[19]中华人民共和国国家知识产权局

[51]Int. Cl[7]
G06F 9/32
G06F 9/38

[12]发明专利申请公开说明书

[21]申请号 00815782.0

[43]公开日 2003年1月8日

[11]公开号 CN 1390324A

[22]申请日 2000.9.21 [21]申请号 00815782.0

[30]优先权

[32]1999.9.24 [33]DE [31]19945940.1

[86]国际申请 PCT/EP00/09267 2000.9.21

[87]国际公布 WO01/22217 德 2001.3.29

[85]进入国家阶段日期 2002.5.16

[71]申请人 印芬龙科技股份有限公司

地址 德国慕尼黑

[72]发明人 聂晓宁

[74]专利代理机构 中科专利商标代理有限责任公司

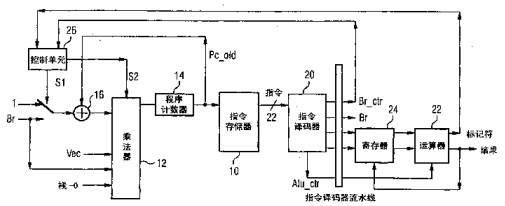
代理人 王仲贤

权利要求书1页 说明书10页 附图3页

[54]发明名称 在流水线结构的处理器中处理条件转移指令的方法和装置

[57]摘要

一种用于在具有流水线结构的处理器中处理条件转移指令的方法和装置,其中对每个根据其将执行条件转移的指令添加一个或多个附加的位,所述附加的位表明在哪种条件下执行条件转移。另外装置可以具有根据附加的用于执行条件转移的位改变程序计数读数的装置。



ISSN1008-4274

1．一种用于在一具有流水线结构的处理器中对条件转移指令处理的方法，其特征在于，对每个根据其将执行条件转移的指令添加一个或多个附加的位，所述附加的位表明在哪种条件下执行条件转移。

2．按照权利要求 1 所述的方法，其特征在于，对每个根据其将执行条件转移的指令附加添加相应的转移地址。

3．按照权利要求 1 或 2 所述的方法，其特征在于，对每个根据其将执行条件转移的指令附加添加一个或多个位，所述位表明在哪种条件下才执行指令。

4．按照权利要求 1 至 3 中任一项所述的方法，其特征在于，在设置相应的位的情况下，对每个具有一个或多个附加位的指令在执行该指令的同时在处理器中将检验与该位或这些位相应的标记符（例如 zero、carry、overflow），并根据检验结果相应地调整处理器的的程序计数器（PC）。

5．一种用于在具有流水线结构的处理器中处理条件转移指令的装置，其特征在于，具有一用于执行条件转移的改变程序计数读数的装置。

6．按照权利要求 5 所述的装置，其特征在于，用于改变程序计数读数的装置具有用于处理器的机器指令中相应的附加位的一个或多个输入端和具有用于处理器的运算器输出的相应的“标记符”信号的一个或多个输入端。

7．按照权利要求 6 所述的装置，其特征在于，机器指令的相应的附加位同时与用于改变程序计数的装置的相应的“标记符”同时存在。

8．按照权利要求 5 至 7 中任一项所述的装置，其特征在于，用于改变程序计数读数的装置具有一个加法器。

在流水线结构的处理器中处理条件转移指令的方法和装置

技术领域

本发明涉及一种在具有“流水线”结构的处理器中处理条件转移指令的方法和装置。

背景技术

用于执行特定指令的必要的周期的数量是处理器的最重要的性能特征之一。为了实现最大的处理速度和最小的功耗，周期的数量必须尽可能少。对此根据已有技术，具有所谓的“流水线”结构的处理器已公知。这意味着，处理器可以同时处理多个指令，其中每个指令分别位于一不同的处理阶段。例如当正在执行一个指令时，下一个指令被译码，并且同时从存储器中调取下下个指令。

特别是在这样一种用于条件转移指令（转移）的“流水线”结构中将导致“冒险”，因此有可能会出现甚至错误的结果。具体地说，在一条件转移指令的情况下，只有当该条件转移指令处理后才能确定下一个指令的地址。采用此方式时只有当从处理器的运算-逻辑单元中得到执行前一个指令的结果时，才能从存储器内调取下一个指令和译码。

迄今已有技术的对该“冒险”问题的解决方案是，紧接转移指令后插入多个空指令（NOP），即非工作-或等待指令，使结果在任何情况下都保持正确。由于必须对空指令进行处理，从而造成多个处理器周期被用于必须处理的空指令。

发明说明

所以本发明的目的在于实现在一个具有流水线结构的处理器中对条件转移指令的处理，而又不会由于空指令造成较大的处理周期的损失。

根据本发明，应用在一具有流水线结构的处理器中处理条件转移指

令的方法实现了该目的，其中对每个根据其将执行条件转移的指令添加一个或多个附加的位，所述附加的位表明在哪种条件下执行条件转移。采用此方式可以较早地建立一个是否执行转移的指令。因此，可以早期地确定一个作为紧接在转移后的下一个指令。所以利用在指令组中的转移预测可以较早地建立条件转移指令的转移目的。

其中特别优选的是，根据将进行条件转移的指令对每个指令附加添加相应的转移地址。采用此方式不仅较早地了解是否应执行条件转移的指令，而且还已经了解了相应的目的地址。因而可以由处理器的主存储器中调取正确的指令。

另外优选对每个指令附加添加一个或多个位，所述位表明在哪种条件下才执行该指令。

为了进一步对处理器的工作速度实现最佳化特别优选的是，当设置相应的位时，在执行每个具有一个或多个附加位的指令的同时，在处理器中将检验与该位或这些位相应的标记符（例如 zero, carry, overflow），并根据检验结果相应地调整处理器的的程序计数（PC）。

本发明的目的同样还通过一种在具有流水线结构的处理器中处理条件转移指令的装置得以实现，其中具有一用于执行条件转移的改变程序计数读出的装置。

其中特别优选的是，用于改变程序计数读出的装置具有针对处理器的机器指令中相应的附加位的一个或多个输入端和具有针对处理器的运算器输出的相应的“标记符”的一个或多个输入端。

其中特别有益的是，保证机器指令的相应的附加位存在的同时存在用于改变程序计数读出的装置的相应的“标记符”。

优选用于改变程序计数读出的装置具有一个加法器。

附图说明

下面将对照附图对本发明做详细的说明。图中示出：

图 1 为具有两级的流水线的处理器的流程图；

图 2 示出本发明的 22 位长的机器指令的结构；

图 3 示出本发明的 25 位长的机器指令的结构；

图 4 为本发明的为执行条件转移改变程序计数读出的装置的示意图；

图 5 示出本发明的另一用于执行条件转移的改变程序计数读出的装置；

图 6 为本发明的具有流水线结构的，利用转移预测执行转移指令的处理器的总体结构的示意图；和

图 7 为本发明的具有“转移预测”装置的处理器的详图。

具体实施方式

本发明以处理器的“流水线”结构为基础。对此在帕特森和亨尼西所著的“计算机的组织和设计”一书中做了说明。

对流水线结构简述如下：

通常利用下述运算由处理器对每个机器指令进行处理：

1．输入指令

2．对指令进行译码

3．执行指令

4．重录

根据已有技术已知，这些运算部分同时进行，其中例如正在执行一个指令，同时另一个指令被译码。在图 1 中的两级的流水线中对此做了表述。

因此处理器应用流水线，以便实现平均在一个处理器周期中处理一个指令。

但这种处理器的流水线结构在必须执行多个条件转移指令时，势必导致出现问题。该问题的专用术语为“转移冒险”。这意味着，一个转移指令，即一个条件转移指令只有在执行完在先的指令后才能表明，是否下一个指令将被继续处理或应被转移到另一个目的地址。

根据已有技术，该问题是通过对在条件转移指令后的时标填充一个“不工作”-指令，即填充等待一个处理器周期的指令解决的。虽然可以保证在任何情况下程序继续正确的运行，但将丧失一个处理器周期并因此丧失最大可能的计算机容量。下面将举例对迄今的已有技术做进一步

说明，所述举例分别涉及对一数字的绝对值的计算：

首先存在条件执行的可能性，例如

```
/* A=| B|  */
LOAD  R1  B
COMPARE R1 0 /*if B≥0, carry=0 */
NEGATIVE R1 on-carry /* negate, if carry=1 */
STORGE R1 A
```

如果只有唯一一个指令被条件地执行并且该指令不含有转移时，则该执行方式是可行的。当仅用一个指令不能表示的较为复杂的功能和任务情况下，则可以如在下面的程序中的描述，分别执行条件转移。如在下面黑框内的程序段所示，必须在两个转移指令之后加入一个“不工作”指令。在两级的流水线的情况下，当流水线较长时，则相应有较多的“不工作”指令：

```
LOAD     R1      B
COMPARE R1       0
JUMP ON CARRY   L1
JUMP     L2
NO OP
L1:  NEGATIVE   R1
L2:  STORE      R1  A
```

最后根据已有技术还有一种推断执行的方案。这意味着，可以简单地实施一种方案，并且假设，具有大约大于 50%的概率可以正确的继续进行。为此将付出很大的硬件代价，这是因为当预测不正确的情况下必须将一些指令“拆散”。另外当“估计错误”时，将因此丧失处理器周期。

而且迄今根据已有技术还没有对该问题相应的解决方案，该问题即这种“转移冒险”，在条件转移时出现的问题将导致流水线结构的处理器的工作周期的损失。根据本发明通过带有“条件执行”的指令和“转移运算”-指令的结合以如下方式解决了该问题：

下面举一简单的例子，即指令“Add R2 to R1, if R1 then=0, jump to L1”。该问题用“C”语言编写如下：

```
R1=R1+R2
if (R1= =0)
    GO TO L1
L1:.....
```

为此根据本发明采用了机器指令 ADD R1, R2,#JMP, ON ZERO。 其中#JMP 表示至进入点 L1 的转移地址。

为此我们将指令由“后条件”扩展到已知的“前条件”。例如：P1，ADD R1， R2， #JMP，Q1。

其中 P1 表示：当满足 P1 时，执行 R1=R1+R2。根据本发明 Q1 表示：当根据 R1=R1+R2 计算满足 Q1 时，利用 JMP 执行转移。

为此可以将下述的“C”-程序：

```
if (A=1)
B=A;
else
C=A;
```

译成下述机器码：

```
LOAD                R1    A
Q1    TEST          R1    1  #  L /* if A=1 jump L */
P1    Q1    STORE   R1    B
STORE               R1    C
```

如图 2 和 3 所示，根据本发明在指令编码中既设有用于“前条件”的位，又设有用于“后条件”的位。

图 2 示出一个仅具有 22 位长的指令的例子，其中一位 1 用于“前条件”，一位 2 用于“后条件”，8 位 3 至 10 用于相对转移值（位移）和分别用于两个寄存器地址的每一个 3 位，和 6 位指令码。

在实际中通常需要检查诸如“前条件”和“后条件”等多个条件。所以必须设有如图 3 所示的相应的多个位。

如图 3 所示，位 0 至 1 含有用于后条件的信息，位 2 至 3 含有用于前条件的信息，位 4 至 10 含有相对的转移地址，即转移距离。

本发明的方法与程序循环结合在一起使用是特别有效的，例如用于

下述的“C”程序：

```
for (i=1;i<5;i+ + ) {
    x [i]=i;
}       /* C-Program */
```

根据本发明然后将此转换成特别简化的机器程序：

```
Load      R1      5
Load      R2      x    /* Address of  x[5]*;
L1=STORE-INDEXED R2  R1  /* x[i]=i */
Q1 DECREMENT      R1  #1  L1
          ADD     R2   1    /*i=i+1*/
```

其中“后条件 Q1”表示：当 R1=R1-1 不等于 0 时的条件转移。

本发明实现的编程简化的另一个例子是下述用于对循环缓冲存储器处理的程序。

根据已有技术必须对该问题编程如下：

```
TST (R3)  #buffer-end  // ring buffer end reached
BNZ NEXT               // if no
NOP
LDI (R3) #buffer-start// else set the pointer to buffer
                          again
```

根据本发明下述的两个指令足以替代上述程序：

```
TST  (R3)  #buffer-end
LDI  (R3)  #buffer-start
```

但要注意的是，本发明的该方案并不适用于所有循环结构。根据本发明的所有循环结构可以编程如下：

```
LDI (R4) #loop-cnt-minus-1//init loop counter
WHILE-LOOP:
FIRST-PC                //code sequency
SUBI(R4) #1 #loop-flag //decrement by 1 and
                          indicate loop end
```

```
BNZ WHILE-LOOP          //if not zero go to loop
                        begin
```

根据本发明替代通常的减法机器指令 SUB 采用机器指令 SUBI，该机器指令被扩展，从而其具有一个标志符-位，该标志符-位用于指示在条件转移指令 BNZ 前的一个周期，该标志符-位构成条件转移时的正确的转移，从而在任意的两级的流水线的情况下不会出现处理器周期的损失。指令 LDI 表示循环开始。

典型的避免“转移冒险”的方案在于，对条件转移的有待转移的目的进行预测。

循环的实施一般需要如下三个步骤：

1．启动循环计数器

2．递减或递增循环计数

3．在循环结束时的转移

条件转移时的周期损失在于，在转移后执行的下一个指令取决于循环条件的满足。该事实导致在条件转移指令后必须必须插入空-指令 NOP。通过在运算指令中采用诸如 ADD 或 SUB 等循环-标志符在执行相加或相减指令结束时将对循环条件进行检查。然后检查“0-标志符”，即运算器在 0 上的显示，以便确定应设置在处理器的程序计数器的哪个地址上。“循环-标志符”可以解释为“启动-位移-标志符”或通常的地址移动。

图 4 示出本发明执行“循环”-标志符的最简单的基本原理。

程序存储器 10 在此通过一乘法器 12 与程序计数器 14 连接。程序计数器（PC）14 与逻辑门 16 连接，逻辑门将程序计数器的输出值与一个常数或循环-标志符逻辑连接在一起。该逻辑电路 16 的输出端与乘法器（MUX）12 的一个输入端连接，乘法器的另一输入端与程序存储器 10 连接，并且乘法器的输出端与程序计数器 14 连接。由处理器通过控制信号对乘法器 12 进行控制。

本发明的另一改进在于通过对循环开始的缓冲存储，可以省去转移指令：

```
LDP (R4) #loop-cnt-minus-1
WHILE-LOOP:
```

FIRST-PC

SUBI (R4) #1 #Loop-flag

NEXT-INS:

在此需要一个附加的指令 LDP，该指令示出一个循环的开始。下一个程序码地址然后作为循环开始被缓冲存储。采用指令 LDI 并将下一个计数值显示装入缓冲存储器中。但因此当然还需要一个附加的指令。指令 SUBI 具有一个循环标志符，该循环标志符用于说明就条件转移而言哪个是正确的转移。对 zero-标志符进行检验，以便判决是否应返回环路开始，或执行下一个指令（NEXT-INS），该指令由#-Loop-标志符指示出。

如图 5 所示这种简化的循环结构处理需要较为复杂的电路结构。

如图 4 所示，具有一个程序存储器 10，该程序存储器与乘法器 12 的输入端连接，乘法器的输出端与程序计数器（PC）14 连接。程序计数器（PC）的输出端同样与逻辑门 16 连接，逻辑门对程序计数器的输出值与循环标志符进行逻辑连接。逻辑电路 16 的输出端与乘法器 12 的另一输入端连接。但在本实施例的情况下，乘法器还具有一输入端，该输入端与缓冲存储器 18 连接，可以将程序计数器 14 的值装入缓冲存储器的输入端。采用此方式时，显示指令“Load the next program counter reading into the buffer”是多余的。

图 6 示出处理器的整个结构，所述处理器具有处理本发明的指令的能力。图 6 中具有与图 4 和 5 相同的附图标记。程序计数器（PC）14 又对程序编码存储器 10 进行存取，并分别对有待处理的程序行进行存取。由程序存储器 10 将相应的指令编码输送给指令译码器（IDEC）20，指令译码器接着将相应的控制指令传送给运算器（ALU）22 和寄存器组 34。然后根据需要将寄存器的内容写入运算器 22，或如箭头所示由运算器重写入寄存器组，装入积存器内容。运算器的标志符 zero、carry 和 overflow 然后同时被输送给指令译码器（IDEC）20 和乘法器（MUX）12 的控制输入端。乘法器 12 的两个输入端被值 1 和由指令译码器 20 输出相对的转移值#JMP 占用。乘法器 12 的输出端与一加法器 16 连接，加法器 16 的另一输入端与程序计数器 14 的输出端连接。

当流水线-级多于两个时必须注意的是，标志符-信号 zero, carry,

overflow 和所属的相对的转移值#JMP 必须同时加在乘法器 12 上。但如果是一个本实施例所述的两级的流水线，此点则是不必要的。下面将对具有本发明的“后-条件”的指令编码加以说明。再重新对照图 2，在该图中示出长度为 22 位的本发明的最简单的指令语句。

最高的 6 位（21 至 16）包含指令码（OPCODE），例如：相加。下面的三位包含长度为三位的在 15、14、13 位的第一寄存器的地址（REG A）（通常的处理器大多采用的寄存器不超过 8 个），接着是第二个，在本情况时有待选址的寄存器（REG B）在 12、11、10 位上。

在该指令中处理器的运算器将寄存器 A 和 B 的内容相加并存储在寄存器中。根据本发明这时对该指令附加另外几位，即 9 至 2 位（位移），所述位对下述转移条件下的转移距离加以说明。接着的是条件-位 1 和 0，其中位 1（后）表示后条件，而位 0（前）表示前条件。

其中处理过程如下：对指令进行调取和译码。为此处理器在一特定的程序计数读出状态上，例如 PC=0 时开始。

在该程序计数读出的情况下，由程序存储器中调取出一个 22 位指令，该指令位于与程序计数器读数相符的存储器中的地址上。

然后由指令译码器（IDEC）20 对该指令进行处理。

其中首先检查是否设置了相应的前条件-位 1。如果是此情况，则在没有满足相应的前条件的情况下，尚未执行指令。

本发明与已有技术的区别在于后条件-位。

根据后条件-位产生信号“BR-CTR”。同时按如下方式进行相加：

为运算器产生一个控制信号 ALU-CTR 以及读出-和写入地址和允许信号。指令译码器 20 同时提供转移距离“BR”。“BR-CTR”-信号按下述方式进行转移控制：

1. 当后条件-位=0，即 PCNEW=PCOLD+1 时，没有转移；

2. 当后转移-位=1 和条件得到满足，例如 0-标志符=1 时，则执行相对转移。程序计数器将被设置在新值 PCMEW=PCOLD+BR。

如图 3 所示，例如可以采用一个以上的后条件-位。此时可以对多个条件进行检查（例如 zero、carry、overflow）。

根据本发明在对指令译码时由指令译码器 20 同时首次产生运算器的

控制信息和有关转移目的地址的信息。

这时指令被执行并且必要时被转移。

为此运算器（ALU）执行运算。运算结果被重新写入寄存器中。同时相应的 zerro、carry 等等标志符被加在运算器的输出端。

其中同步向转移控制提供各个标志符的位、“BRCTR”和“BR”值。如图 7 所示，控制单元“Cond”26 产生两个控制信号 S1 和 S2。S1 用于对不进行转移或预测相对转移进行控制。S2 用于通过乘法器 12 接通相对转移地址“PCNEW”。

因此对转移可以节省附加于相应的运算指令的附加指令。从而可以减少所需指令的数量并随之提高处理器的吞吐量。

在图 7 中详细地示出了用于处理具有本发明的“后-条件-位”的处理指令的处理器的结构。在图 7 中用与图 4、5 和 6 相同的附图标记标示相同的单元。

而且如图 7 所示，还具有程序计数器 14，该程序计数器对指令存储器（CODEROM）10 进行选址。由指令存储器将为 22 位的指令宽度输送给指令译码器（IDEC）20。该指令译码器产生用于控制寄存器 24 和运算器（ALU）22 的通常的信号。根据本发明指令译码器还产生信号“BR”（该信号包括多个位）并说明转移距离，以及信号“BR-CTR”，该信号表明一有待处理的条件转移和有待检查的运算器的相应的标志符-位。

运算器 22 将运算结果和表示特定条件（例如 zero、overflow 、carry 等）的相应的标志符加在输出端。当然也可以将结果重新输送给寄存器 24。由 ALU 输出的“BR-CTR”-信号和标志符被输送给另外一个逻辑单元（Cond）26。该逻辑单元根据相应的 BR-CTR-信号和附属的标志符产生信号 S1 和 S2，所述信号对乘法器 12 和一个位于加法器 16 输入端前的开关进行控制。该开关器在 1 和作为满足标志符条件的函数的“BR”之间进行转换。加法器的另一输入端与程序计数器 14 的输出端连接。

采用本发明的方式可以以处理器的较少的附加的技术代价实现对条件转移速度的大幅度的改善。

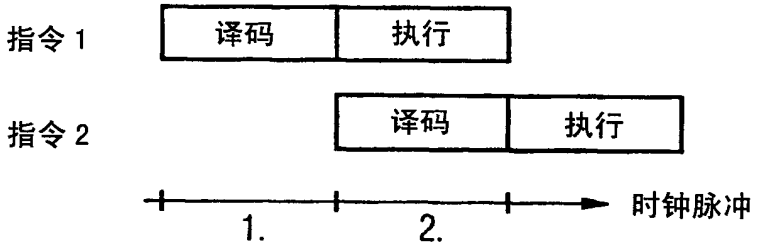

指令 1
译码
执行
指令 2
译码
执行
时钟脉冲
1.
2.

图 1

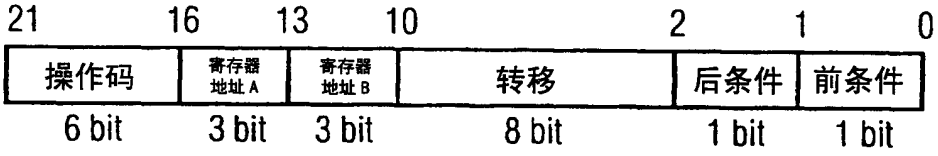

21
16
13
10
2
1
0
操作码
寄存器 地址 A
寄存器 地址 B
转移
后条件
前条件
6 bit
3 bit
3 bit
8 bit
1 bit
1 bit

图 2

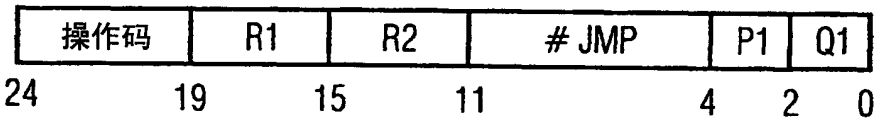

操作码
R1
R2
JMP
P1
Q1
24
19
15
11
4
2
0

图 3

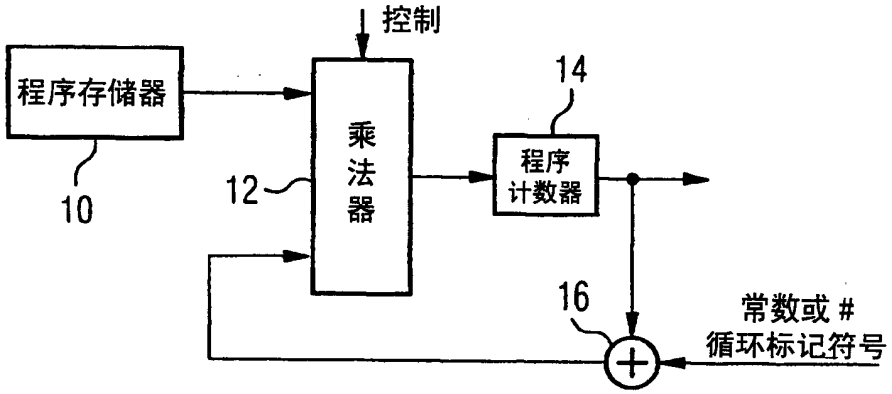

控制
程序存储器
10
12
乘法器
14
程序计数器
16
常数或 # 循环标记符号

图 4

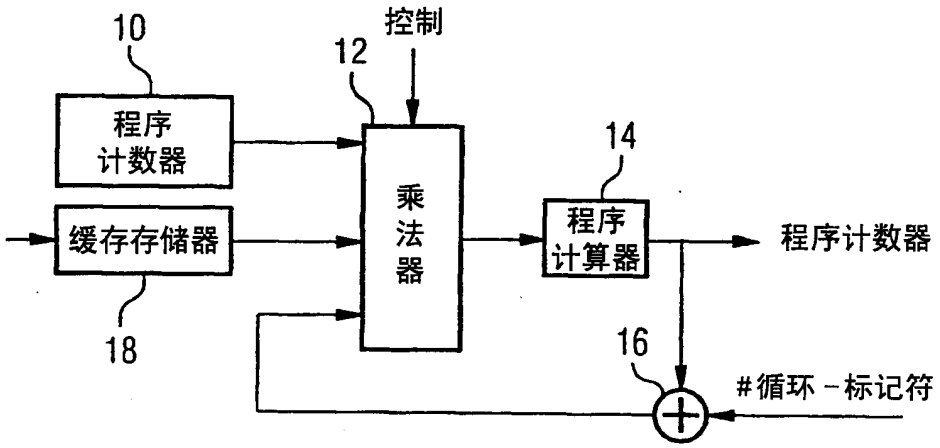

10
控制
12
程序
计数器
缓存存储器
18
乘
法
器
14
程序
计算器
程序计数器
16
#循环－标记符

图 5

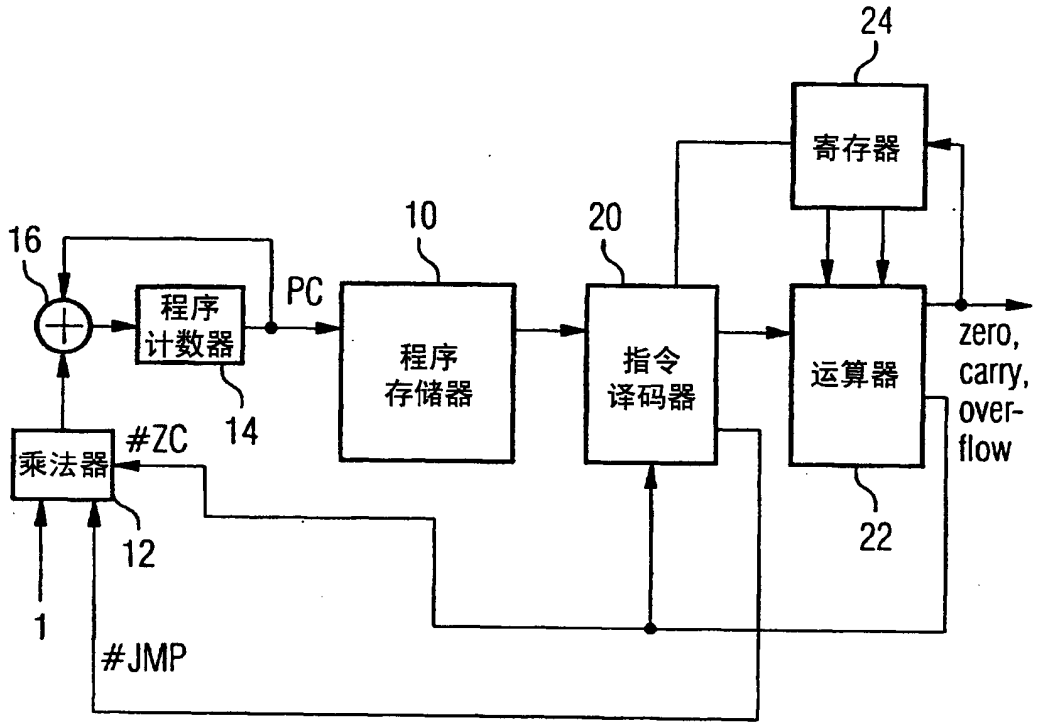

24
寄存器
16
10
20
PC
程序
计数器
14
程序
存储器
指令
译码器
运算器
zero,
carry,
over-
flow
乘法器
#ZC
12
22
1
#JMP

图 6

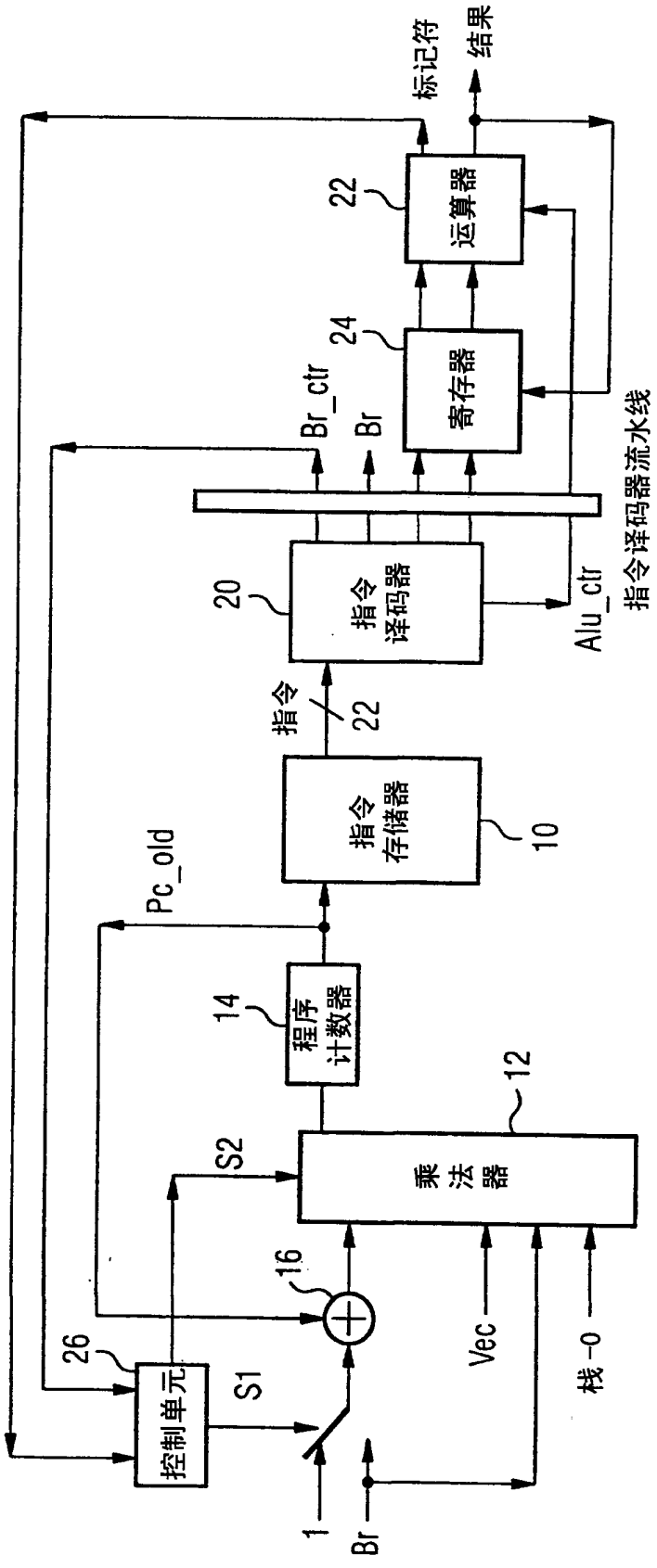

标记符
结果
22
运算器
24
寄存器
Br_ctr
Br
20
指令
译码器
Alu_ctr
指令译码器流水线
指令
22
指令
存储器
10
Pc_old
14
程序
计数器
12
乘
法
器
S2
16
26
控制单元
S1
Vec
栈-0
1
Br

图 7