

在 CPU 中执行指令的方法

技术领域

[01]本说明书一个或多个实施例涉及计算机硬件芯片领域,尤其涉及在 CPU 中执行指令的方法。

5 背景技术

[02]在当前的大数据云环境下,需要对海量数据进行存储和处理,对数据的计算速度提出了更高的要求。众所周知,计算速度的决定性因素为中央处理单元 CPU 的性能。为了实现更高速度的运算,CPU 在各个方面,从物理工艺到逻辑控制,都在不断进行改进。

[03]例如,为了提升并行处理能力,提出 CPU 超线程技术,即利用特殊字符的硬件指令,把两个逻辑内核模拟成物理芯片,让单个处理器能使用线程级并行计算,从而兼容多线程并行计算。也就是说,超线程 CPU 可以在一个物理核的基础上,并行运行 2 个或多个线程,从而得到更多可并行指令,提升总体运行性能。

[04]另一方面,为了更有效地利用 CPU 的时钟周期,避免流水线停顿或等待,采用指令预测的方案,进行指令预取和指令预执行。

15 [05]这些方案都在一定程度上提升了 CPU 的执行效率。然而,指令预测并不总是准确,在指令预测失误的情况下,会严重降低 CPU 的执行效率。

[06]因此,希望能进行进一步改进,从而进一步提升 CPU 效率。

发明内容

20 [07]本说明书一个或多个实施例描述了一种 CPU 中执行指令的方法,在原有指令预测的基础上,避免预测错误的指令被执行,进一步提升 CPU 执行效率。

[08]根据第一方面,提供了一种在 CPU 中执行指令的方法,包括:

[09]从当前线程队列中依次提取指令形成指令块,以送入 CPU 执行单元执行,所述指令块中包含单条跳转指令,且该跳转指令为所述指令块中最后一条指令;

[10]将至少一条指令补充到当前线程队列,形成待执行线程队列;

25 [11]根据 CPU 执行单元的执行结果,确定所述跳转指令的目标指令;

[12]判断所述待执行线程队列中是否包含所述目标指令;

[13]在所述待执行线程队列中不包含所述目标指令的情况下,清除所述待执行线程队列,获取所述目标指令,并将所述目标指令添加到待执行线程队列中。

[14]根据一个实施例,通过以下方式形成指令块:

- 5 [15]从当前线程队列中读取预定阈值数目的指令,所述预定阈值数目依赖于所述 CPU 执行单元的数目;判断所述预定阈值数目的指令中是否包含跳转指令;在包含跳转指令的情况下,以该跳转指令为结尾向前截取,将截取的指令作为所述指令块。

[16]根据另一实施例,通过以下方式形成指令块:

- 10 [17]从当前线程队列中读取第一指令;在当前指令块中指令数目未达到预定阈值的情况下,将所述第一指令添加到当前指令块中,其中所述预定阈值依赖于所述 CPU 执行单元的数目;判断该第一指令是否为跳转指令;在该第一指令为跳转指令的情况下,将所述当前指令块作为所述指令块。

[18]在一种可能的设计中,通过以下方式补充指令到当前线程队列:根据指令预测的预测分支,将预测分支对应的至少一条指令补充到当前线程队列。

- 15 [19]在一种可能的方案中,从已解码缓存中读取对应的指令补充到当前线程队列,以形成待执行线程队列,其中已解码缓存中存储有预取并解码的多条指令。

[20]在一种可能的实施方式中,跳转指令为寄存器操作指令,所述指令块还包含至少一条内存操作指令。

- 20 [21]进一步地,在一个实施例中,在所述至少一条内存操作指令执行完毕之前,即可确定出所述跳转指令的目标指令。

[22]根据一种可能的设计,通过以下方式获取目标指令:

[23]判断已解码缓存中是否包含所述目标指令,其中已解码缓存中存储有预取并解码的多条指令;

[24]在包含的情况下,从所述已解码缓存获取所述目标指令;

- 25 [25]在不包含的情况下,从内存获取所述目标指令。

[26]根据第二方面,提供一种 CPU 控制器,包括:

[27]指令提取逻辑,用于从当前线程队列中依次提取指令形成指令块,以送入 CPU 执行

单元执行,所述指令块中包含单条跳转指令,且该跳转指令为所述指令块中最后一条指令;

[28]指令补充逻辑,用于将至少一条指令补充到当前线程队列,形成待执行线程队列;

[29]目标确定逻辑,用于根据 CPU 执行单元的执行结果,确定所述跳转指令的目标指令;

5 [30]判断逻辑,用于判断所述待执行线程队列中是否包含所述目标指令;

[31]队列操作逻辑,用于在所述待执行线程队列中不包含所述目标指令的情况下,清除所述待执行线程队列,获取所述目标指令,并将所述目标指令添加到待执行线程队列中。

[32]根据第三方面,提供一种中央处理单元 CPU,包含第二方面的控制器。

[33]通过本说明书实施例提供的方案,按照原有方式进行指令预取,放到已解码缓存和
10 线程队列中,但是执行时,确保同时执行的代码块中至多在最后包含一条跳转指令,并且在跳转指令确定出目标指令之前,不再为线程队列中的后续指令进行重命名&分配可执行资源阶段。在确定出目标指令之后,比较线程队列中的指令与目标指令是否匹配,以此保证只有正确分支得到执行。如此,在利用原有指令预测方案的优势的基础上,避免预测错误的指令被执行导致的回滚耗时,提升 CPU 总体执行效率。

15 附图说明

[34]为了更清楚地说明本发明实施例的技术方案,下面将对实施例描述中所需要使用的附图作简单地介绍,显而易见地,下面描述中的附图仅仅是本发明的一些实施例,对于本领域普通技术人员来讲,在不付出创造性劳动的前提下,还可以根据这些附图获得其它的附图。

20 [35]图 1 为根据一种实施方案的 CPU 执行过程;

[36]图 2 示出根据一个实施例在 CPU 中执行指令的方法;

[37]图 3 示出根据一个实施例的 CPU 控制器的功能框图。

具体实施方式

[38]下面结合附图,对本说明书提供的方案进行描述。

25 [39]图 1 为根据一种实施方案的 CPU 执行过程。如图 1 所示,整个执行过程分为多个阶段。首先是取指令阶段。当前的主流 CPU 每个指令周期可以取 16 字节,大约每次为 4

条指令。接着进行指令预解码。预解码阶段的主要工作是辨别指令长度，同时标注跳转指令。通常来说，主流 CPU 在该阶段有 5 指令/周期的吞吐量。

[40] 预解码之后即进入解码阶段。解码阶段主要将复杂指令转变为精简指令(固定长度)，同时指定操作类型。通常该阶段也有 5 指令/周期的吞吐量。 解码后的指令会放入到已
5 解码缓存。

[41] 已解码缓存作为一个指令缓存池，其中可以存储多条已解码的指令，供下一阶段，即线程队列，读取。已解码缓存到下一阶段的吞吐量可以达到每个周期 6 跳指令。

[42] 如前所述，对于超线程 CPU，可以存在多个线程并行执行。在执行过程中，每个线程都会读取接下来待执行的指令，形成自己的线程缓存队列，后续又称为线程队列。在
10 已解码缓存中存在上述待执行指令的情况下，就使用已解码缓存中存储的指令，否则，从前端（内存）得到相应指令添加到队列中。在图 1 中示例性示出了线程 A 和线程 B 各自的线程队列，但是可以理解，超线程 CPU 还可以支持更多线程的并行执行。

[43] 接着，从形成线程队列进入下一阶段：重命名&分配可执行资源。从线程队列到该阶段的吞吐量可以达到每个周期 5 跳指令。而在重命名&分配可执行资源阶段，主要工
15 作是解决寄存器读写依赖关系，去除不必要的依赖关系，力求得到指令更多并行执行能力，同时分配执行时所需要的各种资源。

[44] 在分配好执行所需资源之后，指令才会被送入到 CPU 的执行单元进行执行。目前 CPU 拥有多个执行单元，当前最普遍的 CPU 具有 8 条可以并行执行的流水线，也就是
20 每个周期可以执行 8 个微操作，虽然可以乱序执行，但是最后指令提交的顺序与程序的顺序相同。

[45] 如前所述，为了避免指令缺失带来的流水线停顿或等待，目前几乎所有 CPU 都会采用指令预测，又称为分支预测（Branch Prediction）方案进行指令的预测和预取。在每个
25 周期结束之后，预测单元根据其包含的历史执行状态表预测将要预取的指令。如果指令没有跳转，在前述取指令阶段，就取当前取指地址加 16 字节的指令块。如果指令存在跳转，则根据指令预测结果，获取预测分支的指令。

[46] 经过不断改进，当前的指令预测方案的预测准确度已经可以超过 90%，有些方案的预测准确度甚至可达 98%。但是，仍然存在预测错误的可能，此时很有可能将错误的指令块输入到可执行单元中。

[47] 例如，假定存在指令 L1, L2, L3, L4, L5，其中 L2 是跳转指令，该指令规定，在

某判断条件满足时，跳转至指令 L5，否则顺序执行指令 L3 和 L4。如果在指令预测时，预测该跳转指令 L2 的目标分支是 L3，那么在取指令阶段就会读取 L3 和后续指令，并且在后续执行阶段，有可能将 L1，L2，L3，L4 送入 CPU 执行单元进行执行。如果实际上 L2 的执行结果指示，应该跳转到 L5，那么 L3 和 L4 就被错误地执行。在这样的情况下，CPU 不得不重新刷新整条流水线，回滚到之前的分支，然后重新热启动，选择另一条分支执行。尽管指令预测错误的概率并不高，但是一旦出现，就需要进行上述操作，这样的操作非常耗时，致使 CPU 效率最大只能在 75% 左右。

[48] 为此，本说明书的实施例在此基础上进行进一步改进，尽可能保留和利用高准确率的指令预测所带来的优势，同时防止预测失败情况下错误指令被执行。根据一个或多个实施例，仍然按照原有方式进行取指令，同时放到已解码缓存和线程队列中，但是在跳转指令获取到有效的目标地址，也就是确定出目标指令之前，不再执行代码块的重命名 & 分配可执行资源阶段，保证后续执行操作都是正确完成，没有预测失败而导致的效率下降。下面描述上述构思的实现方式。

[49] 图 2 示出根据一个实施例在 CPU 中执行指令的方法流程图，如图 2 所示，该方法包括：步骤 21，从当前线程队列中提取指令形成指令块，送入 CPU 执行单元执行，所述指令块中包含单条跳转指令，且该跳转指令为该指令块中最后一条指令；步骤 22，将至少一条指令补充到当前线程队列，形成待执行线程队列；步骤 23，根据 CPU 执行单元的执行结果，确定所述跳转指令的目标指令；步骤 24，判断所述待执行线程队列中是否包含所述目标指令；在不包含所述目标指令的情况下，在步骤 25，清除所述待执行线程队列，获取所述目标指令，并将所述目标指令添加到待执行线程队列中。下面描述以上各个步骤的具体执行方式。

[50] 如前所述，根据本说明书的实施例，为了利用原有指令预测方案的优势，仍然按照原有方式进行取指令，同时放到已解码缓存和线程队列中。也就是说，按照原有方式执行图 1 的取指令阶段、预解码阶段、解码阶段，并将解码的指令放入已解码缓存中。各个线程可以从已解码缓存中读取指令，形成线程队列。因此，假定在步骤 21 之前，已经按照已有方式形成了线程队列。

[51] 在步骤 21，从当前线程队列中提取指令形成指令块，以送入 CPU 执行单元执行。

[52] 如果从当前线程队列提取的指令不包含跳转指令，那么仍按照正常方式，形成与硬件最大处理能力对应的最大长度的指令块。通常，CPU 硬件的最大处理能力取决于包含的执行单元的数目，可以根据执行单元的数目确定一预定阈值，作为指令块的最大长度。

例如，当前最普遍的 CPU 具有 8 条可以并行执行的流水线，那么可以将该预定阈值设为 8，相应地，指令块的最大长度为 8。在提取的指令不包含跳转指令的情况下，可以仍按照正常方式提取 8 条指令作为指令块。

5 [53]与常规方案不同的，在要提取的指令包含跳转指令时，则要确保送入 CPU 执行单元的一个指令块只包含一条跳转指令，且该跳转指令为指令块中最后一条指令。也就是说，在从线程队列向 CPU 执行单元传送指令时，对指令类型进行判断，并以跳转指令为界划分指令块，使得跳转指令作为送入 CPU 执行单元的一组指令中的最后一条指令。

10 [54]可以采用多种方式形成上述指令块。在一个实施例，从当前线程队列中一次读取预定阈值数目的指令，该预定阈值数目对应于 CPU 最大处理能力，或者说依赖于 CPU 执行单元的数目。然后，判断这些指令中是否包含跳转指令。如果不包含，如前所述，则将这些读取的指令作为上述指令块。如果这些指令中包含跳转指令，那么以该跳转指令为结尾向前截取，将截取的指令作为所述指令块。

15 [55]例如，假定预定阈值数目为 8，则一次从当前线程队列中读取 8 条指令。如果这 8 条指令中没有跳转指令，则直接将这 8 条指令作为上述指令块，送入 CPU 执行单元。如果这 8 条指令中包含跳转指令，那么从跳转指令的位置向前截取，来形成指令块。例如，如果第 5 条指令为跳转指令，那么就从以第 5 条指令为结尾向前截取，也就是将第 1 条到第 5 条指令作为指令块。

20 [56]通过以上方式，确保跳转指令一定是送入 CPU 执行单元进行执行的一组指令中的最后一条指令。可以理解，在执行跳转指令之前，无法准确确定下一条要执行的指令，即目标指令，而指令预测方案是将预测的目标指令预取到线程队列中。如果确保跳转指令是一次送入 CPU 执行单元的指令组中的最后一条指令，相当于在该跳转指令与后续的预测的目标指令之间建立了一个隔离或中断，确保预测的目标指令不会与该跳转指令一起被送入 CPU 执行单元执行。如此，为准确确定目标指令，进而在预测失误的情况下纠正错误的目标指令提供了时机和可能性。

25 [57]例如，在前述例子中，指令 L1, L2, L3, L4, L5 中 L2 是跳转指令。即使跳转指令 L2 的目标分支被错误地预测为 L3，那么根据以上实施例，只会将 L1 和 L2 作为一个指令块，送入 CPU 执行单元进行执行，而不会同时将 L1, L2, L3, L4 一起执行。在执行 L1 和 L2 时，就为确定 L2 的准确目标分支，并纠正错误的预测分支提供了时机。

[58]如上，从线程队列中提取出至少一部分指令送入 CPU 执行单元，那么线程队列中待

执行指令数目会暂时变少。因此，在形成指令块送入 CPU 执行单元的同时或之后，对线程队列进行补充，以维持其队列长度。也就是，在步骤 22，将至少一条指令补充到当前线程队列，形成待执行线程队列。可以理解，待执行线程队列用于形成送入 CPU 执行单元执行的下一指令块。

5 [59]根据一种实施方式，在该步骤中，可以按照常规方式，根据指令预测的预测分支，向线程队列补充指令，形成待执行线程队列。在一个实施例，根据指令预测结果，从已解码缓存中读取对应的指令补充到当前线程队列，其中已解码缓存中存储有预取并解码的多条指令。在极为罕见的情况下，也可以从前端（例如内存）请求指令，解码后补充到线程队列中。

10 [60]另一方面，在步骤 21 形成的指令块被送入 CPU 执行单元后，CPU 执行单元就会将这些指令加入流水线，进行执行。特别地，上述指令块的最后一条指令为跳转指令，通过执行该跳转指令，才可以准确确定跳转的目标地址，也就是确定出目标指令。也就是，在步骤 23，根据 CPU 执行单元的执行结果，确定跳转指令的目标指令。

[61]接着，在步骤 24，判断步骤 23 中补充后的待执行线程队列中是否包含上述目标指令。如果包含，则意味着，接下来要执行的目标指令已经被放入待执行线程队列中，指令预测结果是正确的，不需要进行额外操作；当前指令块执行完毕之后可以按照图 2 的方法继续从线程队列取得下一指令块进行执行。

[62]但是，如果待执行线程队列中不包含上述目标指令，那就意味着，接下来应该执行的指令（目标指令）没有被放入线程队列，反过来说，待执行线程队列中包含的指令序列并不是接下来应该被执行的指令。造成的原因可能就是指令预测失误，而将错误的分支指令预取到了线程队列中。在这样的情况下，在步骤 25，清除（flush）当前的待执行线程队列，获取所述目标指令，并将所述目标指令添加到待执行线程队列中。

25 [63]具体而言，在步骤 25，由于当前的待执行线程队列包含了错误的指令，不应该被执行，因此首先对当前的待执行线程队列进行清除（flush）操作。清除 flush 是 CPU 中的一种操作，可以对操作对象中存储的数据全部清理。清除当前的待执行线程队列就意味着，删除队列中的所有指令，将该队列清空。

[64]与清除错误的指令相应地，还要获取正确的目标指令，将其添加到待执行线程队列中。

[65]在一个实施例中，首先判断已解码缓存中是否包含正确的目标指令；在包含的情况

下，从已解码缓存获取该目标指令。可以理解，尽管之前的线程队列添加了错误的分支指令，但是往往只是指令执行顺序上的错误，而基于指令预测方案的指令预取会不断地预取到许多条指令，解码后放入已解码缓存。因此，在绝大多数的情况下，都可以从已解码缓存中获取到正确的目标指令，将其添加到待执行线程队列。进一步地，在添加目标指令之外，还相应地将该目标指令所在分支的后续指令也添加到待执行线程队列中。

[66]另一方面，在极其罕见的情况下，已解码缓存中不包含目标指令。这时，可以从内存请求获取该目标指令，解码后加入到待执行线程队列中。

[67]经过步骤 25 的操作，确保待执行线程队列中包含有待执行的正确指令分支，于是后续送入 CPU 执行单元的指令块也是正确的指令块，预测错误的指令不会被执行单元真正执行。

[68]下面结合一个具体例子描述以上过程。假定存在以下一段指令（其中/*...*/中间的内容是对指令的解释）：

```

Loop: 1. fld f0,0(x1) /*按照 x1 存的地址，将内存内容传到寄存器 f0 */
      2. fadd.d f4,f0,f2 /*将 f0+f2 的结果存入到寄存器 f4 */
      3. fsd f4, 0(x1) /*按照 x1 存的地址，将 f4 中的数据存于内存中*/
      4. addi x1,x1-8 /*x1 的内容-8 同时存于寄存器 x1 */
      5. bne x1, x2, Loop /*如果 x1 不等于 x2 跳转到 Loop, 执行指令 1, 否则进入指令 6
*/
      6. addi x2, x2+1/*x2 的内容+1 同时存于寄存器 x2 */
      7...
      8...

```

[69]在这一段指令中，指令 5 为跳转指令，根据 x1 与 x2 是否相等，跳转到不同分支，第一分支的目标指令为指令 1，第二分支的目标指令为指令 6。假定这些指令已经按照常规方式预取、解码，放入到已解码缓存。另外，假定根据指令预测的结果，指令 5 之后会执行第二分支，因此，将指令 1 到指令 8 依次放入到线程队列中。

[70]如果按照常规方式，则有可能将指令 1 到指令 6 作为一个指令块，送入 CPU 执行单元进行执行。一旦指令预测失误，指令 5 之后本应该执行第一分支，即指令 1，但是指令 6 被错误地执行，就会引起耗时的清除回滚操作，急剧降低 CPU 效率。

[71]根据本说明书实施例的方案，在步骤 21，由于指令 5 为跳转指令，只会将指令 1 到指令 5 形成一个指令块，送入 CPU 执行单元执行。

[72]在步骤 22，从已解码缓存中补充指令到线程队列，形成待执行线程队列。由于指令预测的分支是指令 6，因此这时待执行线程队列可以包括：指令 6，指令 7 以及补充进来的其他指令。

[73]在步骤 23，根据 CPU 执行单元的执行结果，可以确定跳转指令 5 的目标指令实际应该为第一分支中的指令 1。

[74]接着，在步骤 24，可以判断出，待执行线程队列中并不包含目标指令 1。这时，在步骤 25，清除待执行线程队列，从已解码缓存中获取指令 1，将其添加到待执行线程队列中，以按照正确分支重新形成待执行线程队列。

[75]如此，在以上实施例中，按照原有方式进行指令预取，放到已解码缓存和线程队列中，但是执行时，确保同时执行的代码块中至多在最后包含一条跳转指令，并且在跳转指令确定出目标指令之前，不再为线程队列中的后续指令进行重命名&分配可执行资源阶段。在确定出目标指令之后，比较线程队列中的指令与目标指令是否匹配，以此保证只有正确分支得到执行，不会产生预测失败而导致的效率下降。

[76]可以理解，CPU 执行效率的提高需要尽可能利用每个执行周期，减少流水线停顿和等待，避免无指令空操作。以上的方案为了防止指令预测失误导致的耗时回滚和效率下降，在确定跳转指令的目标指令之后，判断和比较线程队列，以确保后续操作的正确性。这样的“附加”操作是否会带来流水线等待或者空操作，进而影响 CPU 执行效率，是需要考虑的问题。不过，经过发明人的研究和分析，以上操作并不会造成 CPU 执行周期的浪费，影响执行效率。下面是对该问题的论证。

[77]首先本说明书的优化方案并不会带来中间数个周期的无指令空操作问题。通常，CPU 每个周期可以取得多条指令，例如 5 条指令。经过统计，在 5 条指令，平均含有 1 条跳转指令，1.4 条读内存指令，1 条写内存指令，以及 1.6 条计算操作。而统计表明，绝大多数跳转指令都会依赖简单的寄存器操作，因此大多数跳转指令可在 1 个周期完成。

[78]而另一方面，任何一条内存操作都会有较长周期的延迟。具体地，如果内存操作中 L1 缓存，则需要 4 个周期的延迟；如果命中 L2 缓存，则需要 12 个周期的延迟；如果没有命中缓存，需要到内存中读取数据，则至少需要 50 个周期的延迟。因此，跳转指令会率先执行完毕，因此可以尽早决定目的地址，而不会造成等待。

[79]例如,以上指令 1 到指令 6 的例子中,指令 1 到指令 5 形成指令块,送入执行单元。其中指令 1 需要从内存取得数据,即使 L1 缓存命中,也需要 4 个周期。由于依赖关系必须依次执行指令 2 和指令 3,该过程需要再次增加 2 个周期。而指令 4,5 为寄存器操作,可以在一个周期内完成,因此在指令 1,2,3 完成之前,就可以预先知道指令 5 是否需要跳转。

[80]此外,现有的主流 CPU 都是采用多线程操作,例如若有两个线程,CPU 会为每个线程轮流取指,比如先给线程 A 取指令,再给线程 B 取指令,再次为线程 A 取指令。这样对于线程 A 就会有一个周期的等待,在这样原本就有的等待周期之内,就可以确定出跳转指令的目标指令。

10 [81]实际上,当前在大数据云服务器环境下,大多数的性能问题集中于访问内存的延迟,而这种延迟往往达到上百个指令周期,这样的指令周期足以被利用来确定跳转指令的目标指令,并确认线程队列的正确性,而不影响 CPU 运行。因此通过上面的处理,该方案不会引入 CPU 无指令空操作和等待问题。

[82]另外,还需要考虑以上的优化方案与原有指令预测方案带来的指令预取问题和数据预取问题。原有的指令预测方法可以在指令预执行阶段运行没有确定的指令,从这个角度来看原有的方法会帮助 CPU 预取到很多将要执行的代码,减少指令缺失带来的延迟。在本方案中可以继续采用这样的方案,只不过只有在明确跳转指令的目标地址之后才执行正确的有效指令。关于数据预取,原有的指令预测执行方法可以在指令预执行阶段运行没有确定的指令,所以可以预先把即将用到数据从内存中放入 CPU 缓存。在本说明书的实施例中的也可以继续采用这样的已有方案,同时,只有在明确目标地址之后才执行有效指令,读取需要的数据。

15

20

[83]因此,本说明书实施例的方案,可以充分利用现有预测方法的优势,在此基础上结合当前 CPU 的各项已有的特征和主要运行环境,最终充分提升 CPU 可利用率,提升云计算集群吞吐量。

25 [84]如本领域技术人员所知,CPU 中指令的执行过程通过控制器来控制。控制器是整个 CPU 的指挥控制中心,用于协调各个部件之间的操作。控制器一般包括指令控制逻辑、时序控制逻辑、总线控制逻辑、中断控制逻辑等几个部分。指令控制逻辑要完成取指令、分析指令和执行指令的操作。

[85]根据以上描述的实施例的方案,对原有的指令控制过程进行了优化和调整,因此相

应地，可以在硬件层面上修改控制器电路，特别是其中的指令控制逻辑，使其完成以上实施例描述的控制过程。

[86]图 3 示出根据一个实施例的 CPU 控制器的功能框图。如图 3 所示，控制器 300 可以包括，指令提取逻辑 31，用于从当前线程队列中提取指令形成指令块，以送入 CPU 执行单元执行，所述指令块中包含单条跳转指令，且该跳转指令为所述指令块中最后一条指令；指令补充逻辑 32，用于将至少一条指令补充到当前线程队列，形成待执行线程队列；目标确定逻辑 33，用于根据 CPU 执行单元的执行结果，确定所述跳转指令的目标指令；判断逻辑 34，用于判断所述待执行线程队列中是否包含所述目标指令；队列操作逻辑 35，用于在所述待执行线程队列中不包含所述目标指令的情况下，清除所述待执行线程队列，获取所述目标指令，并将所述目标指令添加到待执行线程队列中。

[87]以上各个逻辑可以根据需要采用各种电路元件实现，例如采用若干比较器来实现判断逻辑等。

[88]通过以上的控制器，可以实现如图 2 所示的控制过程，从而在利用指令预测和预取的优势的基础上，防止和避免预测错误导致的效率下降，全面提升 CPU 执行效率。

[89]本领域技术人员应当理解，以上所述的具体实施方式，对本发明的目的、技术方案和有益效果进行了进一步详细说明，所应理解的是，以上所述仅为本发明的具体实施方式而已，并不用于限定本发明的保护范围，凡在本发明的技术方案的基础之上，所做的任何修改、等同替换、改进等，均应包括在本发明的保护范围之内。