[19] 中华人民共和国国家知识产权局

[51] Int. Cl[7]
G06F 7/72

[12] 发明专利申请公开说明书

[21] 申请号 02824949.6

[43] 公开日 2005 年 4 月 6 日

[11] 公开号 CN 1605059A

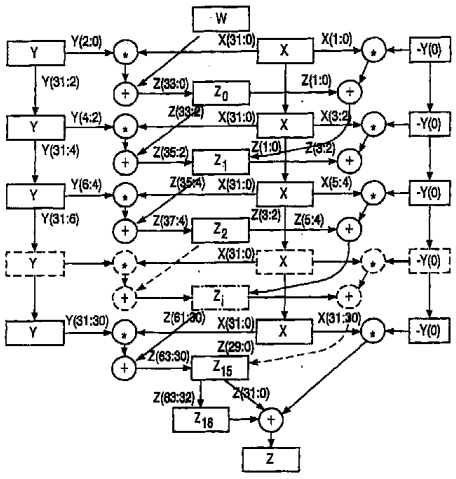
[22] 申请日 2002.12.5 [21] 申请号 02824949.6
[30] 优先权
[32] 2001.12.14 [33] EP [31] 01204908.6
[86] 国际申请 PCT/IB2002/005210 2002.12.5
[87] 国际公布 WO2003/052584 英 2003.6.26
[85] 进入国家阶段日期 2004.6.14
[71] 申请人 皇家飞利浦电子股份有限公司
地址 荷兰艾恩德霍芬
[72] 发明人 G·T·M·胡伯特
[74] 专利代理机构 中国专利代理(香港)有限公司
代理人 李亚非 王 勇

权利要求书 1 页 说明书 18 页 附图 4 页

[54] 发明名称 蒙哥马利乘法器中的流水线内核

[57] 摘要

一种安排用于以素数为模将第一长整数实体与第二长整数实体进行相乘的乘法器设备。特别地，它包括一种流水线乘法器内核，同时以蒙哥马利方式执行全部的乘法运算。



ISSN1008-4274

1．一种乘法器设备，该设备被安排用于以素数为模将第一长整数实体与第二长整数实体相乘，所述设备其特征在于包括一个流水线乘法器内核，同时以蒙哥马利方式执行全部的乘法。

2．如权利要求 1 所述的设备，还被安排用于实现改进的布斯算法。

3．如权利要求 2 所述的设备，还被安排用于以并行结构减去 y_0*X。

4．如权利要求 1 所述的设备，被安排用于以各个多项式的形式乘以所述第一和第二整数实体，并且在伽罗瓦域 GF(2^n)中以一个不可约的多项式为模。

5．如权利要求 1 所述的设备，其被应用于执行一种类似 RSA 的计算。

6．如权利要求 1 所述的设备，其被应用于执行一种椭圆曲线计算。

蒙哥马利乘法器中的流水线内核

本发明涉及一种乘法器设备，用于以素数为模将第一长整数实体乘以第二长整数实体，该乘法器设备包括一种流水线乘法器内核，并被安排用于以蒙哥马利方式执行所有的乘法运算。

发明背景

以素数为模的长整数乘法是一种基本和重复的运算，这些运算用于所谓的公共密钥系统和各种其它的应用．为了有效地使用这种应用，应该将乘法的执行时间最小化．因此，本发明尤其涉及一种如权利要求 1 前序部分所述的设备。为了上述目的，已经提出了通过使用一种乘法和归约运算(reduction operation)的组合的各种方法和设备。特别是，所述结果的最高有效部分实际上常常用来作用于所述归约。然而，本发明已经意识到这种最高有效部分的展开将实质上延迟所有的运算，尤其当现有硬件应该以最大可能高的工作周期使用时，这时由于这种乘法的次序必须以一种连续的方式来执行。

发明概述

因此，本发明的一个目的是提供一种如前序部分所述的乘法器设备，其中与现有技术相比，使用实际结果的最低有效部分作用于归约，同时提高实际使用所述硬件的时间片，以便能够处理这种基本的和长序列的乘法。

因此，现在根据本发明的其中一个方面，其特征在于根据权利要求 1 的特征部分。实际上，根据本发明的组合使其更容易保持流水线被充满，因此在叠加运算中缩短平均计算时间。所述乘法器设备可用于 GF(p)以及 GF(2^n)中的运算，其中 p 为素数。

在独立权利要求中阐述了本发明的其它有利方面，这些权利要求限定了本发明的基本组件应用的有利扩展，或者应用领域。

附图说明

参照公开的优选实施例，并特别参照所附附图，在下文将更加详

细地讨论本发明的这些和其它方面以及优点，附图示出了：

图 1，流水线乘法器的方框图；

图 2，用于长整数乘法 X*Y+W 的方框图；

图 3，用于长整数乘法 $X_i.Y+W$ 的方框图；

图 4，用于伽罗瓦域(Galois field)GF(2ⁿ)的流水线乘法器；

图 5，无进位传送加法器的结构；

图 6，图示中间进位和求和量的图。

优选实施例详述

1. 流水线乘法器

流水线乘法器被设计成在每个时钟周期接收两个将被相乘的新数字。两者的乘积在许多级中计算，并当所有级已经运算过时准备好该乘积。例如，对于 32*32 位的乘法，级数可以是 17（16 个用于乘法和 1 个用于最后的加法）。每个时钟周期，计算下一个乘积，但是仅在 17 个时钟周期后准备好结果。所以在同一个时钟周期至多同时处理 17 个乘积。

对于一种高效的流水线乘法器，已经设计出长整数计算从而保持流水线被充满。一种必须避免的情况是新的计算依靠正在进行的计算结果。在这种情况必须插入等待状态。这就是为什么设计蒙哥马利乘法器用于不同于类似 RSA 计算的椭圆曲线计算的原因。

紧接着所述乘法，乘法器执行两步加法 P=X.Y+A+B。其中一步加法两个长整数乘法所必需的，其中该乘法必须被分成许多基本的 32*32 位乘法。

流水线乘法器能够被设计用于不同的位数，例如 8*8 或 16*16。

2. 蒙哥马利乘法

蒙哥马利乘法计算乘积 $P=x.y.R^{-1} \bmod p$。这里，x 和 y 是要被相乘的输入，p 是乘法的模数。此外，$R=2^n$，其中 n 是该系统的位数，例如 1024 用于类似 RSA 的系统，和 160 用于椭圆曲线。作为一个实例选择一种具有 17 级的 32*32 位乘法器。

3. 利用基数 B 的蒙哥马利乘法

这种方法适用于很大的值 N_w 以及 RSA.

- $B=2^{32}$，假定 32 位的处理器字长.
- $R=B^{N_w}$，N_w 是长整数的 32 位字的数目.
- a[i]是数字 a 的第 i 个 32 位字.
- T[0]是临时变量 T 的最低有效 32 位部分.

预先存储的常量:

- $m'=-(p^{-1}) \bmod B$（32 位宽）
- 素数 p

输入: a mod p, b mod p

输出: 蒙哥马利乘积: $MonPro(a,b)=a.b.R^{-1} \bmod p$

运算如下:

```
T=0
For i=0 to Nw-1
  { T=T+a[i].b;                    //Nw 乘法
    Ui=T[0].m' mod B;              //1 乘法
    T=(T+Ui.p)/B                   //Nw 乘法
  }
if T>p then T=T-p
```

4. 对于 512 位操作数的计算

在我们的实例中，a 和 b 由 16 个 32 位字组成. 首先从 i=0 开始计算 T=T+a[i].b.

第一个计算在时隙 0 开始，最后一个计算在时隙 15 开始 . 在时隙 16 增加一个等待周期.

在时隙 17 准备好第一个结果 T[0].然后我们从这个时隙开始计算乘积 Ui=T[0].m'，该乘积在时隙 34 输出.

下一系列的计算是 T=(T+Ui.p)/B，其在时隙 34 开始，并在时隙 49 结束. 其第一个结果在时隙 51 输出，但因为该结果总是为零所以将其丢弃. 第二个结果是在时隙 52 输出.

从时隙 52 开始，重新开始循环. 当前一轮结果准备好时，它立刻使用前一轮的结果.

这里有 16 轮，所以时隙总数是 16*52=832。

在时隙 848 准备好完全的结果。

5. 对于 1024 位操作数的计算

首先从 i=0 开始计算 T=T+a[i].b。

我们以计算最初的 17 个乘积开始。

第一个结果 T[0]在时隙 17 准备好。在该时隙中我们计算乘积 Ui=T[0].m'，该乘积在时隙 34 输出。从时隙 18 直到 32，我们计算 T=T+a[i].b 的剩余的乘积。

下一系列计算(T+Ui.p)/B 从时隙 34 开始并在时隙 65 结束。当在时隙 66 开始新一轮计算时，准备好第一个结果。

这里有 32 轮，所以时隙总数是 32*66=2112。

在时隙 2128 准备好完全的结果。

6. 对于 2048 位操作数的计算

首先从 i=0 开始计算 T=T+a'[i].b'。

我们以计算最初的 17 个乘积开始。

第一个结果 T[0]在时隙 17 准备好。在该时隙中我们计算乘积 Ui=T[0].m'，该乘积在时隙 34 输出。从时隙 18 到 N_w，我们计算剩余的乘积。

下一系列计算(T+Ui.p)/B 从时隙 N_w 开始，并在时隙 $2N_w-1$ 结束。

当在时隙 $2N_w$ 开始新一轮计算时，准备好第一个结果。

这里有 N_w 轮，所以时隙总数是 $N_w.(2N_w+1)$。

在时隙 $N_w.(2N_w+1)+17$(对于 2048 位，等于 8273)准备好完全的结果。

7. 利用基数 R 的蒙哥马利乘法

该算法适用于小值 N_w，也适用于椭圆曲线。

- $B=2^{32}$（假定为 32 位的处理器字长）
- $R=B^{N_w}$（N_w 是长整数的 32 位字的数目）

预先存储的常量:

• $m'=-(p^{-1}) \bmod R$（m'是 N_w 32 位宽）

• 素数 p

输入：a mod p，b mod p

输出：$MonPro(a,b)=a.b.R^{-1} \bmod p$

T=a.b

U=T.m' mod R

T'=T+U.p

T=T/R

if T>p then T=T-p

对于 GF(2^n)上的系统，所有加法都是按模 2 计算。这里，m'是多项式 $B=\alpha^{32}$ 倒数。

8. 计算方法

首先，计算 T=a.b 的完全积。这需要 N_w^2 次乘法。然后准备好 T 的第一个结果，因此我们在此之后就可以立刻开始。对于乘积 T.m'，我们仅需要计算比 R 更小的乘积。

• 在时隙 17 准备好乘积 T[0]。T[0]*(m[0]..m[N_w-1])的计算在时隙 N_w^2 开始，并需要 N_w 次乘法。

• 在时隙 17+N_w 准备好乘积 T[1]。T[1]*(m[0]..m[N_w-2])的计算在时隙 $N_w^2+N_w$ 开始，并需要 Nw-1 次乘法。

• 在时隙 17+2N_w 准备好乘积 T[2].T[2]*(m[0]..m[N_w-3])的计算在时隙 $N_w^2+2N_w-1$ 开始，并需要 N_w-2 次乘法，等等。

• 在时隙 17+j.N_w 准备好乘积 T[j]。T[j]*(m[0]..m[N_w-j-1])的计算在时隙 $N_w^2+(2N_w-j+1).j/2$ 开始，并需要 N_w-j 次乘法，等等。

• 在时隙 17+(N_w-1).N_w 准备好乘积 T[N_w-1].T[N_w-1]*m[0]的计算在时隙 $N_w^2+(N_w+2).(N_w-1)/2$ 开始，并需要 1 次乘法。

可以证明，对于 $N_w \geq 5$，在新的乘积 T[j]*m[0]开始之前，总是准备好乘积 T[j]。因此，不需要等待周期。

• U[0]在时隙 N_w^2+17 准备好。从该时刻起，计算乘积 U.p。

• 最后一次乘法在时隙 $N_w^2+(N_w+2).(N_w-1)/2+1$ 开始。对于 N_w=5，最后一次乘法在在时隙 40 开始，并且 U[0]在时隙 42 开始。因此这里

就要求有两个等待周期．对于更大的值 N_w，就不需要有等待周期．

- U.p 的计算占用 N_w^2 个时隙．
- 对于 $N_w>5$，时隙总数是 $2.N_w^2+(N_w+2).(N_w-1)/2+1$
- 对于 $N_w=5$，时隙总数是 67 个时隙．
- 完全的结果在时隙 $2.N_w^2+(N_w+2).(N_w-1)/2+18$ 准备好．

9．改进的布斯(BOOTH)算法

改进的布斯算法设计为用被乘数的两位来作部分乘法．这使部分乘法的数量减少了一半．

首先重新编码乘数 Y，其中 y'_i 可以具有值-2，-1，0，+1 和+2（有符号数字符号）．

$y'_i=-2.y_{i+1}+y_i+y_{i-1}$（仅为 i 的偶数值定义）

$Y=y'_{30}.2^{30}+y'_{28}.2^{28}+...+y'_0.2^0$

$y_i'=-2.y_{i+1}+y_i+y_{i-1};$

$y_n=0$

$$p=\sum_{i\ 1,odd}^{n-1} y'_i.2^i.X=\sum_{i-1,odd}^{n-1}(-2y_{i+1}+y_i+y_{i-1}).2^i.X=$$

$$=-X\sum_{i=1,odd}^{n-1} y_{i+1}.2^{i+1}+X.\sum_{i-1,odd}^{n-1} y_i.2^i+2X.\sum_{i-1,odd}^{n-1} y_{i-1}.2^{i-1}=$$

$$=-X\sum_{i-2,even}^{n} y_i.2^i+X.\sum_{i=1,odd}^{n-1} y_i.2^i+2X.\sum_{i=0}^{n-2} y_i.2^i=$$

$$=-X\sum_{i-0,even}^{n-2} y_i.2^i+X.\sum_{i=1,odd}^{n-1} y_i.2^i+2X.\sum_{i=0}^{n-2} y_i.2^i+y_0.X=$$

$$=X.\sum_{i-0}^{n-1} y_i.2^i+y_0.X=X.Y+y_0.X$$

为了得到正确的结果，我们必须从乘积减去 $y_0.X$．

10．并行进行的减法

$X=x_{31}.2^{31}+x_{30}.2^{30}+...x_1.2^1+x_0.2^0$

$Y=y_{31}.2^{31}+y_{30}.2^{30}+...y_1.2^1+y_0.2^0$

$W=w_{31}.2^{31}+w_{30}.2^{30}+...w_1.2^1+w_0.2^0$

$Z=X.Y+W.$

在这方面，图 1 是流水线乘法器实施例的方框图．这里，具有星号的圆圈执行乘法运算，但具有加号标记的圆圈执行加法运算，诸如

一种包括进位存储运算的加法运算。各种方框将临时保留那里指示的量。为了更清楚起见，各种相互连接示出了一道将被传送的位的位排列顺序。在右侧，一列方框用于引入必要的修正项。

左侧部分计算 $Z=X.Y+W+y_0.X$。这个末项是该算法的人为项，右侧部分与其它计算并行减去该末项。这就是本发明。

以下实施例公开了乘法是怎样建立的，但这种实现可以在细节上有所偏差。

在第一个时隙计算 $Z_0=X.Y(1:0)+W_0$ 并将它存储在寄存器 Z_0 中。X 被传送到第二 X 寄存器，且 Y(31:2)被传送到第二 Y 寄存器。

在第二个时隙计算 $Z1=X.Y(3:2)+Z_0$ 并将它存储在寄存器 Z_1 中。此外，X 被传送到第三 X 寄存器，且 Y(31:2)被传送到第三 Y 寄存器。

而且计算 $-y_0*X(1:0)$，并把它加到 Z(1:0)，等等。

在第 16 个时隙计算 $Z_{15}=X.Y(31:30)+Z_{15}$，并将它存储在寄存器 Z_{15} 中。

此外还计算 $-y_0*X(31:30)$，并把它加到 Z(31:30)。

现在，Z_{15} 包含了 64 位。

在最后的时隙(#17)，较高的 32 位传送到 Z_{16} 和 Z_{15}，并且两个修正位被加到前面的 Z_{16} 值，然后输出。

当进行长整数乘法时，与 X0，X1，...，XN_{w-1} 组合输入 N_w 次 Yi。在长整数计算开始时，Z_{16} 设置为 0。仅当 $X_0.Y_i+W$ 达到输出 Z 时，然后才加 $Z_{16}=0$。

11. 对于 GF(2^n)的蒙哥马利乘法

椭圆曲线计算也可以通过域 GF(2^n)来定义。

在这个域中所有的加法(这里通过“+”来表示)都是模 2 计算(异或)。

这个域中的多项式具有最多 n-1 次的幂。

所以当 n=32 时，X 和 Y 的多项式定义如下(所有系数都是 0 或 1)：

$$X=x_{31}.\alpha^{31}+x_{30}.\alpha^{30}+...x_1.\alpha^1+x_0.\alpha^0$$

$$Y=y_{31}.\alpha^{31}+y_{30}.\alpha^{30}+...y_1.\alpha^1+y_0.\alpha^0$$

还有一个不可约的多项式 p，被定义为：

$$p=p_n.\alpha^n+p_{n-1}.\alpha^{n-1}+...p_1.\alpha^1+P_0\alpha^0.$$

乘积 $P=X.Y \bmod p$ 通过以下式子来计算:

$(x_{n-1}.\alpha^{n-1}+x_{n-2}.\alpha^{n-2}+...x_1.c^1+x_0.\alpha^0).(y_{n-1}.\alpha^{n-1}+y_{n-2}.\alpha^{n-2}+...y_1.\alpha^1+y_0.\alpha^0) \bmod p$.

然后由多项式 p 来除乘积 X.Y，余数就是结果。余数的幂总是小于 p 的幂。

X，Y 和 p 都可用长整数来表示。

约化乘积的计算可以通过正常乘积计算来进行，并具有这样的修改，内部加法利用模 2 计算进行。然后除了模 2 加以及保留余数以外，照常规进行除法。

然而，还能够更快地进行蒙哥马利乘法。

12. 蒙哥马利乘法

蒙哥马利乘法把许多素数（不可约多项式）加到（部分）乘积中，这样乘积被一个诸如 α^{32} 或 α^{160} 的适当因数 R 可除尽。对于给定多项式的二进制表示，可以考虑用 2^{32} 或 2^{160} 代替。这里，m'定义为 $m'=p^{-1} \bmod R$，其中 p^{-1} 定义为 $p.p^{-1} \bmod R=1$

现在，通过以下修改能够应用相同的算法:

- 乘法器中的加法运算是模 2 的.
- 省略最后的减法.

这种方案的进一步细节。除了上述的以外，各种其它细节、实施例、和说明将在下面作为补充的形式提出。

13. 长整数乘法器和加法器

定义:

- $X=x_{Nw-1}.B^{Nw-1}+...+x_2.B^2+x_1.B^1+x_0.B^0$
- $Y=y_{Nw-1}.B^{Nw-1}+...+y_2.B^2+y_1.B^1+y_0.B^0$
- $P_i=p_{iNw}.B^{Nw-1}+...+p_{i2}.B^2+p_{i1}.B^1+p_{i0}.B^0$
- $P=p_{2Nw-1}.B^{2Nw-1}+...+p_2.B^2+p_1.B^1+p_0.B^0$
- $B=2^{32}$
- $m=N_{w-1}$

长整数乘法涉及很多两个 32 位字的乘法。该实施例使用一种流水线 32 位乘法器（参见图 1 和 4），这种乘法器在每个时隙接收三个

新 32 位操作数(X*Y+Z)。这种乘法器速度非常快。然而，这种乘法的输出仅在 17 个时隙后准备好。所以最多可以有 17 个乘法同时计算。可是，当想要与正在进行相乘的结果相乘的时候，必须等待将该结果被准备好。这样能够引入等待周期，因此它将降低性能。

$Z=X.Y+W$

$Z=X.\{Y_0.B^0+Y_1.B^1+..Y_n.B^n\}+W$

Z, X 和 W 具有大小为 N_w-1 的 32 位字。Y_i 具有 32 位宽。

$W=W_0.B^0+W_1.B^1+..W_n.B^n\}$

中间结果 $W_i=W_{i1}.B^0+W_{i2}.B^1+..W_{i,m+1}B^n$

- 计算 $P_0=X.Y_0+W$。结果以 $P_0=W_0.B+Z_0$ 分解。
- 计算 $P_1=X.Y_1+W_0$。结果以 $P_1=W_1.B+Z_1$ 分解。
- 计算 $P_2=X.Y_2+W_1$。结果以 $P_2=W_2.B+Z_2$ 分解。
- ...
- 计算 $P_n=X.Y_n+W_{n-1}$。
- 对于 $j\geq m$，$Z_j=P_{n.j}$。

所以我们需要一个函数来计算 $P_i=X.Y_i+W_i$。

在这方面，图 2 是一个用于计算(X*Y+W)的方案的方框图。

$P_i=X.Y_i+W$

这个计算是前述计算的一部分。

X 和 W 具有大小为 $m=(N_w-1)$ 的 32 位字。

Y_i 具有 32 位宽。

- 计算 $S_1=x_0.y_i+w_0$。S_1 以 $Z_1.B+P_0$ 分解。
- 计算 $S_2=x_1.y_i+w_1+Z_1$。S_2 以 $Z_2.B+P_1$ 分解，等等。
- 计算 $S_n=X_n.y_i+W_n+Z_{n-1}$。S_n 以 $P_{n+1}.B+P_n$ 分解。

在图 3 中已经示出了相关的实施例，该图是一个用于根据(X*Y+W)执行长整数乘法的方案的方框图。

利用图 1 的流水线乘法器计算 S=x.y+w+z，已在上文讨论。

- 为了在 GF(2ⁿ)上计算，加法是模 2 计算的。所以没有进位。

14. 流水线乘法器

$X=x_{31}.2^{31}+x_{30}.2^{30}+...x_1.2^1+x_0.2^0$

$Y=y_{31}.2^{31}+y_{30}.2^{30}+...y_1.2^1+y_0.2^0$

$W=W_{31}.2^{31}+W_{30}.2^{30}+...W_1.2^1+W_0.2^0$

$Z=X.Y+W.$

左侧部分计算 $Z=X.Y+W+y_0.X$。末项是所用算法的人为项。右侧部分减去所述末项。

下面给出了一种想法，即乘法是怎样建立的，但实现可能在细节上有所偏差。

在第一个时隙中，计算 $Z_0=X.Y(1:0)+W_0$ 并把它存储在寄存器 Z_0 中。X 被传送到第二 X 寄存器并且 Y(31:2)被传送到第二 Y 寄存器。

在第二个时隙中，计算 $Z_1=X.Y(3:2)+Z_0$ 并把它存储在寄存器 Z_1 中。X 被传送到第三 X 寄存器并且 Y(31:2)被传送到第三 Y 寄存器。

此外，计算 $-y_0*X(1:0)$ 并把它加到 Z(1:0)。

...

在第 16 个时隙中，计算 $Z_{15}=X.Y(31:30)+Z_{15}$ 并把它存储在寄存器 Z_{15} 中。

此外，计算 $-y_0*X(31:30)$ 并把它加到 Z(31:30)。现在 Z_{15} 包括 64 位。

在最后一个时隙（#17）中，较高的 32 位被传送到 Z_{16} 和 Z_{15} 中，并且两个修正位被加到输出的 Z_{16} 的前值。

进行如在第 13 段中描述的长整数乘法，然后与 $X_0, X_1, ..., X_{NW-1}$ 组合输入 N.次 Y_i。当 $X_0.Y_i+W$ 达到输出 Z 时，那么就代替加 Z_{16} 的内容而不加任何内容。Z_{16} 具有第 13 段中 Z_i 的函数：从一个乘法传送到下一个乘法的那个部分。

15. 改进的布斯算法

首先重新编码乘法器 Y，其中 y'_i 仅可以具有值-2，-1，0，+1 和+2（有符号的数字表示）。

$y'_i=-2.y_{i+1}+y_i+y_{i-1}$（仅对 i 的偶数值定义）

$Y=y'_{30}.2^{30}+y'_{28}.2^{28}+...+y'_0.2^0$

例如当 $y=29_{dec}=011101_{bin}$ 时，那么 $y'=(2\underline{1}1)_{sd}=2.2^4-1.2^2+1=29_{dec}$，其中$\underline{1}$表示-1。

使用的这些公式是在以前段落中公开的有关改进布斯算法的内容

（第 9 段）。

为了得到正确的结果，我们必须从乘积 $y_o.X$ 中减去。

用 2 相乘就是左移 1 位被乘数。

部分乘积被以一种 2 根值表示法编码，其中每个乘积都可以具有值-1，0 或+1。

现在以 16 级来计算乘积。在每级中计算 $y'_i.X.2^i$ 的部分乘积，并把它相加到在前结果中，例如当 $x=53_{dec}=110101_{bin}$ 和 $y=29\ (y'=(2\underline{1}1)_{sd})$ 时，那么

对于 32 位操作数，要进行 15 个加法。在一个普通全加器中，这花费非常长的时间，因为进位必须波动通过。为了防止这种现象，我们将使用一种无进位传送加法器。在这方面，图 5 示例了一种无进位传送加法器的方案。

16. 冗余二进制计数法

加法器的被加数和加数都使用一种冗余二进制计数法，这种方法也是一种有有符号数字表示法。具有一个固定根 2 和一个数字组 $\{\underline{1}, 0, 1\}$，其中 $\underline{1}$ 表示-1。一个 n 位数字的冗余二进制整数 Y 具有值 $y_{n-1}2^{n-1}+y_{n-2}2^{n-2}+...+y_1.2^1+y_o.2^0$，其中 y_i 可以具有值-1，0 或 1。

在冗余二进制计数法中可以有多种方式表示一个整数，例如 $[0101]_{SD2}=[011\underline{1}]_{SD2}=[1\underline{1}01]_{SD2}=[1\underline{11}\underline{1}]_{SD2}=[10\underline{11}]_{SD2}=5_{dec}$。只有‘0’具有唯一的表示：[00...0]。

从标准二进制计数法变换到冗余二进制计数法是简单的：两者都是一样的。

从冗余二进制计数法变换到标准二进制计数法通过以下减法来进行：$X_{bin}=X^+-X^-$，其中 X^+ 是通过用‘0’替代所有的‘$\underline{1}$’而从 X_{sd1} 中得到的，X^- 是通过用‘0’替代所有的‘1’和用‘$\underline{1}$’替代所有的‘1’而从 X_{sd2} 中得到的。

例如，当 $X=[10\underline{1}\underline{1}]_{SD2}=5_{dec}$ 时，那么 $X^+=[1000]_{bin}=8_{dec}$ 和 $X^-=[0011]_{bin}=3_{dec}$。

通过用‘$\underline{1}$’替代所有的‘1’和用‘1’替代所有的‘$\underline{1}$’来对一个变量求反。例如，当 $X=[10\underline{1}\underline{1}]_{SD2}=5_{dec}$，那么 $-X=[\underline{1}011]_{SD2}=-5_{dec}$

我们将一个变量编码如下（参见表 1）：

X	输出 x^+x^-	输入 x^+x^-
0	00	00
1	10	1x
$\bar{1}$	01	x1

表 1 对于 GF(p)以冗余二进制计数法对 X 进行编码（x=任意）。从来不使用组合 11。

因此，当输入 X 并且 X=1 时，比条件 $x^+=1$ 充分。同样，当 X=$\bar{1}$ 时，也比 $x^-=1$ 充分。

17. 无进位传送加法器

所选择的表示法是，在下一个数字吸收可能的进位，并不影响下一个进位。因此，这种加法器的速度比 32 位全加器快得多。

至于 32*32 位乘法器，有 16 个加法（包括在前乘法的上面最高有效字）。然后仅在末端，所述冗余二进制计数法转换成标准二进制计数法。这种转换不是自由传送的。

加法以（在概念上）2 步来进行。首先计算中间和 s_i 以及中间进位 c_i。在下一步，两者都转换成最终和（sum_i）。该中间进位可以最可能取决于在前和当前的数字值，但不取决于再早一些的值。

c_i 和 s_i 满足以下等式：$2c_i+s_i = x_i+y_i$。而且 c_{i-1} 和 s_i 这样选择，使得两者既不是 1 也不是$\bar{1}$。

在这方面，图 6 所示为生成中间进位和求和量的图。

和 $S_i=c_i+s_{i-1}$ 将不再生成一个新的进位：

- 类型 1，3，4 和 6：$c_{i-1}+s_i=c_{i-1}$
- 类型 2a，5a：$c_{i-1}\neq\bar{1}$，也就是 0 或 1，所以 $c_{i-1}+s_i$ 是$\bar{1}$或 0。
- 类型 2b，5b：$c_{i-1}\neq 1$，也就是 0 或$\bar{1}$，所以 $c_{i-1}+s_i$ 是 1 或是 0。

这通过以下实例说明：

```
X        [10101001]sd2=87dec
Y        [11100111]sd2=101dec
         ----------+
S         01001110
C        11000101
```

```
         ---------+
Sum      11100010=188dec
```

18. 转换为标准二进制计数法

在最后一级，结果被转换成标准二进制计数法。$X=X^{+}-X^{-}$，其中 X^{+} 由所有 x_i^{+} 构成，以及 X^{-} 由所有的 x_i^{-} 构成。

因为 x_i^{+} 和 x_i^{-} 决不会同时为 1，所以我们不需要全减器。因此，我们尝试一种不同的方法。

我们将从右边到左边移去所有的1。

当不从右边借位时:

• 当下一个数字是 '1' 时，那么就保存该数字并且没有到左边的借位。

• 当下一个数字是 '0' 时，那么就保存该数字并且没有到左边的借位。

• 当下一个数字是 '1' 时，那么就用 '1' 替代该 '1' 并且有到左边的一个借位。

当有来自右边的借位时:

• 当下一个数字是 '1' 时，那么就用 '0' 替代该 '1'，并且没有到左边的借位。

• 当下一个数字是 '0' 时，那么就用 '1' 替代该 '0'，并且有到左边的一个借位。

• 当下一个数字是 '1' 时，那么就用 '0' 替代该 '1' 并且有到左边的一个借位。

然而，当最左的数字是 '1' 和最右的数字是 '1' 时，并且它们之间的所有数字是 '0' 时(10...01)，这将产生一个非常大的延迟。

为了减小该延迟，我们把 32 位分成 8 组，每组 4 位。

• 当最左的非零数字是 '1' 时，那么产生一个到下一个左边组的借位。

• 当该组中有至少一个 '1' 时，从右边组来的一个借位不传送到

下一个组。

19. 对于 GF(2N)的乘法器逻辑

$X=x_{31}.\alpha^{31}+x_{30}.\alpha^{30}+...x_1.\alpha^1+x_0.\alpha^0$

$Y=y_{31}.\alpha^{31}+y_{30}.\alpha^{30}+...y_1.\alpha^1+y_0.\alpha^0$

$W=w_{31}.\alpha^{31}+w_{30}.\alpha^{30}+...w_1.\alpha^1+w_0.\alpha^0$

对于这些向量的表示，在上述公式中可以看到用‘2’来代替‘α’.

$Z=X.Y\oplus W.$

$$Z=\sum_{i=0}^{31}\sum_{j=0}^{31}x_i.y_j\alpha^{i+j}$$

$Z_i=(y_{2i}\oplus y_{2i+1}.\alpha).X+Z_i$

在 GF(2ⁿ)中没有等式用于布斯编码。

在第一个时隙中（参见图 4），计算 X.Y(1:0)⊕W 并把它存储在寄存器 Z_0 中。X 被传送到第二 X 寄存器以及 Y(31:2)被传送到第二 Y 寄存器（Y）。

在第二个时隙中，计算 X.Y(3:2)⊕Z_0 并把它存储在寄存器 Z_0 中。X 被传送到第二 X 寄存器以及 Y(31:4)被传送到第三 Y 寄存器（Y）。

在第 16 个时隙中，计算 Z_{15}=X.Y(31:30)⊕Z_{15} 并把它存储在寄存器 Z_{15} 中。

现在 Z_{15} 包括 64 位。

在最后一个时隙(#17)中，较高 32 位被传送到 Z_{16} 以及 Z_{15} 被加到输出的在前 Z_{16} 值。

进行如在第 13 段中描述的长整数乘法，那么与 $X_0, X_1, ..., X_{NV-1}$ 组合输入 N_w 次 Y_i。当 $X_0.Y_i\oplus W$ 达到输出 Z 时，那么代替加 Z_{16} 的内容而不加任何内容。Z_{16} 具有第 13 段中 Z_i 的函数：从一个乘法传送到下一个乘法的那个部分。

特别是，图 4 示例了一种用于在 GF 2ⁿ 中运算的流水线乘法器实施例的方案。

加法

加法是 2 变量的 exor。没有进位。

编码

因为我们想与 GF(p)的逻辑组合所述逻辑，所以我们将使用以下冗余编码。这里 $X=x^+ \wedge x^-$，其中^表示一种逻辑“OR”功能。

X	输出 x^+x^-	输入 x^+x^-
0	00	0x
1	10	1x
1	10	X1

表 2 用于对 GF(2^n)以冗余二进制计数法编码 X（x=任意）

20. 用于 GF(p)和 GF(2^n)的逻辑

两种乘法器级都将使用以下结构 $z_j=a_i . x_{j-1} \oplus b_i . x_j$

GF(p)

GFp=1

$y'_i=-2 . y_{i+1}+y_i+y_{i-1}$(仅用于奇数 i，参见以下表 3)

y_{i+1}	y_i	y_{i-1}	y'_i	a_i	b_i	$sign_i$
0	0	0	0	0	0	1
0	0	1	1	0	1	1
0	1	0	1	0	1	1
0	1	1	2	1	0	1
1	0	0	-2	1	0	0
1	0	1	-1	0	1	0
1	1	0	-1	0	1	0
1	1	1	0	0	0	0

表 3 用于 GF(p)的编码

GF(2^n)

GFp=0

$a_i=y_{i-1}$

$b_i=y_i$

$z_j^+=z_j=a_i . x_{j-1} \oplus b_i . x_j$

$z_j^-=0$

组合

$$a_i=\left(\overline{y_{i+1}} . y_i \wedge . \overline{GFp}\right) y_{i-1} \wedge y_{i+1} . \overline{y_i} . \overline{y_{i-1}} . GFp$$

$b_i=y_i \oplus GFp.\ y_{i-1}$

$z_j^+=(a_i.\ x_{j-1} \oplus b_i.\ x_j).\ y_{i+1}$

$z_j^-=(a_i.\ x_{j-1} \oplus b_i.\ x_j).\ \overline{y_{i+1}}.\ GFp$

21. **无传送进位加法**

类型	x_i	y_I	x_{i-1}	y_{i-1}	f_2	f_5	h	中间进位 c_i	中间和 s_i
1	1x	1x	xx	xx	0	0	x	10	00
2a	1x	00	x0	x0	1	0	1	10	01
	00	1x	x0	x0	1	0	1		
2b	1x	00	x1	xx	1	0	0	00	10
	1x	00	xx	x1	1	0	0		
	00	1x	x1	xx	1	0	0		
	00	1x	xx	x1	1	0	0		
3	00	00	xx	xx	0	0	x	00	00
4	1x	x1	xx	xx	0	0	x	00	00
	x1	1x	xx	xx	0	0	x		
5a	00	x1	x0	x0	0	1	1	00	01
	x1	00	x0	x0	0	1	1		
5b	00	x1	x1	xx	0	1	0	01	10
	00	x1	xx	x1	0	1	0		
	x1	00	x1	xx	0	1	0		
	x1	00	xx	x1	0	1	0		
6	x1	x1	Xx	xx	0	0	x	01	00

表 4 中间进位及和

表 4 与利用根据表 1 编码的图 6 类似.

$f_2=x_i^+.\overline{y_i^+}.\overline{y_i^-} \wedge \overline{x_i^+}.\overline{x_i^-}.y_i^+$

$f_5=\overline{x_i^+}.\overline{x_i^-}.y_i^- \wedge x_i^-.\overline{y_i^+}.\overline{y_i^-}$

$h=\overline{x_{i-1}}.\overline{y_{i-1}^-}$

$c_i^+=x_i^+.y_i^+ \wedge f_2.h$

$c_i^-=x_i^-.y_i \wedge f_5.\bar{h}$

$s_i^+=(f_2 \wedge f_5)\bar{h}$

$s_i^-=(f_2 \wedge f_5)h$

$S_i=c_{i-1}+s_i.$

$c_{i-1}^{+}c_{i-1}^{-}$	$s_i^{+}s_i^{-}$	$S_i^{+}S_i^{-}$
00	00	00
00	1x	10
00	x1	01
1x	00	10
1x	1x	-
1x	x1	00
x1	00	01
x1	1x	00
x1	X1	-

表 5 总和 S_i

22. GF(2^n)

$S_i=x_i \oplus y_i$

看来如果我们取消 GF(p)系统中的进位，根据 GF(p)规则生成的 S_i 则产生正确的答案，S_i根据表 2 被编码。

组合逻辑

$$f_2=x_i^{+}.\overline{y_i^{+}}.\overline{y_i^{-}} \wedge \overline{x_i^{+}}.\overline{x_i^{-}}.y_i^{+}$$

$$f_5=\overline{x_i^{+}}.\overline{x_i^{-}}.y_i^{-} \wedge x_i^{-}.\overline{y_i^{+}}.\overline{y_i^{-}}$$

$$h=\overline{x_{i-1}}.\overline{y_{i-1}}$$

$$c_i^{+}=(x_i^{+}.y_i^{+} \wedge f_2.h).GFp$$

$$c_i^{-}=(x_i.y_i \wedge f_5.\overline{h}).GFp$$

$$s_i^{+}=(f_2 \wedge f_5)\overline{h}$$

$$s_i^{-}=(f_2 \wedge f_5)h$$

$$S_i^{+}=\overline{c_{i-1}^{+}}.\overline{c_{i-1}^{-}}.s_i^{+} \wedge c_{i-1}^{+}\overline{s_i^{+}}.\overline{s_i^{-}}$$

$$S_i^{-}=\overline{c_{i-1}^{+}}.\overline{c_{i-1}^{-}}.s_i \wedge c_{i-1}^{-}\overline{s_i^{+}}.\overline{s_i^{-}}$$

23. 从冗余二进制到二进制的转换

输入是一个 $x_i=\{\underline{1}, 0, 1\}$ 的向量 X。输出是一个 $y_i=\{0, 1\}$ 的向量 Y。

向量 X 被划分为每组 4 位的 8 个组，$i=4m+n$ (n=0..3, m=0..7)。

在各组之间：

- 当这个组中最左边非零数字是 '$\underline{1}$' 时，生成组借位 $g_{..}$

• 当所述组没有任何‘1’时，传送组借位 g_{n-1}。

在各组中：

• 当数字是‘1’时，生成一个借位 b_i

• 当数字不是‘1’时，传送一个借位 b_{i-1}：$b_i=b_{i-1}$

b_{i-1}	$x_i^+x_i^-$	y_i	b_i
0	00	0	0
0	1x	1	0
0	x1	1	1
1	00	1	1
1	1x	0	0
1	x1	0	1

表 6 一个组中的输出和借位

$$g_{4i+3}=x_{4i+3}^- \wedge \overline{x_{4i+3}^+}.x_{4i+2}^- \wedge \overline{x_{4i+3}^+}.\overline{x_{4i+2}^+}x_{4i+1}^- \wedge \overline{x_{4i+3}^+x_{4i+2}^+x_{4i+1}^+}x_{4i}^- \wedge \overline{x_{4i+3}^+}.\overline{x_{4i+2}^+}.\overline{x_{4i+1}^+}.\overline{x_{4i}^+}.g_{4i-1}$$

对于 $i\neq 0,4,8,\ldots,28$，$y_i=(x_i^+ \wedge x_i^-) \oplus b_{i-1}$

对于 $i=0,4,8,\ldots,28$，$y_i=(x_i^+ \wedge x_i^-) \oplus g_{4i-1}$

$b_i=x_i^- \wedge \overline{x_i^+}.b_{i-1}$

24. GF(2^n)

由于不存在借位，因此转换很简单。如果我们抑制所有的借位，那么 GF(p)的电路将给出正确的答案。

组合逻辑：

$$g_{4i+3}=x_{4i+3}^- \wedge \overline{x_{4i+3}^+}.x_{4i+2}^- \wedge \overline{x_{4i+3}^+}.\overline{x_{4i+2}^+}x_{4i+1}^- \wedge \overline{x_{4i+3}^+x_{4i+2}^+x_{4i+1}^+}x_{4i}^- \wedge \overline{x_{4i+3}^+}.\overline{x_{4i+2}^+}.\overline{x_{4i+1}^+}.\overline{x_{4i}^+}.g_{4i-1}$$

对于 $i\neq 0,4,8,\ldots,28$，$y_i=(x_i^+ \wedge x_i^-) \oplus (b_{i-1}.GFp)$

对于 $i=0,4,8,\ldots,28$，$y_i=(x_i^+ \wedge x_i^-) \oplus (g_{4i-1}.GFp)$

$b_i=x_i^- \wedge \overline{x_i^+}.b_{i-1}$

流水线乘法器

如图 1 中所示的流水线乘法器也能用于 GF(2^n)，但是在右侧部分的-Y[0]被设置为‘0’。上面描述了所有其它改进。

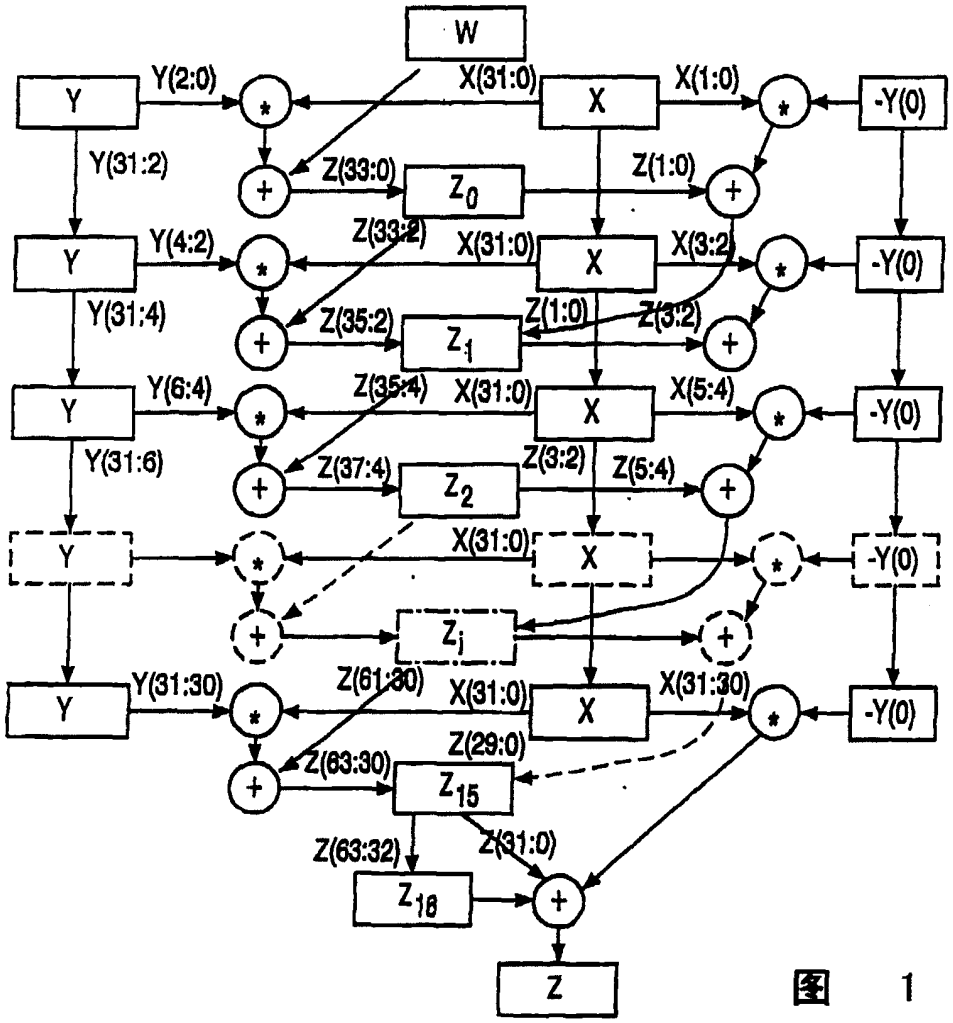

W
Y
Y(2:0)
X(31:0)
X
X(1:0)
-Y(0)
Y(31:2)
Z(33:0)
Z0
Z(1:0)
Y(4:2)
Z(33:2)
X(3:2)
Y(31:4)
Z(35:2)
Z1
Z(3:2)
Y(6:4)
Z(35:4)
X(5:4)
Y(31:6)
Z(37:4)
Z2
Z(5:4)
Zi
Y(31:30)
Z(61:30)
X(31:30)
Z(29:0)
Z(63:30)
Z15
Z(63:32)
Z(31:0)
Z18
Z

图 1

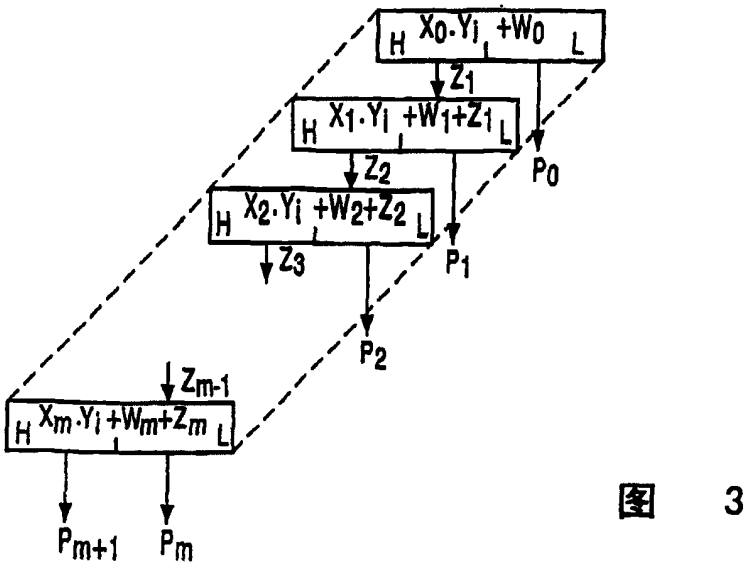

H X0·Yi +W0 L
Z1
H X1·Yi +W1+Z1 L
P0
Z2
H X2·Yi +W2+Z2 L
Z3
P1
P2
Zm-1
H Xm·Yi +Wm+Zm L
Pm+1
Pm

图 3

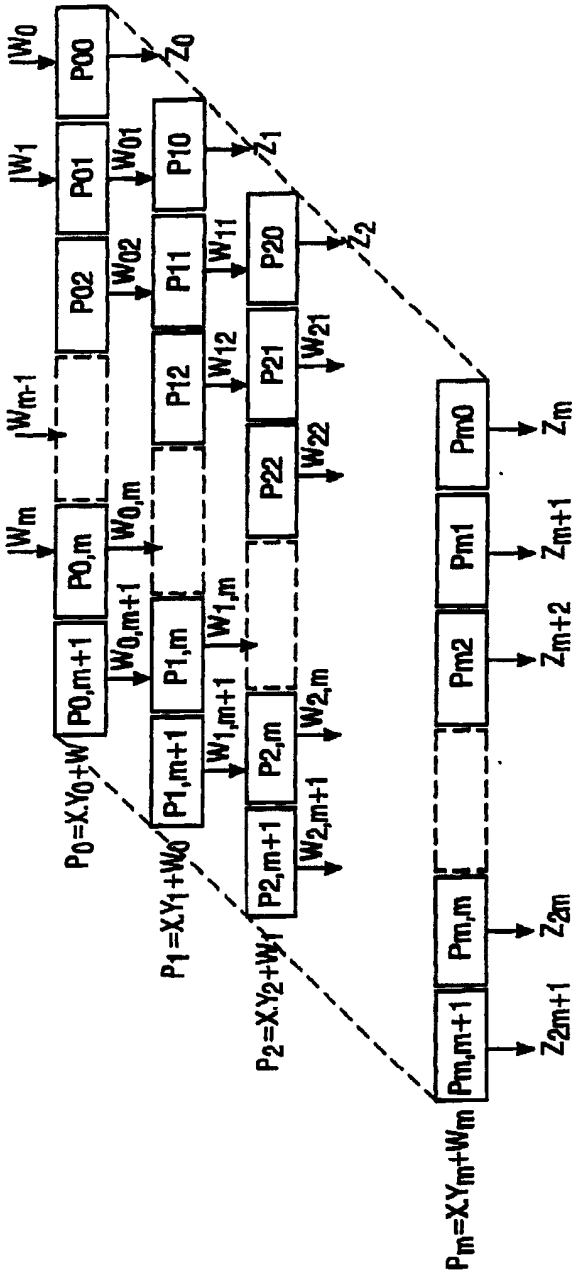

W0
W1
Wm-1
Wm
P00
P01
P02
P0,m
P0,m+1
P10
P11
P12
P1,m
P1,m+1
P20
P21
P22
P2,m
P2,m+1
Pm0
Pm1
Pm2
Pm,m
Pm,m+1
W01
W02
W0,m
W0,m+1
W11
W12
W1,m
W1,m+1
W21
W22
W2,m
W2,m+1
Z0
Z1
Z2
Zm
Zm+1
Zm+2
Z2m
Z2m+1
P0=XY0+W0
P1=XY1+W1
P2=XY2+W2
Pm=XYm+Wm

图 2

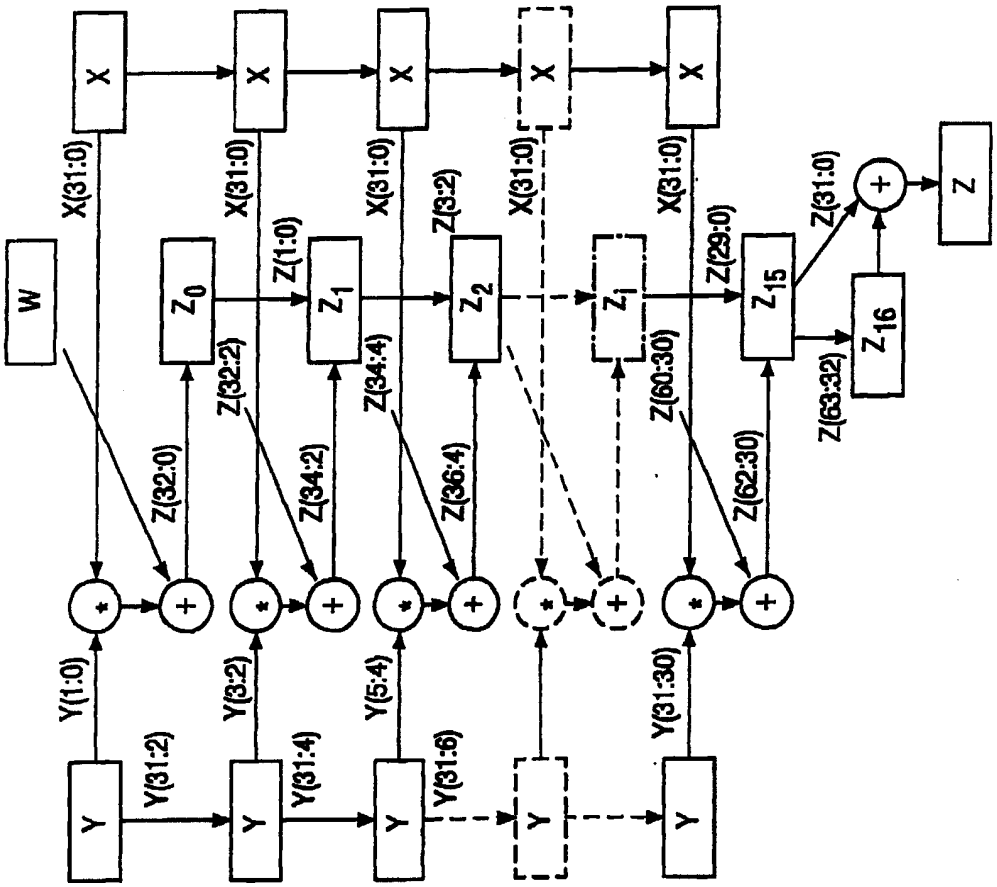

X
W
Y
Z0
Z1
Z2
Zi
Z15
Z16
Z
X(31:0)
Z(1:0)
Z(3:2)
Z(29:0)
Z(31:0)
Z(32:0)
Z(32:2)
Z(34:2)
Z(34:4)
Z(36:4)
Z(60:30)
Z(62:30)
Z(63:32)
Y(1:0)
Y(3:2)
Y(5:4)
Y(31:2)
Y(31:4)
Y(31:6)
Y(31:30)

图 4

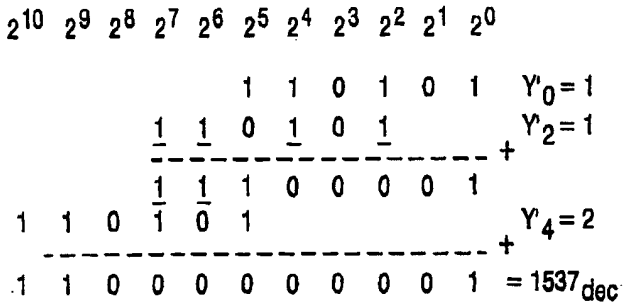


图 5

类型	被加数位 x_i	被加数位 y_i	前一数字 x_{i-1}, y_{i-1}	中间进位 c_i	中间求和 s_i
1	1	1	任意	1	0
2a	1 0	0 1	全为非负	1	$\underline{1}$
2b	1 0	0 1	一个或两个为负	0	1
3	0	0	任意	0	0
4	1 $\underline{1}$	$\underline{1}$ 1	任意	0	0
5a	1 $\underline{1}$	$\underline{1}$ 0	全为非负	0	$\underline{1}$
5b	1 $\underline{1}$	$\underline{1}$ 0	一个或两个为负	$\underline{1}$	1
6	$\underline{1}$	$\underline{1}$	任意	$\underline{1}$	0

图 6