[19]中华人民共和国专利局

[51]Int.Cl[6]
G06F 7/52

[12] 发明专利申请公开说明书

[21] 申请号 97115450.3

[43]公开日 1998 年 3 月 11 日

[11] 公开号 CN 1175730A

[22]申请日 97.7.24

[30]优先权

[32]96.7.25 [33]US[31]685995

[71]申请人 摩托罗拉公司

地址 美国伊利诺斯

[72]发明人 罗格·A·史密斯

[74]专利代理机构 中国国际贸易促进委员会专利商标事务所

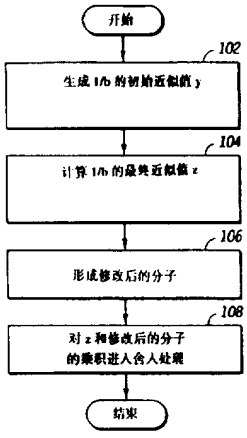
代理人 于 静

权利要求书 4 页 说明书 12 页 附图页数 7 页

[54]发明名称 使用浮点运算硬件进行微处理器整数除法操作的方法和装置

[57]摘要

一种在多个整数分子被一个公共整数分母除时用于产生多个整数商的方法和装置，通过如下方式实现：用整数分母倒数的浮点数近似值乘以浮点数分子，进行舍入前，对产生的浮点商进行偏差处理。通过计算分母倒数的限定精度平方根的平方计算分母倒数的一个初始近似值。使用有限次幂级数计算最终倒数。计算修正分子同倒数的多次乘积并据要求对乘积进行向上或向下舍入取整处理。



(BJ)第 1456 号

权 利 要 求 书

1. 一个产生整数分子(a)和整数分母(b)的整数商的方法，其特征在于：

整数分子(a)、整数分母(b)和整数除法指令一起提供给中央处理单元(CPU)；

使用中央处理单元(CPU)确定整数分母(b)倒数的初始近似值(y)；

对初始近似值(y)进行精细处理，产生更为精确的整数分母(b)倒数的近似值(z)；

通过偏差值(p)对整数分子(a)进行偏差处理，产生偏差分子(a')；并通过将更精确的近似值(z)和偏差分子(a')相乘来确定a/b的整数商，以此响应 CPU 中发布的控制信号并在 CPU 中执行舍入操作以确保a/b的整数商是一个整数。

2. 根据权利要求 1 中的方法，提供整数分子(a)和整数分母(b)的这一步骤的特征在于：

以一种整数格式提供整数分子(a)和整数分母(b)，该整数格式在处理之前由 CPU 中的一个浮点运算单元转换为浮点格式。

3. 根据权利要求 1 中的方法，确定初始近似值(y)这一步骤的特征在于：

从存储在存储器中的一张表中找到初始近似值(y)的小数部分，并

由整数分母(b)的指数部分来计算初始近似值(y)的指数部分。

4. 根据权利要求 1 的方法，确定整数分母(b)倒数的初始近似值(y)并精细处理(y)从而生成一个更为精确的近似值(z)，上面这一步骤的特征在于：

使用整数分母(b)作为一个高速缓存标志值，通过存取该高速缓存找到更精确的近似值(z)。

5. 根据权利要求 1 的方法，精细处理初始近似值(y)以生成一个更精确的近似值(z)这一步骤的特征在于：

按$e = 1 - b * y$确定一个误差值(e)；并

据$z = f(e,y)$找到一个更精确的近似值(z)。

6. 一种用于产生整数分子(a)和整数分母(b)的整数商的方法，其特征在

于：

整数分子(a)、整数分母(b)和整数除法指令一起提供给中央处理单元(CPU)；

使用中央处理单元(CPU)确定整数分母(b)倒数的初始近似值(y)；

确定整数分母(b)的平方根倒数的近似值(r)，即$r \approx (1/\sqrt{b})$；并且

计算初始近似值(y)同平方根倒数近似值(r)和平方根倒数近似值(r)的乘积，即$y = r * r$；

精细化初始近似值(y)得到一个更精确的整数分母(b)的倒数近似值(z)，这一步骤的特征在于：

按$e = 1 - b * y$确定一个误差值(e)，并据$z = f(e,y)$找到一个更精确的近似值(z)；　通过偏差值(p)对整数分子(a)进行偏差处理，产生偏差分子(a')；这一步骤的特征在于：

按$a' = a + p$找到偏差分子(a')，这里的p等于$\pm 1/2$；并且

通过将更精确的近似值(z)和偏差分子(a')相乘来确定a/b的整数商，以此响应 CPU 中发布的控制信号并在 CPU 中执行舍入操作以确保a/b的整数商是一个整数。

7. 一种在流水线浮点处理器中执行整数除法的方法，其特征在于：

提供一个整数分子(a)和一个整数分母(b)给流水线浮点处理器；

通过流水线浮点处理器确定整数分母(b)倒数的初始近似值(y)，即$y \approx 1/b$；

通过流水线浮点处理器找到初始近似值(y)的误差值(e)，$e = 1 - b * y$；

通过在浮点处理器中至少一次的循环处理，循环确定更精确的分母(b)倒数的近似值(z)，不断进行循环处理直到更精确近似值(z)在指定的误差公差(δ)内，更精确近似值(z)是初始近似值(y)和误差值(e)的函数。

对整数分子(a)进行偏差处理，得到偏差值(a')，在$a' = \pm(a + p)$中的$|p|$是一个用于下限取整操作的小于 1.0 的小数值；在$a' = \pm(a - p)$中的$|p|$则是一个用于上限取整操作的小于 1.0 的小数值；

通过将整数分子(a')与更精确近似值(z)相乘，确定一个初步结果；然后对该初步结果进行舍入处理，得到一个整数值。

8. 一个整数除法器，其特征在于：

浮点运算装置，该装置用于确定整数分母(b)倒数的初始近似值(y)；

浮点运算装置，该装置用于将初始近似值(y)精细处理为整数分母(b)倒数的一个更为精确的近似值(z)；

浮点运算装置，该装置通过一个偏差值(p)对整数分子(a)进行偏差处理，从而得到一个经偏差化的分子(a')；并且

该装置在 CPU 内通过对更精确近似值(z)和偏差分子(a')的乘积执行舍入操作来确定a/b的整数商，从而确保a/b的整数商是一个整数。

9. 存储在计算机可读取介质中的一个整数除法器，计算机可读取介质的特征在于：

至少有一条计算机指令，该指令用于确定整数分母(b)倒数的初始近似值(y)；

至少有一条计算机指令，该指令用于对初始近似值(y)进行精细处理，从而得到整数分母(b)倒数的一个更精确的近似值(z)；

至少有一条计算机指令，该指令通过偏差值(p)对整数分子(a)进行偏差处理，从而生成一个经偏差的分子(a')；并且

至少有一条计算机指令，该指令在 CPU 内通过对更精确近似值(z)和偏差分子(a')的乘积执行舍入操作来确定a/b的整数商，从而确保a/b的整数商是一个整数。

10. 一种用于得到整数分子(a)和整数分母(b)的整数商的装置，该装置的特征在于：

包含流水线浮点运算电路的中央处理单元(CPU)；

同中央处理单元(CPU)相连的存储器，该存储器包含一或多条计算机指令，这些指令由 CPU 进行存取并经解码执行下面的操作：

使用流水线浮点运算电路确定整数分母(b)倒数的初始近似值(y)；

使用流水线浮点运算电路对初始近似值(y)进行精细处理，得到整数分母(b)倒数的一个更精确的近似值(z)；

使用流水线浮点运算电路通过一个偏差值(p)对整数分子(a)进行偏差处理，从而得到一个经偏差化的分子(a')；并且

使用流水线浮点运算电路在 CPU 内通过对更精确近似值(z)和偏差分子(a')的乘积执行舍入操作来响应 CPU 内发布的控制信号，确定a/b的整数商，从而确保a/b的整数商是一个整数值。

使用浮点运算硬件进行微处理器整数除法操作的方法和装置

本发明主要涉及微处理器除法，特别涉及使用能进行浮点加乘运算的浮点运算硬件得到整数除法结果的方法和装置。

在现代计算机界，使用计算机系统中的中央处理单元(CPU)执行整数除法的必要性在逐步增长。整数除法操作包括一个整数分子和一个整数分母，在许多应用中都将用到这一操作。典型地，整数除法可用于 Internet 上 MPEG 序列或 JPEG 图象的计算。整数除法可用于计算机图形处理、各种绘图算法、三维屏幕显示、图形用户界面(GUIs)以及其他使用整数除法进行比例换算和/或压缩的应用。同线性代数和矩阵计算紧密相连的多媒体应用需要使用整数除法。视频处理、整数线性规划、多倍精度(multiple-precision)算术以及欧几里德(Euclid)的最大公分母(GCD)算法的使用也需要大量的整数除法。因此，随着这些应用的逐步扩展，以一种效果和效率不断提高的方式执行整数除法对于现代计算机和微处理器设计已变得越来越重要。

现在，整数除法是通过中央处理单元(CPU)的除法操作来完成的。整数除法由 CPU 的整数执行单元(integer execution unit)执行，其精度一般不超过 32 位(bits)。大多数除法操作需要至少 20 至 40 个时钟周期，并且对于大多数现代计算机的中央处理单元(CPU)，除法操作通常是最耗时的算术操作。正如业界已认识到的，计算机运行得越快，其价值越高，因此除法操作的使用并不是优化的。在上面谈到的应用中，整数除法典型发生在比例换算中，总是涉及相同的整数分母。当执行除法操作时，分母连续保持相同值，每次除法必须彼此独立地串行执行，无法通过并行操作获取加速优势或者在遇到后面的除法操作是相同分母时，无法通过在除法操作之间传递信息来加速随后的除法处理。总之，当上面谈到的应用要求在合理的时间范围内以合理的效率执行整数除法操作时，在整

数执行单元中执行整数除法操作并不是一种有效的方法。现在的整数除法操作速度慢，对于相同整数分母的聚集比例换算缺乏效率和精度，总之对上面的应用来说是一种无效的方法。同时，由通常的经验可知，浮点处理器可以为浮点数的加、减和乘操作提供有效的流水线处理。这些浮点处理器的精度典型可达 53 位。

因此，需要一种新的装置和方法来进行以浮点格式表示的整数除法操作，这样，这些整数除法操作的执行速度会得到提高，操作精度也将得到提高，并且当使用相同分母进行多个除法操作时，可以以更佳的时间效率方式进行流水线处理。

图 1 是数字直线图，用于举例说明本发明所阐述的问题之一；

图 2 是数字直线图，用于举例说明本发明；

图 3 是高层流程图，用来举例说明本发明的操作；

图 4 到图 6 都是流程图，用来举例说明图 3 中的操作步骤；

图 7 是一张调度说明表，用来举例说明本发明在流水线处理器系统中的操作；

图 8 是方框图，用来举例说明本发明的集成电路实现设计；

图 9 和图 10 都是方框图，用来举例说明本发明的数据处理系统实现。

总的说来，本发明涉及在中央处理单元(CPU)内部执行整数除法操作，但避免使用除法硬件和算法。相反，本文所述的硬件和方法通过使用 CPU 内的浮点运算硬件的更有效的加法和乘法资源，以一种既精确又快捷有效的方式来处理整数除法操作。现有技术使用算术除法操作来实现整数除法功能。除法操作涉及整数分子 a 和整数分母 b，$\frac{a}{b}$ 的结果约等于某个整数值 x。$\frac{a}{b}$ 向下取整，记作 $\left\lfloor \frac{a}{b} \right\rfloor$，是不大于 $\frac{a}{b}$ 的最大整数，$\frac{a}{b}$ 向上取整，记作 $\left\lceil \frac{a}{b} \right\rceil$，是不小于 $\frac{a}{b}$ 的最小整数。在本文的方法和装置中，整数除法 $\frac{a}{b}$ 是通过两个数的乘法实现的，一个是 b 的倒数即 $\frac{1}{b} = b^{-1}$ 的近似值；另一个数是经偏差处理的分子 $a' = \pm(a + p)$，两数相乘的结果将近似为最接近的最大整数或最小整数。

需要重点说明的是，在使用本方法时，由于值$\frac{1}{b}=b^{-1}$在 CPU 中不能表示为无穷位或完整精度值，因此，需要这里所讨论的a的附加修正偏差。例如，值 1/3 在大多数 CPU 中不能表示为无穷位或完整精度值。所以，当中央处理单元(CPU)接收到整数分子a和整数分母b时，这两个数或者以浮点格式或者转化为浮点格式提供以进行浮点处理。这里的a进行了偏差校正以补偿$\frac{1}{b}=b^{-1}$取近似值带来的精度损失。

实验表明，使用整数单元进行的典型整数除法操作可能占用 20 至 40 个时钟周期，数据精度为 32 位。采用浮点运算单元中的加乘方法来实现数据精度为 50 位的整数除法操作，则占用 30 个时钟周期。如果除法操作使用相同的分母b，则第二个除法可能仅需要 1 个时钟周期，第三个除法同样也仅再附加 1 个时钟周期，附加的除法操作相当节省时间。换而言之，一个典型整数除法单元如果计算 24/11， 101/11 和 65/11 大约需要 60 至 120 个时钟周期，而计算$a'b^{-1}$ 的近似值的方法在 32 个时钟周期内就可以得到这三个除法的整数商。另外，这些结果是用浮点格式表示的，这使得在具有更佳流水线性能的浮点运算单元中直接利用这些结果变得更容易了。所需时间比在浮点运算单元中进行单个浮点数除法更快，因为进行提供整数商结果的浮点操作时，多个附加计算可能被覆盖。

所以，这里所述的整数除法过程及装置可提供浮点数计算，这一计算在浮点运算硬件上进行，这一方式较之整数硬件单元具有更有效的流水线性能。采用浮点运算单元，即使只进行一个整数除法操作，速度也可明显得以改善，对于具有相同分母b(例如 JPEG 中的比例换算等)的多个除法操作，在性能上则可以得到很大提高。浮点运算硬件单元提供更多位精度，因此本文所述的方法和装置可以为整数执行单元中通常使用的超过 32 位的整数提供准确的整数除法精度。另外，整数值以浮点格式进行维护，因此减少了整数和浮点数之间的转换时间。总之，本文所述用于计算整数除法的方法可以进行快速计算，从而适用于 MPEG 应用、JPEG 处理，图形界面、三维图形、最大公分母(GCD)计算、线性代数计算、矩阵计算、视频处理、数字音响处理、整数线性规划等等，使这些

应用以先前不可能达到的速度进行计算。

参照理论讨论部分及图 1 至图 10 可以进一步理解本发明。

在举例说明用于不同计算机应用中整数除法的具体的方法和装置之前，下面的理论讨论将有助于理解本文所述的整数除法处理。

值小于某个最大值 A 的所有整数都可表示为浮点格式。设$|a|<A$和$A>b>0$是提供给 CPU 的两个整数值。整数下限$\left\lfloor \frac{a}{b} \right\rfloor$和整数上限$\left\lceil \frac{a}{b} \right\rceil$也是范围处于(-A， A)之间的整数。通过使用$z \approx b^{-1}$的近似值及计算

$$\left\lfloor \frac{a}{b} \right\rfloor = \left\lfloor (a+\frac{1}{2})z \right\rfloor$$

和/或

$$\left\lceil \frac{a}{b} \right\rceil = \left\lceil (a-\frac{1}{2})z \right\rceil$$

达到不进行精确除法计算而进行下限取整计算和上限取整计算的目标。这种 CPU 计算方式可以以较快的式进行，并且具有前面讨论的所有优点。如果z有足够的精确度近似等于$\frac{1}{b}=b^{-1}$，使 $\delta=1-bz$ 满足$|\delta|<\frac{1}{2A+1}$，则上面的计算可以得到正确的结果。

在 CPU 中计算z的一种方便方法是得到$\frac{1}{b}=b^{-1}$的一个初始估计值，记为近似值y，计算$e=1-by$，最后通过 CPU 中一或多次循环处理由y得到z。经截尾的以幂级数表示的近似值记为$z=y(1+e+e^2+e^3+e^4+...+e^m)$。如果精确计算这一被截尾的以幂级数表示的近似值，结果会得到一个误差值$\delta=e^{m+1}$。可以只使用一次(也就是说，一次循环)或按序列(也就是说，多次循环)使用这个表达式来进行计算。对于具有中等或较高时延的流水线处理器，使用m具有较高阶的单元多项式比较合适；对于具有较低时延的处理器，则重复使用低阶多项式更合适一些。另外，对于m具有确定值的多项式，更容易进行因式分解，从而可以比其他方式更快地进行单元高阶多项式的估算。上面所述的这种情况，如下例所示，特别合适。

对于现代具有流水线的微处理器(见图 8 至 9)，通常存在一条指令用于计算浮点数倒数的近似值。如果这一指令以 8 位精度来计算y的近似值，

则 m=6 是满足 z 的精度大于 50 位(如果直接计算)的最小 m 值。如果这一指令执行得快，则这是一个好的开端。

如果这一指令执行得慢或者没有提供，则使用一条指令，用来计算浮点数平方根倒数的近似值。如果这一指令执行 $\frac{1}{\sqrt{b}}$ 的近似值为 r，r 可精确到 5 位，则 $\frac{1}{b}=b^{-1}$ 的近似值 y 可通过 $y=r^2$ 来计算，精度可以小于 4 位。这种方法较之上一段中描述的初始近似值的计算方法，计算速度明显加快，m 可以取到 16 。

使用 y 这一精度小于 4 的值，而同时为保证正确， A 取值为 2^{50}，这看上去是一个令人生畏的问题，比较适宜的方法是取阶 m=16 的单元多项式。经过因式分解，近似值最终可表示为：

$$z=y+ye(1+e)(1+e^2)(1+e^4)(1+e^8)$$

这一因式分解可以通过选择算术操作被执行的阶以多种方式进行计算。当组合乘法/加法指令可用时，有一种特别快捷和方便的方法，即

$$z=y+(e+e^2)(y+y\cdot e)((e+e^2)+(e+e^2)\cdot e^4)(1+e^8)$$

为了减小计算 z 的时间，还有其他一些选择阶的方法，一种特别好的方法示例如下。

如果进行精确计算，则 z 将满足 δ 所表示的误差范围。但是，在计算 z 时，必然会发生多次数学舍入。可被引入的最大总误差(the maximum total error)具体依赖于操作发生时的阶。对于图 7 所描述操作的具体选择，若同时考虑精度差异和算术舍入的不精确性，可以用枯燥但却直接了当的方式来表示最终的 δ，δ 限定在小于为给定值 A 而设定的所需值范围内。对于四种 IEEE 算术舍入模式的任何一种，这一精确性都可以得到保证。注意，在许多 CPU 或数字信号处理器(DSP)中以硬件线路实现的乘-加指令对于有效地及精确地实现这一多项式的估计是相当有用的。尽管该方法可能在“舍入正无穷(round-to-plus-infinity)模式”下执行，但为了最终的舍入处理，在“舍入 0(round-to-0)”或“舍入负无穷(round-to-negative-infinity)”两种模式下执行更为方便。

借助把 $a'z$ 加到一个大数 M 的技巧，使用标准浮点指令可以有效得到

最终整数结果，此时，在舍入负无穷或舍入 0 模式下可以提供小于 $M+a'z$(值 $M+a'z$ 永不可能精确为一个整数)的最大整数值，在舍入正无穷(round-to-plus-infinity)模式可提供大于 $M+a'z$(值 $M+a'z$ 永不可能精确为一个整数)的最小整数值。用结果减去 M 或者用 M 减去结果，可以得到正确的整数下限或整数上限。在舍入 0 或舍入负无穷模式下，

$$\left\lfloor \frac{a}{b} \right\rfloor = (M+(a+\frac{1}{2})z) - M$$

$$\left\lceil \frac{a}{b} \right\rceil = M - (M+(\frac{1}{2}-a)z)$$

而在舍入正无穷模式下，

$$\left\lfloor \frac{a}{b} \right\rfloor = M - (M+(-a-\frac{1}{2})z)$$

$$\left\lceil \frac{a}{b} \right\rceil = (M+(a-\frac{1}{2})z) - M$$

使用这些方法，可在现代具有流水线处理方式的 CPU(见图 9)上计算出范围小于或等于 2^{50} 的整数 a 和 b 比率的正确整数下限和整数上限，还可在现有技术基础上得到如前描述的所有优点。注意，相同的 z 值可以用于所有具有相同分母 b 的分数中。还要注意，对于这些整数，a 的偏差结果用双精度来表示。总之，如果希望精确处理 N 位整数，则需要在浮点运算时加入一小部分附加位以完成计算 z 过程中产生的算术舍入。

参照图 1-10 可以进一步理解本发明。

图 1 是数字直线图，用于举例说明本发明所阐述问题之一。该数字直线图上的刻度(tick)示出了一整数被整数 b 除后的可能结果值。这些结果用整数 n 加上 $\frac{1}{b}$ 的倍数表示。在两个相邻整数 n 和 $n+1$ 之间，可能的结果值有 n，$n+\frac{1}{b}$，$n+\frac{2}{b}$，如此直到 $n+\frac{b-1}{b}$，最终到 $n+1$。数字直线上部示出了精确的可能合理结果值。如果这些结果仅进行近似计算，如乘以一个倒数的近似值而不是除以分母的精确值，则在这些精确点上会延伸出一些分支。在精确点上部的短线段指示了这一延伸的最大范围。如果近似结果处于整段之上的某个下半段中，则随后的截尾操作会产生一个错误的整数下限结果。例如，线 n 左面的某个值会被截尾为 $n-1$，尽管其实际整数

下限值是n。如果近似结果处于整段之上的某个上半段中，则随后的舍入操作会产生一个错误的整数上限结果。例如，对线n右半段的值，其整数上限会是$n+1$，而不是正确结果n。

图 2 是本发明的示例数字直线图。如前所示，该数字直线图上的刻度示出了一整数被整数b除后的可能合理结果值。数字直线上部示出了一整数加上$\frac{1}{2}$再除以b后的精确的可能合理结果。这些结果相对于图 1 有$\frac{1}{2b}$的偏差。精确偏差点上部的短线段示出了与精确值倒数的近似值相关的不确定延伸。如果如图示偏移靠右并且延伸小于$\frac{1}{2b}$，则偏差结果的近似值总可以向下舍入到下一个最小整数以得到精确值$\left\lfloor \frac{a}{b} \right\rfloor$，但如果偏移靠左(图未示出)并且延伸小于$\frac{1}{2b}$，则偏差结果的近似值总可以向上舍入到下一个整数以得到精确值$\left\lceil \frac{a}{b} \right\rceil$。因此无论$a$取何值都不会导致计算失败。将偏移和用于得到$\frac{1}{b}$足够精确近似值的有效方法相结合是本发明的关键。

注意图 2 所示的向上偏移，该偏移同向下舍入相结合用来产生下限取整函数。向下偏移同向上舍入相结合用来产生上限取整函数。各种等价性，如$\lfloor x \rfloor = -\lceil -x \rceil$，可以用来提供示例方法的轻变式(slight variants)。

图 3 是本发明的高层示例流程图。首先在步骤 102，生成$\frac{1}{b}$的初始近似值y。接着，在步骤 104 计算$\frac{1}{b}$的最终近似值z。在步骤 106 形成修正后的分子。最后在步骤 108，对z与修正后的分子的乘积进行舍入处理。

图 4 是步骤 102 的示例流程图。首先在步骤 112 计算$\frac{1}{\sqrt{b}}$的近似值r，接着在步骤 114 通过计算r^2来得到y。

图 5 是一个示例流程图，用来说明步骤 104 中$\frac{1}{b}$的最终近似值z的计算过程。图 5 中的两个输入值是方框 120 中的y和方框 122 中的b。方框 120 中的值y在步骤 114 和图 4 中计算。方框 122 中的输入值b是分母。值e在步骤 124 中按$e = 1 - b * y = (1 - by)$进行计算。在本等式及随后的等式中，中间项表示待执行的操作，右边项表示经精确计算得到的精确代数值。值e

用来参与步骤 126 中的$s=e*e=e^2$及步骤 128 中的$n=e+e*e=e(1+e)$的计算。

步骤 126 中的值s用来参与步骤 130 中$k=y+y*s=y(1+e^2)$的计算。步骤 126 中的值s也用来参与步骤 132 中$q=s*s=e^4$的计算。步骤 132 中的值q用来参与步骤 134 中$d=1+q*q=1+e^8$的计算。步骤 132 中的值q及步骤 128 中的值n用来参与步骤 136 中$c=n+n*q=e(1+e)(1+e^4)$的计算。步骤 130 中的值 k 和步骤 134 中的值d用来参与步骤 138 中$m=k*d=y(1+e^2)(1+e^8)$的计算。最后，方框 120 中的值y，步骤 138 的m以及步骤 136 中的c用来参与步骤 140 中$z=y+m*c=y+ye(1+e)(1+e^2)(1+e^4)(1+e^8)$的计算。精确结果为$y(1+e+e^2+...+e^{16})$，这对于示例实现而言已足够精确了。

图 6 是图 3 步骤 106 和 108 的示例流程图。在步骤 150 进行舍入类型(下限取整函数或上限取整函数)判断。如果舍入类型表明是下限取整函数，则值 R 按$R=M+(\frac{1}{2}+a)*z$在步骤 152 来计算，结果$\left\lfloor\frac{a}{b}\right\rfloor$在步骤 156 通过从 R 中减去 M 来计算。如果步骤 150 指示是上限舍入，则 R 在步骤 154 按$R=M+(\frac{1}{2}-a)*z$来计算，结果$\left\lceil\frac{a}{b}\right\rceil$在步骤 158 通过从 M 中减去 R 来计算。这里所示例的舍入计算方法假设浮点计算是在前面所讨论的两种模式，即舍入 0 或舍入负无穷模式下进行的。

图 7 是一张指令调度表，用来说明实现本发明的流水线 CPU 的执行过程。该表有四列，列 1 包含周期号，列 2 包含所进行的操作，本表示例了$\left\lfloor\frac{a}{b}\right\rfloor$，$\left\lceil\frac{a'}{b}\right\rceil$和$\left\lfloor\frac{a''}{b}\right\rfloor$的计算过程。注意，周期 20，25 和 28 仅是用于计算$\left\lceil\frac{a'}{b}\right\rceil$的附加周期，周期 22，26 和 29 仅是用于计算$\left\lfloor\frac{a''}{b}\right\rfloor$的附加周期。第三列表明第二列所示操作的结果在哪个周期可为随后的操作使用。调度假设每个时钟周期发布一次，所有指令的时延为 3 个时钟周期。在周期 0，计算$r=\frac{1}{\sqrt{b}}$，计算结果在周期 3 可用。然后在周期 3，计算$y=r*r$，其结果在周期 6 准备好，以用来计算$e=1-b*y$。流水线特性在周期 9 和周期 10 表现明显，值e用来计算周期 9 中的值s和周期 10 中的值n。最后第四列

包含描述某些操作结果的注解。本调度执行图 5 所描述的计算。注意，结果$\left\lfloor \frac{a}{b} \right\rfloor$在周期 30 可用，结果$\left\lfloor \frac{a'}{b} \right\rfloor$在周期 31 可用，而结果$\left\lfloor \frac{a''}{b} \right\rfloor$在周期 32 可用。在 32 个周期内除了得到三个运算结果外，另外还初始化和完成了 10 个浮点操作，这些操作没有占用额外的时间。如果仅计算$\left\lfloor \frac{a}{b} \right\rfloor$，在前 30 个周期内将执行 16 个附加的浮点操作，但是如果同时计算$\left\lfloor \frac{a}{b} \right\rfloor$和$\left\lceil \frac{a'}{b} \right\rceil$，则在前 31 个周期内要执行 14 个附加的浮点操作。在同一处理器条件下，这些运算的执行速度一般均比单个双精度浮点除法快。另外，通常除法操作不能发布其他浮点操作指令。

图 8 示出了一个用来执行图 1 至图 7 操作的定制设计的集成电路。图 8 示出了执行图 8 所清晰标注的步骤 102 到步骤 108 的硬件。换句话说，图 8 中的硬件单元 102 用来执行图 3 中步骤 102，图 8 中的硬件单元 104 用来执行图 3 中步骤 104，硬件单元 106 用来执行图 3 中步骤 106，图 8 中的硬件单元 108 用来执行图 3 中步骤 108。

在图 8 中，输入整数分子a和整数分母b通过导线 162 和 164 提供，而控制是上限取整操作还是下限取整操作的信号通过导线 166 提供。

硬件单元 102 说明该硬件用来提供输入值b的倒数的初始近似值y。值b如浮点格式 170 所示，格式 170 有一个指数部分和一个小数部分，小数部分包含标记为f_{high}的高阶位和标记为f_{low}的低阶位。指数部分用来构造y_0的指数部分，这里的y_0是b倒数的初始近似值。格式 170 小数部分的f_{high}部分用来存取存储在存储器中的一张表。存储在存储器中的表 174 典型地可以存于 ROM 存储器或 RAM 存储器中，该表可提供y_0,的小数部分f'_{high}的高阶位，而小数部分的低端部分则如图 8 中格式 176 所示填为 0。数值y_0,是通过导体 162 输入的值b的倒数的初始近似值，格式 176 示出了值y_0,的指数部分和小数部分。值y_0通过导体 178 提供给硬件的下一部分。通过硬件单元 104 以循环方式或流水线方式对值y_0进行处理，得到精确到限定公差范围内的$\frac{1}{b}$的最终近似值。

硬件单元 104 通过导体 178 将值y_0和通过导体 162 将值b作为其输入。

一旦导体 178 和 162 提供了这些输入值，则或者单一硬件单元以循环方式逐步计算更精确的倒数近似值，或者一组以流水线方式排列的硬件单元以按序流水线操作来计算更精确的值b的倒数近似值。所示例的实现方法利用了易于流水线化的简单公式。如果这些计算阶段共享硬件，则较前阶段较之后面阶段可以进行较低精度运算。如果在这些计算阶段循环使用同一硬件，则通过执行计算的一部分可以在较前阶段减小乘方。

在硬件图 104 示例了一个更为精确的用以计算b倒数近似值的方法。在流水线的第一个阶段或者在循环计算的第一个步骤中，硬件 180 通过硬件 182 来计算误差项。通过硬件 184 可以使用误差项和值y_0来计算更精确的b的倒数近似值。通过循环处理的后续步骤或流水线处理的阶段 180'，使用示例所示的循环处理方式或流水线处理方式来执行 180 中示例的类似操作以进一步对b的倒数近似值进行精确化。一旦在硬件 104 中执行了足以达到指定精度公差的循环次数或流水线阶段，则输出z通过导体 186 提供。为了硬件单元 102 和 104 能并行处理，输入整数分子a通过硬件 106 来处理。硬件单元 106 和 108 通过导体 166 接收下限取整及上限取整控制信息。线 166 上的信号用于控制执行操作是下限取整操作还是上限取整操作。在步骤 106，这一控制信号用于确定a的偏差方式(+ 表示下限取整，- 表示上限取整)。偏差值 R 通过导体 188 提供给硬件 108。硬件 108 接着执行通过导体 186 提供的值z和通过导体 188 提供的值 R 的乘积。然后这一乘积被舍入为最接近的最小整数值。导体 166 的控制信号确定是否改变整数的取整记号(no 表示下限取整，yes 表示上限取整)。硬件 108 的输出是通过导体 168 提供的整数商。如果b的值不变，则根据图 3 结构对图 8 所进行的划分使得易于保持z的值；如果最近使用过b，则图 8 的划分使得易于从高速缓存(cache)中得到z。正如前面所讨论的，在实现时进行少许变换就可以利用数学等价性来提供功能等价性。

图 9 和图 10 一起示例了一个数据处理系统，该系统可用来执行这里所

述的整数除法操作。图 9 和图 10 示例了中央处理单元(CPU)部分 200，该部分通过一条 32 位地址总线 272 和一条 64 位数据总线 272 同外部存储器 280 相连。中央处理单元(CPU)200 有一个提取单元(fetch unit)212，通过 128 位总线 258，利用指令存储器管理单元(IMMU)250，该提取单元 212 负责从 16KBI(I 表示指令)高速缓存 254 提取计算机指令。如图 10 所示，提取单元 212 所提供的指令将填入一个容量为 8 条指令的队列 214。提取单元 212 以每次 4 条指令的速度不断提取指令以确保队列 214 充满用于 CPU200 处理的指令。包含分支判断信息的分支处理单元(branch processing unit)216 用于控制提取器 212 以便维护指令队列 214 中正确的指令执行流。如图 9 和图 10 的中央部分所示，一个分派单元(dispatch unit)218 被用来解码和发布指令到适当的执行单元。分派单元 218 可为图 9 和图 10 中央部分所示的四种执行单元类型之一提供解码后的指令。这四种类型是浮点运算单元 240，装入/存储单元(load/store unit)234，单循环整数运算单元 228 及多循环整数运算单元 224。单元 218，216，214 和 212 是一个较大指令单元 210 的所有部分，该指令单元负责为图 9 和图 10 中的多个执行单元之一提供连续的指令流。

如图 9 和图 10 所示，所提供的指令通过 128 位总线 220 送到保留站(reservation station)222，226，232 和 238。保留站 222，226，232 和 238 中的每个保留站为执行单元 224，228，234 和 240 中的一或多个执行单元输送指令。224，228，234 和 240 这些执行单元将执行由指令单元 210 提供的经解码的指令。在指令执行期间，存储在通用寄存器(GPR)文件 230 区的多个通用寄存器和存储在浮点寄存器(FPR)文件 236 区的多个浮点寄存器可被执行单元存取。另外，装入/存储单元 234 可以存取存储队列(store queues)和装入完成(finish load queues)队列 244，数据高速缓存 264 同 D(数据)高速缓存标志(data cache tags)262 和数据高速缓存管理单元 260 相联。装入/存储单元 234 可存取这一标志信息以维护处理单元内部数据的完整性。

总线接口单元(bus interface unit)270 提供到外部总线 272 和 274 的接

口。总线接口单元 270 将指令放入与指令标志 252 相联的 I(指令)高速缓存 254 中。数据也通过总线接口单元 270 从外部存储器进行读取，读入后放入 D(数据)高速缓存 264 中。如上所讨论的，指令也通过指令存储器管理单元(MMU)250 被送往指令提取单元 212 。一旦指令被提供并通过执行单元 224 ， 228 ， 234 和 240 得以执行后，指令会进入到一个结束单元(completion unit)，该结束单元通常在处理器 200 所执行的流水线序列的最后，对异常的执行和错误的分支判断进行补救处理。 32 位地址总线 272 和 64 位数据总线 274 与外部存储器 280 和计算机可读介质 286 相连。外部存储器 280 中包含四组指令，用于执行硬件单元 200 中特定的操作。特别地，存储器 280 包含四个软件程序单元 102 ， 104 ， 106 和 108 ，它们分别对应图 3 中的步骤 102 ， 104 ， 106 和 108 。因此， CPU200 通过存取外部存储器 280 可以执行这里所述的图 1 至图 7 的操作。存储在存储器 280 中的软件单元 102 ， 104 ， 106 和 108 总是从计算机可读介质 286 装入存储器 280 ，可读介质有 EPROM ， EEPROM ， ROM ， DRAM ， SRAM ，磁存储器，带存储器，光存储器，激光磁盘(CD)，闪速存储器，网络存储器，通过通信连接的另一台计算机，或者象用于计算机可执行代码或计算机数据的存储设备。

熟练的技术人员能够识别本发明的修改和变动而不偏离本发明的宗旨，因此，本发明计划完成所有象附加的权利说明书中的修改和变动。

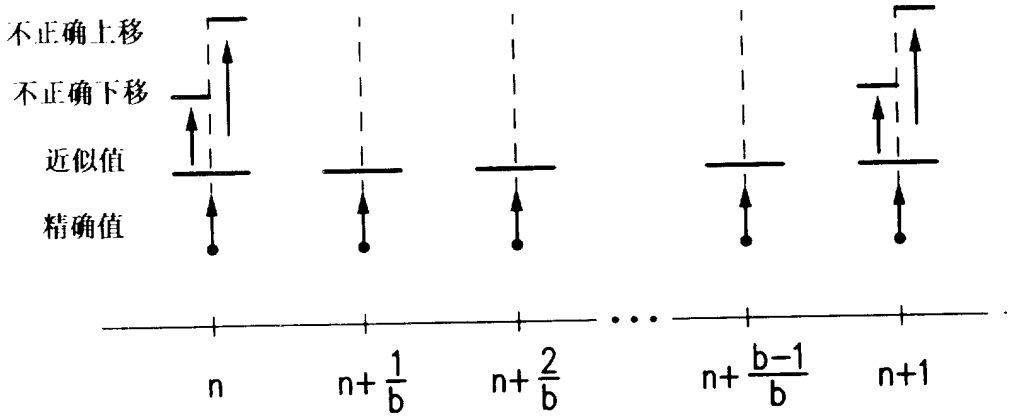

不正确上移
不正确下移
近似值
精确值
n
n+1/b
n+2/b
n+(b-1)/b
n+1

图1

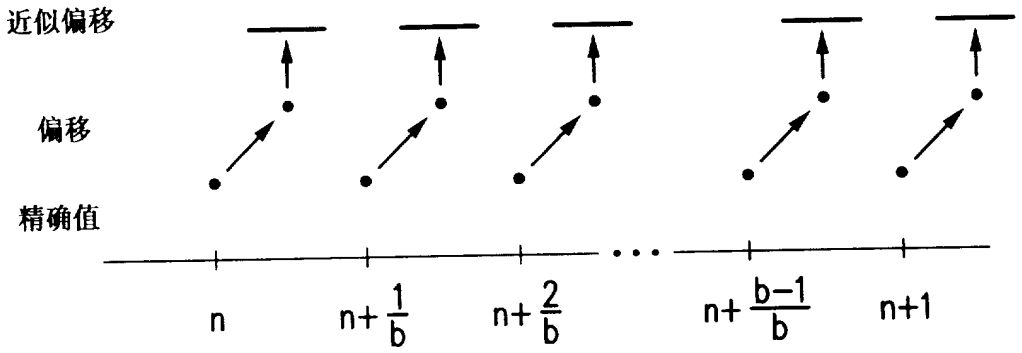

近似偏移
偏移
精确值
n
n+1/b
n+2/b
n+(b-1)/b
n+1

图2

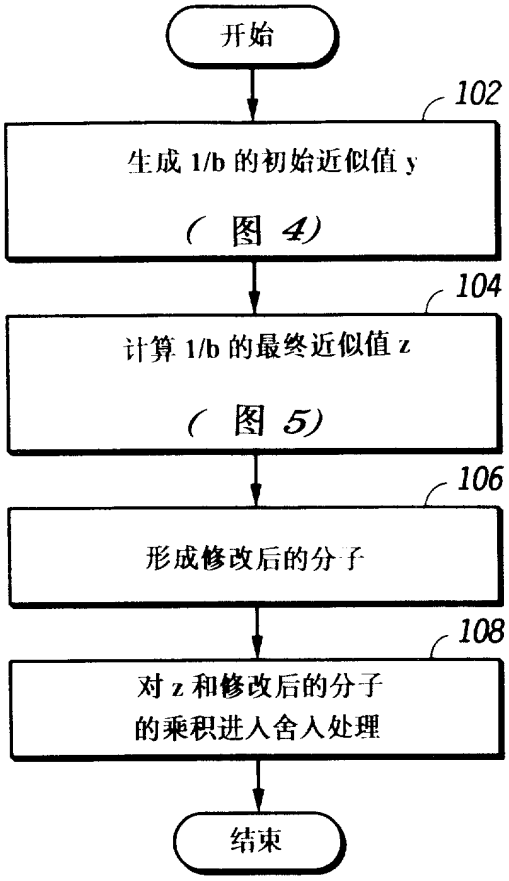

开始
102
生成 1/b 的初始近似值 y
(图 4)
104
计算 1/b 的最终近似值 z
(图 5)
106
形成修改后的分子
108
对 z 和修改后的分子
的乘积进入舍入处理
结束

图3

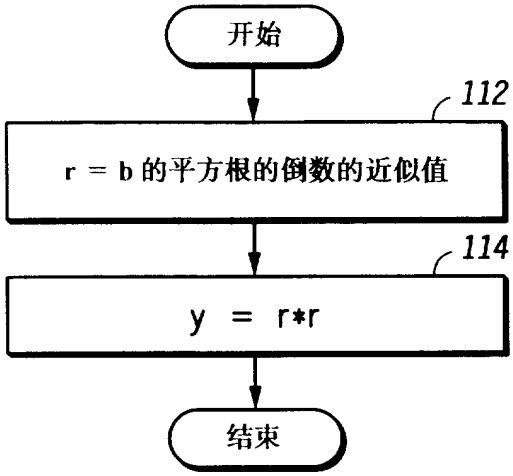

开始
112
r = b 的平方根的倒数的近似值
114
y = r*r
结束

图4

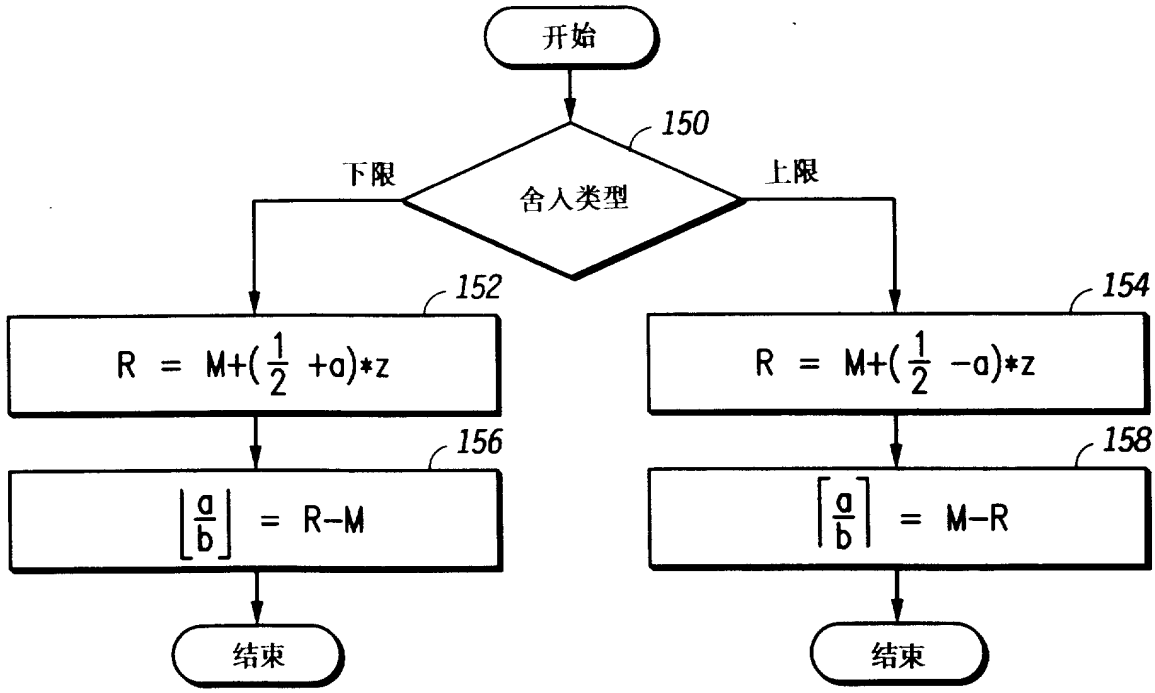

开始
150
舍入类型
下限
上限
152
R = M+(1/2 +a)*z
156
⌊a/b⌋ = R-M
结束
154
R = M+(1/2 -a)*z
158
⌈a/b⌉ = M-R
结束

图6

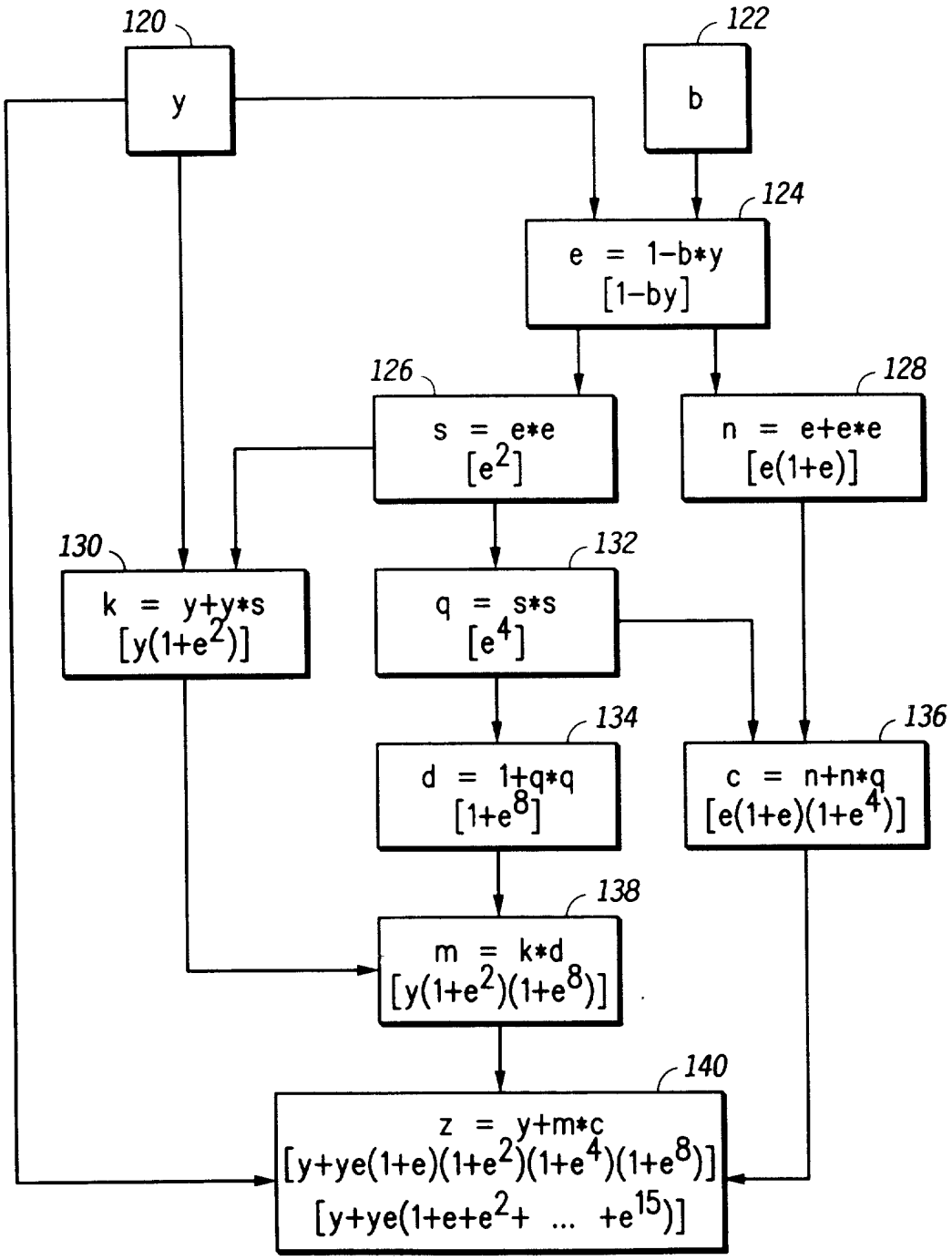

120
y
122
b
124
e = 1−b*y
[1−by]
126
s = e*e
[e^2]
128
n = e+e*e
[e(1+e)]
130
k = y+y*s
[y(1+e^2)]
132
q = s*s
[e^4]
134
d = 1+q*q
[1+e^8]
136
c = n+n*q
[e(1+e)(1+e^4)]
138
m = k*d
[y(1+e^2)(1+e^8)]
140
z = y+m*c
[y+ye(1+e)(1+e^2)(1+e^4)(1+e^8)]
[y+ye(1+e+e^2+ ... +e^15)]

图5

图7

周期	操作	准备完毕	注释
0	$r = 1/\sqrt{b}$	3	
1			
2			
3	$y = r*r$	6	
4			
5			
6	$e = 1-b*y$	9	
7			
8			
9	$s = e*e$	12	e^2
10	$n = e+e*e$	13	$e(1+e)$
11			
12	$q = s*s$	15	e^4
13	$k = y+y*s$	16	$y(1+e^2)$
14			
15	$d = q*q$	18	$(1+e^8)$
16	$c = n+n*q$	19	$e(1+e)(1+e^4)$
17			
18	$m = k*d$	21	$y(1+e^2)(1+e^8)$
19	$P = \frac{1}{2}+a$	22	
20	$(Q = \frac{1}{2}-a)$	23	
21	$Z = y+m*c$	24	$y+ye(1+e)(1+e^2)(1+e^4)(1+e^8)$
22	$(R = \frac{1}{2}+a'')$		
23			
24	$S = M+P*Z$	27	
25	$(T = M+Q*Z)$	28	
26	$(U = M+R*Z)$	29	
27	$\lfloor \frac{a}{b} \rfloor = S-M$	30	
28	$(\lceil \frac{a'}{b} \rceil = M-T)$	31	
29	$(\lfloor \frac{a''}{b} \rfloor = U-M)$	32	

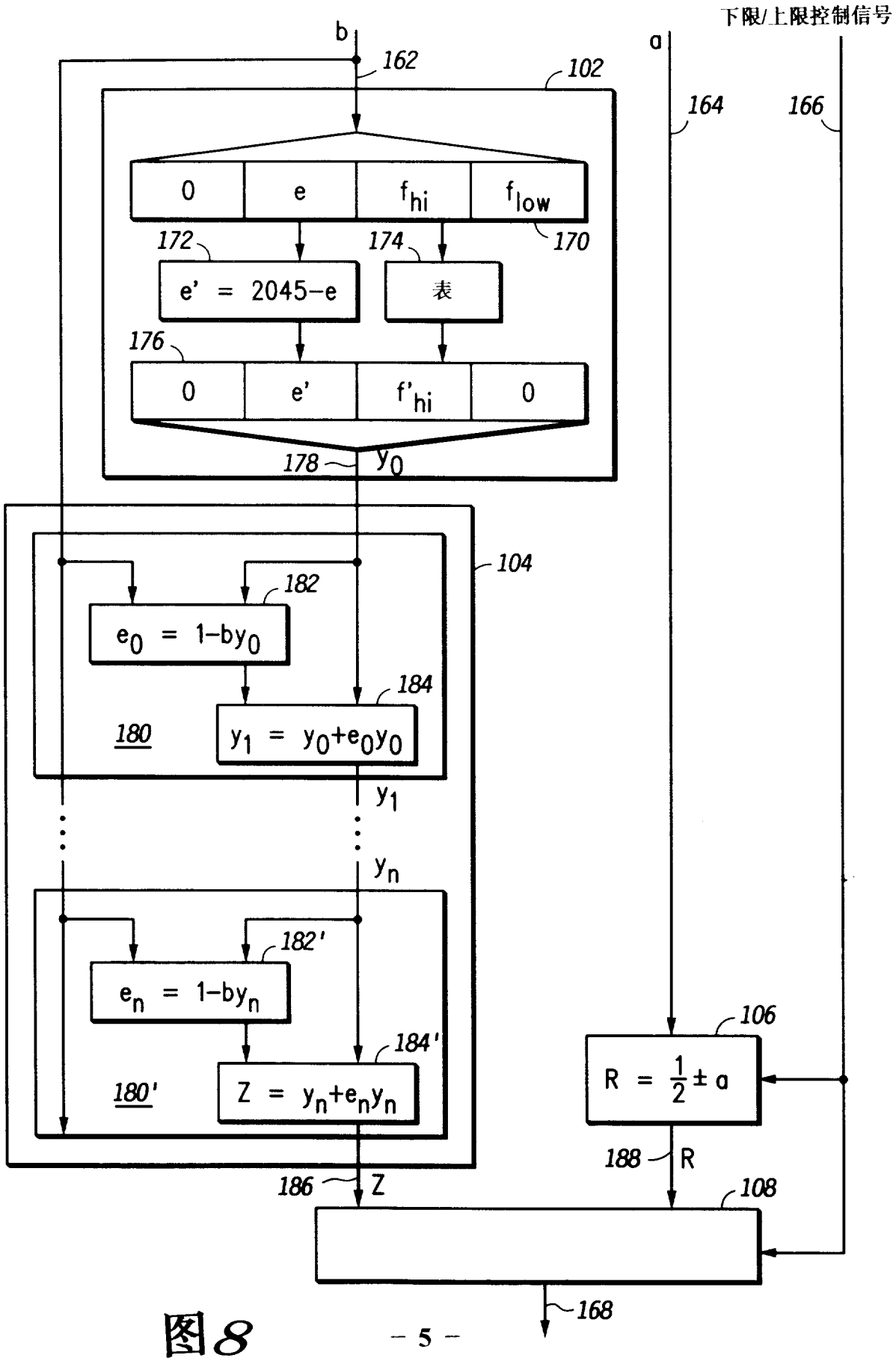

下限/上限控制信号
b
a
162
102
164
166
0
e
f_{hi}
f_{low}
170
172
174
e' = 2045−e
表
176
0
e'
f'_{hi}
0
178
y_0
104
182
$e_0 = 1-by_0$
184
180
$y_1 = y_0+e_0y_0$
y_1
y_n
182'
$e_n = 1-by_n$
184'
180'
$Z = y_n+e_ny_n$
106
$R = \frac{1}{2} \pm a$
188
R
186
Z
108
168

图8

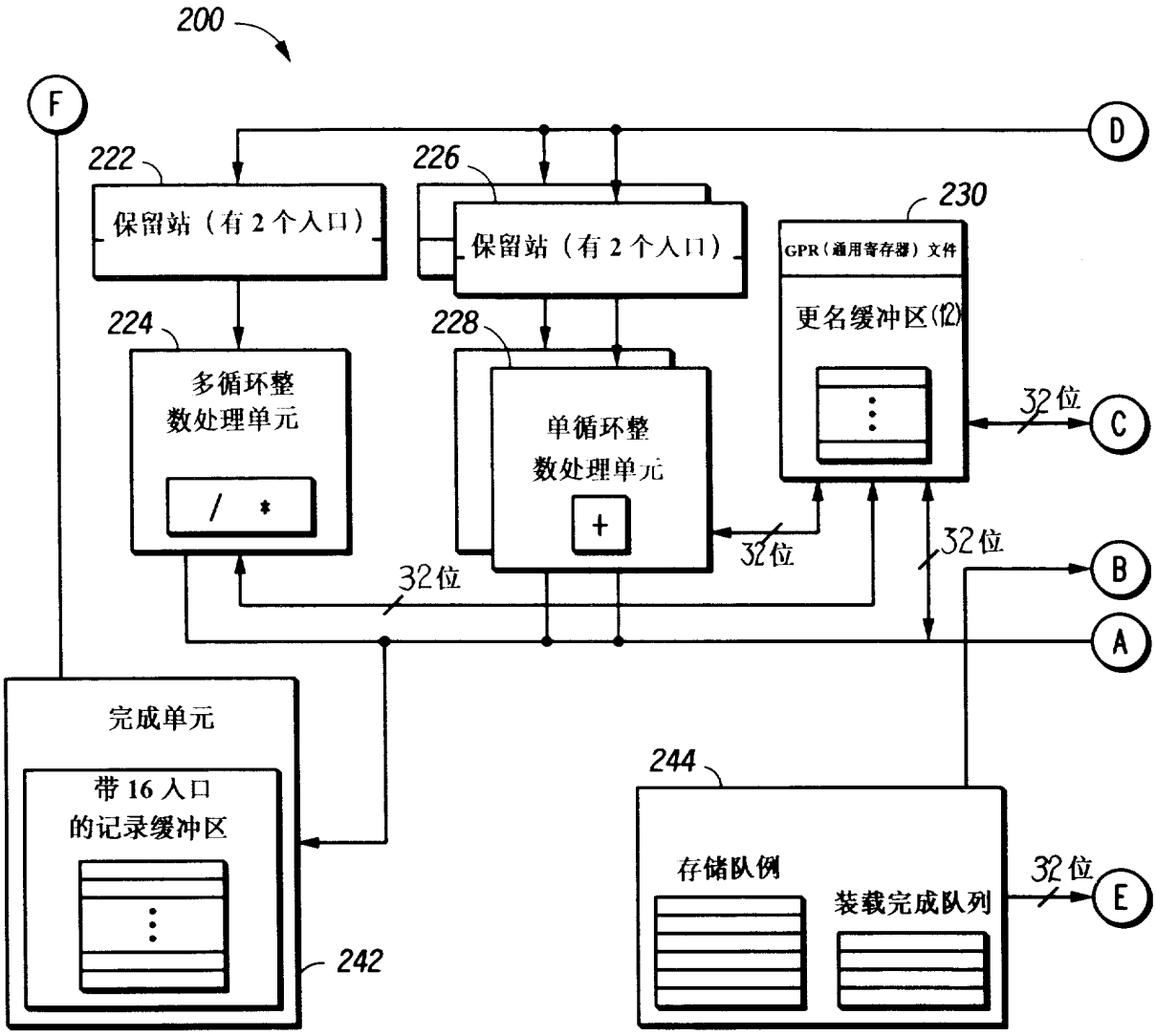

200
F
D
222
保留站（有 2 个入口）
226
保留站（有 2 个入口）
230
GPR（通用寄存器）文件
更名缓冲区(12)
224
多循环整
数处理单元
/ *
228
单循环整
数处理单元
+
32位
C
32位
32位
B
32位
A
完成单元
带 16 入口
的记录缓冲区
242
244
存储队例
装载完成队列
32位
E

图9

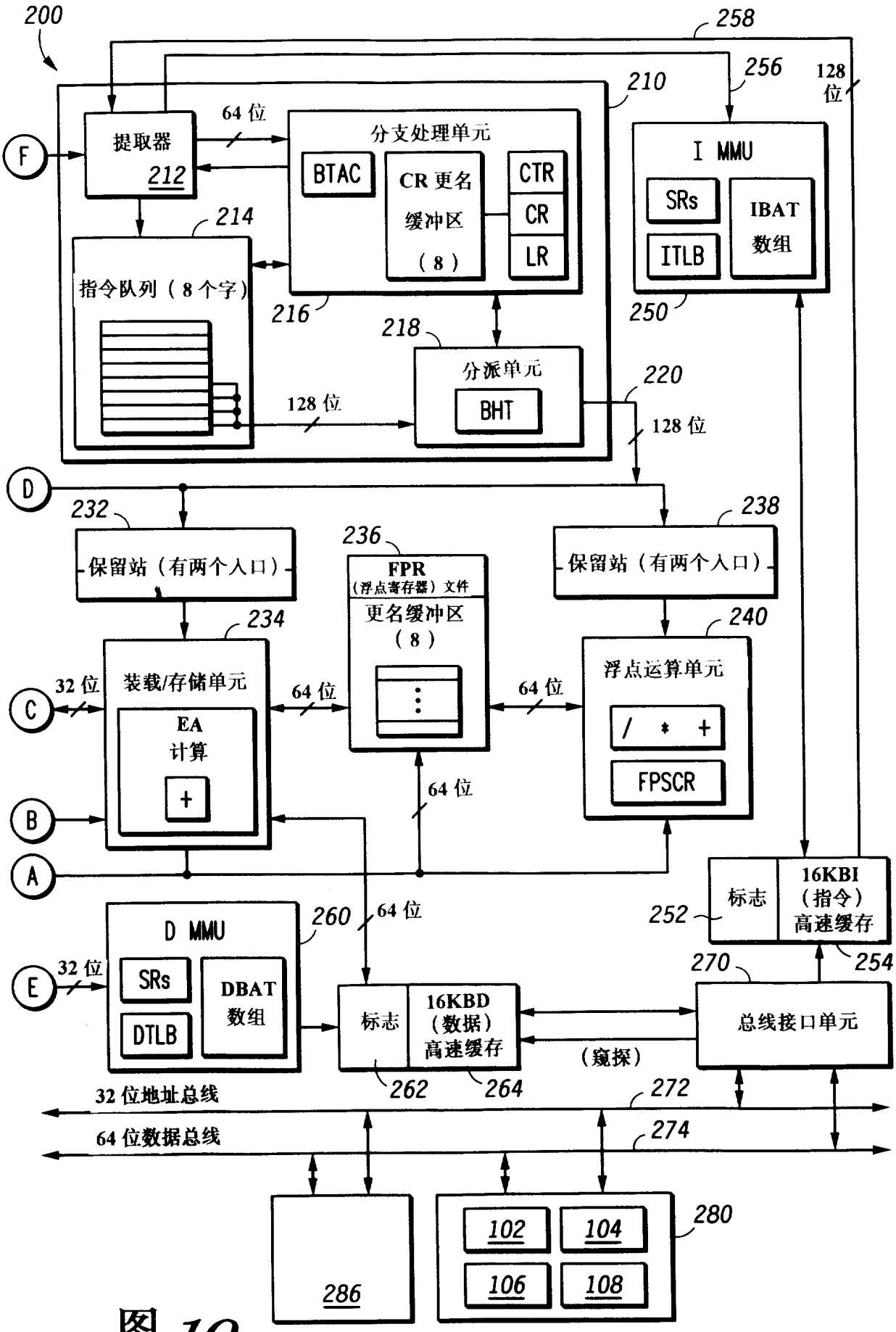

200
258
256
128 位
210
64 位
F
提取器
212
分支处理单元
BTAC
CR 更名缓冲区（8）
CTR
CR
LR
I MMU
SRs
IBAT 数组
ITLB
214
指令队列（8 个字）
216
250
218
分派单元
BHT
220
128 位
128 位
D
232
保留站（有两个入口）
236
FPR（浮点寄存器）文件
更名缓冲区（8）
238
保留站（有两个入口）
234
240
32 位
C
装载/存储单元
EA 计算
+
64 位
64 位
浮点运算单元
/ * +
FPSCR
64 位
B
A
标志
16KBI（指令）高速缓存
252
254
D MMU
260
64 位
32 位
E
SRs
DTLB
DBAT 数组
标志
16KBD（数据）高速缓存
270
总线接口单元
（窥探）
262
264
32 位地址总线
272
64 位数据总线
274
286
102
104
106
108
280

图 10