(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 105005463 B
(45)授权公告日 2018.03.06

(21)申请号 201510206131.X

(22)申请日 2015.04.27

(65)同一申请的已公布的文献号
申请公布号 CN 105005463 A

(43)申请公布日 2015.10.28

(30)优先权数据
61/984,709 2014.04.25 US
14/530,370 2014.10.31 US

(73)专利权人 安华高科技通用IP(新加坡)公司
地址 新加坡新加坡市

(72)发明人 索菲·威尔逊 约翰·雷德福
杰弗里·巴雷特 塔里克·库尔德

(74)专利代理机构 北京律盟知识产权代理有限责任公司 11287
代理人 张世俊

(51)Int.Cl.
G06F 9/30(2006.01)
G06F 13/38(2006.01)

(56)对比文件
US 6370637 B1,2002.04.09,
EP 0863460 A2,1998.09.09,
US 5881262 A,1999.03.09,
US 2012/0226894 A1,2012.09.06,

审查员 白桦

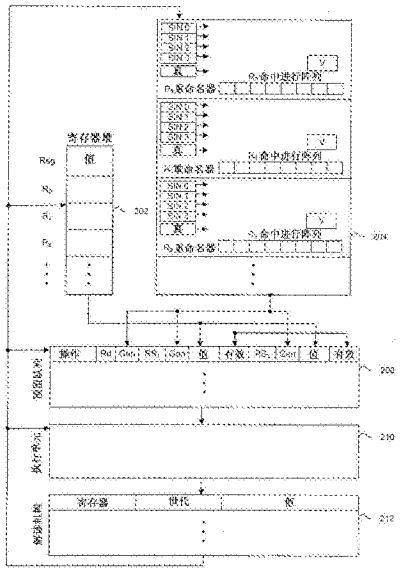
权利要求书3页 说明书37页 附图18页

(54)发明名称

具有世代重命名的计算机处理器

(57)摘要

本公开涉及一种具有世代重命名的计算机处理器，该处理器包括寄存器堆，寄存器堆具有多个寄存器并且被配置用于乱序指令，该处理器进一步包括重命名器单元，重命名器单元产生与寄存器堆地址相关联的号，以提供从现有版本的寄存器临时偏移的重命名版本的寄存器，而非指定非程序员可见的物理寄存器作为重命名寄存器。该处理器包括耦接至指令高速缓存并且被配置为将推测性地址提供给指令高速缓存的小型重置DHL Gshare分支预测单元。



CN 105005463 B

1.一种处理器,包括:

预留队列;

重命名器,所述重命名器耦接至所述预留队列并且被配置为产生世代号且将所述世代号通信至所述预留队列;

寄存器,所述寄存器耦接至所述预留队列并且被配置为存储值;

执行管线,所述执行管线耦接至所述预留队列;以及

解读机构,所述解读机构耦接至所述执行管线,所述解读机构包括:

第一存储器,所述第一存储器包括多个寄存器,所述多个寄存器经配置以将暂时将执行管线结果作为中间结果存储;

第二存储器,所述第二存储器通信耦接至所述第一存储器,所述第二存储器经配置以接收且存储包括所述中间结果中的一或多者的多个存储结果;

第一仲裁器,所述第一仲裁器通信性耦接至所述第一存储器和所述第二存储器,所述第一仲裁器经配置以接收多个中间结果和所述多个存储结果,且从所述多个中间结果和所述多个存储结果中选择将引退的所述第一仲裁器的输出;

第二仲裁器,所述第二仲裁器经通信耦接以接收执行管线结果和所述第一仲裁器的输出;以及

第三存储器,所述第三存储器经通信耦接以从所述第二仲裁器接收多个推测执行管线结果。

2.根据权利要求1所述的处理器,进一步包括:

小型重置DHL Gshare分支预测单元。

3.根据权利要求1所述的处理器,进一步包括:

加载/存储单元,所述加载/存储单元被配置为针对多个排队等待的加载和存储指令中的每个调度对内存的访问;

其中,所述加载/存储单元包括被配置为防止冲突的内存操作被同时调度的资源锁定电路。

4.一种处理器,包括:

指令高速缓存;

指令解码器,所述指令解码器耦接至所述指令高速缓存;

预留队列,所述预留队列耦接至所述指令解码器;

寄存器,所述寄存器耦接至所述预留队列并且被配置为存储值;

重命名器,所述重命名器耦接至所述预留队列并被配置为产生世代号且将所述世代号传送到所述预留队列;

小型重置DHL Gshare分支预测单元,所述小型重置DHL Gshare分支预测单元耦接至所述指令高速缓存并且被配置为生成地址且将所述地址通信至所述指令高速缓存;

多个执行管线,所述多个执行管线耦接至所述预留队列;以及

解读机构,所述解读机构耦接到所述多个执行管线中的每一执行管线,所述解读机构包括:

第一存储器,所述第一存储器包括多个寄存器,所述多个寄存器经配置以将暂时将多个执行管线结果作为多个中间结果存储;

第二存储器，所述第二存储器通信耦接至所述第一存储器，所述第二存储器经配置以接收且存储包括所述多个中间结果中的一或多者的多个存储结果；

第一仲裁器，所述第一仲裁器通信耦接至所述第一存储器和所述第二存储器，所述第一仲裁器经配置以接收所述多个中间结果和所述多个存储结果，且从所述多个中间结果和所述多个存储结果中选择将引退的所述第一仲裁器的输出；

第二仲裁器，所述第二仲裁器经通信耦接以接收执行管线结果和所述第一仲裁器的输出；以及

第三存储器，所述第三存储器经通信耦接以从所述第二仲裁器接收多个推测执行管线结果。

5.一种处理器，包括：

指令高速缓存；

指令解码器，所述指令解码器耦接至所述指令高速缓存；

分支预测单元，所述分支预测单元耦接至所述指令高速缓存并且被配置为生成指令地址且将所述指令地址提供至所述指令高速缓存；

寄存器堆，所述寄存器堆被耦接为接收寄存器地址；

重命名器单元，所述重命名器单元耦接至所述指令高速缓存并且耦接至所述指令解码器；

多个预留队列，所述多个预留队列中的每个预留队列均耦接至所述寄存器堆、所述指令解码器、以及所述重命名器单元；

多个执行管线，所述多个执行管线中的每个执行管线均耦接至所述多个预留队列中的相应一个，并且被配置为从所述多个预留队列中的所述相应一个接收指令和数据且执行所述指令；

解读机构，所述解读机构耦接至所述多个执行管线中的每个执行管线并且被配置为接收所述多个执行管线中的每个执行管线的输出，所述解读机构包括：

第一存储器，所述第一存储器包括多个寄存器，所述多个寄存器经配置以将暂时将多个执行管线结果作为多个中间结果存储；

第二存储器，所述第二存储器通信耦接至所述第一存储器，所述第二存储器经配置以接收且存储包括所述多个中间结果中的一或多者的多个存储结果；

第一仲裁器，所述第一仲裁器通信耦接至所述第一存储器和所述第二存储器，所述第一仲裁器经配置以接收所述多个中间结果和所述多个存储结果，且从所述多个中间结果和所述多个存储结果中选择将引退的所述第一仲裁器的输出；

第二仲裁器，所述第二仲裁器经通信耦接以接收执行管线结果和所述第一仲裁器的输出；以及

第三存储器，所述第三存储器经通信耦接以从所述第二仲裁器接收多个推测执行管线结果；

其中，响应于所述指令解码器的输出，所述寄存器堆被配置为提供指令中指定为来源的寄存器的内容；并且响应于所述指令解码器的所述输出，所述重命名器被配置为同时提供世代号。

6.根据权利要求5所述的处理器，其中，所述解读机构的输出耦接至所述多个执行管线

中的每个执行管线的输入，并且进一步耦接至所述多个预留队列中的每个预留队列的输入。

7.根据权利要求5所述的处理器，其中，所述多个预留队列中的每个预留队列均被配置为存储指令操作码、目的地寄存器地址、以及目的地寄存器世代号。

8.根据权利要求5所述的处理器，进一步包括指令取出单元。

9.根据权利要求8所述的处理器，其中，所述指令取出单元包括：

循环计数寄存器，所述循环计数寄存器被配置为存储循环计数值或者存储推测性循环计数值；

循环开始寄存器，所述循环开始寄存器被配置为存储循环开始地址；以及

循环匹配寄存器，所述循环匹配寄存器被配置为存储循环匹配地址。

10.根据权利要求8所述的处理器，其中，所述指令取出单元耦接至所述指令解码器和SIN分配器，并且所述指令取出单元被配置为将指示表示推测性循环执行的循环计数值的信号提供给所述SIN分配器。

具有世代重命名的计算机处理器

[0001] 相关申请的交叉引用

[0002] 该非临时性申请要求保护于2014年4月25日提交的题为“具有世代重命名的计算机处理器”的美国临时申请号61/984,709、于2014年4月25日提交的题为“零开销循环”的美国临时申请号61/984,710,、于2014年4月25日提交的题为“分支预测”的美国临时申请号61/984,711、于2014年4月25日提交的题为“解读机构(stunt box)”的美国临时申请号61/984,708、于2014年4月25日提交的题为“用于VLIW处理器中的加载/存储调度的资源锁定”的美国临时申请号61/984,707、以及于2014年4月25日提交的题为“具有世代号的重命名”的美国临时申请号61/984,706的权益,通过引用将所有公开内容结合在此。

技术领域

[0003] 本公开整体涉及计算机处理器。

背景技术

[0004] 半导体制造的超前发展使得可以整合集成电路中的大量逻辑线路。从而,这又导致了数字系统架构的超前发展。数字系统架构中从包括单集成电路中的各种逻辑线路的功能而受益极大的一方特殊领域就是处理器设计。

发明内容

[0005] 根据本发明的一个方面,提供了一种处理器,包括:预留队列(reservation queue);重命名器,所述重命名器耦接至所述预留队列并且被配置为产生世代号(generation number)且将所述世代号通信至所述预留队列;寄存器,所述寄存器耦接至所述预留队列并且被配置为存储值;执行管线(execution pipe),所述执行管线耦接至所述预留队列;以及缓冲器,所述缓冲器耦接至所述执行管线并且被配置为接收所述执行管线的输出,并且所述缓冲器进一步耦接至所述预留队列且被配置为将所述执行管线的所述输出通信至所述预留队列。

[0006] 根据一个实施方式,该处理器进一步包括:小型重置DHL Gshare分支预测单元。

[0007] 根据一个实施方式,该处理器进一步包括:加载/存储单元,所述加载/存储单元被配置为调度对多个队列加载和存储指令中的每个的内存的访问;其中,所述加载/存储单元包括被配置为防止冲突的内存操作被同时调度的资源锁定电路。

[0008] 根据本发明的另一个方面,提供了一种处理器,包括:指令高速缓存(instruction cache);指令解码器,所述指令解码器耦接至所述指令高速缓存;预留队列,所述预留队列耦接至所述指令解码器;寄存器,所述寄存器耦接至所述预留队列并且被配置为存储值;小型重置DHL Gshare分支预测单元,所述小型重置DHL Gshare分支预测单元耦接至所述指令高速缓存并且被配置为生成地址且将所述地址通信至所述指令高速缓存;执行管线,所述执行管线耦接至所述预留队列;缓冲器,所述缓冲器耦接至所述执行管线并且被配置为接收所述执行管线的输出,并且所述缓冲器进一步耦接至所述预留队列且被配置为将所述执

行管线的所述输出通信至所述预留队列。

[0009] 根据本发明的另一个方面，提供了一种处理器，包括：指令高速缓存；指令解码器，所述指令解码器耦接至所述指令高速缓存；分支预测单元，所述分支预测单元耦接至所述指令高速缓存并且被配置为生成指令地址且将所述指令地址提供至所述指令高速缓存；寄存器堆(register file)，所述寄存器堆被耦接至接收寄存器地址；重命名器单元，所述重命名器单元耦接至所述指令高速缓存并且耦接至所述指令解码器；多个预留队列，所述多个预留队列中的每个预留队列均耦接至所述寄存器堆、所述指令解码器、以及所述重命名器单元；多个执行管线，所述多个执行管线中的每个执行管线均耦接至所述多个预留队列中的相应一个，并且被配置为从所述多个预留队列中的所述相应一个接收指令和数据且执行所述指令；解读机构(stunt box)，所述解读机构耦接至所述多个执行管线中的每个执行管线并且被配置为接收所述多个执行管线中的每个执行管线的输出；其中，所述寄存器堆被配置为响应所述指令解码器的输出提供指令中指定为寄存器来源的内容；并且所述重命名器被配置为响应所述指令解码器的所述输出同时提供世代号。

[0010] 根据一个实施方式，所述分支预测单元包括小型重置DHL Gshare分支预测器。

[0011] 根据一个实施方式，所述指令高速缓存被配置为接收并且存储VLIW指令。

[0012] 根据一个实施方式，所述解读机构的输出耦接至所述多个执行管线中的每个执行管线的输入，并且进一步耦接至所述多个预留队列中的每个预留队列的输入。

[0013] 根据一个实施方式，处理器进一步包括：加载/存储单元，所述加载/存储单元耦接至所述解读机构并且耦接至所述寄存器堆；和内存，所述内存耦接至所述加载/存储单元；其中，所述加载/存储单元被配置为接收来自所述解读机构的输入并且接收来自所述寄存器堆的输入。

[0014] 根据一个实施方式，所述多个预留队列中的每个预留队列均被配置为存储指令操作码、目的地寄存器地址、以及目的地寄存器世代号。

[0015] 根据一个实施方式，所述多个预留队列中的每个预留队列进一步被配置为存储第一源寄存器地址、第一寄存器世代号、第一寄存器值、第一有效位、第二源寄存器地址、第二寄存器世代号、第二寄存器值、以及第二有效位。

[0016] 根据一个实施方式，所述指令解码器识别指定成对寄存器操作的指令。

[0017] 根据一个实施方式，所述寄存器堆包括多个寄存器；并且所述重命名器单元包括用于所述多个寄存器中的每个寄存器的寄存器重命名块。

[0018] 根据一个实施方式，所述解读机构被配置为将双宽输出(pair-wide output)和单宽输出(single-wide output)同时放置在所述操作数复制网络上。

[0019] 根据一个实施方式，处理器进一步包括指令取出单元。

[0020] 根据一个实施方式，所述指令取出单元包括：循环计数寄存器，所述循环计数寄存器被配置为存储循环计数值或者存储推测性循环计数值；循环开始寄存器，所述循环开始寄存器被配置为存储循环开始地址；以及循环匹配寄存器，所述循环匹配寄存器被配置为存储循环匹配地址。

[0021] 根据一个实施方式，所述指令取出单元耦接至所述指令解码器和SIN分配器，并且所述指令取出单元被配置为将指示表示推测性循环执行的循环计数值的信号提供给所述SIN分配器。

[0022] 根据一个实施方式，所述指令解码器耦接至所述循环计数寄存器、所述循环开始寄存器、以及所述循环匹配寄存器。

[0023] 根据一个实施方式，所述指令取出单元被配置为生成循环匹配信号。

[0024] 根据一个实施方式，所述指令解码器被配置为从所述指令取出单元接收所述循环匹配信号。

附图说明

[0025] 参考附图描述了示例性实施方式。在附图中，类似参考标号指相同或者功能相似的元件。此外，参考标号中最左侧的数字确定其中参考标号首先出现的附图。

[0026] 图1是示例性处理器的高级框图。

[0027] 图2是实现寄存器重命名的示例性处理器的高级框图。

[0028] 图3是示出了根据示例性实施方式的寄存器重命名过程的流程图。

[0029] 图4是具有资源锁定的示例性处理器的高级框图。

[0030] 图5是示出了根据示例性实施方式的状态确定过程的流程图。

[0031] 图6是示出了根据示例性实施方式的指令状态更新过程的流程图。

[0032] 图7是实现解读机构的示例性处理器的高级框图。

[0033] 图8是实现解读机构的另一示例性处理器的高级框图。

[0034] 图9是示出了根据示例性实施方式的结果选择过程的流程图。

[0035] 图10示出了根据本公开的实施方式的示例性分支预测单元。

[0036] 图11进一步详细地示出了根据本公开的实施方式的分支预测单元。

[0037] 图12示出了根据本公开的实施方式的全局分支历史寄存器的示例性更新。

[0038] 图13示出了根据本公开的实施方式的全局分支历史寄存器的预置。

[0039] 图14示出了根据本公开的实施方式的提供访问更大分支历史表中的条目的索引的示例性系统。

[0040] 图15示出了根据本公开的实施方式的提供访问小型分支历史表、混合选择器、以及更新计数器中的每一个中的条目的索引的系统。

[0041] 图16示出了根据本公开的实施方式的用于更新存储在大型分支历史表和小型分支历史表中的分支预测条目的示例性状态机。

[0042] 图17示出了用于实现根据本公开的实施方式的零开销(zero overhead)循环的架构的一部分。

[0043] 图18示出了用于实现根据本公开的实施方式的零开销循环的架构的一部分。

[0044] 图1至图18示出了本领域中易于实施的各个部件、其布置、以及互连，图不一定按比例绘制。

具体实施方式

[0045] 下列细节描述参考了示出示例性实施方式的附图。在细节描述中，参考“一种示例性实施方式”、“示出性实施方式”、“示例性实施方式”等指所描述的示例性实施方式可包括具体特性、结构、或者特征，但是，每种示例性实施方式不一定必须包括该具体特性、结构、或者特征。而且，该短语不一定必须指相同的示例性实施方式。此外，当结合示例性实施方

式描述具体特性、结构、或者特征时，相关领域技术人员利用常识结合其他示例性实施方式(无论是否明确进行描述)影响该特性、结构、或者特征。

[0046] 出于示出性之目的而非限制之目的提供此处所描述的示例性实施方式。其他示例性实施方式是可以的，并且在本公开的实质和范围内可以对示例性实施方式做出更改。

[0047] 应当理解的是，此处的措辞或者术语出于描述而非限制之目的，由相关领域技术人员根据此处的教导解释本说明书的术语或者措辞。

[0048] 术语

[0049] 在电子领域中，术语“芯片”、“裸片”、“集成电路”、“半导体器件”、以及“微电子器件”等通常可相互使用。

[0050] 如此处使用的，FET指金属氧化物半导体场效应晶体管(MOSFET)。n沟道FET被称之为NFET。p沟道FET被称之为PFET。

[0051] CMOS是代表互补金属氧化物半导体的简称，并且指其中NFET和PFET形成在同一芯片上的半导体制造工艺。

[0052] CMOS电路指其中一起使用NFET和PFET的电路。

[0053] SoC是代表片上系统的简称，并且指包括通常通过总线互连的两个或者多个电路块的芯片，其中，这些电路块提供如此高级的功能以至于之前将这些块视为系统级部件。例如，至今，具有必备功能的电路块包括标量、超标量、以及极长指令字处理器、DRAM控制器(例如，DDR3、DDR4、以及DDR5)、闪存存储控制器、通用串行总线(USB)控制器等。该列表旨在为示出性并且并不具有限制性。描述SoC的另一种常见方式是包括实现诸如计算机系统或者基于计算机的系统等电子系统所需的所有部件的芯片。

[0054] VLIW是极长指令字的简称。

[0055] 如在此处描述示例性实施方式时所使用的，VLIW指令指用于呈现给指令解码器的分组到一起的一组指令。该组指令中的独立指令被分配给用于执行指令的多个执行管线中的一个。

[0056] IC0指到指令高速缓存的输入端上的伪阶段(pseudo-stage)。

[0057] IC1指指令高速缓存阶段。在该周期中，发出对指令高速缓存的取出请求，并且同时计算以确定下一步取出哪一PC。在该阶段，提供之前请求的VLIW指令。

[0058] DE1指的是指令解码器的第一阶段。

[0059] DE1_操作指在指令解码器的第一阶段中执行的逻辑操作。

[0060] DE1_时间指其中执行DE1_操作的一个周期。

[0061] DE2指的是指令解码器的第二阶段。

[0062] DE2_操作指在指令解码器的第二阶段中执行的逻辑操作。

[0063] DE2_时间指其中发生读取和重命名通用寄存器堆(GRF)和预测寄存器堆(PREG)的一个周期。

[0064] RS指预留站。存在若干个不同预留站，指令或操作可被入队(enqueue)至这些预留站。在最佳情况下，这是一个单周期阶段，然而，在这里，操作可在多个周期中结束排队。

[0065] Exn指执行管线的第n个阶段。执行管线的实施例包括ALU短管线和长管线、BRANCH、以及加载存储单元。

[0066] SHP指短执行管线。使用短执行管线执行单周期操作。

[0067] LOP指长执行管线。使用长执行管线执行需要2至8个周期才可完成的指令。

[0068] LSU指加载存储单元。

[0069] DTCM指数据紧密耦接内存。

[0070] PBUS指连接至外围内存的总线。

[0071] DCACHE指用于缓存访问外围内存的数据缓存。

[0072] 入队指其中将DE2中的VILW指令分割成其分量操作并且然后使管线向下前移至预留站中的动作。

[0073] 发布指将操作从预留站移至执行单元。当操作从预留站移至执行单元时，操作被称之为发布。操作是VLIW指令的组成部分。

[0074] 当前PC指用于当前在给定阶段中的指令的程序计数器(PC)的值。管线的每个阶段具有自身版本的当前PC。

[0075] 下一PC指从Icache取出的下一个PC。对于直线码，则为当前PC+当前指令宽度，对于重定向码，则为新目标PC。

[0076] 循环开始地址指循环体中的第一指令的地址，即，被分支到以开始新循环迭代的地址。

[0077] 循环结束地址指循环体之后的第一指令的地址，即，被分支到以自然退出该循环的地址。

[0078] 循环体指以循环开始地址开始并且以循环匹配地址结束的指令。

[0079] 循环匹配地址指循环体中的最后指令的地址。

[0080] 循环计数指应执行的循环的迭代次数。这来自用于LOOP操作的中间字段，或者来自用于一个ZLOOP和多个ZLOOP操作的通用寄存器。

[0081] SIN指推测索引号，即，用于识别在分支附近处被推测地入队的指令。

[0082] SIN决策指确定是否正确推测分支。在EX1中执行SIN决策。

[0083] SIN验证指EX1中正确推测的分支，其从而对与正确推测分支附近处的操作相关联的SIN进行验证。验证操作是更新架构状态的一种操作。

[0084] SIN撤消指的是EX1中不正确推测的分支，其从而将撤消所有突出的SIN并且执行EX1重定向，从而从执行管线有效移除位于分支附近的所有操作。在一种实施方式中，移除位于不正确推测分支附近的操作包括改变与执行管线中的每个指令相关联的位的状态。

[0085] 状态一致性执行(SCE)指由内部机构执行的防止未来操作发生不一致机器状态的动作。

[0086] 捕获事件指同步、异步、以及故障事件的集合。

[0087] 同步捕获事件涉及特定指令并且被及时检测，以防止导致该事件的指令入队。管理程序调用(SVC)指令填充该目录。由于同步捕获事件发生在指令流中的架构限定位置，所以其比较精确。

[0088] 异步捕获事件(中断)与当前指令序列单独发生。异步异常符合这种情况。

[0089] 故障捕获事件防止程序流恢复。故障捕获事件的例子是未对准PC和数据中止。具有寄存器目的地的故障操作必须完成寄存器值。

[0090] 处理器概况

[0091] 公开了一种处理器架构，该处理器架构包括具有多个寄存器的寄存器堆、被配置

为用于乱序指令执行、并且进一步包括重命名器单元，重命名器单元产生与寄存器堆地址相关联的世代号，从而提供与现有版本的寄存器临时偏移的重命名版本的寄存器、而非指定非程序员可见物理寄存器作为重命名寄存器。处理器架构包括耦接至指令高速缓存并且被配置为将推测性地址提供至指令高速缓存的小型重置双重历史长度(DHL)Gshare分支预测单元。处理器架构适于实现集成电路。通常，但不一定必须是，利用CMOS线路实现集成电路。

[0092] 在示例性实施方式中，在作为嵌入式处理器的集成电路中实现根据本公开的处理器。

[0093] 图1是示出了根据本公开的示例性处理器的主要块的高级框图。示例性处理器包括指令高速缓存102，指令高速缓存102被耦接为从分支预测单元104接收VLIW指令地址，并且进一步被耦接为将输出提供至分支预测单元104、指令解码器106、寄存器堆108、以及世代重命名器110。世代重命名器110耦接至分支执行单元118以接收SIN控制信号、耦接至SIN分配器以接收SIN号、耦接至解读机构124以接收来自操作数(oprand)复制网络的输出、并且耦接至分支预留队列112、执行管线预留队列114A、114B、114C、114D、和加载/存储预留队列116。寄存器堆108耦接至解读机构124，以接收来自操作数复制网络的输入，并且进一步耦接至分支预留队列112、执行管线预留队列114A、114B、114C、114D、以及加载/存储预留队列116。分支预留队列112耦接至分支执行单元118。执行管线预留队列114A、114B、114C、114D各自分别耦接至相应的执行管线120A、120B、120C、以及120D。执行管线120A、120B、120C、以及120D各自耦接为将输出提供至解读机构124。执行管线120A、120B、120C、以及120D各自分别耦接为将其输出提供回至其输入，并且每个进一步耦接至分支执行单元118的输出，以接收SIN控制信号。内存122耦接至加载/存储单元116。并且加载/存储单元116进一步耦接至系统总线126。

[0094] 指令高速缓存102保存之前通过指令取出单元(未示出)而取出的VLIW指令。通常，从设置在处理器本身的外部的内存取出VLIW指令。分支预测单元104被示出为耦接至指令高速缓存102。分支预测单元104提供供取出的VLIW指令的地址。如果请求的VLIW指令存在于指令高速缓存102中，则将其提供给指令解码器106。如果请求的VLIW指令不存在于指令高速缓存102中，则发生缓存未命中(miss)，并且从设置在处理器外部的内存取出请求指令。

[0095] 分支预测单元104具有若干种功能，其中包括提供指令高速缓存102所需的程序计数值，以及整个处理器的不同阶段和逻辑块需要的程序计数值。对于连续执行的程序代码，程序计数值仅通过刚刚取出的指令长度而改变。但是，当检测到分支指令时，分支预测单元104则确定下一指令应从哪一地址取出什么。在该示例性处理其中红，分支预测单元104使用小型重置DHL Gshare分支预测机构来确定下一指令地址。

[0096] 指令解码器106解码VLIW指令的内容并且将控制信息提供给处理器的各个其他块。

[0097] 寄存器堆108包含预定数目的程序员可见寄存器。这些寄存器保存在程序执行过程中所使用的值。

[0098] 将从VLIW指令获得的各个指令入队到选择的预留队列中。当执行入队指令所需的操作数变得可用时，则将该指令发布给与选择的预留队列相关联的执行管线。

[0099] 世代重命名器110用于通过指令将世代号分配给寄存器实例，而这些寄存器实例通常被再分配给不同的非程序员可见的物理寄存器。

[0100] 预留队列保存等待被发布的指令。

[0101] 解读机构124提供用于接收和分发执行管线的输出的机构。解读机构124将数据提供给操作数复制网络。操作数复制网络允许处理器中的其他块可使用执行管线的所有结果。同样，等待从执行另一指令时产生的操作数的指令无需等待将该操作数写回至寄存器堆中并且然后从寄存器堆中被读取。更确切地，经由操作数复制网络可使等待该特定结果的处理器中的所有位置使用请求操作数。

[0102] 系统总线126提供一种使嵌入式处理器与位于处理器本身外部的集成电路上的其他逻辑块通信的机构。

[0103] 利用世代号的重命名

[0104] 处理器包括多种不同的物理资源，例如，寄存器、执行单元、算术单元、内存、控制逻辑等。处理器中包括的物理资源中的一种就是程序员在创建软件时可使用的一组架构可见寄存器。这些寄存器根据软件的指示物理地存储处理器所使用的信息。基于处理器的设计和实现的指令集确定该组寄存器中的可用寄存器数目。因为存在一定数目的寄存器，所以通常要求程序员在创建其软件时再次使用寄存器。

[0105] 处理器在执行软件时需要确保正确值及时在正确时隙(instant)位于各个寄存器中，以确保正确执行程序。这对于根据可用的资源以不同顺序执行指令的乱码机如此，并且甚至对于能够取出、解码、以及并行执行多个指令的极长指令字(VLIW)处理器更是如此。

[0106] 为了解决这些问题，开发了世代重命名。世代重命名允许在寄存器堆中保存各个架构寄存器的值的过程，同时允许各个寄存器的多个版本、命名世代存在于该机器中。每一代均代表将或者可存储在寄存器中的新值。不同寄存器可同时使用不同的世代。例如，该机器可在使用R1的第5代和第7代的同时使用R0的第2代、第3代、以及第4代。就架构而言，每个寄存器均具有与其相关联的值、保存当前使用的是哪一代的装置、以及用于识别最后已知正确使用的世代和任何推测使用的世代的一组指针。下面将更为详细地描述这些部件。

[0107] 当各个指令处于命中进行(in flight)时，例如，等待被执行、正在被执行、或者等待被隐退(retired)，与各个指令相关联的寄存器保存指示该指令所使用的每个寄存器的世代的描述符。因此，在允许如预期地执行软件的同时，多个指令使用同一寄存器处于命中进行。指令可被保存至直至其源寄存器中的每个寄存器的正确世代可用。即使指令使用同一目的地寄存器，也可执行指令。并且即使修改同一寄存器，也可引退指令，而不在程序执行时产生错误。通过主要与将正确值转发(forward)至正确指令有关的最小附加逻辑，处理器可在保存与架构可见寄存器相关联的寄存器堆的同时执行寄存器堆。例如，当引退指令时，可将目的地寄存器的值转发给需要具有该代的寄存器的命中进行指令。在实施例中，如果这也是命中进行寄存器的最后一代，则还可将还值写入寄存器堆中。

[0108] 每一代均允许处理器在使用同一架构寄存器的同时取出、排队、以及执行多个指令。这就减少或者消除了拖延执行管线等待先前指令的结果的需求。例如，程序员可能希望仅使用6个寄存器执行下列代码：

[0109] Inst 0:Add R0,R1,R2

[0110] Inst 1:Mul R1,R0,R0

[0111] Inst 2:Add R0,R4,R5

[0112] Inst 3:Mul R4,R0,R0

[0113] 在该代码中，程序员将值添加到R1和R2中并且将结果存储在R0中。然后，求该结果(存储在R0中)的平方并且将其存储在R1中。接着，将值添加到R4和R5中并且再次将结果存储在R0中。同样，求该结果的平方并且将其存储在R4中。

[0114] 通常，程序员必须要么拖延Inst 2直至Inst 1完成(或者当Inst 1执行时，冒着R0包含错误的值的危险)，要么使用程序员不可见的寄存器扩大该组可用寄存器并且移除因再利用寄存器R0而产生的读后写(WAR)故障。如果不完成此操作，则R0中的值可能不正确，从而将不期望的值存储在R1中。在每次遭遇故障时拖延机器可明显降低处理器的性能。增加存储架构寄存器和非程序员可见寄存器的寄存器堆的尺寸可缓解这个问题。但是，这需要增强转发逻辑和重命名逻辑，以跟踪哪些寄存器存储相关指令正在使用的架构寄存器的值。并且物理寄存器数目的增加使功耗和芯片区域增加。

[0115] 世代重命名允许处理器通过架构寄存器保持寄存器堆，同时甚至在这些指令指定使用同一寄存器时，允许处理器并行排队和执行指令。具体地，这允许两个指令改变同一寄存器的值以被同时入队、发布、以及执行，而不影响根据任一指令执行的软件的余下操作。每次将寄存器用作目的地时，其将被加标签(tag)有新世代号。在这之后使用该寄存器的每个指令将使用该新的世代，直至世代号再次增加。一旦执行初始指令并且确定该寄存器的值，则可更新使用该寄存器和该世代号的所有未决指令，以指示其具有该寄存器的有效值。

[0116] 返回上述实施例，我们将使用惯例Rn®m表示Rn第m代。对于上述代码片段，我们假定寄存器的世代信息为R0®0、R1®1、R2®1、R4®3、以及R5®0。在示例性实施方式中，当将指令排队时，世代号被分配如下：

[0117] Inst 0:AddR0®1, R1®1, R2®1(增加R0的世代,R1和R2保持不变)

[0118] Inst 1:MulR1®3, R0®1, R0®1(增加R1的世代,R0保持不变)

[0119] Inst 2:AddR0®2, R4®0, R5®1(增加R0的世代,R4和R5保持不变)

[0120] Inst 3:MulR4®4, R0®2, R0®2(增加R4的世代,R0保持不变)

[0121] 因此，世代重命名允许在不影响Inst 1的结果的情况下发布并且执行Inst 2。在上述实施例中，如果在Inst 1之前执行Inst 2，则将Inst 2的结果转发至使用R0®2的所有其他指令，而不影响使用R0的其他世代的指令。例如，一旦被执行，则该指令的结果可被等待这些结果的任何命中进行指令复制。如下所讨论，例如，如果我们具有两个执行单元，则可将我们指令中的每个排队到不同的执行单元。如果Add指令比Mul指令执行地更快，则我们可能具有类似如下的程序：

[0122] 时间1:将Inst 0放入队列0中;将Inst 1放入队列0中;将Inst 2放入队列1中;将Inst 3放入队列1中;

[0123] 时间2:执行Inst 0和Inst 2(不根据任一之前指令的结果)

[0124] 时间3:引退R0®1;适当地更新Inst 1和Inst 3;利用来自Inst 0和Inst 2的结果(假定对于该处理器可将来自Inst 2的结果转发至Inst 3)，执行Inst 1和Inst 3。)

[0125] 可替代地：

[0126] 时间1:将Inst 0放入队列0中;将Inst 1放入队列0中;将Inst 2放入队列1中;将Inst 3放入队列1中;

[0127] 时间2:执行Inst 0和Inst 2(不根据任一之前指令的结果)

[0128] 时间3:引退R0®1;适当地更新Inst 1;利用来自Inst 0的结果,开始执行Inst 1。

[0129] 时间4:引退R0®2;适当地更新Inst 3;利用来自Inst 2的结果,开始执行Inst 3。

[0130] 因此,世代重命名允许多个指令同时存在于处理器中。可乱序执行这些指令,而看似按顺序执行(即,乱序执行与按顺序执行的结果相同)。通过允许多个版本的各个寄存器存在于处理器中,世代重命名允许执行此操作,因此,指令可更新并且使用同一寄存器,而不妨碍彼此的执行。

[0131] 世代重命名结构

[0132] 参考图2,处理器的示例性实施方式包括寄存器堆202、重命名器204、预留队列208、执行单元210、以及解读机构212。

[0133] 寄存器堆202被配置为存储各个架构寄存器的值。例如,寄存器堆202可存储通用寄存器R0至R32、预测寄存器P0至P7、以及乘法累加(MAC)寄存器M0至M7的值。在实施方式中,寄存器堆202包括控制来自执行管线的结果将何时被写入寄存器堆的架构寄存器中的写控制线路。

[0134] 在实施方式中,处理器包括用于一个或者多个架构寄存器的重命名器204。重命名器204可包括命中进行阵列216、真值指针218、一个或者多个推测指令号(SIN)指针220、使能电路222、以及确认电路224。每个命中进行阵列216均指示处理器当前正在使用的寄存器的世代。例如,命中进行阵列216可指示处理器当前正在使用R0的第2代、第4代、以及第5代。

[0135] 因此,每个命中进行阵列216保存处理器中的指令当前正在使用的每个寄存器的所有世代的列表。命中进行阵列216可以是循环(circular)阵列。因此,在阵列中分配最后世代之后,例如,第7代,分配的下一世代将是0代。可以使用预留队列、加载/存储(L/S)队列、执行单元、解读机构等中的这些世代。

[0136] 为了确定哪一世代是使用中的最后一代,可使用各个架构寄存器的真值指针218。每个真值指针218均指向相关联的命中进行阵列216中代表该寄存器中作为指令目的地的最后一代的位置。例如,在上述代码段中,在取出和重命名Inst 3之后,R0的真值指针218将指向R0®2、R1的真值指针218将指向R1®3,并且R4的真值指针218将指向R4®4。可以若干种方式使用真值指针218。例如,在对指令进行解码的同时,真值指针218指向寄存器的用于指令来源的世代。对于目的地,可将该代设置为真值指针之后的下一代。

[0137] 在实施方式中,当引退结果时,可发生若干不同的事情。可以将该寄存器的世代值转发给所有当前位于预留队列中的使用该寄存器的该世代的指令并且设置这些寄存器的有效位。有效位允许处理器确定准备好发布执行哪些指令。在实施方式中,如果该值不具有推测性,即,该值不是在我们猜测其结果的条件分支指令之后生成的,则清除相关联命中进行阵列216中有关该代的命中进行位。

[0138] 使能电路222还利用引退的结果值、引退的结果世代号、以及真值指针218确定是

否应将该结果写入寄存器堆202中。真值指针218指向为非推测执行的寄存器的最后一代。即，真值指针218指向在处理器推测性地预测任何条件分支之前分配的寄存器的最后一代。如果引退的结果世代号与真值指针218匹配，则不存在写入该寄存器中的处于命中进行的任何非推测性指令。此时，可将该值写入寄存器堆202中。因此，使能电路222可将该寄存器的写入设置为启用，并且将引退的结果值写入寄存器堆202中。从而限制寄存器堆202中不必要的写入数目。

[0139] 除真值指针218之外，还将一个或者多个SIN指针220分配给各个寄存器。SIN指针220用于跟踪被分配给推测性地取出、排队、或者执行的指令的寄存器的世代。例如，下列代码片段从Inst 0中的内存中检索一个值并且执行Inst 1至6等于该检索值的次数。Inst 1至6占据从内存位置0x00200020开始的阵列并且使每种元素加倍：

[0140] Inst 0:LD R0,[0x10]

[0141] Inst 1:ADD R1,0x00200020

[0142] Inst 2:LD R2,[R1]

[0143] Inst 3:MUL R2,R2,0x2

[0144] Inst 4:ST R2,[R1]

[0145] Inst 5:SUB R0,R0,0x01

[0146] Inst 6:BNZ Inst 1

[0147] 当处理器到达Inst 6时，处理器必须做出决定，要么返回Inst 1要么继续执行下一连续指令。该决定基于R0中的值。该指令试图利用位于内存位置0x10中的值加载R0。从而使得从内存进行检索要花费一些时间，尤其是如果该值位于主内存中而非位于缓存中。从内存检索该值的同时，处理器可决定推测性地执行其他指令。可基于之前的历史确定分支回Inst1。

[0148] 由于从主内存检索信息需要时间，所以可多次推测性地执行该循环。尽管处理器获知来自第一执行的结果是正确地，然而，每个其他的迭代均将是推测性的，直至我们从内存接收到Inst 0请求的值。

[0149] 因此，对于Inst 6的每次推测执行，则分配新的SIN指针220。对于每个SIN指针220，则分配新版本的R1和R0(见Inst 1和Inst 5)，并且分配两个新版本的R2(见Inst 2和Inst 3)(存储指令-Inst 4-并不具有寄存器目的地)。在运行该循环4次之后，R1和R2看起来应该如下：

[0150] R1.真值指针=>R1®1

[0151] R1.SIN0指针=>R1®2

[0152] R1.SIN1指针=>R1®3

[0153] R1.SIN2指针=>R1®4

[0154] R1.SIN3指针=>R1®5

[0155] R1.命中进行阵列=011111000…0

[0156] R2.真值指针=>R2®0

[0157] R2.SIN0指针=>R2®2

[0158] R2.SIN1指针=>R2®4

[0159] R2.SIN2指针=>R2®6

[0160] R2.SIN3指针=>R2®8

[0161] R2.命中进行阵列=1111111110…0

[0162] 此时，如果完成对R0的加载并且其具有大于4的值，则确认推测执行指令中的每个，将这些寄存器中的每个的SIN3指针复制到真值指针中，并且在不推测的情况下继续执行。但是，如果该值小于4，例如，该值是2，则一次或者多次执行必须被展开(unwound)。将正确指针(例如，SIN1指针)复制到真值指针中，清除新的真值指针与最近分配的SIN指针(例如，SIN3指针)之间的所有命中进行位，例如，R1命中进行阵列位4至5与R2命中进行阵列位5至8。

[0163] 因为不将来自推测执行指令的结果加载到寄存器堆202中，所以不需要对寄存器堆202做出任何改变。

[0164] 在实施方式中，重命名器204还可确定寄存器堆中的值是否有效。当将指令解码并且将其添加到预留队列中的行列时，则将存储在寄存器堆202中的寄存器的值也复制到预留队列中。此外，设置指示该源寄存器值是否有效的有效位。确认电路224处理设置该位。如果真值指针(或者最近分配的SIN指针，如果我们推测性地执行该指令)指向的寄存器的世代为1，则清除该有效位，并且如果其是0，则设置该有效位。即，如果仍在计算寄存器的该代的值(即，仍处于命中进行)，则寄存器堆202中的值无效，但是，如果将该值引退，则寄存器堆202中的值有效。一旦设置所有的源寄存器有效位，则指令准备好发布执行。

[0165] 世代重命名过程

[0166] 在示例性实施方式中，处理器可被配置为执行世代重命名，而对VLIW指令进行解码。在另一实施方式中，可在对VLIW指令进行解码之后执行世代重命名。

[0167] 在步骤302，处理器取出VLIW指令。例如，处理器取出程序计数器(PC)指向的下一缓存线。该缓存线可包括由多个单独指令组成的VLIW指令。例如，根据实施细节，VLIW指令可由2个、4个、8个单独指令组成。本领域技术人员应当理解的是，在其他实施方式中，根据实施细节，VLIW指令可由其他数目的单独指令组成，例如，3个，

[0168] 在步骤304，可以识别用作VLIW指令中的来源和目的地的寄存器。如下面更细描述的，可将该信息提供给指令解码单元、寄存器堆、以及重命名器的其余部分。作为指令解码的一部分，可以识别与每个指令相关联的寄存器(即，源寄存器和目的地寄存器)。在另一实施方式中，可将寄存器识别为独立单元的一部分并且将其转发至指令解码单元、寄存器堆、以及重命名器。

[0169] 在步骤306，可以对VLIW指令中的一个或者多个指令进行解码。在一种实施方式中，这是指令解码单元的第二阶段的一部分，第一阶段包括识别寄存器。执行两步解码的一个优点是可在该周期的早期简单并且快速地执行寄存器识别，从而允许其他元件处理依赖于所识别寄存器的指令。

[0170] 在步骤308，可以访问寄存器堆，例如，寄存器堆202，并且可以识别VLIW指令中所使用的任何源寄存器的世代的值。此时，识别源寄存器中被写入寄存器堆中的最后一代。该值可以有效或者无效。因此，该值与VLIW指令推测关联。如下面讨论的，将设置有效位指示

该值是否是VLIW指令的源寄存器所需的世代的正确值。

[0171] 在步骤310，世代号与在步骤304中识别的寄存器号相关联。该世代号指示完成指令需要与寄存器号相关联的哪个值。例如，如果VLIW指令指示器将寄存器R0用作源寄存器，则处理器当前正在使用的R0中的最后一代可与VLIW指令相关联。其可以是当前处于命中进行并且远期某一时刻变得可用的R0中的世代。或者，其可以是之前计算并且可供使用的R0的世代。在另一实施例中，如果VLIW指令指示寄存器号是目的地寄存器，则R0中下一可用的非命中进行世代与该VLIW指令相关联。例如，如果第3代至第6代当前处于命中进行，则R0中的第7代将与VLIW指令相关联。

[0172] 在实施方式中，诸如上面讨论的真值指针或者SIN指针等指针可识别当前处于命中进行的寄存器的最后一代。例如，如果当前取出的指令为非推测性的指令(即，已经解析出所有之前的分支指令)，则可使用真值指针标识寄存器的最后有效世代。如果推测性地取出指令，则可使用SIN指针标识寄存器的最后有效世代。

[0173] 在实施方式中，可将源寄存器加标签，以指示源寄存器使用由VLIW指令产生的值。例如，第一单独指令可递增由第二单独指令检索的值，其中，第一单独指令和第二单独指令被包含在相同的VLIW指令中。在该实施方式中，为第一单独指令标识的源寄存器可包括VLIW指令中指示其使用来自VLIW指令中的另一单独指令的结果的标签。如果被这样加标签，而非与寄存器可用的最后命中进行世代相关联，则第一单独指令的源寄存器将与下一可用的非命中进行世代相关联。即，第二单独指令的目的地的同一代，从而指示在可用时可将第二单独指令的结果转发至第一单独指令。

[0174] 在步骤312，对于被识别为VLIW指令的源寄存器的被标识的寄存器，确定从寄存器堆检索的值是否有效。例如，可检查有关寄存器号的命中进行阵列。如果所使用的指向最后一代的指针指示该代当前不处于命中进行并且所使用的最后一代与VLIW指令的源寄存器的世代匹配，则可设置有效位。从寄存器堆检索的值是有关正在使用的源寄存器的世代的正确版本。在实施方式中，如果不存在任何SIN指针(即，不推测性地执行任何指令)和真值指针，则仅设置有效位。

[0175] 在步骤314，将各个指令添加到预留队列中。这包括添加被执行的指令的操作码、源寄存器号和目的地寄存器号、相关联的世代号、源寄存器的值，并且还将源有效位添加到预留队列中。

[0176] 用于加载/存储调度的资源锁定

[0177] 除使用相同的寄存器之外，程序中的指令通常指定使用相同的资源。资源可包括但不限于指定寄存器(例如，乘法累加(MAC)寄存器)或者内存(例如，内存阵列的每个库)。在实施方式中，可以同时访问这些类型资源中的一些或者全部。在实施方式中，存在与不同资源相关联的限制，例如，不能同时将两个指令写入MAC中，或者，如果将两个指令写入不同的地址或者同一地址中的不同位，则同时仅可将两个指令写入同一内存库中。

[0178] 在实施方式中，对于更新资源的指令，可以使用逻辑线路确定何时可以随时更新物理资源。对于更新MAC寄存器的指令，处理器的控制逻辑可评估更新MAC指令的其他未决指令。如果任何其他未决指令更新同一MAC寄存器，则需要在试图更新MAC寄存器之前拖延该指令。

[0179] 在实施方式中，每次取出和排队更新资源的指令时，可设置与更新资源的所有未

决指令相关的指令的状态信息。可以在处理器时钟周期期间保存该状态。

[0180] 在实施方式中，对于一个或者多个指令入队到其中的更新资源每个周期，处理器可识别具有资源独立性的任何指令。在不与任何其他未决或者执行指令冲突的情况下执行的指令具有资源独立性。在实施方式中，具有资源独立性的指令是并不更新与任何更早入队指令相同的资源的指令。例如，如果指令是MAC指令并且更新MAC3，并且如果也没有任何更早入队的指令更新MAC3，则处理器将该指令识别为具有资源独立性。否则，将该指令被标识为对与更早指令相同的资源具有相关性，直至清除所有的资源相关性。

[0181] 在实施方式中，处理器可标识具有资源独立性的所有指令。该列表代表了可能在该周期发布的所有指令。处理器则可从被发布的队列中选择一个或者多个指令。在实施方式中，处理器可基于指令的位置进行选择，例如，选择更晚指令之前的更早指令。在实施方式中，处理器可基于更新的资源数目进行选择，例如，选择更新更多资源的指令而非更新更少资源的指令(可能因为这些指令阻塞更多的指令)。处理器可基于指令执行的操作类型从该组资源独立的指令中选择准备好发布的指令。例如，处理器可允许根据处理器的设计发布额外的指令(例如，如果处理器可发布2个MAC指令，但是，仅一个预测更新指令，并且MAC和预测更新指令都是资源，则处理器可优选为发布MAC指令，而非预测更新指令)。

[0182] 在实施方式中，可以将加载/存储(L/S)单元划分成两个部分。一旦取出L/S指令，则将其放在L/S单元的第一部分中。该指令则保持位于L/S单元的第一部分中，直至准备好执行该指令。对于加载指令，一旦确定加载信息的内存地址，则准备好执行该指令。对于存储指令，一旦确定存储的值和存储该值的内存地址，则准备好执行该指令。

[0183] 一旦准备好执行L/S指令，则将其从L/S单元中移出并且放置在L/S单元的第二部分中的指令队列的首位。在实施方式中，L/S单元的第二部分中的每个指令均与可用于选择一个或者多个指令进行发布的其他状态信息相关联。

[0184] 在实施方式中，状态信息包括关于存储指令的操作数是否具有推测性的状态。存储指令改变内存。对内存的改变难以取消(undo)或者回退。因此，在一些实施方式中，不执行存储指令，并且不对内存做出改变，直至处理器确定可以使用正确的操作数执行存储指令。

[0185] 在实施方式中，状态信息可显示L/S指令与其他L/S指令的关系，具体地，是否可发布L/S指令、或者L/S单元中的一个或者多个其他L/S指令是否阻塞发布该L/S指令。例如，如果当前L/S单元中存在三个L/S指令，则当将新的L/S指令添加到L/S单元中时，新L/S指令的状态信息将包含有关该三个其他L/S指令中的每个的状态信息。

[0186] 在实施方式中，状态信息可包括显示是否准备好发布L/S指令。对于任何给定的周期，可准备好发布多个L/S指令。一旦对L/S单元中的所有L/S指令进行分析，则选择从L/S指令中发布指示其已准备好发布的一个或者多个L/S指令。在实施方式中，可选择最早的准备好发布的L/S指令。在另一实施方式中，在准备好发布的存储指令之前可选择准备好发布加载指令。根据处理器与系统设计的平衡，可使用选择和发布L/S指令的其他方法。

[0187] 在实施方式中，状态信息可显示两个L/S指令的地址之间的关系。在实施方式中，状态信息可显示四种状态---独立、相互排斥、可合并、以及冲突。在上述所述划分的L/S单元实施方式中，因为将每个L/S指令从L/S单元的第一部分移至L/S单元的第二部分，所以可生成器与当前L/S单元中的每个其他指令相关的状态信息。可以保存生成的状态信息，直至

发布L/S指令，并且可在发布其他L/S指令之前对其进行更新。

[0188] 在实施方式中，如果将L/S指令标记为与更早的指令独立，则指如果两个指令就绪，则可发布该两个指令。在实施方式中，如果执行的每个指令均访问不同的内存资源，则可将指令标记为与内存资源中的另一指令独立。例如，如果内存阵列具有4个库，其中每个均可以被读入或者写入、独立于其他内存库，则可将被读入或者写入不同内存库中的两个指令标记为独立。

[0189] 在实施方式中，如果将L/S指令标记为与更早的指令相互排斥，则指如果两个指令就绪，则可以发布任一指令，但是，在同一周期中，不可同时发布两个指令。因此，如果在同一周期中，两个指令准备好发布，则可发布更晚的一个，或可发布更早的一个，但是，不能同时发布两个指令。在上述实施例中，内存阵列具有四个独立的内存库，如果执行的每个指令被读出或者写入同一库中，但是是该库内不同索引，则任一指令可在就绪时执行，但是，因为在一个单周期中，仅可写入或者读出每个内存库一次，所以不能一起执行两个指令。

[0190] 在实施方式中，如果将L/S指令标记为与更早指令可合并，则指更晚指令可与更早指令一起或者在更早指令之后执行，但不是在之前执行。因此，如果准备好发布更晚指令，则可在与更早指令相同的周期中或者之后的任一周期中发布更晚指令，但是，如果更早指令未发布或者未被选择发布，则不能发布更晚指令。例如，内存阵列具有四个独立的内存库，如果两个指令皆是存储指令并且每次执行写入一个或者多个相同的内存位置(例如，如果两个指令皆具有要写入的重叠字节)，则将更晚指令标记为可合并。在该实施例中，如果能够发布更早指令而非更晚指令，则不存在任何问题。但是，更晚指令必须等待执行更早指令，否则重叠字节可能以错误的值结束(将在未来发布的更早指令的值，而非准备好发布的更晚指令的值)。并且如果两个指令都准备好发布，则可同时发布两个指令。可将重叠字节的值设置成将更晚指令存储在该位置中的值。

[0191] 在实施方式中，如果将L/S指令标记为与其他指令冲突，则指必须在更早指令之后发布更晚指令。例如，在上述实施例中，内存阵列具有四个独立的内存库，如果更早指令是存储指令并且更晚指令是加载指令，并且两个指令皆访问内存中的至少一个相同的位置，则更晚指令必须等待在其自身发布之前发布更早指令。如果在更早指令之前发布更晚指令，则将对于任何重叠的位置均检索错误值。

[0192] 下面是在L/S单元中如何跟踪L/S指令及其相关联的状态指示的实施例。下面实施例使用了简化内存结构，以示出上述条件和关系。如下所示，该实施例使用了上述所述使用的四个库内存实施例，其中，每个库均包括多个索引(即，行)。

[0193]

	B0	B1	B2	B3
索引 0				
索引 1				
索引 2				

[0194]

索引 3				
索引 4				
索引 5				
索引 6				
索引 7				

[0195] 在该实施例中，在时间1，将存储指令移至L/S单元。该存储指令在索引3将数据存储到库0和1中。因为在该实施例中，此时L/S单元中不存在任何其他指令，所以其状态是空(clear)的。此外，预备状态是空的，指示该指令是当前推测的。在下面实施例中，我们将使用下列惯例方法描述L/S指令——St[R0]，$R1_{I\ 3,B0/1}$。在该常规方法中，“St”是这样一种指令，即，“St”用于存储或者“Ld”用于加载。“[R0]”(即，第一操作数)是目的地。在该实施例中，“[ ]”指示该目的地是R0指向的地址。例如，如果R0是0x00010000，则该指令最终将值存储到内存的地址0x00010000中。“R1”(即，第二操作数)是来源。在该实施例中，这是将被存储在R0指向的地址中的值。

[0196]

时间 1：

$St\ [R0], R1_{I\ 3, B0/1}$

就绪		0	1	2	3	发布
	0.St					

[0197] 在时间2，将第二存储指令移至L/S单元。该存储指令也将数据存储到库0中，不过是在索引1。然后，处理器更新与所有现有指令有关的指令的状态。该指令并不是推测的，所以其准备好对被评估。因为该指令被写入与第一存储指令相同的内存库中，但是被写入不同的索引中，所以指令相互排斥。任一指令均可在相对于另一个的任一时间执行，但是，这些指令不能同时执行。

[0198] 时间2：

[0199]

$St\ [R2], R3_{I\ 1, B0/0}$

$St\ [R0], R1_{I\ 3, B0/1}$

就绪		0	1	2	3	发布
Y	1.St	ME				
	0.St					

[0200] 在时间3，将加载指令移至L/S单元。该加载指令从库1中在索引3检索信息。该加载指令访问的内存的至少一部分与由第一存储指令写入至的内存匹配。当将该指令移至L/S单元时，处理器更新该指令相对于所有现有指令的状态。加载指令始终准备好被评估。因为该指令从与第一存储指令相同的内存库读出，具有同一索引，并且存在内存重叠，所以指令发生冲突。任一指令均可在相对于另一个的任意时间内执行，则这些指令不能同时执行。因为将该指令写入不同于第二指令的库中，所以这两个指令彼此独立。因此，状态看上去如下：

[0201] 时间3：

[0202]

Ld R4, [R5]I 3, B1/1 (匹配St R0)
St [R2], R3I 1, B0/0
St [R0], R1I 3, B0/1

就绪		0	1	2	3	发布
Y	2.Ld	C	I			
Y	1.St	ME				
	0.St					

[0203] 在时间4，将第三存储指令移至L/S单元。该存储指令在索引3将信息写入库1中。该存储指令中的地址的任何部分均不与之前的加载指令重叠。当将该指令移至L/S单元时，处理器更新该指令相对于所有现有指令的状态。该存储指令不具有推测性，因此该存储指令已准备好被评估。因为该指令写入与第一存储指令相同的内存库和索引中，所以可以将指令合并。该指令可在与第一存储指令相同的时间内或者在之后的任一时间执行。将该指令写入不同于第二指令的库中，因此，各指令彼此独立。由于通过同一索引将该指令写入与加载指令相同的内存库中，但是，不存在任何内存重叠，所以该指令相互排斥。

[0204] 时间4：

[0205]

St [R6], R7I 3, B1/1 (不匹配Ld R4)
Ld R4, [R5]I 3, B1/1 (匹配St R0)
St [R2], R3I 1, B0/0
St [R0], R1I 3, B0/1

就绪：		0	1	2	3	发布
Y	3.St	M	I	ME		
Y	2.Ld	C	I			
Y	1.St	ME				
	0.St					

[0206] 在时间5，将第二加载指令移至L/S单元。该存储指令在索引3从库1中检索信息。该加载指令中的地址的任何部分均不与第一存储指令重叠，但是，多个部分与第三存储指令重叠。当将该指令移至L/S单元中时，处理器更新该指令相对于所有现有指令的状态。加载指令一直准备好被评估。因为通过同一索引从与第一存储指令相同的内存库读取指令，但是，不存在任何内存重叠，所以这两个指令相互排斥。而该指令从不同于第二存储指令的库中加载，所以这两个指令彼此独立。从同一索引和同一库中读取第一存储指令和第二存储指令。因此，各指令彼此独立。一个指令在另一个之前执行并没有任何阻碍。因为通过同一索引将该指令读入与第三存储指令相同的内存库中，并且存在内存重叠，所以这两个指令相互冲突。

[0207] 时间5：

[0208]

Ld R8, [R9]I 3, B1/1 (不匹配 St R0, 匹配 St R6)

St [R6], R7I 3, B1/1 (不匹配 Ld R4)

Ld R4, [R5]I 3, B1/1 (匹配 St R0)

St [R2], R3I 1, B0/0

St [R0], R1I 3, B0/1

就绪		0	1	2	3	发布
Y	4.Ld	ME	I	I	C	
Y	3.St	M	I	ME		
Y	2.Ld	C	I			
Y	1.St	ME				
	0.St					

[0209] 在实施方式中，一旦内存变得可用，例如，当系统总线可用时，处理器可对L/S单元中的未决L/S指令进行分析，以确定准备发布哪些指令。在实施方式中，分析从最更早指令至最更晚指令的L/S指令，并且将每个指令标识为已准备好发布或者未准备好发布。在实施方式中，准备好发布可包括关于L/S单元中的其他指令的发布状态的额外信息(例如，准备好发布，准备好与Inst X一起发布，或者如果Inst X未发布，则准备好发布)。这对于可合并和相互排斥的指令比较重要。

[0210] 继续上述实施例，在时间6，如果处理器能够访问内存，则其必须确定其可以执行哪一L/S指令或者哪些L/S指令。因此，处理器开始评估就绪指令。在时间6开始时，L/S单元看上去如下：

[0211] 时间6：

[0212]

Ld R8, [R9]I 3, B1/1 (不匹配 St R0, 匹配 St R6)

St [R6], R7I 3, B1/1 (不匹配 Ld R4)

Ld R4, [R5]I 3, B1/1 (匹配 St R0)

St [R2], R3I 1, B0/0

St [R0], R1I 3, B0/1

就绪		0	1	2	3	发布
Y	4.Ld	ME	I	I	C	
Y	3.St	M	I	ME		
Y	2.Ld	C	I			
Y	1.St	ME				
	0.St					

[0213] 在实施方式中，从最早指令至最晚指令对第二L/S单元中准备好被评估的指令进行评估。因为第一存储指令未准备好被评估，所以不可发布第一存储指令。处理器能够临时修改与第一指令相互排斥的所有指令，以指示其现在与第一存储指令独立。因此，假定没有任何其他指令阻塞它们，第二存储指令和第二加载指令现在可发布。

[0214] 接着，对第二存储指令进行评估。第二存储准备好被评估。第二存储相对于第一存储相互排斥。因为并未发布第一存储，所以该状态可临时被修改成独立。对状态的修改是临时的，当该周期结束时，则退回所有状态改变。因为没有待评估的任何其他状态确定，所以

可将第二存储标记为准备好发布。

[0215] 时间6：

[0216]

Ld R8, [R9]I 3, B1/1 (不匹配 St R0, 匹配 St R6)

St [R6], R7I 3, B1/1 (不匹配 Ld R4)

Ld R4, [R5]I 3, B1/1 (匹配 St R0)

St [R2], R3I 1, B0/0

St [R0], R1I 3, B0/1

就绪		0	1	2	3	发布
Y	4.Ld	ME> I	I	I	C	
Y	3.St	M	I	ME		
Y	2.Ld	C	I			
Y	1.St	ME> I				Y
	0.St					

[0217] 接着，对第一加载指令进行评估。准备好对第一加载指令进行评估。第一加载指令与第一存储相互冲突。因为第一存储尚未被发布，所以不能发布该加载指令。第一加载与第二存储指令独立。但是，直至发布第一存储指令之后，才可发布第一加载指令。因此，不准备发布第一加载指令。

[0218] 接着，对第三存储指令进行评估。准备好对第三存储指令进行评估。第三存储指令可与第一存储指令合并。因此，第三存储指令可与第一存储指令一起发布或者在第一存储指令之后发布，但是，因为第一存储指令尚未被发布，并且在该周期中不被发布，所以也不能发布第三存储指令。即使第三存储指令与第二存储指令独立并且与未准备好被发布的第一加载指令相互排斥，情况也是这样。因此，并未准备好发布第三存储指令。

[0219] 时间6：

[0220]

Ld R8, [R9]I 3, B1/1 (不匹配 St R0, 匹配 St R6)

St [R6], R7I 3, B1/1 (不匹配 Ld R4)

Ld R4, [R5]I 3, B1/1 (匹配 St R0)

St [R2], R3I 1, B0/0

St [R0], R1I 3, B0/1

就绪		0	1	2	3	发布
Y	4.Ld	ME> I	I	I	C	
Y	3.St	M	I	ME> I		
Y	2.Ld	C	I			
Y	1.St	ME> I				Y
	0.St					

[0221] 接着，对第二加载指令进行评估。已准备好对第二加载指令进行评估。第二加载指令与第一存储指令相互排斥。未准备好发布第一存储指令，因此，不存在与第一存储指令的任何冲突。第二加载指令可与第二存储指令和第一加载指令独立执行，因此，也不与这些指令中的任一个存在冲突。但是，第二加载指令从与第三存储指令写入的相同的索引和相同

的库中读取，并且该加载中不存在与该存储重叠的部分，因此，各指令相互冲突。直至第三存储指令之后，才可发布第二加载指令。

[0222] 接着，在完成评估之后，处理器可选择指令进行发布。在以上这种情况下，准备好发布的唯一指令是第二存储指令，因此，发布第二存储指令。

[0223] 继续上述实施例，如果在时间7，处理器能够访问内存，则其必须确定其将执行哪一L/S指令或者哪些L/S指令。此时，第一存储指令不再具有推测性并且已准备好被评估。在时间6开始时，L/S单元看上去如下：

[0224] 时间7：

[0225]

Ld R8, [R9]I-3, T1/1 (不匹配 St R0, 匹配 St R6)

St [R6], R7I-3, T1/1 (不匹配 Ld R4)

Ld R4, [R5]I-3, T1/1 (匹配 St R0)

St [R0], R1I-3, T0/1

就绪		0	1	2	3	发布
Y	3.Ld	ME	I	C		
Y	2.St	M	ME			
Y	1.Ld	C				
Y	0.St					

[0226] 此时，已准备评估第一存储指令。因为不存在与其冲突的更早指令，所以已准备好发布第一存储指令。处理器能够临时修改可与第一指令合并的所有指令，以指示所有指令现在与第一存储指令独立，即，如果不存在其他冲突，所有指令可与第一指令一起执行。因此，第二存储指令被修改为指示器与第一存储指令独立。

[0227] 时间7：

[0228]

Ld R8, [R9]I-3, T1/1 (不匹配 St R0, 匹配 St R6)

St [R6], R7I-3, T1/1 (不匹配 Ld R4)

Ld R4, [R5]I-3, T1/1 (匹配 St R0)

St [R0], R1I-3, T0/1

就绪		0	1	2	3	发布
Y	3.Ld	ME	I	C		
Y	2.St	M>I	ME			
Y	1.Ld	C				
Y	0.St					Y

[0229] 接着，准备好评估第一加载指令。因为如果与第一存储指令冲突，则直至发布第一存储指令之后，才可发布第一加载指令。因此，为准备好发布第一加载指令。处理器能够临时修改与第一加载指令相互排斥的所有指令，以指示其现在与第一存储指令独立，即，如果不存在任何其他冲突，由于第一加载指令并未执行，因此可执行这些指令。因此，第二存储指令被修改为指示其与第一加载指令独立。

[0230] 接着，准备好评估第二存储指令。由于上述所述状态变化，假定第一存储指令发

布，那么可发布第二存储机，这是因为该状态初始指示他们是可合并的，即，可在第一存储指令之后或者与第一存储指令一起发布第二存储。

[0231] 最后，准备好评估第二加载指令。因为第二加载指令与第一指令相互排斥并且第一指令准备好发布，所以不能发布第二加载指令。此外，第二加载指令与第二存储指令冲突，所以直至发布第二存储指令之后，才可发布第二加载指令。

[0232] 此时，已经评估了L/S队列中的所有未决指令。准备好发布两个存储指令。此时，根据内存系统的实施方式，可执行一个或者多个L/S指令。例如，内存系统可被设计成每周期仅处理一个L/S指令、每周期处理多个L/S指令、或者加载指令与存储指令的某种组合(例如，每周期处理1个加载指令和2个存储指令)。

[0233] 时间7：

[0234]

Ld R8, [R9]I-3, T1/1 (不匹配 St R0, 匹配 St R6)

St [R6], R7I-3, T1/1 (不匹配 Ld R4)

Ld R4, [R5]I-3, T1/1 (匹配 St R0)

St [R0], R1I-3, T0/1

就绪		0	1	2	3	发布
Y	3.Ld	ME	I	C		
Y	2.St	M>I	ME>I			Y
Y	1.Ld	C				
Y	0.St					Y

[0235] 在示出性实施例中，内存系统被设计成每周期处理2个L/S指令。这可以是2个加载指令、2个存储指令、或者1个加载指令和1个存储指令。在标识两个存储指令可以发布之后，处理器可以在该周期发布两个指令。在实施方式中，一旦发布指令，则可以从L/S单元中移除该指令，将与该指令有关的所有依赖性改变成独立，并且保持状态表的其余部分不变。在实施方式中，状态表可保持不变，直至L/S指令被放入其最后条目中，此时，可巩固该表格，以移除已经发布的条目。在实施方式中，一旦发布指令，不仅移除了该指令，而且表格中所有其余的条目也相应地移位。

[0236] 在该实施例中，每次发布L/S指令时，状态表则被折叠(collapse)。在时间8开始时，已经发布了存储指令，状态表看上去如下：

[0237] 时间8：

[0238]

Ld R8, [R9]I-3, T1/1

Ld R4, [R5]I-3, T1/1

就绪		0	1	2	3	发布
Y	1.Ld	I				
Y	0.Ld					

[0239] 如上所述，此时，如果准备好评估额外的L/S指令，则可将其移至L/S单元，并且更新状态表。如果处理器能够访问内存，则可发布这两个加载指令(即，同时准备好发布该两个指令，并且第二加载可独立于第一加载指令而执行，因此，同时准备好发布第二加载和第一加载指令)。

[0240] 资源锁定结构

[0241] 参考图4，加载/存储(L/S)队列的示例性实施方式包括L/S预留队列1 402和L/S预留队列2 404。

[0242] L/S预留队列1 402可以是上述所述L/S单元的第一部分。当取出L/S指令时，首先，将L/S指令放置在L/S预留队列1 402中。一旦解析出L/S指令的操作数，则可以将其放置在L/S预留队列2 404中。加载指令从内存中的某一位置取出数据并且将其加载到目的地，通常是寄存器。因此，对于加载指令，所有需要解析出的是内存的位置。对于存储指令，将通常存储在寄存器中的值存储到内存的某一位置中。因此，对于被移至L/S预留队列2 404中的存储指令，必须解析出被存储的值和将该值存储在其中的位置。

[0243] L/S预留队列2 404可以是上述所述L/S单元的第二部分。L/S预留队列2 404包括状态选择逻辑406、状态更新和发布确定逻辑408、以及L/S指令状态队列410。如上所述，当将L/S指令接收到L/S预留队列2 404中时，将其放置在L/S指令状态队列410中。

[0244] 在实施方式中，当将L/S指令放置在L/S指令状态队列410中时，状态选择逻辑406基于L/S预留队列2 404中当前未决的所有其他L/S指令确定该指令的初始状态。如上所述，在实施方式中，状态选择逻辑406设置每个L/S指令的初始状态。状态选择逻辑406指示存储指令知否具有推测性。对于不具有推测性的这些存储指令，状态选择逻辑406将其放入L/S指令状态队列410中，但是，将数据就绪状态设置成“否”。对于所有其他L/S指令，状态选择逻辑406则将数据就绪状态设置成“是”。如上所述，状态选择逻辑406还设置初始指令冲突状态。该状态允许状态更新和发布确定逻辑408可基于由之前L/S指令读出和写入的内存确定是否发布指令。

[0245] 在实施方式中，状态更新和发布确定逻辑408确定是否准备发布指令、是否选择被发布的指令、以及当发布指令时是否更新L/S指令状态队列410。如上所述，当L/S指令的操作数就绪时，状态更新和发布确定逻辑408对指令冲突状态和之前L/S指令的发布状态进行分析，以确定是否准备发布L/S指令。一旦对L/S预留队列2 404中的所有指令分析完毕，状态更新和发布确定逻辑408则选择待发布的指令。可优选为发布更早的指令而非新指令、发布存储指令而非加载指令、发布从内存中的特定区域读取或者被写入内存中的特定区域中的指令等。

[0246] 在实施方式中，L/S指令状态队列410将所有L/S指令的状态保存在L/S预留队列2 404中。在实施方式中，L/S指令状态队列410包括指示是否可执行该指令的数据就绪指示器。例如，不可发布推测性取出的存储指令。由于推测性取出的存储指令影响内存，所以不能容易回退已发布的存储指令。因此，将这些指令保存在L/S指令状态队列410中，直至其不再具有推测性。在实施方式中，L/S指令状态队列410包括内存指令自身或者指向该内存指令的指针。从而允许处理器在其就绪时发布该指令。在实施方式中，L/S指令状态队列410包括保存L/S指令之间的关系的指令冲突状态。在实施方式中，一旦设置任何给定指令的指令冲突状态，则可将其保存，直至发布该指令。因此，并不需要在处理器每次访问内存时进行重新计算。在实施方式中，L/S指令状态队列410包括可发布指令。该指示允许L/S预留队列2 404跟踪每个周期可发布的指令。因此，一旦处理器访问内存，则其可选择执行一个或者多个L/S指令。

[0247] 状态选择过程

[0248] 在示例性实施方式中，L/S单元可被配置为确定新的L/S指令与所有未决L/S指令之间的关系。

[0249] 图5中示出的过程描述了L/S单元如何比较新的L/S指令与未决L/S指令。

[0250] 在步骤502，L/S单元确定是否将新的L/S指令写入与未决L/S指令相同的块中。

[0251] 如上所述，如果否，则各指令相互独立，并且该过程移至步骤504。在步骤504，L/S单元将与新的L/S指令和未决L/S指令有关的状态设置成“相互独立”并且继续至步骤520。

[0252] 如果将新的L/S指令与未决L/S指令写入同一块中，则该过程移至步骤506。在步骤506，L/S单元确定是否将新的L/S指令写入至与未决L/S指令相同的索引中。

[0253] 如上所述，如果否，则各指令相互排斥，并且该过程移至步骤508。在步骤508，L/S单元将与新L/S指令和未决L/S指令有关的状态设置成“相互排斥”并且继续至步骤520。

[0254] 如果将新的L/S指令与未决L/S指令写入同一索引中，则该过程移至步骤510。在步骤510，L/S单元确定新的L/S指令与未决L/S指令是否属于同一类型的操作，即，其是否皆是加载指令或者皆是存储指令。

[0255] 如上所述，如果是，则将各指令合并，并且该过程移至步骤512。在步骤512，L/S单元将与新的L/S指令和未决L/S指令有关的状态设置成“可合并”并且继续至步骤520。

[0256] 如果新的L/S指令与未决L/S指令属于不同类型的操作，即，一个是加载指令，另一个是存储指令，则该过程移至步骤514。在步骤514，L/S单元确定新的L/S指令和未决L/S指令是否访问至少一个重叠字节，即，两个操作皆访问内存中的存储的相同位。

[0257] 如上所述，如果否，则两个指令相互排斥，并且该过程移至步骤518。在步骤518，L/S单元将与新L/S指令和未决L/S指令有关的状态设置成“相互排斥”并且继续至步骤520。

[0258] 如上所述，如果是，则各指令相互冲突，并且该过程移至步骤518。在步骤518，L/S单元将与新的L/S指令和未决L/S指令有关的状态设置成“冲突”并且继续步骤520。

[0259] 在步骤520，选择下一未决L/S指令，该该过程可再次开始确定与新L/S指令和新选择的未决L/S指令有关的状态。

[0260] 状态更新和发布过程

[0261] 在示例性实施方式中，L/S单元可被配置为更新未决L/S指令之间的关系并且标识准备好发布的L/S指令。

[0262] 图6中示出的过程描述了L/S单元如何更新状态关系并且标识准备好发布的指令。

[0263] 在步骤602，L/S单元识别最早的未评估L/S指令并且开始对其进行评估。

[0264] 在步骤604，L/S单元确定是否准备好发布该指令。如果L/S指令是仍具有推测性的存储指令，则未准备好发布该指令。如果与L/S指令相关联的任何指令冲突状态当前未被设置成“独立”，则未准备好发布L/S指令。否则，准备好发布该L/S指令。

[0265] 如果准备好发布L/S指令，则该过程移至步骤606。在步骤606，标识与该指令有关的所有更晚指令。如果当前将任一更晚指令状态设置成“合并”，则将该状态临时设置成“独立”。这是因为准备好发布当前L/S指令，并且更晚L/S指令指示如果发布当前L/S指令则可发布该更晚的L/S指令。

[0266] 在步骤608，将当前L/S指令标记为准备好发布。该过程则继续至步骤612。

[0267] 如果未准备好发布当前L/S指令，则该过程移至步骤610。在步骤610，标识与该指令有关的所有更晚指令。如果任一更晚指令状态当前设置成“相互排斥”，则将该状态临时

设置成“独立”。这是因为未准备好发布当前L/S指令，并且更晚L/S指令指示如果不发布当前L/S指令则可发布该更晚的L/S指令。该过程则继续至步骤612。

[0268] 在步骤612，如果存在被评估的任何其他L/S指令，则该过程返回至步骤602，以对其他L/S指令进行评估。否则，该过程继续至步骤614。

[0269] 在步骤614，选择准备好被发布的L/S指令。在实施方式中，基于L/S指令的年龄(age)进行选择，例如，首先选择更早的指令。在实施方式中，基于L/S指令的类型进行选择，例如，优选发布存储指令而非加载指令。在实施方式中，可基于处理器的设计选择指令，例如，包括加载和存储带宽。本领域技术人员应当理解的是，可以使用其他选择方法。

[0270] 在步骤616，发布所选择的L/S指令。

[0271] 在步骤618，将L/S单元中的所有指令的指令冲突状态重置回该过程开始之前的原设置，并且该过程在步骤620结束。

[0272] 解读机构

[0273] 在标量处理器设计中，按顺序取出、解码、执行、并且引退各个指令。当被引退时，所执行指令的结果更新远期指令所使用的寄存器堆。可以在对另一指令解码、执行另一指令、并且引退另一指令的同时取出指令。处理器中的这些部分中的每个均可被称为一个阶段。然后，通过指令完成某一阶段所需的最长时间可以估测处理器的速度。因此，如果执行阶段是最长阶段，并且需要1秒钟完成该阶段，则通过需要运行程序的指令数目可以估测程序的执行时间，即，如果该程序是10个指令长，则完成该程序的估测时间将为1分钟。因此，为了增加处理器可以运行的速度，可以将这些阶段中的一个或者多个拆分成更小的部分。但是，因为特定指令可能需要在其被执行之前等待完成其他指令，所以这种估测可能不正确。因此，例如，可以在执行阶段结束时将来自一个阶段的指令的结果转发给其他阶段。

[0274] 超标量处理器更进一步考虑了这些概念。并非仅取出、解码、执行、以及引退一个指令，超标量处理器实现了允许其同时处理多个指令的多个并行管线。因此，超标量处理器可被设计成同时取出、解码、执行、以及引退2个、4个、8个、或者多个指令。但是，超标量处理器可实现的管线数目受处理器的设计需求限制。例如，将大多数指令的结果存储在程序中的寄存器堆仅能够被设计成处理有限数量的写给定空间和布线限制。

[0275] 解读机构提供一种在不需要寄存器堆接收新数据的能力等量提高的情况下而增加并行管线的数目的方式。如图1所示，解读机构接收来自所有执行管线的结果。解读机构还能将一个或者多个结果提供给机器的其他部分。保存从执行管线接收并且未提供给机器其他部分的结果，以在之后将其提供给机器。从而获得许多益处。首先，允许处理器具有比在单一时钟周期中引退的结果数目更多的并行管线。其次，允许处理器在由于其不能够以其创建结果的速度引退结果而必须拖延之前的在更长时间段内继续以全部的带宽执行。例如，如果解读机构可持有20个额外的结果，则处理器可继续运行程序，其中，该程序的特定部分的执行力较强，并且减少程序由于引退拥堵而拖延的时间。当创建比引退更多的结果时，可在不使机器变慢的情况下，将额外结果存储在解读机构中。当创建比引退更少的结果时，处理器可在不使机器变慢的情况下清空解读机构。从下面实施例中可以看出：

[0276] 执行的代码：

[0277] 4个加法指令

[0278] 4个存储指令

[0279] 4个加法指令

[0280] 4个存储指令

[0281]

	不具有解读机构的处理器每个周期可取出 4 个指令并且引退 2 个结果		具有解读机构的处理器每个周期可取出 4 个指令并且引退 2 个结果		
周期	引退的指令	引退结果	引退的指令	引退结果	存储器中的结果
1	2 加法	2	4 加法	2	2
2	2 加法	2	4 存储	2	0
3	4 存储	0	4 加法	2	2
4	2 加法	2	4 存储	2	0
5	2 加法	2			
6	4 存储	0			

[0282] 在上述实施例中，其中，两个处理器每个周期皆可取出4个指令并且引退2个指令，具有解读机构的处理器可更快地完成代码段。出于简便之目的，我们将假定上述所有指令相互独立并且并不致使处理器中发生其他拖延。因为不具有解读机构的处理器每个周期仅可引退2个结果，所以不具有解读机构的处理器一次仅可执行2个加法指令。具有解读机构的处理器每次可执行所有的4个加法指令。该处理器可直接引退2个结果，并且在存储被执行的指令的同时引退这两个其余的结果。下面是更为清晰地示出了使用解读机构的实施例的更长实施例。即使可以取出4个加法指令，然而，因为在任一时钟周期内仅可引退2个结果，所以仅可引退2个加法指令。

[0283] 执行的代码：

[0284] 4个加法指令

[0285] 4个加法指令

[0286] 4个加法指令

[0287] 4个加法指令

[0288] 4个加法指令

[0289] 4个加法指令

[0290] 4个存储指令

[0291] 4个存储指令

[0292] 4个存储指令

[0293] 4个存储指令

[0294] 4个存储指令

[0295] 4个存储指令

[0296]

	不具有解读机构的处理器每个周期可取出 4 个指令并且引退 2 个结果		具有解读机构的处理器每个周期可取出 4 个指令并且引退 2 个结果		
周期	被引退的指令	引退结果	引退的指令	引退结果	存储器中的结果
1	2 加法	2	4 加法	2	2
2	2 加法	2	4 加法	2	4
3	2 加法	2	4 加法	2	6
4	2 加法	2	4 加法	2	8
5	2 加法	2	4 加法	2	10
6	2 加法	2	4 加法	2	12
7	2 加法	2	4 存储	2	10
8	2 加法	2	4 存储	2	8
9	2 加法	2	4 存储	2	6
10	2 加法	2	4 存储	2	4
11	2 加法	2	4 存储	2	2

[0297]

12	2 加法	2	4 存储	2	0
13	4 存储	0			
14	4 存储	0			
15	4 存储	0			
16	4 存储	0			
17	4 存储	0			
18	4 存储	0			

[0298] 解读机构结构

[0299] 图7和图8示出了解读机构的示例性实施方式。

[0300] 参考图7，解读机构的示例性实施方式可包括仲裁器702、临时存储器704、解读机构多路复用器706、以及存储器708。

[0301] 在实施方式中，将来自执行单元的结果(例如，图7中的结果1-N)发布给仲裁器702和临时存储器704。

[0302] 在实施方式中，当接收来自下一周期的新结果时，临时存储器704存储来自处理器的一个周期的之前结果。来自临时存储器704的之前结果将被提供至解读机构多路复用器706。

[0303] 在实施方式中，解读机构多路复用器706接收来自临时存储器704的之前结果并且将非引退的结果发送至存储器708。如下所述，当仲裁器确定将要引退哪些结果时，仲裁器选择应将来自临时存储器704的哪些结果移至存储器708中。然后，仲裁器可更新解读机构多路复用器706的选择线，以确保将正确结果发送至存储器708。

[0304] 在实施方式中，存储器708接收来自解读机构多路复用器706的选择结果并且将一组结果提供给仲裁器702。存储器708被配置为存储尚未被引退的结果。

[0305] 如上所述，因为在给定时间内可以生成比引退更多的结果，所以需要存储这些结果。此外，这些结果可以具有推测性，并且直至处理器确认相关联的指令被认为已经取出，才可以引退这些结果。例如，当遇到条件分支指令时，处理器可推测已选择该分支并且从分支指令所指向的位置取出指令。但是，直至该分支设定的条件已被确定，处理器才可引退这些指令。一旦经过确认，则可沿着推测路径引退指令的结果。但是，如果该推测不正确，则在不引退这些结果的情况下，可以移除这些结果。

[0306] 在实施方式中，由仲裁器702接收执行单元、解读机构多路复用器706、以及存储器708的结果。仲裁器702可被配置为选择应被引退的结果。例如，仲裁器可选择在引退来自解读机构多路复用器706的结果之前引退存储器708中的结果并且在引退来自执行单元的结果之前引退来自解读机构多路复用器706的结果。这可能是因为仲裁器优选引退更早的结果而非新的结果。此外，仲裁器可基于可用的引退资源和结果交互确定应引退哪些结果。例如，解读机构可被配置为每个周期引退两个结果。此外，解读机构可被配置成使得这些结果中的一个可以是较大的结果(例如，128位)并且一个可以是较小的结果(例如，64位)。在实施方式中，基于寄存器堆的设计，128位可包括两个连续的寄存器。例如，寄存器堆可被配置成使得可使用单一较大的输出在同一周期中对寄存器R0和R1(其中每个均是64位)进行写入。这对于所有的偶数/奇数对的连续寄存器皆是如此，例如，R0/R1，R2/R3，R4/R5等。因此，即使R0被存储在存储器708中并且从执行单元接收R1，仲裁器也可选择在较大的输出中组合R0和R1。

[0307] 本领域技术人员应当理解的是，在不背离本公开的情况下，根据处理器的设计，可以做出多种其他变形。

[0308] 参考图8，解读机构的示例性实施方式可包括仲裁器802、临时存储器804、解读机构多路复用器806、存储器808、推测性存储器810、以及等待时间结果仲裁器812。仲裁器802、临时存储器804、解读机构多路复用器806、以及存储器808可执行与仲裁器702、临时存储器704、解读机构多路复用器706、以及存储器708相似的操作，但下面所示除外。

[0309] 在实施方式中，可将其他推测性存储器810包括在解读机构中。存储器808中的问题之一就是存储在存储器808中的所有结果的连通性(connectivity)。存储器808可被配置成使得将存储结果的任一可能组合提供给仲裁器802。当存储在存储器808中的结果数量增加时，选择提供给仲裁器802的结果的复杂性呈指数式增加。

[0310] 推测性存储器810解决了这个问题。可以引退推测性结果两次。首先，引退推测结果，并带有指出其仍具有推测性的指示。这允许其他推测指令还可以继续使用这些结果执

行。这种引退可被称之为推测性引退。最终，如果推测指令被确认，则再次引退推测性结果，从而更新依赖于该结果的任何其他指令并且利用现在的非推测性结果更新寄存器堆。推测性结果的第二次引退的处理略微不同于第一次引退。现在，可以按照与其最初引退相同的顺序引退这些结果。因此，可以使用先进先出(FIFO)队列，而非增大复杂存储器808的大小，来同样存储已经被推测性地引退的推测指令。

[0311] 在实施方式中，存储器808可被配置为存储尚未被引退的所有结果，推测性存储器810可被配置为存储所有推测性地引退的指令，并且仲裁器802可被配置为选择从执行单元、解读机构多路复用器806、存储器808、以及推测性存储器810接收的结果中被引退的结果。

[0312] 在实施方式中，早期(old)结果仲裁器812可被配置为从解读机构多路复用器806和存储器808选择提供给仲裁器802的一个或者多个结果。例如，如果仲裁器802仅可引退2个结果，则早期结果仲裁器812可被配置为仅将两个结果提供给仲裁器802。如上所述，对于仲裁器802，可基于处理器的年龄或者设计选择这些结果。从而允许利用仲裁器802中的简单逻辑选择被引退的指令。仲裁器802仅需要在从执行单元接收的结果与从早期结果仲裁器812接收的结果之间进行选择。

[0313] 在实施方式中，早期结果仲裁器812还可接收来自推测性存储器810的结果。

[0314] 在实施方式中，仲裁器802可被配置为快速引退特定数据，以减少拥堵。在实施方式中，仲裁器802可包括简式数据解读机构。该简式数据解读机构可被配置为从执行单元接收特定类型的数据(例如，预测数据)并且引退与通过仲裁器802引退的其他结果独立的数据。在实施方式中，通过简式数据解读机构处理的数据可以较小，例如，8位宽。将解读机构的主输出、例如上述所述64位输出和128位输出用于小型数据可导致引退带宽的低效使用。尽管通过试图将较小数据与较大数据组合来减轻这种问题，然而，这需要额外的逻辑。通过将该数据移除至简式数据解读机构，还通过减少较小数据的延迟，可以减少主输出的拥堵和延迟。

[0315] 解读机构过程

[0316] 在示例性实施方式中，解读机构可被配置为确定被引退的结果。

[0317] 图9中示出的过程描述了L/S单元如何比较新的L/S指令与未决L/S指令。

[0318] 在早期结果仲裁器812中分别接收推测性存储结果902、存储结果904、以及锁存结果906；并且在仲裁器802中接收未锁存结果。在步骤910，选择早期结果并且将其路由至仲裁器802。在步骤912，仲裁器802选择即将被引退的结果。在步骤914，更新临时存储器，并且在916，就所选择的结果是否具有推测性作出确定。如果确定是肯定的，则在918将这些结果存储在推测性存储器810中；并且如果该决策是否定的，则引退这些结果920。

[0319] 分支预测

[0320] 使用分支指令选择程序应遵循的路径。可以使用分支跳至程序中不同地方的进程。还可使用分支允许重复执行循环体，并且仅在满足某一条件时，才可以使用分支执行一条代码。

[0321] 出于两种原因，分支可致使处理器出现问题。分支可改变程序流程，因此，下一指令并不总是紧随该分支之后的指令。分支还可以条件性的，因此，直至执行该分支，才可以获知被取出的下一指令是否是下一顺序指令或者位于分支目标地址的指令。

[0322] 在初期处理器设计中，一次取出并且执行一个指令。在开始取出新指令时，已经获知了上一分支的目标地址和条件。处理器始终知道下一步取出哪一指令。然而，在管线处理器中，执行的若干个指令重叠。在管线处理器中，在执行该分支之前，需要取出紧随该分支之后的指令。然而，还不知道取出的下一指令的地址。这种问题可被称之为分支问题。因为直至执行该分支之后，才知道该分支的目标地址和条件，所以在准备好执行该分支时利用气泡或者空操作填充执行阶段之前的所有管线阶段。如果在管线的第n个阶段执行指令，则每个分支存在(n-1)个气泡或者空操作。气泡或者空操作中的每个均代表执行指令的错失机会。

[0323] 在超标量处理器中，分支问题更为严重，因为存在两个或者多个管线。对于每个周期能够执行k个指令的超标量处理器，气泡或者空操作的数目为(n-1)x k。每个气泡仍代表执行指令的错失机会。在管线和超标量处理器中由于每个分支错失的周期次数相同，但超标量处理器在该时间段内可以执行更多的操作。例如，考虑了4-发布超标量(即，k＝4)处理器，其中，在第n个管线阶段(n＝6)执行各分支。如果每第五个指令是分支指令，则对于执行的每5个有用指令，存在20个气泡。由于分支问题，仅使用执行带宽的20％执行指令。处理器设计的趋势是朝向更宽的发布和更深的管线，从而使分支问题进一步恶化。

[0324] 分支预测是解决分支问题的一种方式。分支预测器预测是否选择或者不选择某一分支。预测器使用这种预测决定在下一周期中取出下一指令的哪一地址。如果预测选择该分支，则将取出分支目标地址处的指令。如果不根据拆分预测分支，则在分支指令之后取出下一顺序指令。当使用分支预测器时，如果未预测该分支，则仅可看见分支惩罚。因此，高准确度的分支预测器是一种用于减少处理器的分支惩罚的重要机构。

[0325] 图10示出了根据本公开的实施方式的示例性分支预测单元104。图10示出了耦接至分支预测单元的程序计数寄存器1002。程序计数寄存器1002提供当前程序计数器(PC)值。当前PC指有关当前给定阶段中的指令的程序计数器(PC)的值。管线中的每个阶段均具有其自身版本的当前PC。下一PC指从Icache 102中取出的下一指令的地址。对于直线式代码，下一PC将是当前PC+当前指令宽度，对于重定向代码，将是新目标PC。应当认识到，除程序计数寄存器1002之外，可以使用另一来源提供被取出的下一指令的地址。分支预测单元104基于下一PC生成分支方向信号1004。分支方向信号1004指示是否选择分支。

[0326] 图11进一步详细地示出了根据本公开的实施方式的分支预测单元104。分支预测单元104包括预置块1102、哈希块1104、全局分支历史寄存器1106、大型分支历史表1108、小型分支历史表1110、混合选择器表1112、更新计数器表1114、以及多路复用器(mux)1116。预置块1102和哈希块1104耦接至全局分支历史寄存器1106。哈希块1104也耦接至大型分支历史表1108、小型分支历史表1110、混合选择器表1112、以及更新计数器表1114。大型分支历史表1108、小型分支表1110、以及混合选择器1112耦接至多路复用器1116。

[0327] 全局分支历史寄存器1106存储指示在执行程序中的指令的过程中是否选择的分支的位。哈希块1104生成访问大型分支历史表1108、小型分支历史表1110、混合选择器表1112、以及更新计数器表1114中的条目的地址。下面参考图11进一步描述了使用哈希块1104生成访问大型分支历史表1108中的条目的地址。下面参考图15进一步详细描述了使用哈希块1104访问小型分支历史表1110、混合选择器表1112、以及更新计数器表1114中的条目的地址。

[0328] 常规分支预测器可仅使用一个分支历史表。此处所示的实施方式使用了两个分支历史表，即，大型分支历史表1108和小型分支历史表1110。小型分支历史表1110和大型分支历史表1108存储预测分支在正被执行的程序代码中的分支方向的值。小型分支历史表1110具有比大型分支历史表1108更少的条目并且由此是更短的历史，该更短的历史在捕捉仅需要最近分支结果的分支之间的关联性时表现较好。大型分支历史表1108具有比小型分支历史表更多的条目并且更长的历史，捕捉分支之间更为复杂的关联性。下面参考图16描述更新大型分支历史表1108与小型分支历史表1110中的值的状态机。

[0329] 多路复用器1116基于从混合选择器表1112的条目中读取的选择值选择从大型分支历史表1108和小型分支历史表1110读取的分支方向。使用哈希块1104将每个取出分支映射至大型分支历史表1108中的条目、小型分支历史表1110中的条目、以及混合选择器表1112中的选择条目。如果混合选择器表1112中的选择条目具有大于或者等于2的值，则使用从大型分支历史表1108的预测来预测分支的方向，否则，使用从小型分支历史表1110的预测来预测分支的方向。如果在预测该分支时，仅大型分支历史表1108是正确的，则使混合选择器表1112中对应于某一分支的选择条目的值递增。如果在预测该分支时，仅小型分支历史表1110是正确的，则使混合选择器表1114中对应于该分支的选择条目的值递减。如果大型分支历史表1108和小型分支历史表1110对该分支给出相同的方向，则选择条目中的值不变。

[0330] 使用更新计数器表1114确定是否禁止更新大型分支历史表1108中的条目。更新计数器表1114将更新值存储在每个条目中。更新值指示在预测特定分支时是大型分支历史表1108还是小型分支历史表1110更为准确。根据本公开的实施方式，如果更新计数器表1114中对应的更新值指示在预测该分支的分支方向时小型分支历史表1110比大型分支历史表1108更为准确，则不更新大型分支历史表1108中对应于分支指令的值。如果对应于更新计数器表1114中的特定分支的更新值是零，则无论通过大型分支历史表1108或者小型分支历史表1110是否正确预测特定分支，均可禁止更新大型分支历史表1108，否则，允许更新。当小型分支历史表1110错误预测特定分支时，将对应于更新计数器表1114中的特定分支的更新值设置成11。之后每次，如果大型分支历史表1108错误预测特定分支，则使对应于该特定分支的更新值递减。同样，当小型分支历史表1108最近错误地预测特定分支时，仅利用对该特定分支的正确预测更新大型分支历史表1108。从而防止大型分支历史表1108的过期更新，从而导致针对该特定分支更好地训练大型分支历史表1108。

[0331] 图12示出了更新根据本公开的实施方式的全局分支历史寄存器1106的实施例。图12示出了全局分支历史寄存器1106中的不同指令和取出指令时的结果更新。在实施例中，可将全局分支历史寄存器1106初始化，以在启动时存储所有零。在另一实施例中，全局分支历史寄存器可在启动时存储随机值。仅对于条件分支指令，才更新全局分支历史寄存器1106。例如，条件分支指令是基于条件是真或者是假而跳至某一地址的分支指令。是否等于零的分支指令(BREQZ)、是否不等于零(BRNEZ)的分支指令、是否小于或者等于零(BRLEZ)的分支指令、以及是否大于零(BRGTZ)的分支指令皆是条件分支指令的实施例。无条件分支指令是始终从程序执行转换至分支指令中指定的指令地址的分支指令。例如，BR X是从程序执行转换至存储在地址X的指令的无条件分支指令。对于诸如加法(ADD)指令、减法(SUB)指令、乘法(MUL)指令、除法(DIV)指令、加载(LD)、或者存储(SD)指令等不是条件分支的指令，

也不更新全局分支历史寄存器1106。当取出指令(例如,ADD指令402)时,由于其不是条件分支指令,所以不对全局分支历史寄存器1106做出任何改变。因此,取出任何算术指令、加载指令、存储指令、或者无条件分支指令将不更新全局分支历史寄存器1106。

[0332] 返回参考图12,一旦接收BR指令1204,因为BR指令1204不是条件分支指令,所以不更新全局分支历史寄存器1106。当取出SUB指令1206和MUL指令1208时,不更新全局分支历史寄存器1106。一旦接收条件分支指令BREQZ 1210,则更新全局分支历史寄存器1106。在该实施例中,假定预测正被选择的BREQZ指令210,则通过将位"1"移位至全局分支历史寄存器1106中指示其已被选择的最低有效位位置而更新全局分支历史寄存器1106。在实施方式中,如果在之后通过在分支单元118中执行BREQZ指令1210而解析出BREQZ指令1210时,确定预测不正确,则利用BREQZ指令1210的正确预测更新全局分支历史寄存器1106。

[0333] 一旦取出DIV指令1212,则由于DIV 1212指令不是条件分支指令,而不更新全局分支历史寄存器1106。一旦接收BRNEZ指令1214,则由于BRNEZ指令1214是条件分支指令,而不更新全局分支历史寄存器1106。如图12所示,假定预测被选择的BRNEZ指令1214,则通过将位"1"移位至全局分支历史寄存器1106的最低有效位而更新全局分支历史寄存器1106。BRNEZ指令1216因其是条件分支指令而致使更新全局分支历史寄存器1106。假定不选择BRNEZ指令1216,则将位"0"移位至全局分支历史寄存器1106的最低有效位位置。对于下一指令,即,大于零分支(BRGTZ)1218,将再次更新全局分支历史寄存器1106。假定选择大于零分支1218,则将一移位至全局分支历史寄存器1106的最低有效位位置。

[0334] 程序可包括多个进程。进程是程序中一旦执行"调用"指令时而访问的代码片段。调用指令可包括在调用指令之后将执行的程序返回至下一指令的指令。调用指令的实施例是参考下面提供的示例性程序代码而进一步描述的"具有链接的分支"指令:

[0335] 程序代码:

[0336] 0x001 ADD

[0337] 0x002 SUB

[0338] 0x003 BR

[0339] 0x004 BRNEZ

[0340] 0x005 MUL

[0341] 0x006 ADD

[0342] 0x007 BRANCH WITH LINK TO PROCEDURE 1

[0343] 0x008 BRLEZ

[0344] 0x009 BRGTZ

[0345] 0x010 ADD

[0346] 0x011 BRANCH WITH LINK TO PROCEDURE 2

[0347] 0x012 ADD

[0348] 进程1

[0349] 0x014 ADD

[0350] 0x015 SUB

[0351] 0x016 BREOZ

[0352] 0x017 BRNEZ

[0353] 0x018 MUL

[0354] 0x019 DIV

[0355] 结束进程1

[0356] 0x021 ADD

[0357] 0x022 MUL

[0358] 进程2

[0359] 0x024 SUB

[0360] 0x025 MUL

[0361] 0x026 ADD

[0362] 0x027 BREQZ

[0363] 0x028 MUL

[0364] 0x030 BRGTZ

[0365] 结束进程2

[0366] 在上述示例性程序代码中，0xXXX表示将指令存储在指令高速缓存102中的地址。具有链接指令的分支是将执行的程序转换至程序代码中的特定进程的指令。此处，使用从执行程序转换至某一进程的链接指令执行该分支被称之为“调用进程”。具有链接指令的分支包括在具有链接指令的分支之后将执行程序返回至下一指令的指令(未示出)。

[0367] 因为分支通常与之前执行的分支相关联，所以将诸如存储在全局分支历史寄存器1106中的全局分支历史用作访问大型分支历史表1108和小型分支历史表1110中的预测条目的索引。较长的分支历史能够使预测器查看更多之前执行的分支并且基于与这些分支的关联性进行学习。对于与最近分支历史高度关联的分支，全局历史可提供关键预测信息。常规分支预测器仅可依赖于全局分支历史而进行分支预测。然而，并非该程序中的所有分支皆与最近执行的分支相关联。对于不与最近执行的分支相关联的这些分支，在全局历史中编码的附加信息在预测分支时可产生更多的危害而非益处。还增加了训练分支预测器的时间并且明显扩大了分支预测表中混淆现象的范围，从而降低当前分支和其他分支的预测准确度。较长全局分支历史寄存器1106能够提高更多的远端分支之间的关联性，但是，也增加了分支历史中包括的不相关分支的数目。当预测分支时，这些不相关的分支可产生明显的噪音。考虑15位全局分支历史寄存器1106。与3个之前分支高度关联的分支可利用关联预测器，但是，即使在这种情景中，历史也包含12位的无用噪音。这就是指在最坏情况下，需要2^{12}倍以上的条目来预测分支，从而极大地增加了分支预测器的训练时间段以及与其他分支的混淆现象。对于与之前分支不相关的分支，整个15位皆是噪音。进程调用通常代表程序流中的中断。进程调用之前的分支趋于与进程调用内的分支的关联度较低。因此，提供了一种允许某些分支从较大历史中获益、但消除或者减少噪音无用的这些区域中的历史噪音的架构。

[0368] 为了使用全局分支历史寄存器1106提供对分支更好的预测，在给出该进程的分支时，使用进程中的第一指令的开始地址重写全局分支历史寄存器1106中的值。此处，使用被调用进程的开始地址的地址重写全局分支历史寄存器1106中的值被称之为“预置”。如果进程的分支具有推测性并且未被正确预测，则将全局分支历史寄存器1106中重写的值重新存储在全局分支历史寄存器1106中。使用进程中第一指令的开始地址提供了将全局分支历史

寄存器1106预置的每个时间点的唯一历史，从而消除程序代码中不同的阈值点之间的混淆现象，并且确保当程序执行再次调用进程时，将全局分支历史寄存器1106预置成同一值。因为将全局分支历史寄存器1106用作大型分支历史表1108和小型分支历史表1106的索引，以确定分支方向，所以将全局分支历史寄存器1106预置成同一值(即，进程中第一指令的开始地址)确保了将从大型分支历史表1108和小型分支历史表1106检索的分支预测被本地化至被调用的进程且更为准确。

[0369] 图13示出了根据本公开的实施方式的全局分支历史寄存器1106的预置。预置块1102耦接至程序计数器1002和全局分支历史寄存器1106。当具有链接指令的分支(诸如，链接至进程1的分支)致使执行程序跳至进程1时，预置块1102将全局分支历史寄存器1106中的值重写至进程1的开始地址。进程1的开始地址是该进程中第一指令的地址，例如，上面程序代码中的0x014。例如，从程序计数器1002可以访问程序1中的第一指令的开始地址。应当认识到，可以从除程序计数器1002之外的来源接收进程开始时的地址。随后无论何时跳至进程1时，将全局分支历史寄存器1106的值预置为0x014。这不仅提供有关进程1中的分支的更好关联性，而且还加快了大型分支历史表1108与小型分支历史表1110的训练速度。同样，无论何时调用进程2，均将全局分支历史寄存器1106预置成进程2中的第一指令的地址，即，0x024。在进程调用时预置全局分支历史寄存器1106的另一益处在于其支持稀疏的大型分支历史表1108和小型分支历史表1110。这就导致了内存节省。而且，如下面参考图14所描述的，当与常规系统相比较时，需要全局分支历史寄存器1106中的较小散列值，从而访问大型分支历史表1108中的条目。

[0370] 图14示出了根据本公开的实施方式的提供访问大型分支历史表1108中的条目的索引的示例性系统。在图14的实施例中，程序计数器1002和全局分支历史寄存器1106耦接至哈希块1104。哈希块1104耦接至大型分支历史表1108并且提供读取或者写入大型分支历史表1108中的条目的索引。哈希块1104包括XOR门1400和哈希函数1402、1404、以及1406。全局分支历史寄存器中的值是15位宽并且利用哈希块1104被向下散列成12位。例如，哈希函数1402、1404、以及1406将全局分支历史寄存器1106中的值从15位向下散列成12位。程序计数器1002的12个最低有效位是利用XOR门1400将全局分支历史寄存器1106中的值散列成12位的XORed，以生成访问大型分支历史表1108中的条目的12位索引。

[0371] 图15示出了根据本公开的实施方式的提供访问小型分支历史表1110、混合选择器1112、以及更新计数器1114中的每个的条目的索引的系统。全局分支历史寄存器1106与程序计数器1002耦接至哈希块1104。哈希块1104耦接至小型分支历史表1110、混合选择器表1112、以及更新计数器表1114。

[0372] 根据本公开的实施方式，哈希块1104包括XOR函数1400。XOR函数1400将程序计数器1002中的32位程序计数器值散列成10位值。由哈希函数1400生成的10位与全局分支历史寄存器1106的最低有效位组合，以形成11位索引。使用11位索引访问小型分支历史表1110、混合选择器1112、以及更新计数器表1114中的条目。

[0373] 图16示出了根据本公开的实施方式的用于更新存储在大型分支历史表1108和小型分支历史表1110中的分支预测条目的示例性状态机。在初始化过程中，可将随机值存储在大型分支历史表1108和小型分支历史表310的条目中。如果在第一次执行时选择某一分支，则对应于该分支的条目与“弱选择”状态1602相关联并且利用位00进行更新。如果分支

的条目当前处于弱选择状态1602，并且如果在下一次再次选择时执行该分支，则该条目与“强选择”状态1604相关联并且利用位01进行更新。如果分支的当前状态处于弱选择状态1602，则如果在下一次执行时不选择该分支，则将该分支转换成“弱不选择状态”1606并且利用10更新该条目。如果分支当前与弱不选择状态1606相关联并且在下一次执行时选择该分支，则将该分支的状态转换至弱选择状态1602并且利用00更新其相应的条目。如果分支处于弱不选择状态1606，则在下一次再次执行时不选择该分支，然后，将该状态转换至“强不选择”状态1608并且利用11更新该条目。如果分支处于强不选择状态1608，则在选择该分支时，将将其转换成弱不选择状态1606并且利用10更新该条目。如果分支处于强选择状态1604并且在下一次再次执行时选择该分支，则使其保持处于强选择状态1604。如果分支处于强选择状态1604，并且在再一次执行时不选择该分支，则将其转换成弱选择状态1602并且利用00更新该条目。如果分支处于弱不选择状态10并且在下一次执行时选择该分支，则将其转换成弱选择状态00。

[0374] 零开销循环

[0375] 通常，“循环”是响应“循环指令”或者“循环操作”而重复特定次数的一个或者多个指令的序列。通常，重复该序列指令被称之为“循环体”。循环开始地址指循环体中的第一指令的地址，即，分支到开始新循环迭代的地址。循环结束地址指循环体之后的第一指令的地址，即，分支到自然退出循环的地址。循环体还可被称之为以循环开始地址开始并且以循环匹配地址结束的指令。循环的示例性伪代码被示出如下：

[0376]
```
For (i>0; i<n, i++)
{
    INST 1;
    INST 2;
    INST 3;
}
```

[0377] 在上述实施例中，“For(i>0;i<n,i++)”可被称之为循环指令。循环体包括指令INST 1、INST 2、以及INST 3。应当认识到，尽管上面实施例中仅示出了三个指令，然而，循环体中的指令数目是任意的。在上述实施例中，“n”是指示执行循环体中的指令的次数的“循环计数”。在实施例中，n可以是中间值，例如，n＝5，即，在循环指令自身中被编码并且指示必须执行循环体5次。在另一实施例中，从处理器的通用寄存器堆或者从指令管线中的另一地方获得n的值。如果在不知道“n”的值的情况下，取出、解码、并且执行循环指令和循环体，则将循环计数称之为“推测计数”。当通过从处理器中的通用寄存器堆访问循环计数或者从处理器中的另一位置访问循环计数而确定其值时，可以将循环计数“n”称为“被解析”。可以推测性地执行循环指令。当确定猜测执行或者不执行循环指令时，则将循环指令称之为“被解析”。

[0378] 在常规架构中，编译器可合成并且执行与上面所示循环实例等同的下列汇编语言：

[0379] MOV R0,n

[0380] 循环开始BREQZ R0,循环结束

[0381] SUB R0,R0,1;

[0382] INST 1;

[0383] INST 2;

[0384] INST 3;

[0385] BR循环开始

[0386] 循环结束INST 4;

[0387] INST N;

[0388] 在上述实施例中,为了执行循环,编译器首先产生使循环计数n(如果可用)传送至寄存器(例如,寄存器R0)中的传送(MOV)指令。编译器还产生诸如等于零分支(BREQZ)、减法(SUB)指令、以及分支(BR)指令等额外指令。BREQZ、SUB、以及BR指令增加了这些指令的原循环体的大小。

[0389] 通过编译器将BREQZ R0,循环结束插入到循环开始地址。在上述实施例中,“循环开始”指“标签”。标签是被分配给代码内的固定位置的名称或者编号,并且通过在代码中其他地方出现的其他指令可以参考标签。除了标记代码中的指令的一部分之外,标签不具有任何影响。如此处使用的,“循环开始”或者“循环开始地址”指循环体中的第一指令的地址。在该实施例中,循环开始地址指通过编译器可生成的BREQZ指令。如此处使用的,“循环结束”或者“循环结束地址”指循环体之后的第一指令的地址。在上述实施例中,循环结束指循环体之后的第一指令INST 4。当R0中的值不等于0时,BREQZ指令将退出循环并且将执行程序转换至循环结束地址处的INST 4。可替代地,如果推测性地执行循环并且之后在管线中确定不应执行该循环,则再次将执行程序转换至INST 4。在另一实施例中,如果R0中的原始值是0(即,其是空循环),则再次将执行程序转换成INST 4。

[0390] 除MOV和BREQZ指令之外,编译器生成在每次执行循环体时使R0中的值以中间值“1”递减的减法(SUB)指令sub R0,R0,#1。而且,在INST 3之后,编译器还生成分支指令BR循环开始。BR循环开始将执行程序转换回至循环开始地址。在常规汇编语言代码的实施例中,BREQZ指令、SUB指令、以及BR指令代表附加指令或者执行循环所需的“开销”。使用这些开销指令保存执行循环的次数、循环开始的转换、以及何时退出循环。

[0391] BREQZ、SUB、以及BR指令是每次执行循环体时需要执行的开销指令。在常规处理器中,这些开销指令在每次执行循环体时添加三个额外周期。对于短循环,诸如上述所示仅具有三个指令INST 1、INST 2、以及INST 3的一个指令,开销指令几乎使执行循环体所需的周期数目加倍。因此,此处所示出的实施方式提供被称之为明显减少执行循环所需的开销的“ZLOOP”的零开销循环指令。根据本公开的实施方式,下面示出了程序员利用ZLOOP指令编写的示例性高级程序代码:

[0392] ZLOOP n,循环开始

[0393] INST 1;

[0394] INST 2;

[0395] INST 3;

[0396] 循环结束INST 4;

[0397] INST N;

[0398] 根据本公开的实施方式,如下面汇编语言等同物所示,由程序员合成并且执行上

述高级程序代码：

[0399] MOV R0,n；

[0400] BREQZ R0,循环结束:SUB循环计数,R0,#1

[0401] 循环开始INST 1；

[0402] INST 2；

[0403] 循环匹配INST 3:BRNEZ循环计数,循环开始:SUB循环计数,

[0404] 循环计数,#1；

[0405] 循环结束INST 4；

[0406] INST N；

[0407] 示例性合成代码中的“:”指在同一周期中取出通过“:”分离的指令。当与上述常规汇编语言相比较时，循环开始地址之前的上述代码仅具有MOV和BREQZ指令的一次执行开销。BREQZ指令检查循环是否为空，即，R0等于0，并且不应被执行。如下进一步所述，上述指令中的“循环计数”指图17中存储循环计数值(如果是具有循环计数的中间值或者解析值的循环指令)或者推测循环计数值(如果是未解析出循环计数)的循环计数寄存器1700。

[0408] 在示出性实施方式中，由编译器生成指示循环中的最后指令的“循环匹配”标签或者循环匹配地址。循环匹配地址指循环体中的最后指令的地址，即，INST 3。在管线的DE1阶段，一旦检测表示循环体中的最后指令的循环匹配地址，与程序编译器相反的指令解码器106则生成并且排队BRNEZ计数，分支预留队列112中的循环开始指令同时将INST 3和SUB R0,R0,#1排队在一个或者多个预留队列114A中。可替代地，INST 3可以被排队在加载/存储预留队列116中。因为通过与编译器或者程序员相反的指令解码器106生成分支指令，所以此处该分支指令被称之为“合成”分支。因为同时将合成BRNEZ指令与INST 3以及SUB指令一起排队，所以不需要附加时钟周期排队生成的BRNEZ指令和SUB指令，从而产生循环指令的零开销处理。在实施例中，循环体中的最后指令(即，上述实施例中的INST 3)不可以是另一分支指令。尽管在此处提供的实施例中，合成分支指令的生成发生在DE1阶段，然而，应当认识到，合成分支指令的生成应发生在管线的其他阶段并且通过除指令解码器之外的其他单元而生成。

[0409] 根据本公开的进一步实施方式，如果未解析出循环计数，则将继续取出并且推测性地执行循环体，直至解析出循环计数。如果执行的循环体的次数(即，循环计数)小于取出循环体的迭代次数，则撤消与连续取出的循环体迭代相关联的指令。撤消指令必须将其驱逐出管线并且不使由其产生的任何结果处于处理器的架构状态。基本上，撤消指令指从管线中移除该指令，如同从未取出或者执行过该执行。如果在指令中将循环计数编码成中间值，则获知循环计数并且不需要解析出循环计数。在该实施例中，不存在连续取出的循环体迭代。此处，包括中间循环计数值的循环指令被称之为“LOOP”指令。如果循环计数是带符号的值(即，不是中间值)，则该指令被称之为“ZLOOP”指令。下面参考图17和图18中的实施例提供用于实现此处所示出的实施方式的进一步细节。

[0410] 图17示出了用于实现根据本公开的实施方式的零开销循环的架构的一部分。图17示出了指令取出单元1701、指令解码器106、SIN分配器107、程序计数寄存器1703、分支预留队列112、以及分支单元118。指令取出单元1701耦接至指令解码器106、SIN分配器107、程序计数寄存器1703、以及分支单元118。在实施例中，程序计数寄存器1703存储可替代地被称

之为“程序计数值”的“程序计数器”。在可替代的实施方式中，从除程序计数寄存器1703之外的管线中的其他地方可接收程序计数值。分支单元118耦接至指令解码器106。

[0411] 指令取出单元1701包括指令高速缓存(Icache)102、循环计数寄存器1700、循环开始寄存器1702、以及循环匹配寄存器1704。循环计数寄存器1700存储循环计数值(如果是具有循环计数的中间值或者解析值的循环指令)或者推测循环计数值(如果是未解析处循环计数的情况)。当解析出循环计数时(例如，当在EX1阶段，在分支单元118中执行分支时)，则以循环计数取代推测循环计数值。循环开始寄存器1702存储循环开始地址并且循环匹配寄存器1704存储循环匹配地址。

[0412] 现将结合图18描述图17。图18中的列是管线阶段。IC1是指令取出阶段。DE1和DE2是解码阶段。RS1和RS2是预留队列阶段。EX 1是执行阶段。应当认识到，可以存在一个或者多个IC、DE、RS、以及EX阶段。图18中的行表示时钟周期。命名法X:Y表示管线阶段:时钟周期。

[0413] 例如，在IC1:0中，由指令取出单元1701从Icache 102取出ZLOOP指令。在DE1:1中，由指令解码器106接收ZLOOP指令。一旦接收ZLOOP指令，指令解码器106则将循环开始地址存储在循环开始寄存器1702中、将循环匹配地址存储在循环匹配寄存器1704中、并且将循环计数寄存器1700中的值设置为-1。计数当前是指示在未知实际循环计数时推测性地执行循环体的负数。这是因为循环计数可被存储在寄存器R0中或者处理器的另一地方并且尚不可用或者不可访问。直至在分支单元118中解析出循环，才可解析出循环计数。在实施例中，指令解码器106利用图17中的Zloop_检测信号1706设置循环计数寄存器1700、循环开始寄存器1702、以及循环匹配寄存器1704中的值。在DE1:1中，指令解码器106还将SIN_发生_信号1712发送至SIN分配器107，以生成循环体中的ZLOOP指令(或者多个指令)的SIN编号。尽管循环计数具有推测性，然而，指令取出单元1701还判断指示SIN分配器107的循环_计数_推测信号1714应在循环计数具有推测性时继续生成SIN编号。

[0414] 在DE2:2中，因为通过指令取出单元1701推测性地取出ZLOOP指令自身，所以将SIN#0分配给ZLOOP指令。通过SIN分配器107生成SIN#1，以分配给循环体中的指令(即，INST 1、INST 2、以及INST 3)，以用于循环体的第一次迭代。例如，当INST 1达到DE2:3时，将SIN#1分配给INST 1，当INST 2达到DE2:4时，将SIN#1分配给INST 2，并且当INST 3达到DE2:4时，将SIN#1分配给INST 3。

[0415] 当循环匹配标签和INST 3达到DE1:4时，指令取出单元1702通过将循环匹配标签与存储在循环匹配寄存器1704中的值相比较而检测循环匹配标签并且将循环匹配信号1701发送至指令解码器106，以生成合成的BRNEZ指令，如上所述，循环匹配标签指循环体中的最后指令，即，INST 3。响应接收循环匹配信号1710，指令解码器106生成用于在将INST 3排队到预留队列114A至114D中的一个中的相同周期内而排队到分支预留队列112中的合成分支指令BRNEZ。可替代地，INST 3可以被排队到加载/存储预留队列116中。在实施例中，指令解码器106利用信号1720将合成分支指令排队到分支预留队列112中。例如，如果INST 3是算术指令，则将其排队到预留队列114A至114D中的一个中，并且如果INST 3是加载/存储指令，则将其排队到加载/存储预留队列116中的一个中，而同时将合成的BRNEZ指令排队到分支预留队列112中。应当认识到，尽管在此处所示出的实施例中，合成分支BRNEZ是不等于零的分支，然而，在其他实施例中，可代替生成等于零(BREQZ)分支指令、分支小于零

(BRLEZ)指令、或者大于零(BRGTZ)分支指令。

[0416] 在DE1:4中，一旦检测到指循环体中的最后指令的循环匹配标签(即，实施例中的INST 3)，则指令解码器106再次设置Zloop检测信号1706，以设置循环计数寄存器1700中的计数值为-2，从而指示将执行二次推测性地取出并且执行循环体。

[0417] 当带有循环匹配标签的INST 3达到DE2:5时，指令解码器106响应循环匹配标签再次将SIN_发生_信号1712发送至SIN分配器107，以生成分配给用于二次推测性地执行循环体的指令的SIN#2。

[0418] 当ZLOOP指令达到EX1:5时，由分支单元118执行ZLOOP指令并且解析出循环。如果循环为空，即，确定实际循环计数(如R0中的实际值所示)为0，则退出循环，并且由分支单元118利用重定向信号1717将程序计数器重定向为由循环结束标签指示的指令，即，INST 4，并且撤消从循环体推测性地取出的指令。如果循环不为空，即，R0大于0，则经由循环_计数_实际信号1715将R0的值发送至指令取出单元1701，其中，将R0中的值添加到存储在循环计数寄存器1700中的值，以确定循环计数。在该实施例中，R0是10，即，10与-2相加并且结果值为循环计数8。这表明已经执行循环两次并且仍需要执行循环体8次以上迭代。

[0419] 一旦通过确定EX1:5中的R0的值解析出计数，则解除认定循环匹配_推测信号1714。响应正被解除认定的循环匹配_推测信号1714，SIN分配器107停止分配进一步的SIN编号。因此，在该实施例中，仅生成两个SIN编号。如果循环计数(由R0指示)为指示空循环的0，则撤消具有SIN编号#1和#2的所有指令以及具有SIN编号#0的推测性取出的ZLOOP指令。如果在另一实施例中，确定R0为1，则循环中的第一推测性迭代有效，并且具有SIN#2的循环的第二迭代无效并且从管线中驱逐出去。

[0420] 在EX1:5中，在分支单元118中执行循环开始指令，即，具有BRNEZ循环计数的INST 3，以确定是否执行分支。例如，如果由R0指示的循环计数为0或者小于0或如果错误地推测执行ZLOOP自身，则分支单元118可发送重定向信号1717，以将程序计数器寄存器1703重定向为由循环结束地址(即，INST 4)指示的指令，并且撤消与SIN#1和#2相关联的所有指令以及与SIN#0相关联的ZLOOP指令。如果确定执行该分支并且由R0指示的循环计数不是0，则“释放”相应的SIN编号。此处所提及的释放SIN编号指将执行循环体中具有释放SIN编号的指令并且将不从管线中驱逐出其值。在该实施例中，如果R0是2或者大于2，则将释放SIN#1和SIN#2。如果R0是1，则将释放SIN#1并且SIN#2无效，从而驱逐出与SIN#2相关联的指令。

[0421] 结论

[0422] 应当认识到，细节描述部分而非本公开的摘要旨在用于对权利要求进行说明。本公开的摘要可设定一种或者多种、而非所有的示例性实施方式并且由此并不旨在以任一方式限制增补的权利要求。

[0423] 相关领域技术人员应当认识到，在不背离本公开的实质和范围的情况下，可以从形式和细节上做出各种改变。

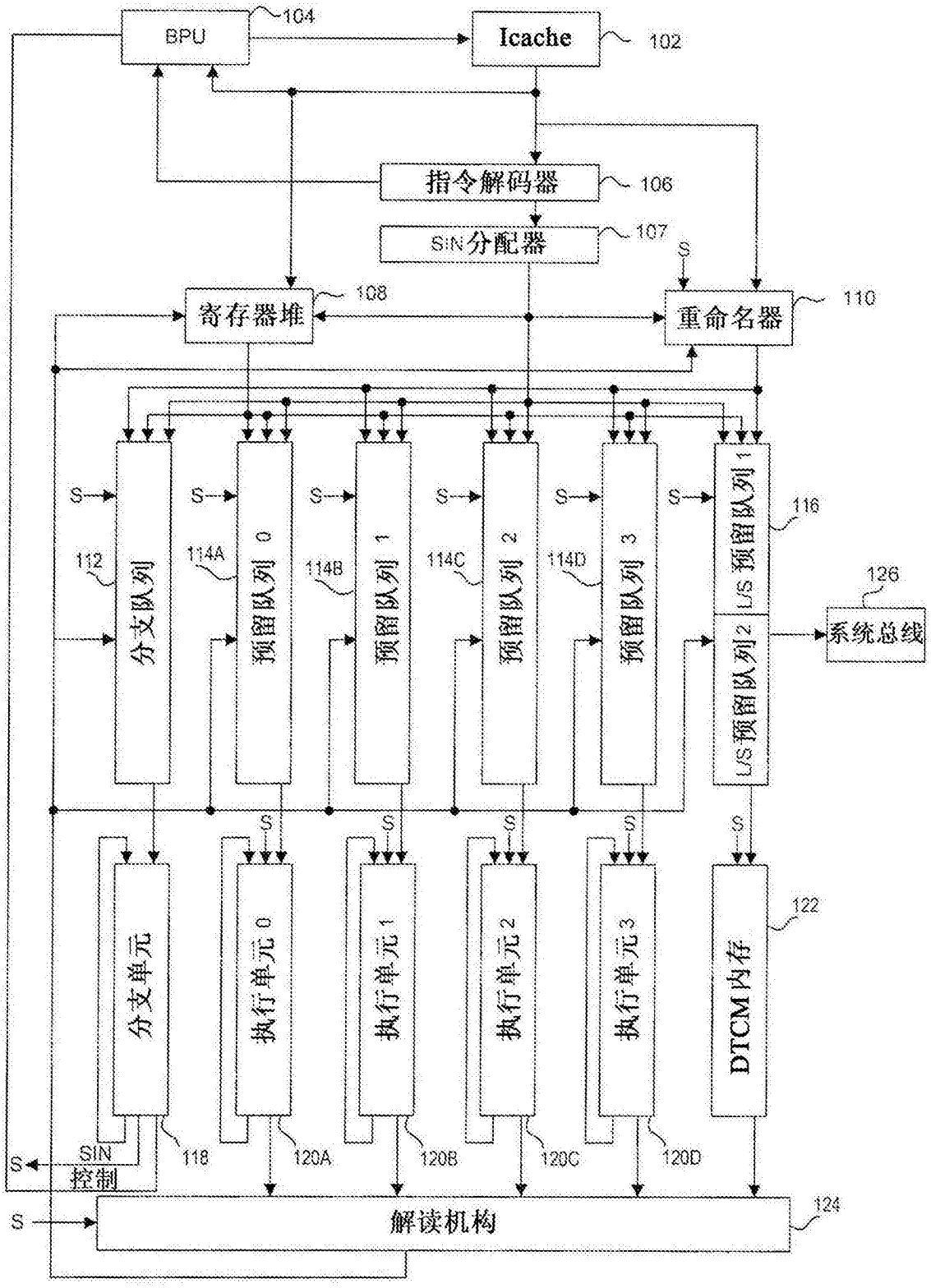

BPU
104
Icache
102
指令解码器
106
SIN分配器
107
S
寄存器堆
108
重命名器
110
S
112
分支队列
S
114A
预留队列 0
S
114B
预留队列 1
S
114C
预留队列 2
S
114D
预留队列 3
S
L/S 预留队列 1
L/S 预留队列 2
116
126
系统总线
S
分支单元
S
执行单元 0
S
执行单元 1
S
执行单元 2
S
执行单元 3
S
DTCM 内存
122
SIN
控制
S
118
120A
120B
120C
120D
S
解读机构
124

图1

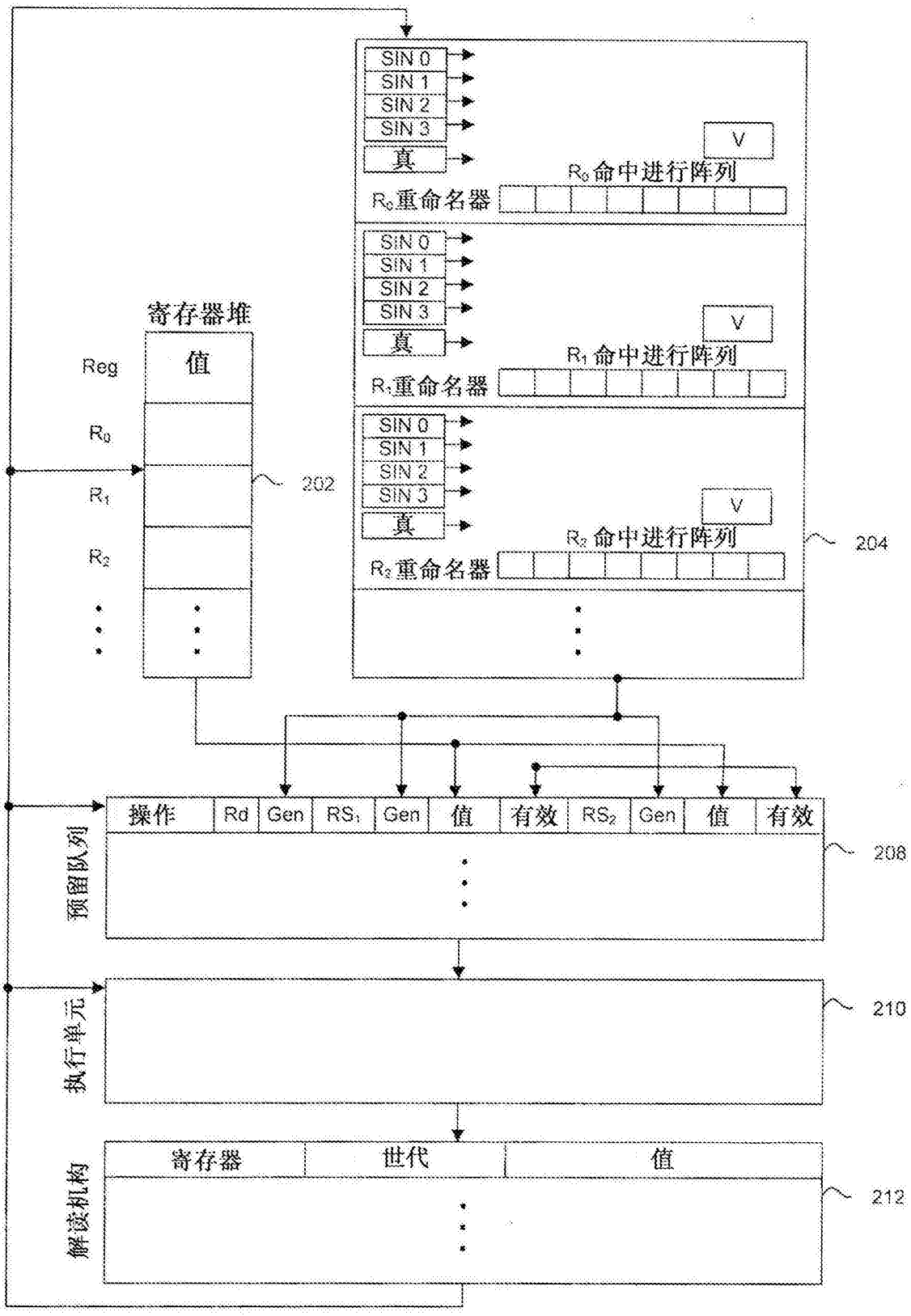

寄存器堆
Reg
值
R_0
R_1
R_2
202
SIN 0
SIN 1
SIN 2
SIN 3
真
V
R_0命中进行阵列
R_0重命名器
R_1命中进行阵列
R_1重命名器
R_2命中进行阵列
R_2重命名器
204
操作
Rd
Gen
RS_1
Gen
值
有效
RS_2
Gen
值
有效
预留队列
208
执行单元
210
寄存器
世代
值
解读机构
212

图2

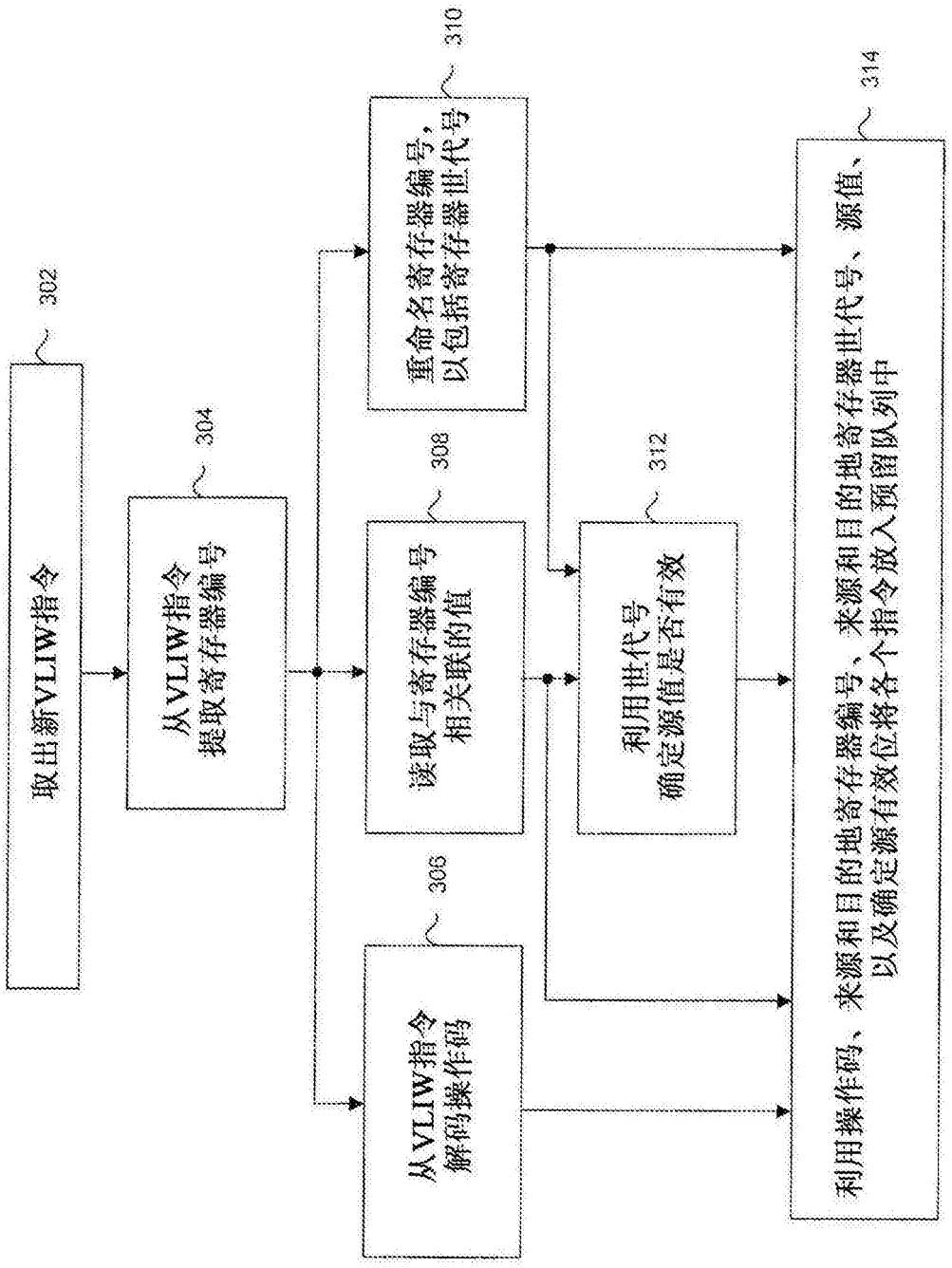

302
取出新VLIW指令
304
从VLIW指令
提取寄存器编号
306
从VLIW指令
解码操作码
308
读取与寄存器编号
相关联的值
310
重命名寄存器编号，
以包括寄存器世代号
312
利用世代号
确定源值是否有效
314
利用操作码、来源和目的地寄存器编号、来源和目的地寄存器世代号、源值、
以及确定源有效位将各个指令放入预留队列中

图3

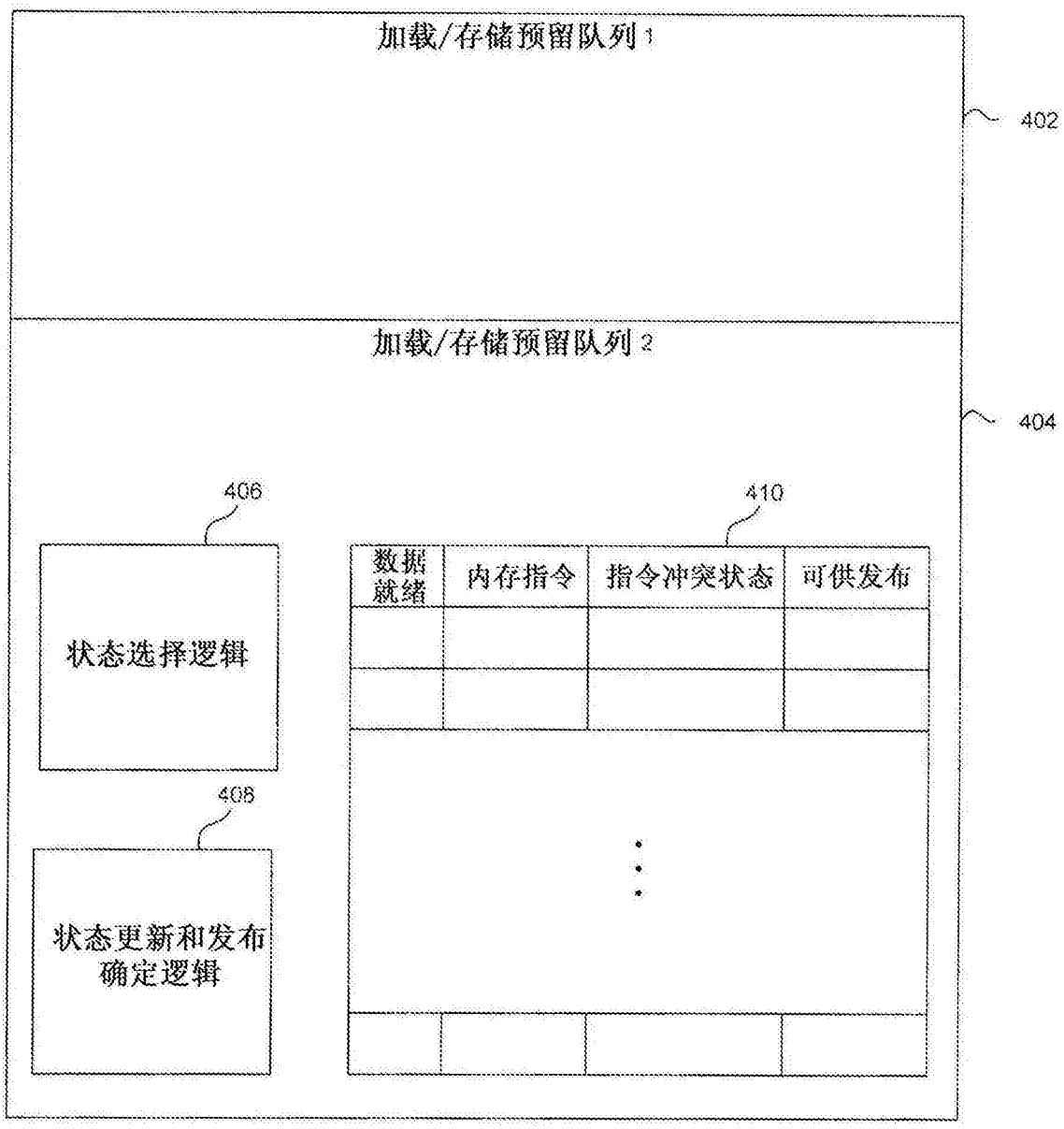

加载/存储预留队列 1
402
加载/存储预留队列 2
404
406
状态选择逻辑
408
状态更新和发布确定逻辑
410
数据就绪
内存指令
指令冲突状态
可供发布

图4

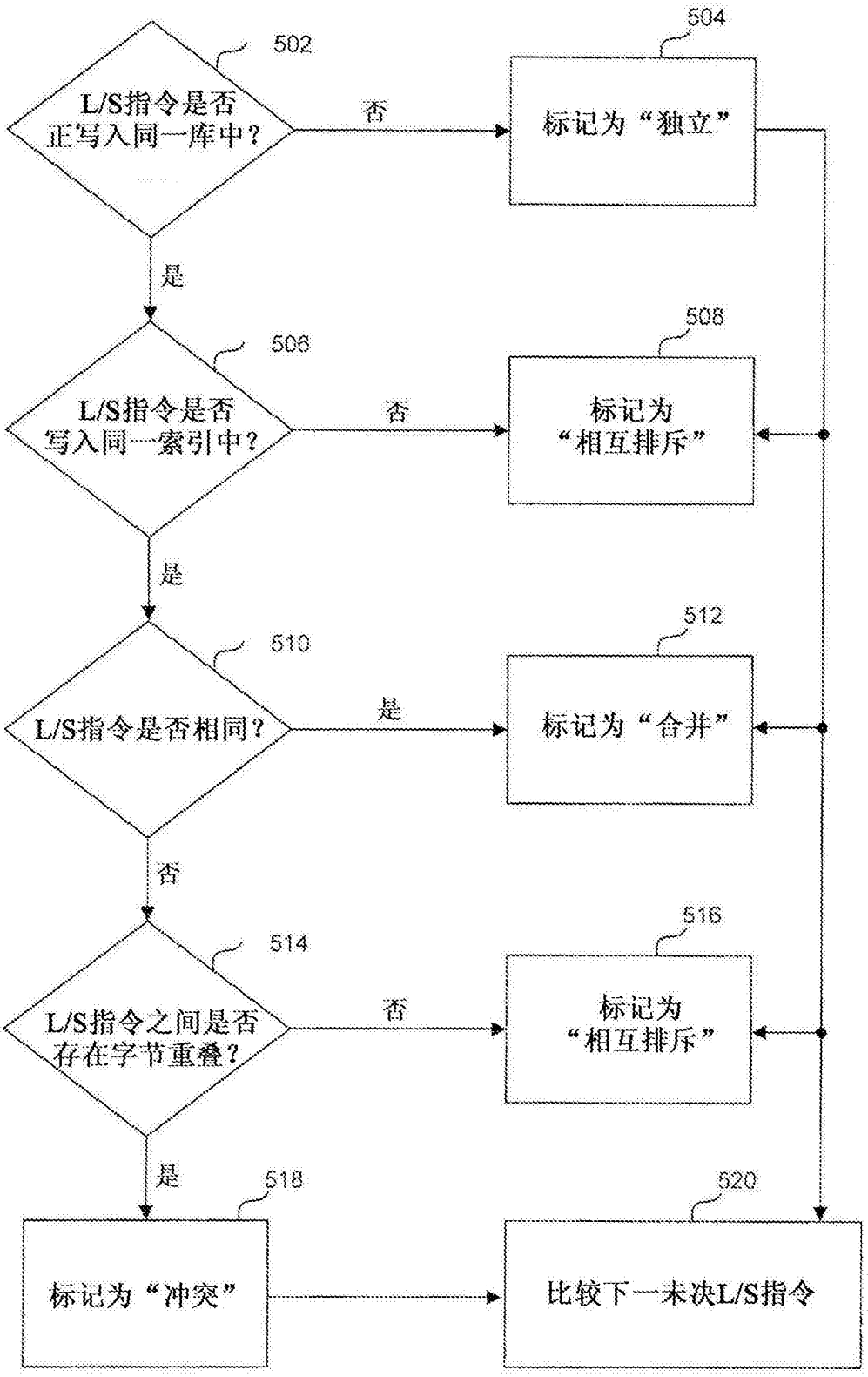

502
L/S指令是否
正写入同一库中？
否
504
标记为“独立”
是
506
L/S指令是否
写入同一索引中？
否
508
标记为
“相互排斥”
是
510
L/S指令是否相同？
是
512
标记为“合并”
否
514
L/S指令之间是否
存在字节重叠？
否
516
标记为
“相互排斥”
是
518
标记为“冲突”
520
比较下一未决L/S指令

图5

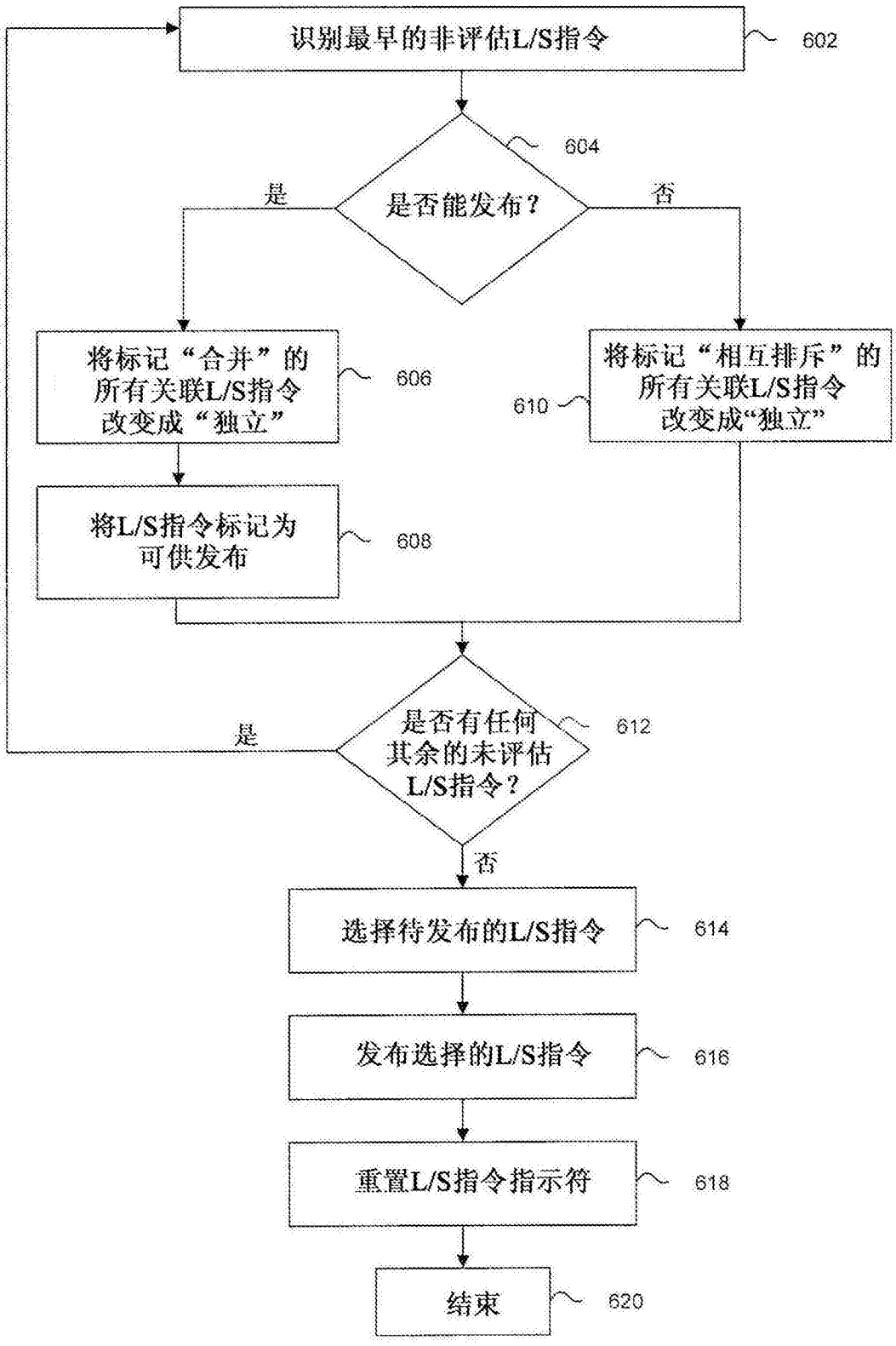

识别最早的非评估L/S指令
602
604
是否能发布？
是
否
将标记“合并”的所有关联L/S指令改变成“独立”
606
将L/S指令标记为可供发布
608
610
将标记“相互排斥”的所有关联L/S指令改变成“独立”
是否有任何其余的未评估L/S指令？
612
是
否
选择待发布的L/S指令
614
发布选择的L/S指令
616
重置L/S指令指示符
618
结束
620

图6

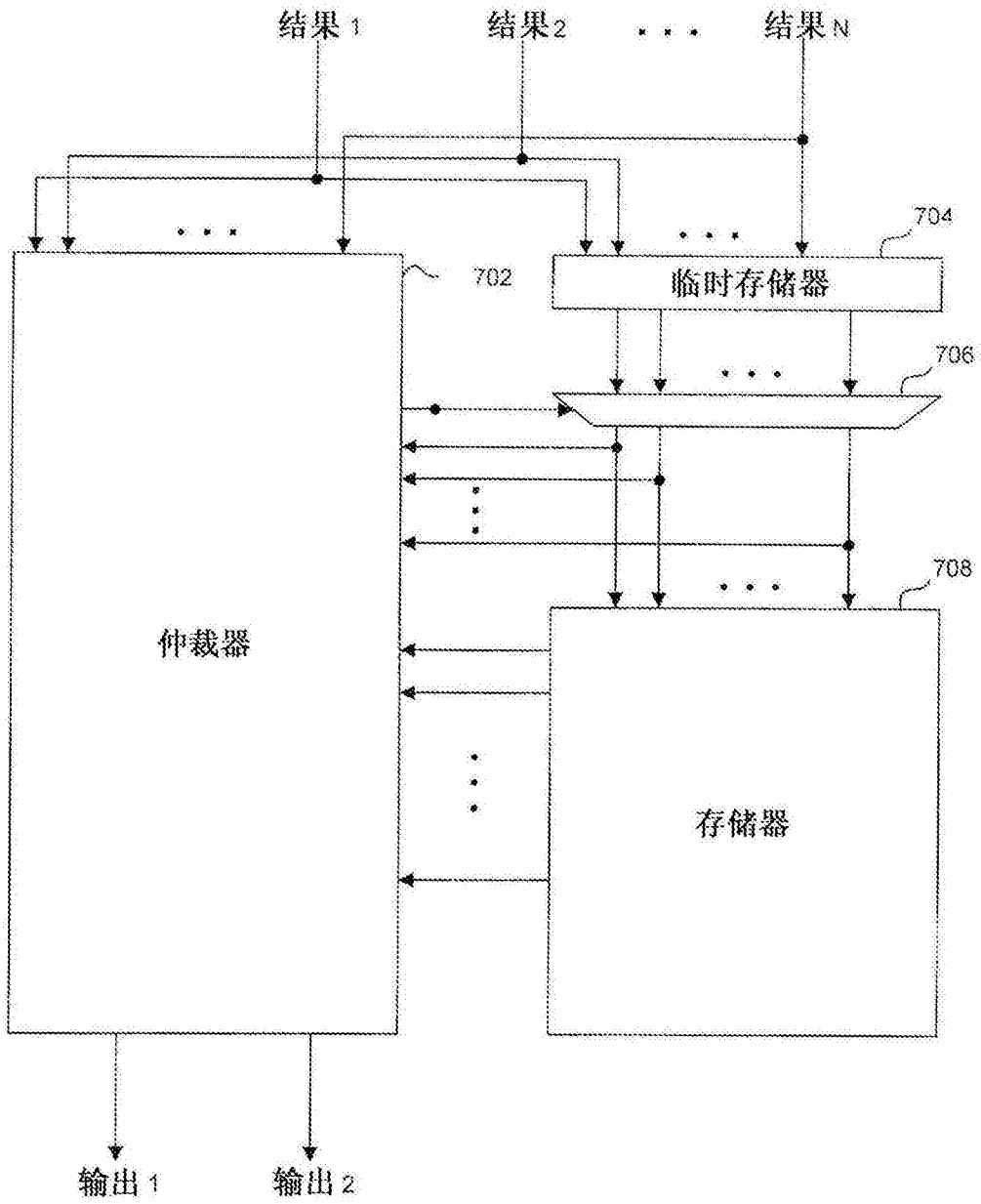


图7

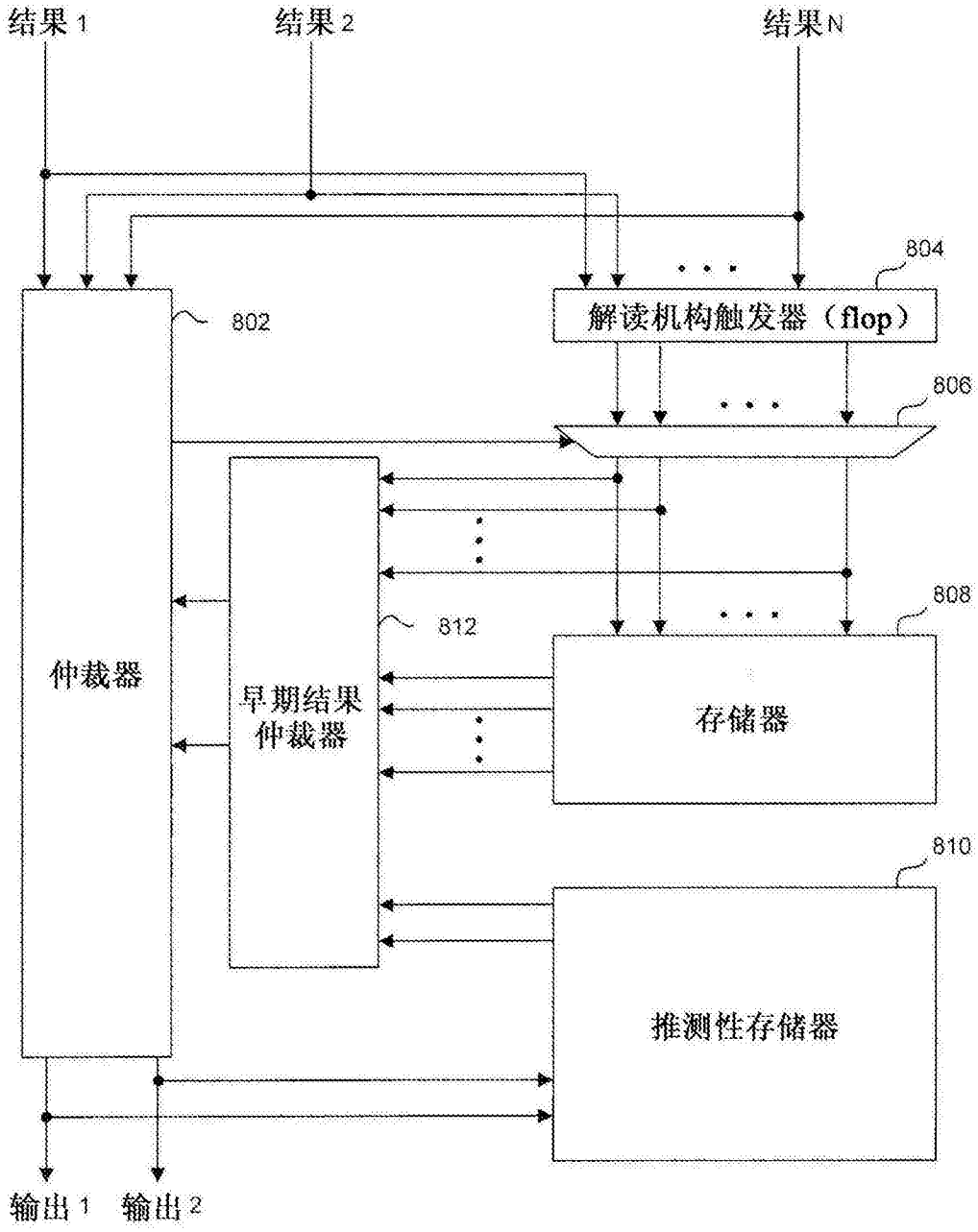

结果 1
结果 2
结果N
804
802
解读机构触发器（flop）
806
808
812
仲裁器
早期结果
仲裁器
存储器
810
推测性存储器
输出 1
输出 2

图8

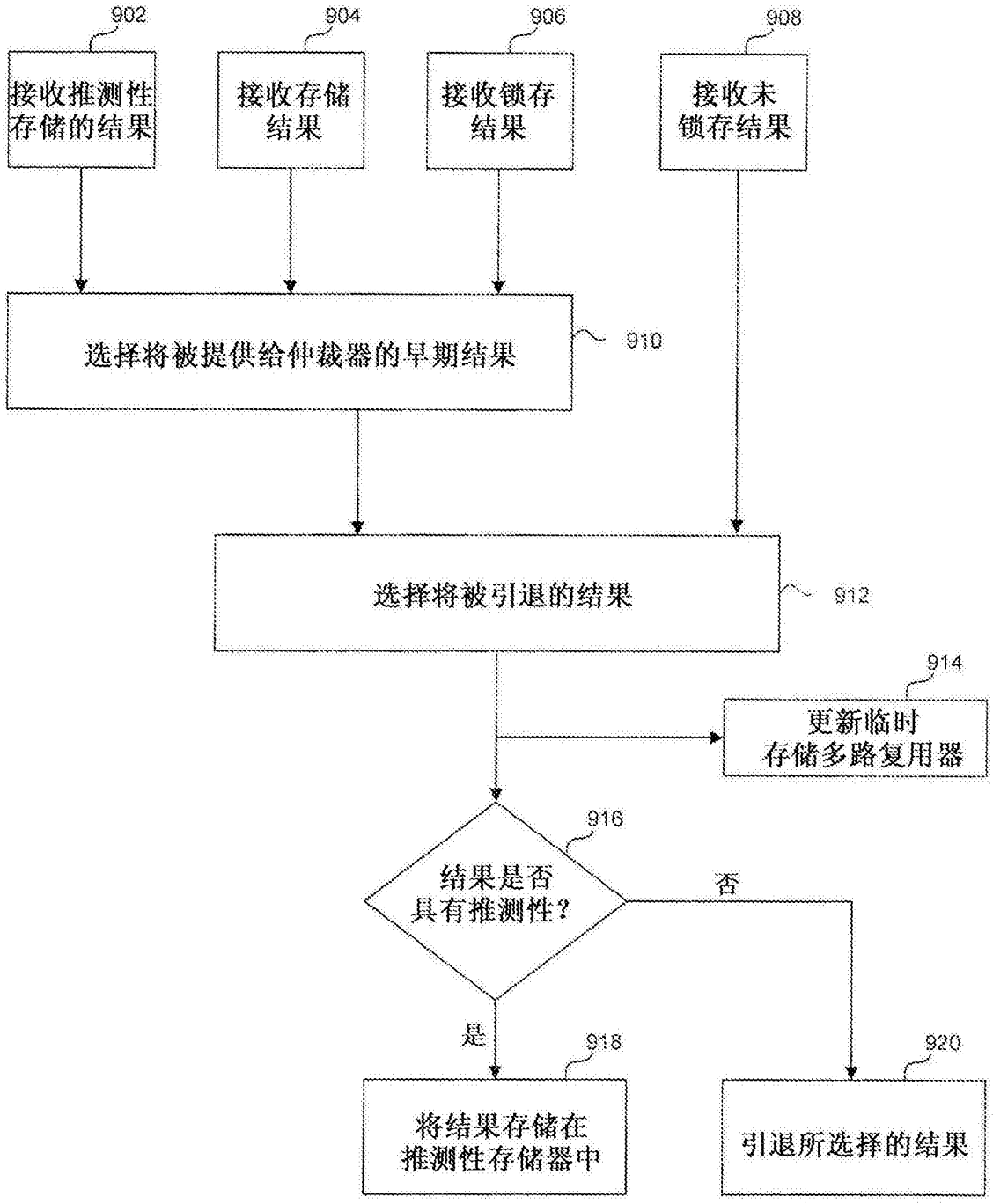

902
接收推测性存储的结果
904
接收存储结果
906
接收锁存结果
908
接收未锁存结果
选择将被提供给仲裁器的早期结果
910
选择将被引退的结果
912
914
更新临时存储多路复用器
916
结果是否具有推测性？
否
是
918
将结果存储在推测性存储器中
920
引退所选择的结果

图9

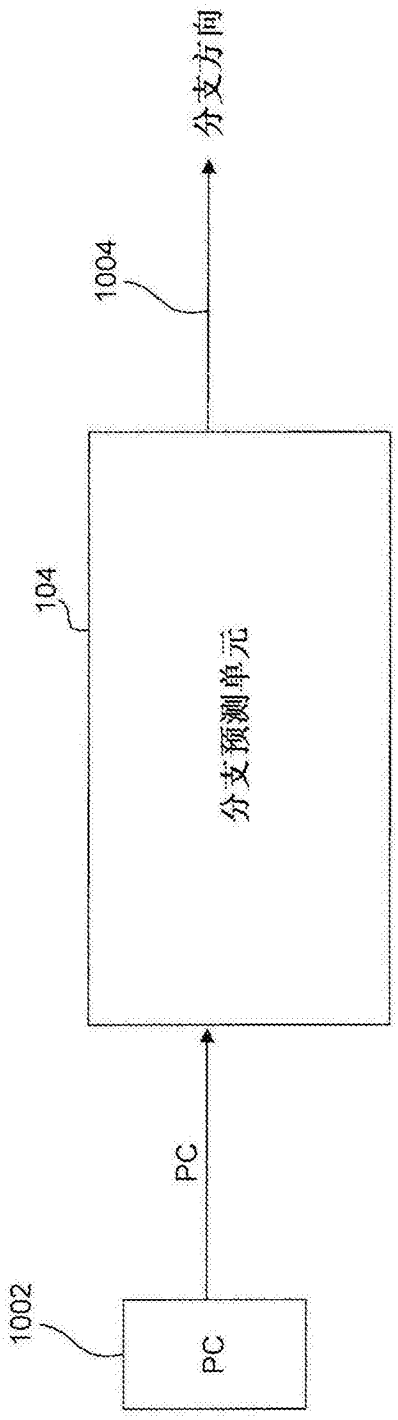


图10

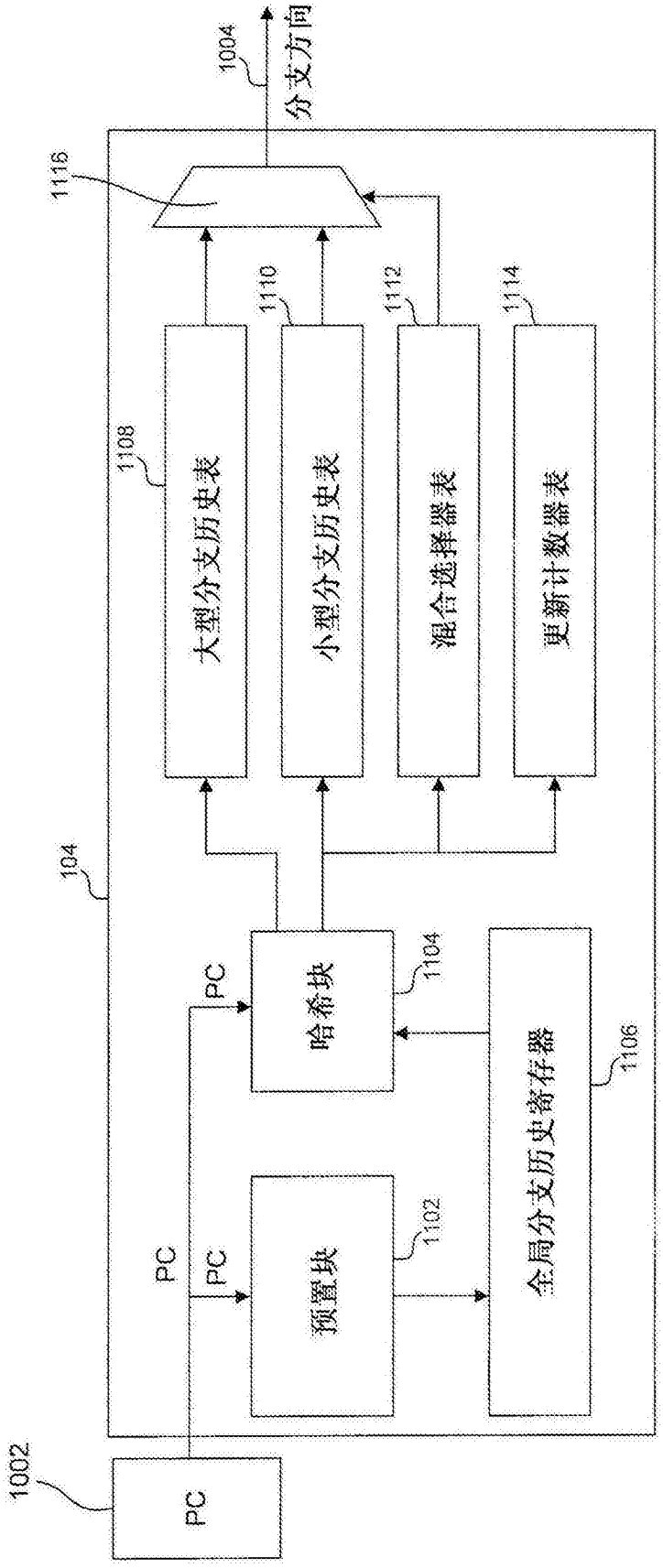

1004
分支方向
1116
1110
1112
1114
1108
大型分支历史表
小型分支历史表
混合选择器表
更新计数器表
104
PC
哈希块
1104
1106
全局分支历史寄存器
PC
PC
预置块
1102
1002
PC

图11

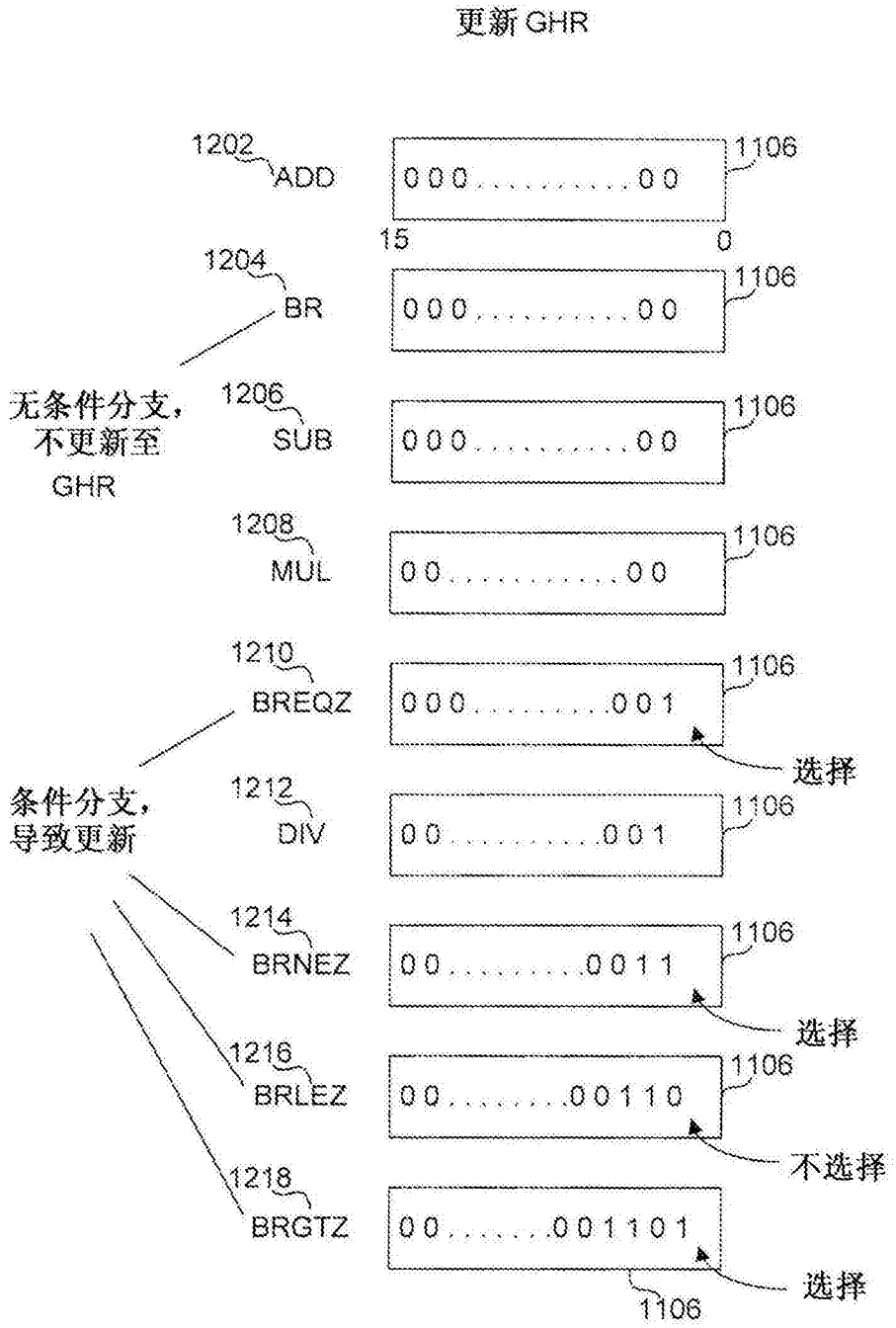

更新 GHR
1202
ADD
0 0 0 . . . . . 0 0
1106
15
0
1204
BR
0 0 0 . . . . . 0 0
1106
无条件分支，
不更新至
GHR
1206
SUB
0 0 0 . . . . . 0 0
1106
1208
MUL
0 0 . . . . . 0 0
1106
1210
BREQZ
0 0 0 . . . . . 0 0 1
1106
选择
条件分支，
导致更新
1212
DIV
0 0 . . . . . 0 0 1
1106
1214
BRNEZ
0 0 . . . . . 0 0 1 1
1106
选择
1216
BRLEZ
0 0 . . . . . 0 0 1 1 0
1106
不选择
1218
BRGTZ
0 0 . . . . . 0 0 1 1 0 1
选择
1106

图12

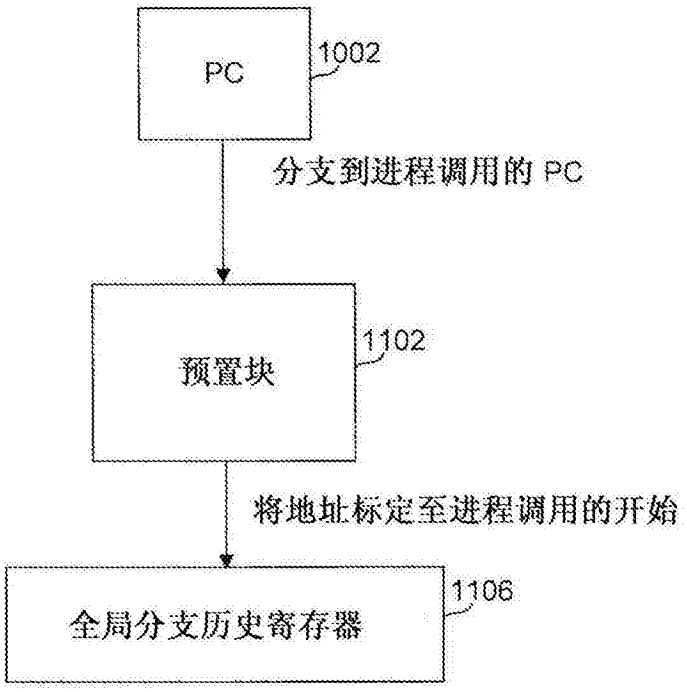


图13

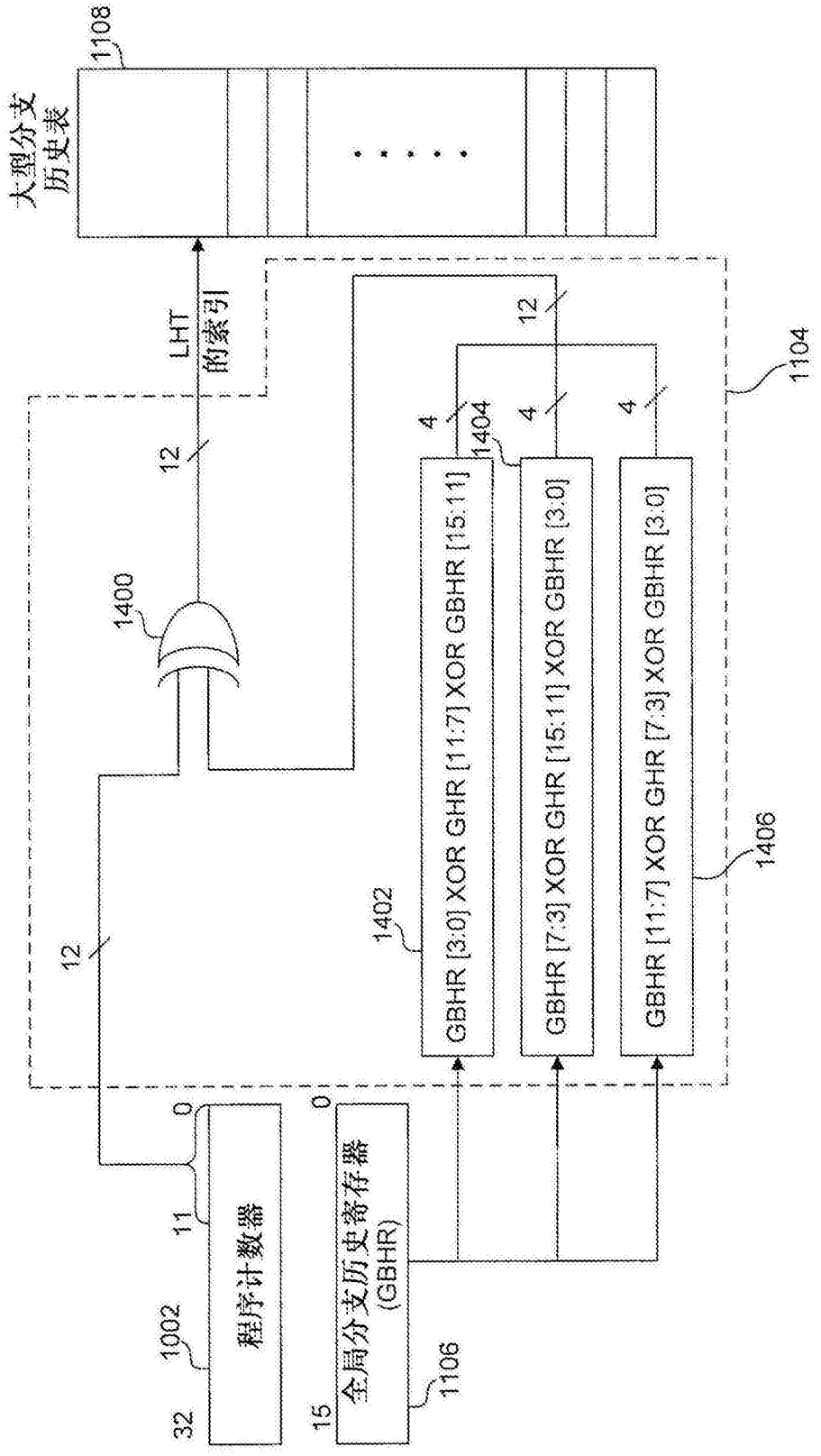

1108
大型分支历史表
LHT的索引
12
1400
1104
12
4
4
4
1404
GBHR [3:0] XOR GHR [11:7]
GBHR [7:3] XOR GHR [15:11]
GBHR [7:3] XOR GHR [3:0]
GBHR [11:7] XOR GHR [7:3]
GBHR [3:0] XOR GHR [3:0]
1406
1402
12
0
11
程序计数器
1002
32
0
15
全局分支历史寄存器 (GBHR)
1106

图14

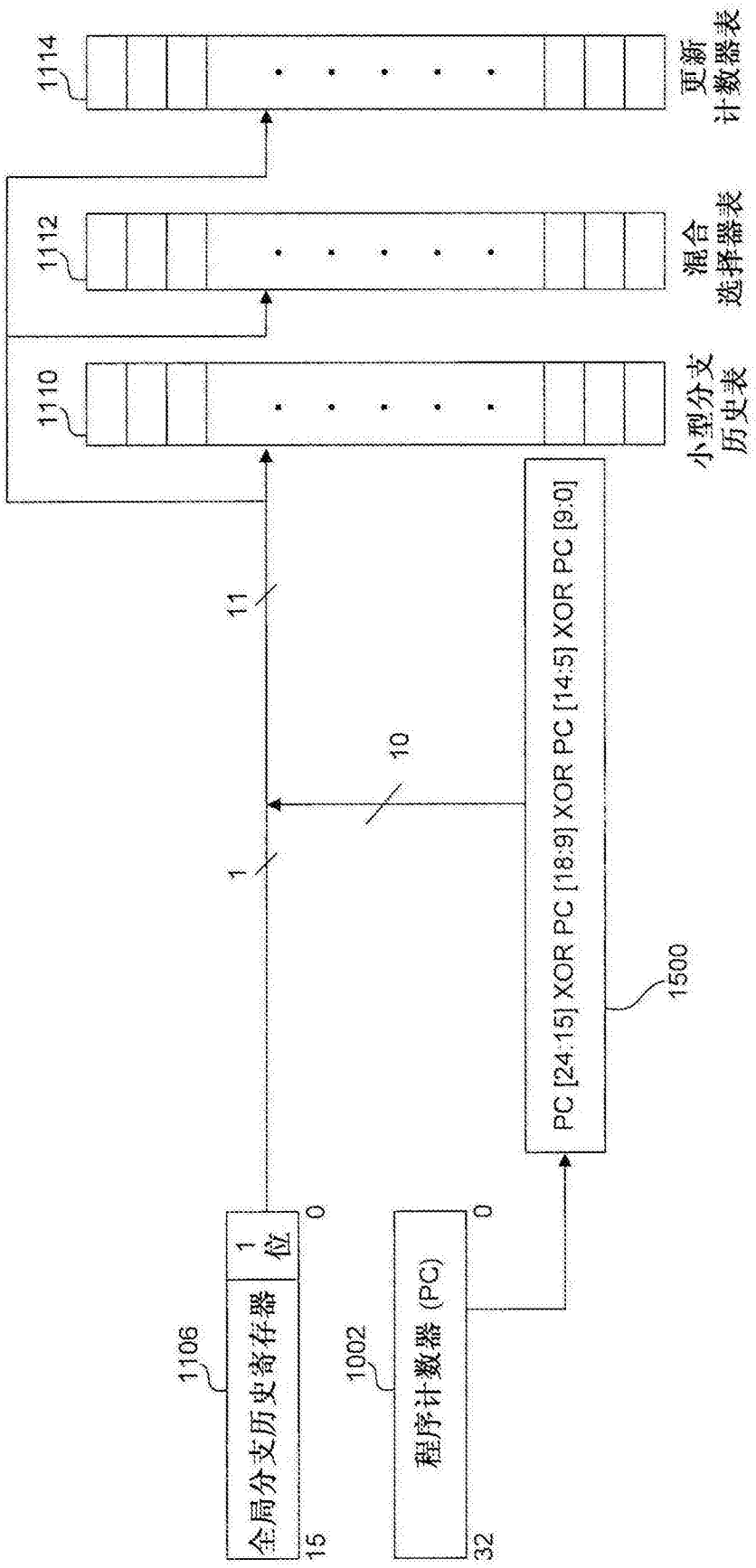

1114
更新计数器表
1112
混合选择器表
1110
小型分支历史表
11
10
1
PC [24:15] XOR PC [18:9] XOR PC [14:5] XOR PC [9:0]
1500
1106
全局分支历史寄存器
1位
0
15
1002
程序计数器 (PC)
0
32

图15

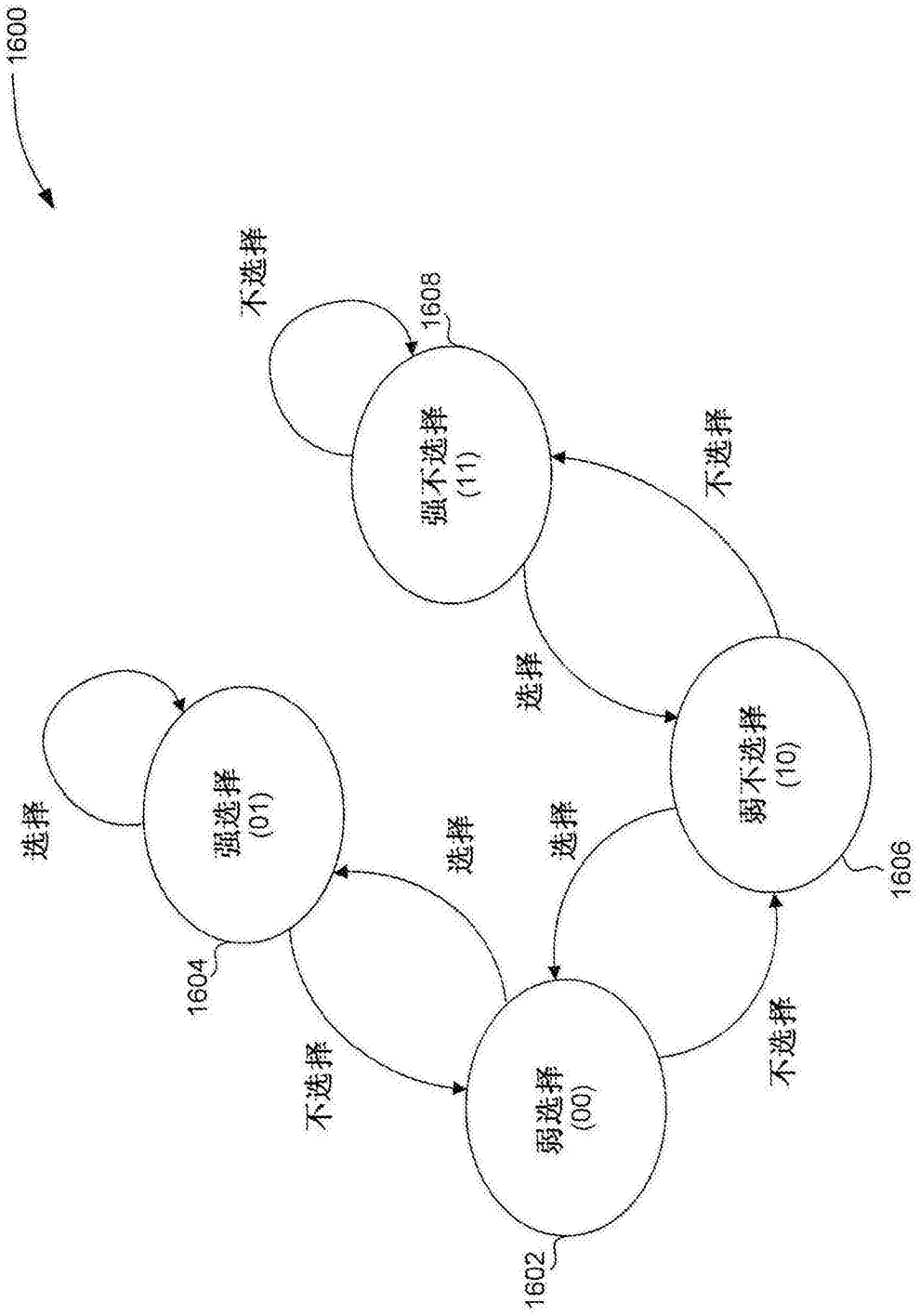

1600
不选择
1608
强不选择
(11)
不选择
选择
选择
强选择
(01)
选择
选择
选择
弱不选择
(10)
1606
1604
不选择
不选择
弱选择
(00)
1602

图16

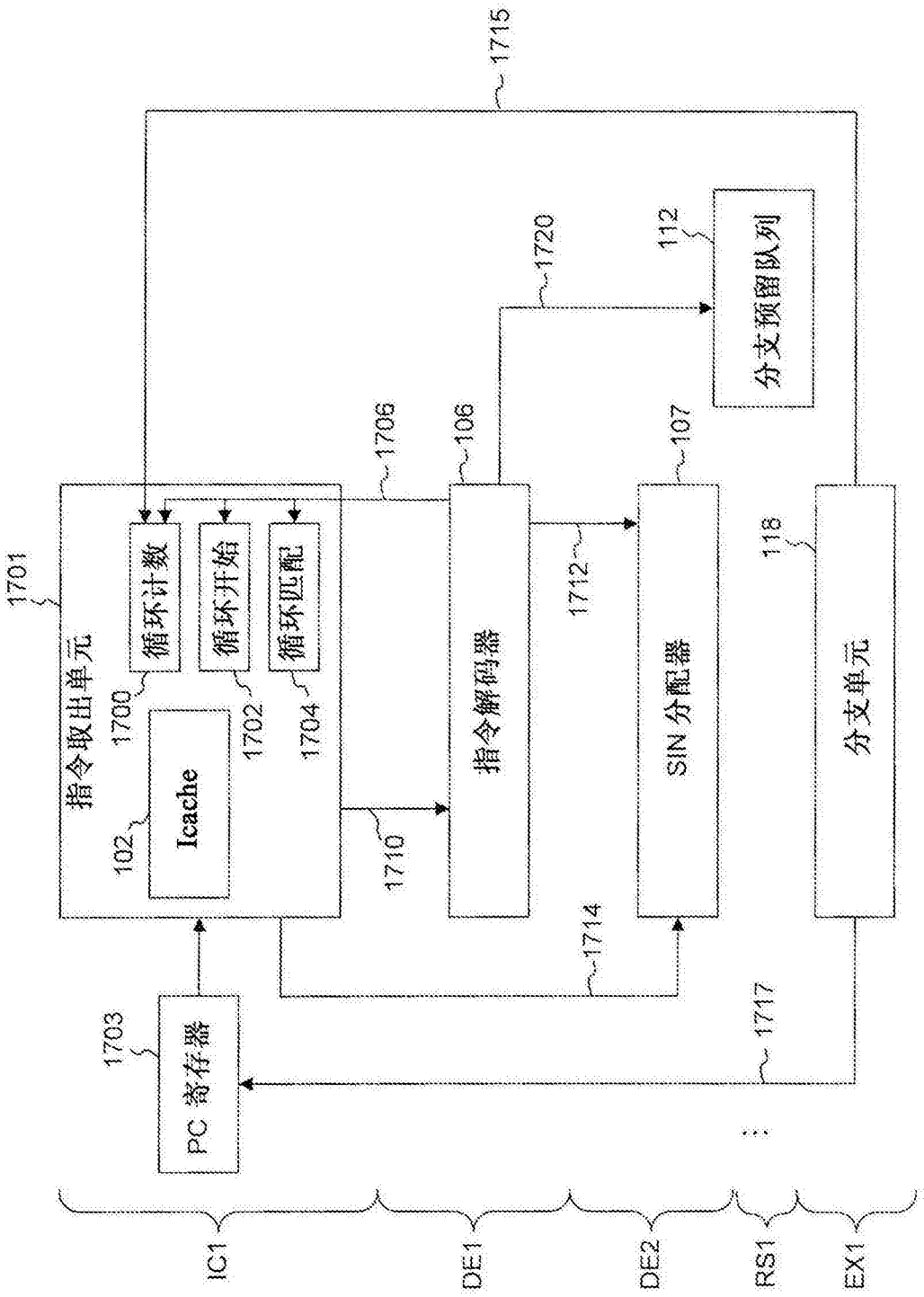

1715
1720
112
分支预留队列
1706
106
107
1701
指令取出单元
循环计数
循环开始
循环匹配
指令解码器
SIN 分配器
118
分支单元
1700
1702
1704
1712
102
Icache
1710
1714
1717
1703
PC 寄存器
IC1
DE1
DE2
RS1
EX1

图17

	IC1	DE1	DE2	RS1	RS2	EX1
0	ZLOOP					
1	INST 1 循环计数 = -1	ZLOOP (设置循环开始，循环匹配，&循环计数寄存器)				
2	INST 2 循环计数 = -1	INST 1	ZLOOP (SIN #0) 生成 SIN #1			
3	循环匹配 Inst 3 循环计数 = -1	INST 2	INST 1 (SIN #1)	ZLOOP (SIN #0)		
4	INST 1 循环计数 = -2	循环匹配 INST 3 (生成合成分支指令)	INST 2 (SIN #1)	INST 1 (SIN #1)	ZLOOP (SIN #0)	
5	INST 2 循环计数 = (循环计数 + R_0) = (-2 + 10) = 8	INST 1	循环匹配 INST 3 (SIN #1) (生成 SIN #2)	INST 2 (SIN #1)	INST 1 (SIN #1)	ZLOOP (SIN #0)
6	循环匹配 Inst 3 循环计数 = 8	INST 2	INST 1 (SIN #2)	循环匹配 INST 3 (SIN #1)	INST 2 (SIN #1)	INST 1 (SIN #1)
7	INST 1 循环计数 =7	循环匹配 INST 3	INST 2 (SIN #2)	INST 1 (SIN #2)	循环匹配 INST 3 (SIN #1)	INST 2 (SIN #1)
8	INST 2 循环计数 = 7	INST 1	循环匹配 INST 3 (不再生成 SIN)	INST 2 (SIN #2)	INST 1 (SIN #2)	循环匹配 INST3 (SIN #1)

图18