处理器100的特定模型寄存器。复数个64比特特定模型寄存器122已揭露于图中,每一特定模型寄存器122具有不同的特定模型寄存器地址(例如0x1110, 0x1234, 0x2220, 0x3330, 0x4440)。如上所述,特定模型寄存器122可视为第1图寄存器文件106中的一部份。

[0179] 第9图是显示x86 ISA程序,具体来说是RDMSR/WRMSR指令124,当指令模式指示符132指示x86 ISA时,x86 ISA程序存取特定模型寄存器122中的一个寄存器。在第9图的实施例中,作为存取的特定模型寄存器122系具有地址0x1234。因此,如x86 ISA所指定的,特定模型寄存器122地址数值已藉由在RDMSR/WRMSR指令124之前的x86程序,而被储存在x86 ECX寄存器106中。此外,在RDMSR指令124的情况下,如x86 ISA所指定的,微处理器100从在地址0x1234的特定模型寄存器122读取64比特数据数值,然后复制到x86 EDX:EAX寄存器106。而在WRMSR指令124的情况下,如x86 ISA所指定的,微处理器100将x86 EDX:EAX寄存器106内之64比特数据数值,复制到在地址0x1234的特定模型寄存器122。

[0180] 第9图亦显示ARM ISA程序,具体来说是MRRC/MCRR指令124,当指令模式指示符132指示ARM ISA时,x86 ISA程序存取特定模型寄存器122中地址为0x1234的一个寄存器。特定模型寄存器122地址数值0x1234已藉由在MRRC/MCRR指令124之前的ARM程序,而被储存在ARM R1寄存器106。此外,在MRRC指令124的情况下,微处理器100从地址0x1234的特定模型寄存器122读取64比特数据数值,然后复制到ARM R2:R0寄存器106;而在MCRR指令124的情况下,微处理器100将ARM R2:R0寄存器106内的64比特数据数值,复制到在地址0x1234的特定模型寄存器122。MRRC/MCRR指令124指定一预设的ARM协同处理器编号。在一实施例中,预设的ARM协同处理器编号是4。MRRC/MCRR指令124亦指定一预设ARM寄存器编号。在一实施例中,预设的ARM寄存器编号是(0, 7, 15, 0),其系分别表示CRn、opc1、CRm以及opc2字段

(field)的数值。在MRC/MCR指令124的情况下、以及MRRC/MCRR指令124的情况下，表示opc1字段为7且CRm字段为15。在一实施例中，若ARM ISA指令124是MRC或MCR指令，那么只有比所指定的64比特特定模型寄存器的低32比特(lower 32 bits)才被读写。

[0181] 在一实施例中，如上所述，由x86 ISA及ARM ISA所定义的通用寄存器，分享寄存器文件106的实体寄存器(physical register)的实例。在一实施例中，对应关系如下表所示。

[0182]

EAX	R0
ECX	R1
EDX	R2
EBX	R3
ESP	R4
EBP	R5
ESI	R6

[0183]

EDI	R7
-----	----

[0184] 上表所示的对应关系可观察到ARM R1寄存器对应到x86 ECX寄存器，且ARM R2:R0寄存器对应到x86 EDX:EAX寄存器，其优点在于可将微码234简单化。

[0185] 虽然可经由上述所揭露的实施例了解到R1寄存器是预设的ARM寄存器，且是用来指定特定模型寄存器122地址，但其他藉由其他方式来指定特定模型寄存器122地址的实施例也被考虑在本发明中，例如，但不限于此，另一通用寄存器是预设寄存器或在MRRC/MCRR指令124本身中来指定寄存器。同样地，虽然上述实施例揭露R2:R0寄存器是预设的ARM寄存器，且是用来处理数据，但其他可设想到的实施例中，用来处理数据的寄存器是藉由其他方式所指定的实施例也被考虑在本发明中，例如，但不限于此，其他通用寄存器是预设寄存器，或是在MRRC/MCRR指令124本身指定寄存器。此外，虽然上述实施例揭露协同处理器4的寄存器(0,7,15,0)是预设ARM协同处理器寄存器，且是用来存取特定模型寄存器122，但其他可设想到的实施例中，是用另一预设ARM协同处理器寄存器也被本发明考虑。最后，虽然上述实施例揭露x86ISA或ARM ISA的通用寄存器分享实体寄存器文件，但它们彼此不分享、或是以不同于前述方式做对应的其它实施例也被考虑在本发明中。

[0186] 请参阅第10图，第10图是一流程图，显示第1图的微处理器100执行存取特定模型寄存器122的指令124。

[0187] 在步骤1002中，微处理器100撷取一ISA指令124，并且将其提供至第1图的硬件指令转译器104，接着执行步骤1004。

[0188] 在步骤1004中，若指令模式指示符132指示x86 ISA，则执行步骤1012，而若指令模式指示符132指示ARM ISA，则执行步骤1022。

[0189] 在步骤1012中，第2图的X86简单指令转译器222遭遇x86 ISARDMSR/WRMSR指令124，并进入陷阱而到第2图的复杂指令转译器206。具体来说，简单指令转译器204提供微码地址252给微程序计数器232，该微程序计数器232进入在微码只读存储器234中用于处理RDMSR/WRMSR指令124的例行程序的入口点。接着执行步骤1014。

[0190] 在步骤1014中复杂指令转译器206利用RDMSR/WRMSR指令124的例行程序的微码只

读存储器指令247,用以产生微指令126来执行RDMSR/WRMSR指令124。第11图是显示处理RDMSR/WRMSR指令124的微码234例行程序的虚拟代码。如第11图所示,TEMP1及TEMP2系指被用来储存暂时数值的暂时(例如非架构)64比特寄存器。接着执行步骤1016。

[0191] 在步骤1016中,执行管线112执行在步骤1014所产生的微指令126,藉以执行RDMSR/WRMSR指令124。也就是说,在RDMSR指令124的情况下,微指令126将特定模型寄存器122内的数值复制到EDX:EAX寄存器,而特定模型寄存器122的地址是由ECX寄存器所指定;相反地,在WRMSR指令124的情况下,微指令126将EDX:EAX寄存器内的数值复制到特定模型寄存器122,而特定模型寄存器122的地址是由ECX寄存器所指定。在执行步骤1016后结束。

[0192] 在步骤1022中,第2图的ARM简单指令转译器224遭遇ARM ISA MRRC/MCRR指令124,并进入陷阱而到复杂指令转译器206。具体来说,简单指令转译器204提供微码地址252给微程序计数器232,此微程序计数器232是在微码只读存储器234中用以处理MRRC/MCRR指令124的例行程序的入口点。接着执行步骤1024。

[0193] 在步骤1024中,复杂指令转译器206利用处理RDMSR/WRMSR指令124的例行程序的微码只读存储器指令247,用以产生微指令126来执行MRRC/MCRR指令124。第11图亦显示处理RDMSR/WRMSR指令124的微码234例行程序的虚拟代码。如第11图所示,共同子程序(RDMSR\_COMMON)可被用以处理RDMSR指令124的微码程序、以及用来处理WRMSR指令124的微码程序两者所呼叫。同样地,共同子程序(WRMSR\_COMMON)可被用来处理MCRR指令124的微码例行程序、以及被用来处理WRMSR指令124的微码例行程序两者所呼叫。这样做是有其优点的,因为大量的操作可藉由共同子程序来执行,使得只需要相对较少的微码234即可支援ARM MRRC/MCRR指令124。此外,处理MRRC/MCRR指令124的例行程序用以确定预设的协同处理器编号已被指定(例如协同处理器4),以及预设的协同处理器寄存器地址已被指定(如(0,7,15,0)),否则,微码将分支到处理存取至其他寄存器的例行程序,如非特定模型寄存器、协同处理器寄存器。

[0194] 在一实施例中,程序亦判断微处理器100不在ARM ISA使用者模式;否则,微码将产生一例外。此外,例行程序判断启动ARM ISA程序来存取特定模型寄存器122的功能已启动;否则,微码把MRRC/MCRR指令124视为无执行任何操作。接着执行步骤1026。

[0195] 在步骤1026中,执行管线112执行在步骤1014产生的微指令126,藉以执行MRRC/MCRR指令124。也就是说,在MRRC指令124的情况下,微指令126将特定模型寄存器122内的数值复制到R2:R0寄存器,而特定模型寄存器122的地址是在R1寄存器内被指定,相反地,在MCRR指令124的情况下,微指令126将R2:R0寄存器内的数值复制到特定模型寄存器122,而特定模型寄存器122的地址是在R1寄存器内被指定。在执行步骤1026后结束。

[0196] 虽然已于第9图至第11图揭露MRRC/MCRR指令124相关的实施例,如上所述的实施例更提供ARM MCR/MRC指令124的功能来存取特定模型寄存器122低32比特。进一步来说,虽然实施例已揭露特定模型寄存器122是经由MRRC/MCRR/MCR/MRC指令124而被存取,但其他的实施例,例如运用ARM ISA LDC/STC指令124来存取特定模型寄存器122也被考虑在本发明中。也就是说,数据是从存储器被读取,或储存在存储器,而不是从ARM ISA通用寄存器(被读取或存储其中)。

[0197] 从上述可了解到本发明实施例是对ARM ISA程序提供一有效的机制来存取微处理器100的特定模型寄存器122。其他可想到的实施例中,每一特定模型寄存器122具有自己的

协同处理器寄存器编号,且协同处理器寄存器编号是在ARM ISA协同处理器寄存器空间的MRRC/MCRR opc1及CRm字段内被指定。本实施例的缺点在于可能会在ARM ISA协同处理器寄存器空间中,消耗相对较多数量的寄存器。此外,还可能需要对现有微码明显扩编,这样将会消耗微码只读存储器234内的有效空间。在一这样的实施例中,ECX数值(或至少较低的比特)被拆散成片段(pieces),并且被分布至opc1及CRm字段位。微码将片段组合成原始的ECX数值。

[0198] 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。可以藉由一般程序语言(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、磁盘或光盘(如CD-ROM及DVD-ROM等)、网络、有线电缆、无线网络或其他通讯介质。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,并且转换为集成电路产品的硬件,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揭露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。

[0199] 惟以上所述者,仅为本发明的较佳实施例而已,当不能以此限定本发明实施的范围,即大凡依本发明权利要求及发明说明内容所作的简单的等效变化与修饰,皆仍属本发明专利涵盖的范围内。另外本发明的任一实施例或权利要求不须达成本发明所揭露的全部目的或优点或特点。此外,摘要部分和标题仅是用来辅助专利文件搜寻之用,并非用来限制本发明的权利范围。

#### [0200] 【相关申请案的参考文献】

[0201] 本申请案是同在申请中美国专利正式申请案的部分连续案,该些案件整体皆纳入本案参考:

#### [0202]

案号	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

[0203] 本申请案引用于以下美国临时专利申请案作优先权,每一申请案整体皆纳入本案参考:

#### [0204]

案号	申请日

[0205]

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

[0206] 美国正式专利申请案

[0207]

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

[0208] 引用下列美国临时申请案的优先权:

[0209]

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

[0210] 以下三个本美国正式申请案

[0211]

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

[0212] 皆是以下美国正式申请式的延续案:

[0213]

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

[0214] 并引用下列美国临时申请案的优先权:

[0215]

61/473,062(CNTR.2547)	04/07/2011
-----------------------	------------

[0216]

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

[0217] 本申请案是以下美国正式专利申请案的相关案:

[0218]

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

13/413,904(CNTR.2568)

03/06/2012

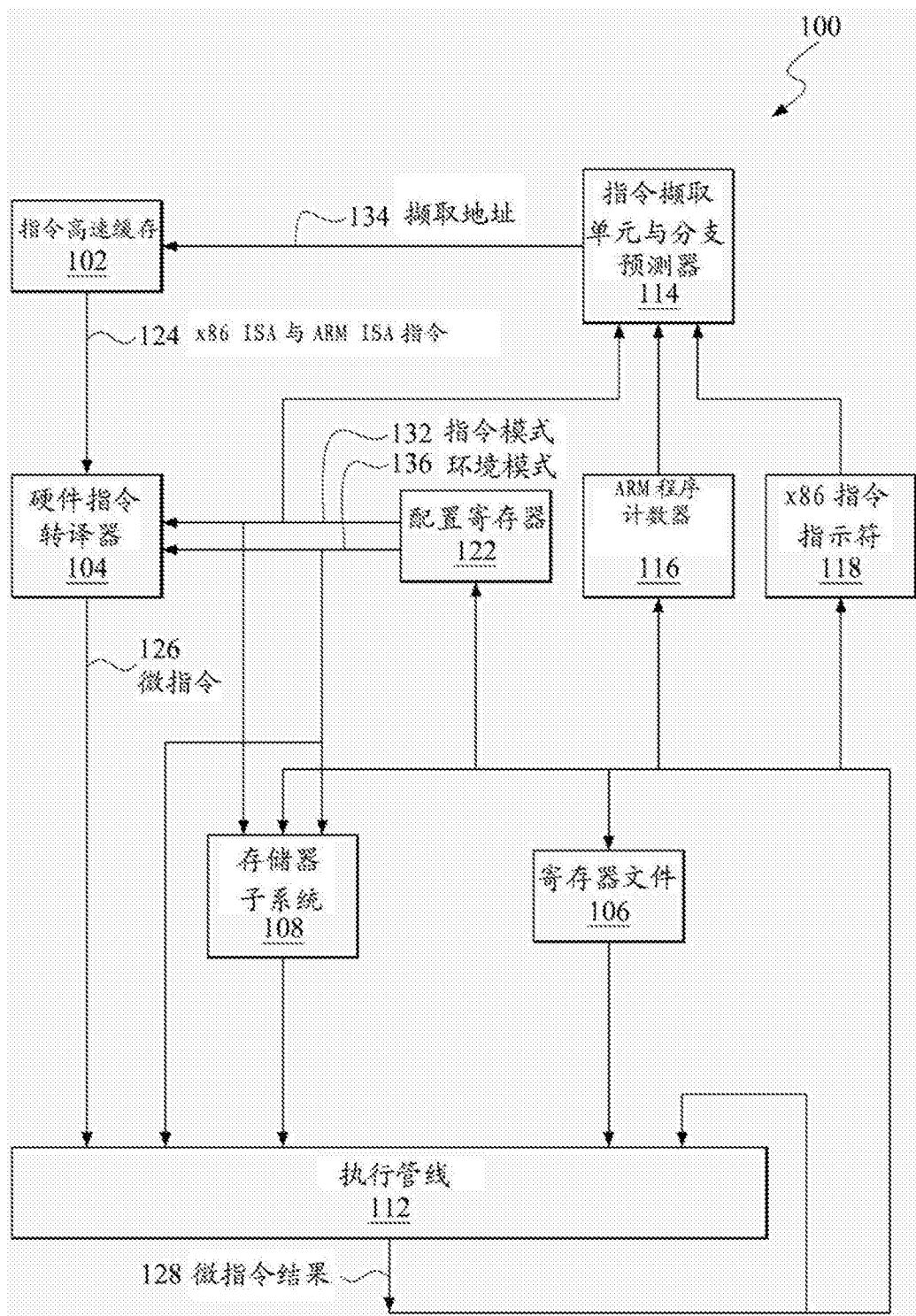


图1

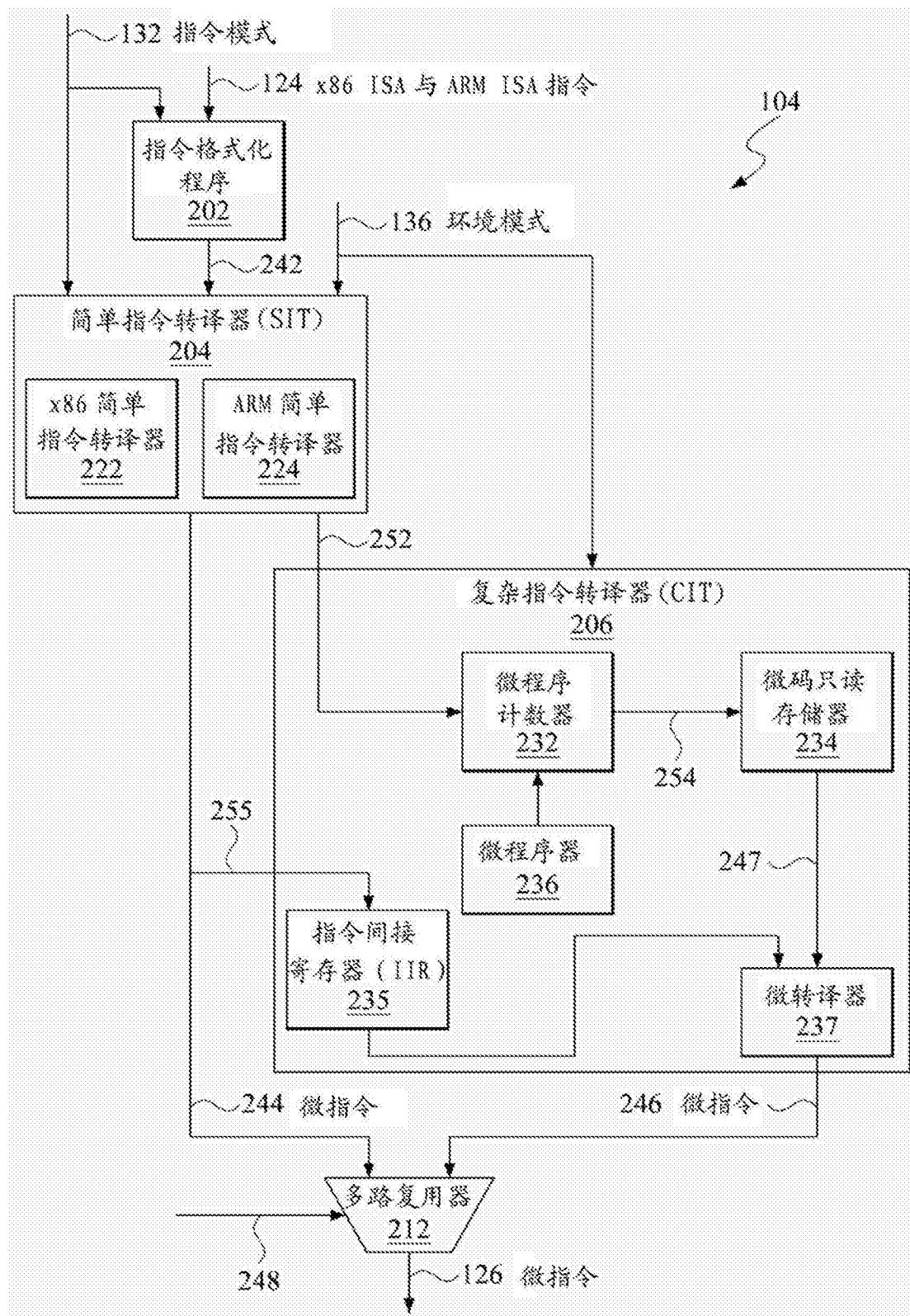


图2

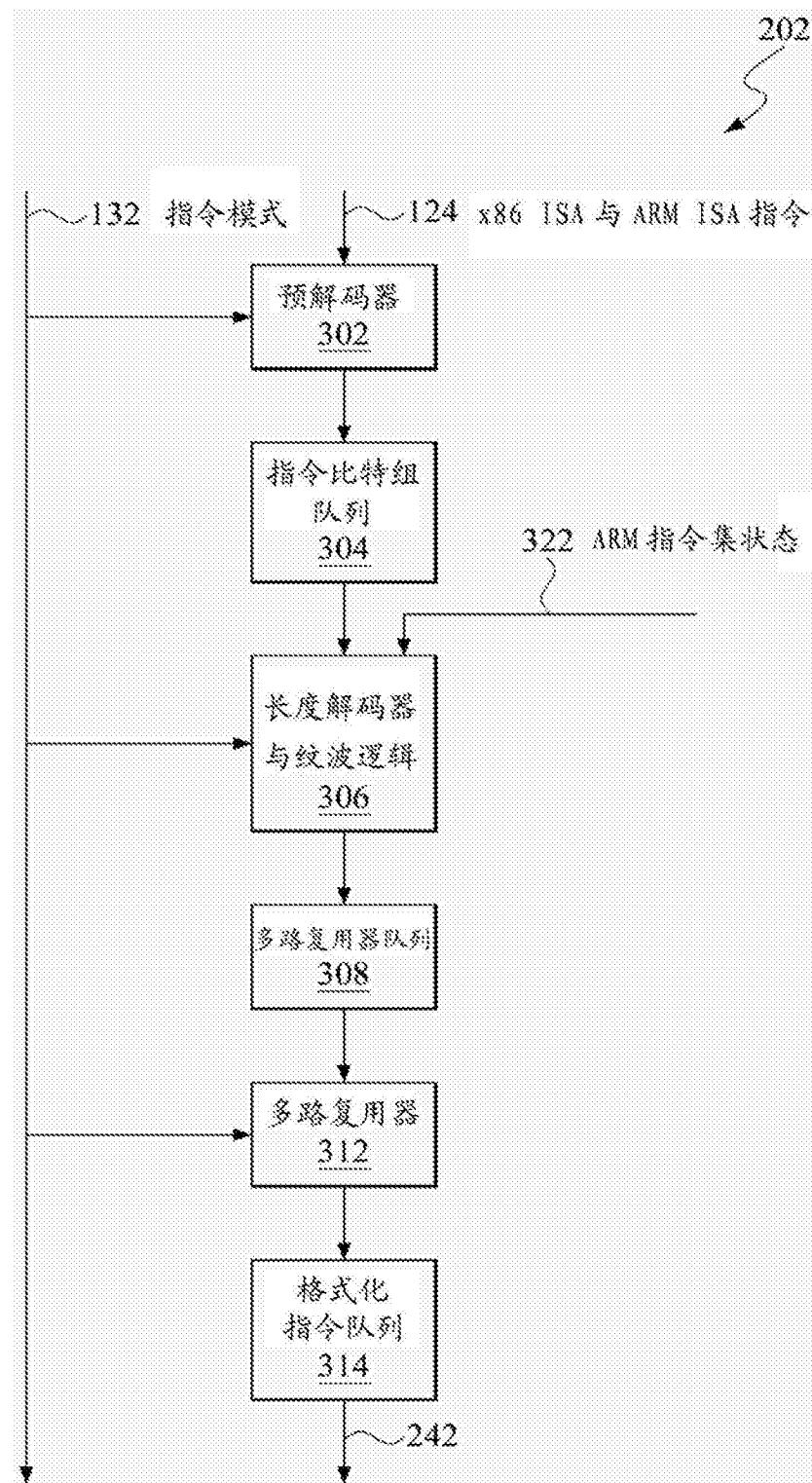


图3

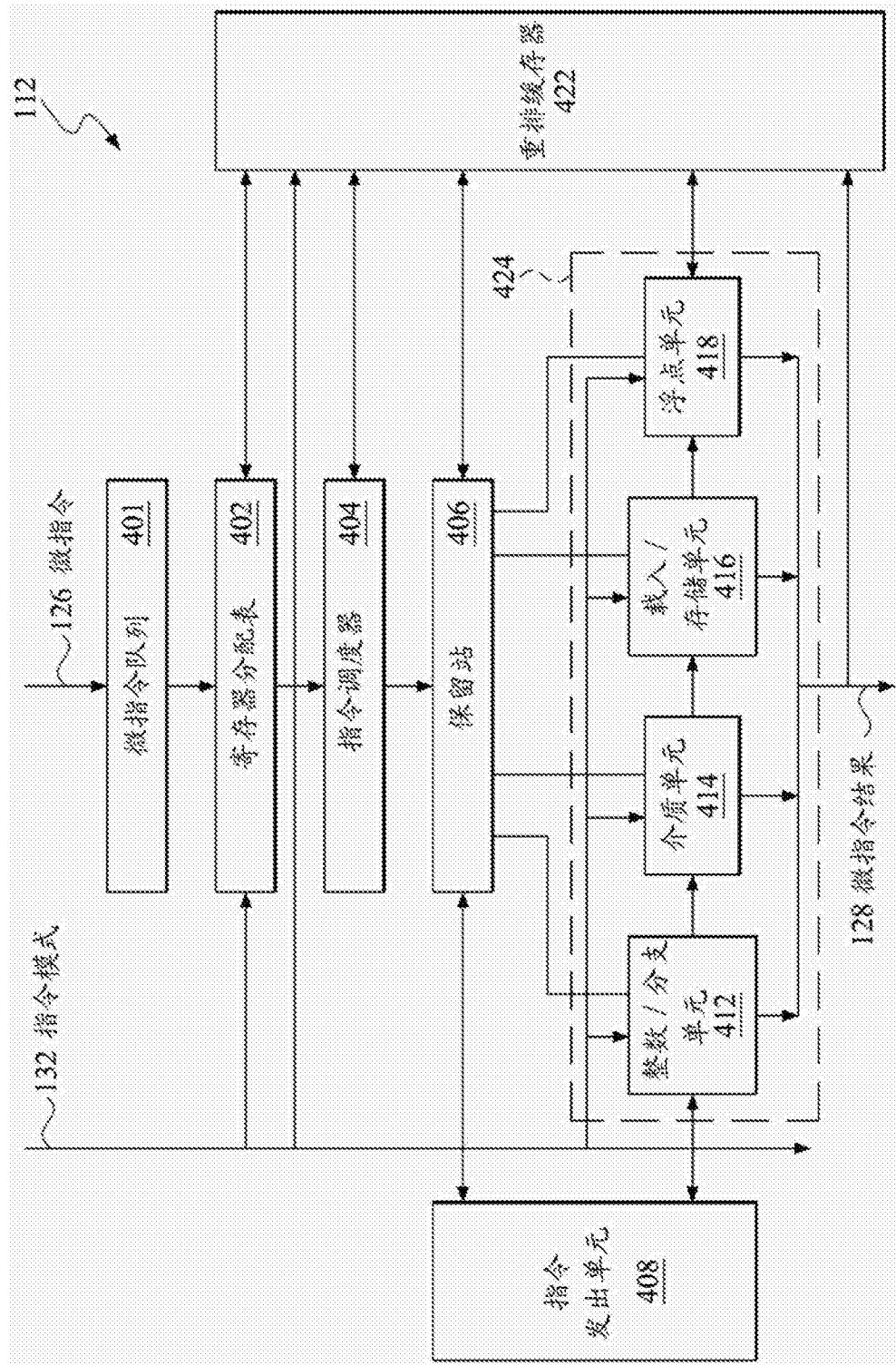


图4

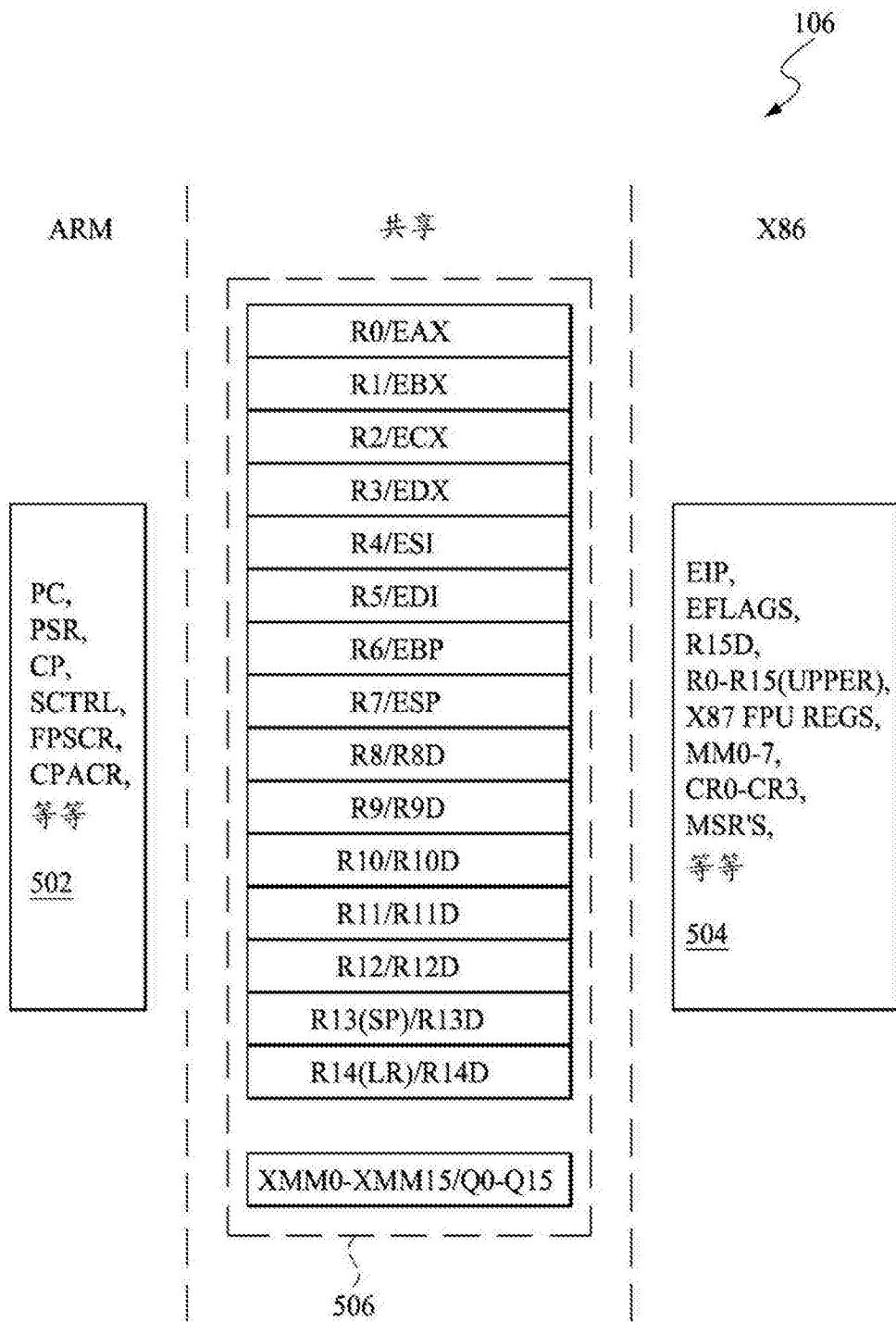


图5

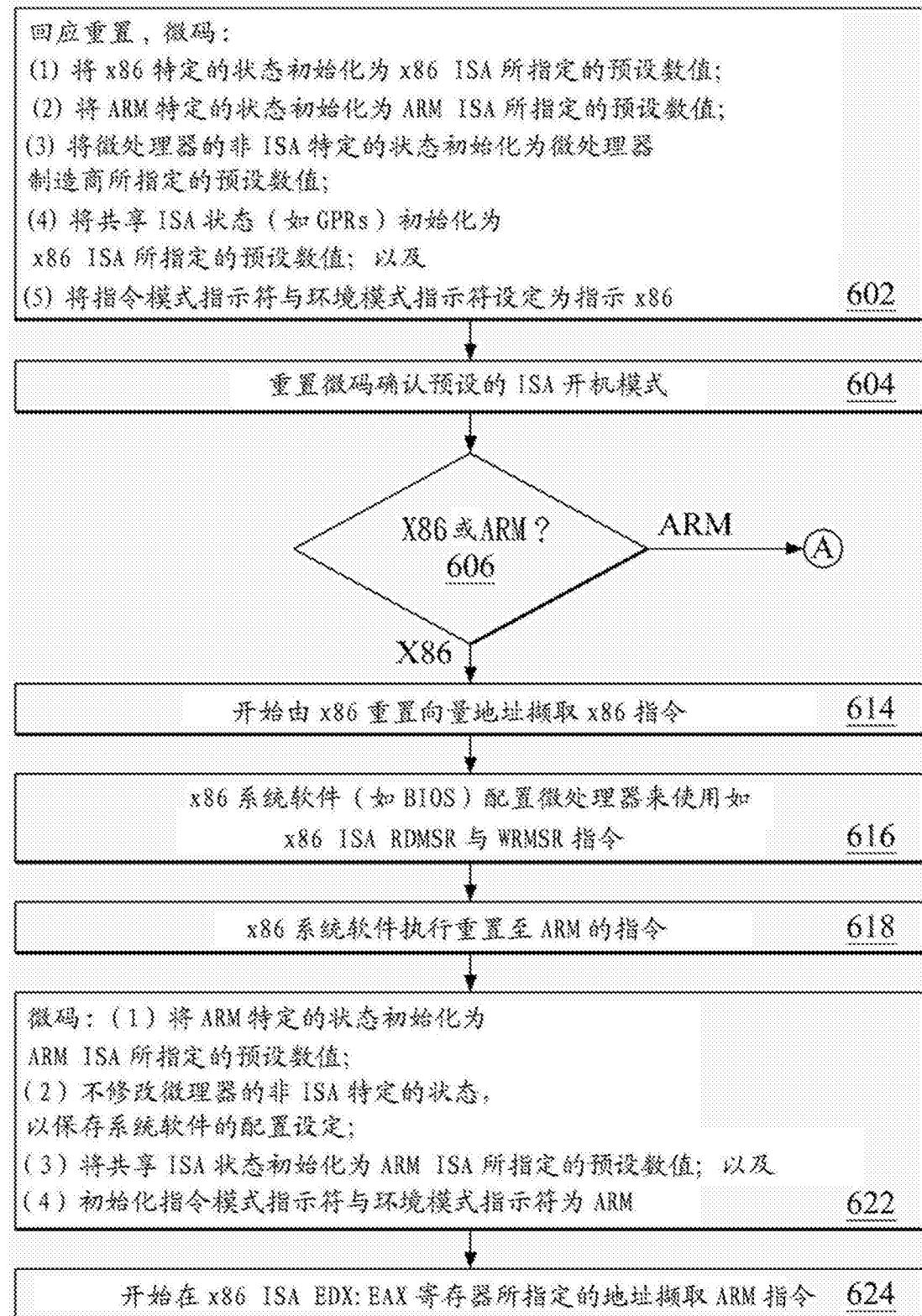


图6A

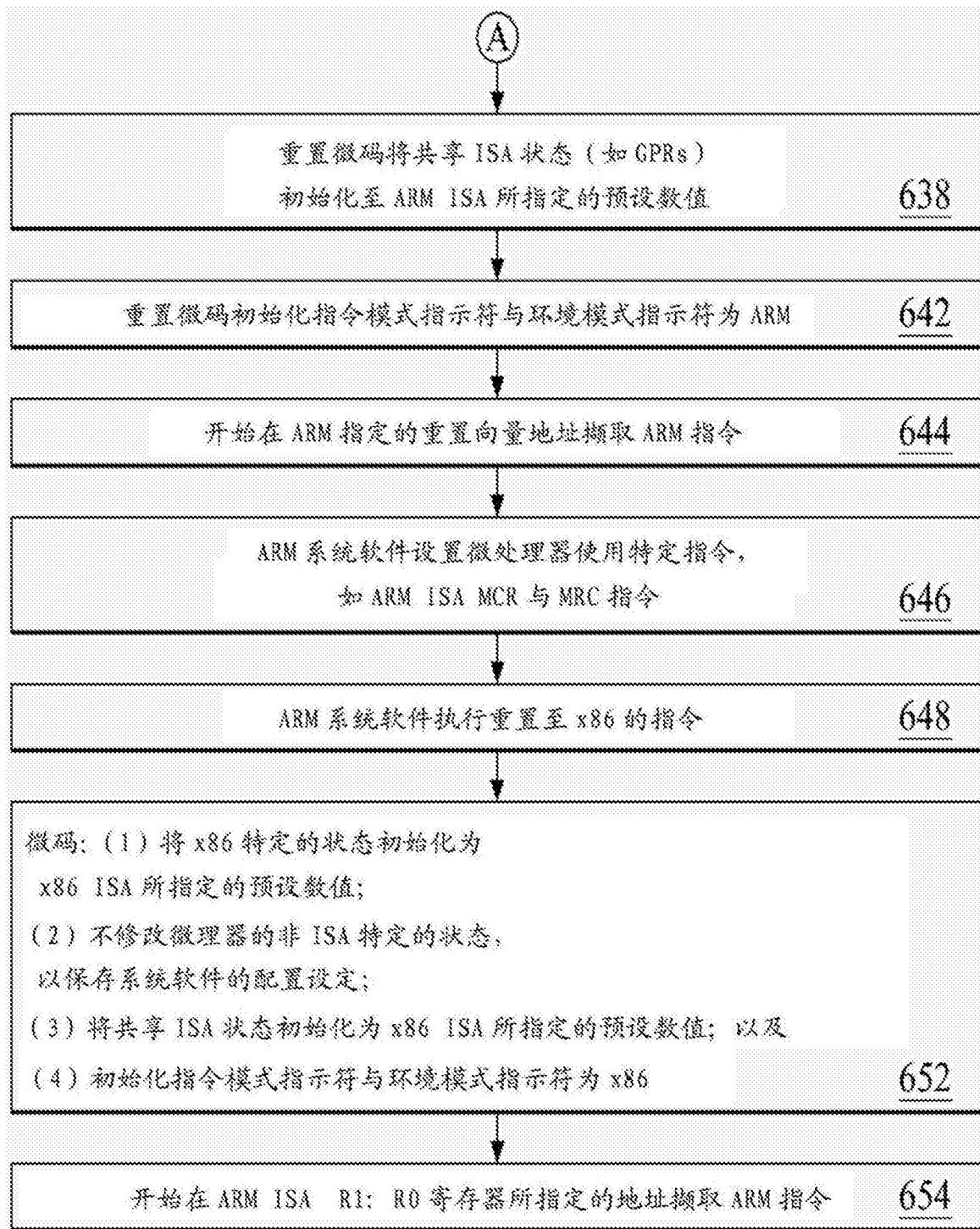


图6B

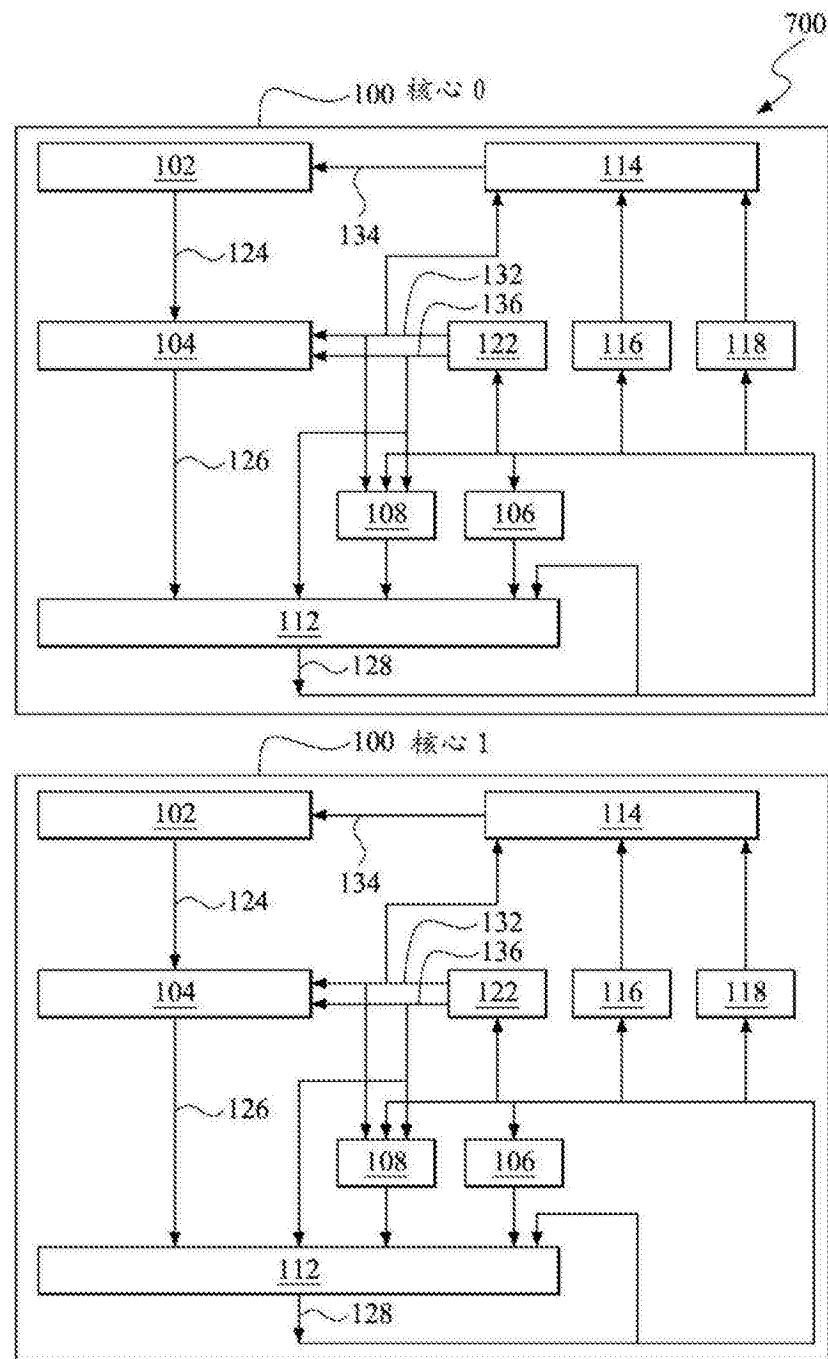


图7

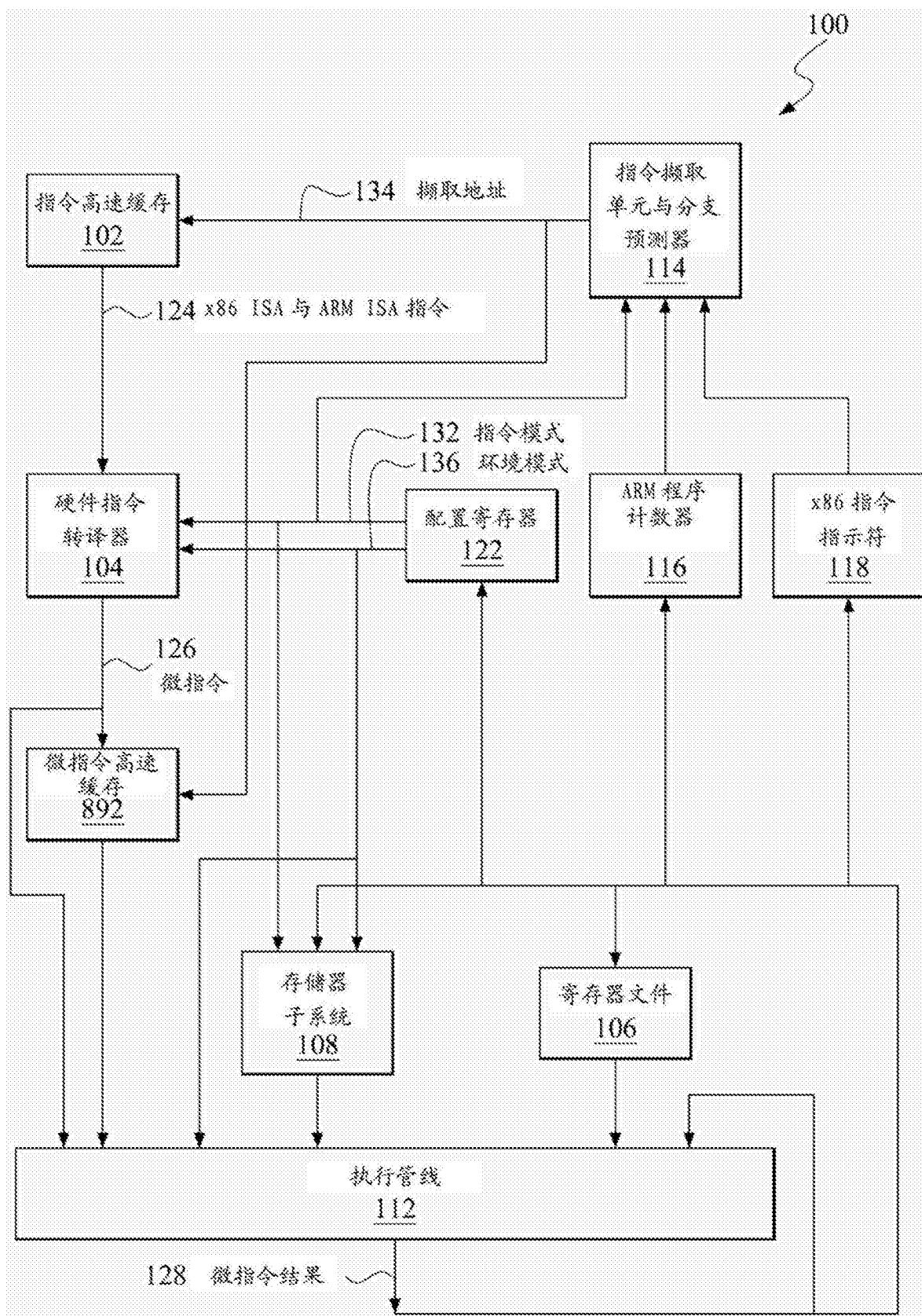


图8

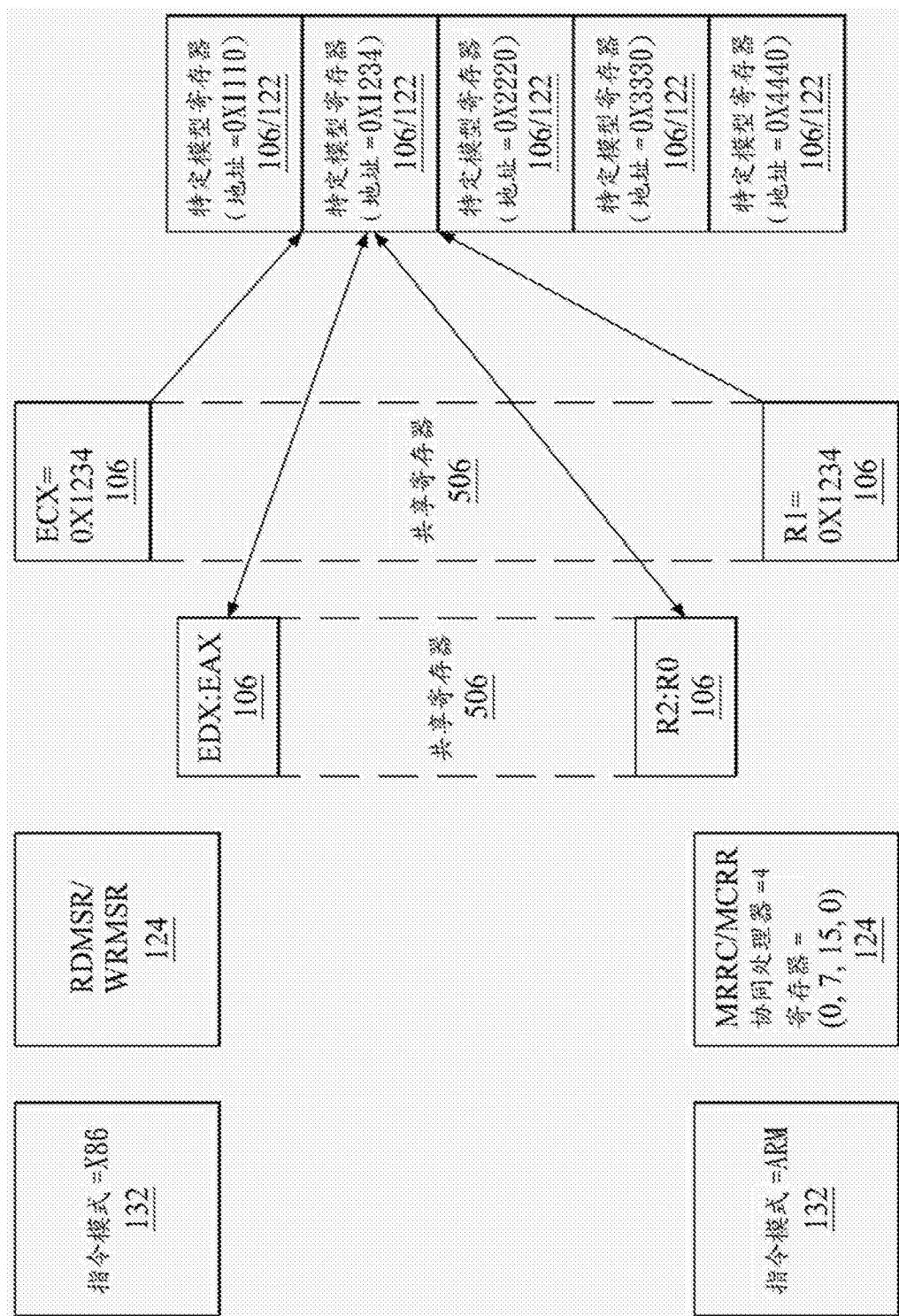


图9

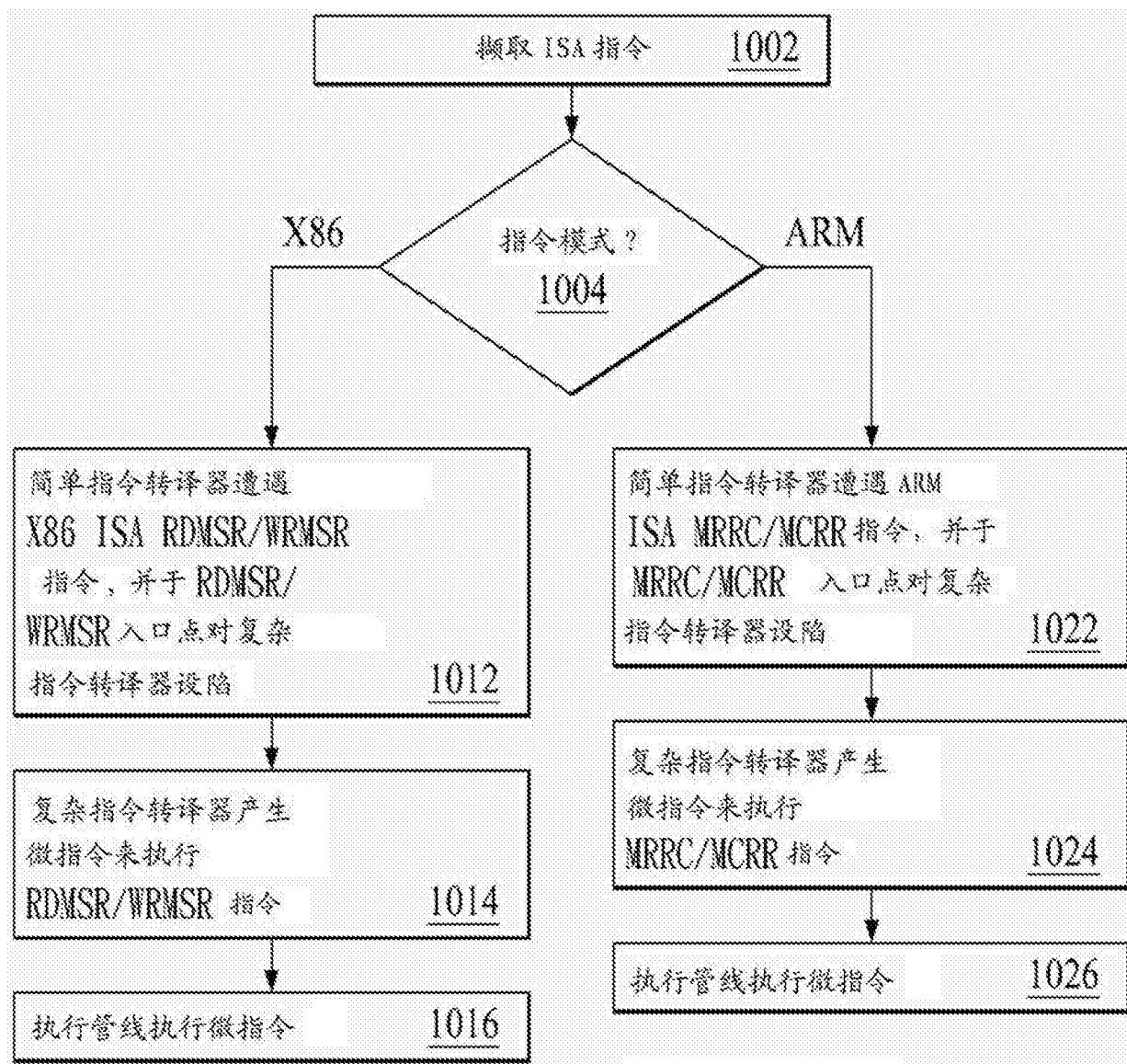


图10

```

MRBC_入口_点：
    确定协调处理器 ==4 ;
    确定暂存器地址 ==(0, 7, 15, 0) ;
    确认不是在使用者模式 ;
    确认已启用从 ARM 存取 MSR 功能 ;
    将 R1 内的数值复制到 TEMP1;
    呼叫 RDMSR_COMMON ;
    将 TEMP2 内的数值复制到 R2; R0;
    从微码返回 ;
    ...

RDMSR_入口_点：
    将 ECX 内的数值复制到 TEMP1 ;
    呼叫 RDMSR_COMMON ;
    将 TEMP2 内的数值复制到 EDX; EAX ;
    从微码返回 ;
    ...

RDMSR_COMMON :
    // 将 TEMP2 内的数值载入到已在 TEMP1 内指定地址的 MSR;
    返回 ;
    ...

MCRR_入口_点：
    确定协调处理器 ==4 ;
    确定暂存器地址 ==(0, 7, 15, 0) ;
    确认不是在使用者模式 ;
    确认已启用从 ARM 存取 MSR 功能 ;
    将 R1 内的数值复制到 TEMP1;
    将 R2; R0 内的数值复制到 TEMP2;
    呼叫 WRMSR_COMMON ;
    从微码返回 ;
    ...

WRMSR_入口_点：
    将 ECX 内的数值复制到 TEMP1;
    将 EDX; EAX 内的数值复制到 TEMP2;
    呼叫 WRMSR_COMMON ;
    从微码返回 ;
    ...

WRMSR_COMMON :
    // 将 TEMP2 内的数值载入到已在 TEMP1 内指定地址的 MSR;
    返回 ;

```

微码 234

图11