[19]中华人民共和国国家知识产权局

[51]Int.Cl[7]
H03M 7/30

[12]发明专利申请公开说明书

[21]申请号 98805358.6

[43]公开日 2000年6月21日

[11]公开号 CN 1257621A

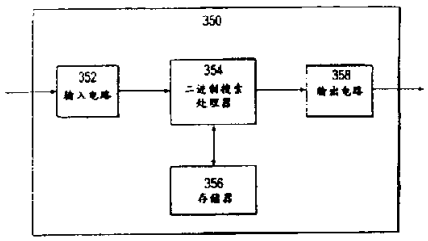
[22]申请日 1998.4.10 [21]申请号 98805358.6
[30]优先权
[32]1997.4.18 [33]US [31]08/837,481
[86]国际申请 PCT/US98/07074 1998.4.10
[87]国际公布 WO98/48516 英 1998.10.29
[85]进入国家阶段日期 1999.11.22
[71]申请人 首要研究有限公司
地址 美国内华达州
[72]发明人 R·E·兰利
[74]专利代理机构 上海专利商标事务所
代理人 钱慰民

权利要求书2页 说明书14页 附图页数7页

[54]发明名称 数字数据无损压缩的方法和设备

[57]摘要

数字数据无损压缩和解压缩的方法和设备。用二进制搜索过程处理输入数据，以便在一次迭代中获得标称压缩。对第一迭代输出进行反复迭代，为所有类型 的数字数据提供很大的压缩。本方法可以在个人计算机上用软件实施，或者利 用大量串联的相同压缩电路在 VLSI 硬件上实施，从而大量串联的压缩电路 形成节律流水式结构。



ISSN1008-4274

权　利　要　求　书

1. 一种数字数据压缩方法，其特征在于，包括以下步骤:

(a) 接收要压缩的数字数据;

(b) 将数字数据分成组，每组具有由 1 和 0 组成的预定序列;

(c) 根据预定过程处理一序列，通过删除位获得具有可变长度的压缩序列值;

(d) 将压缩序列值存储在临时存储区中;

(e) 产生一删除代码，该代码表示被删除位的数目;

(f) 根据预定标准产生一分隔代码;

(g) 按预定关系将删除代码和分隔代码与压缩序列值一起存储在临时存储区中;

(h) 对每个组执行步骤 c 至 g，以获得中间数据组;

(i) 将中间数据组分成序列;

(j) 根据预定标准，通过多次迭代重复进行步骤 c 至 i;

(k) 提供最后一次迭代的中间数据组，将其作为接收数字数据的压缩形式。

2. 如权利要求 1 所述的方法，其特征在于，处理序列的步骤包括用二进制搜索过程获得二进制搜索代码.

3. 如权利要求 1 所述的方法，其特征在于，产生分隔代码的步骤包括根据压缩序列值和删除代码的特性产生分隔代码.

4. 如权利要求 1 所述的方法，其特征在于，产生删除代码的步骤包括产生比被删除位数小一位的代码.

5. 如权利要求 1 所述的方法，其特征在于，产生删除代码的步骤包括产生这样的代码，该代码具有一个低位 0 位和多个高位 1 位，比从序列中删除的位数小 2 位，从而形成压缩序列值.

6.一种对经压缩的原始数字数据进行解压缩的方法，其特征在于，包括以下步骤:

(a) 接收输入数据，所述输入数据是经压缩的原始数字数据，包括由分隔

代码分隔的各种长度的压缩数据代码和删除代码，还包括一个迭代计数；

(b) 用分隔值从输入数据获得数据代码；

(c) 处理数据代码，以便获得解压缩值；

(d) 临时存储所述解压缩值；

(e) 重复步骤(b)、(c)和(d)，直至处理完所有的输入数据；

(f) 对计数器增值；

(g) 按分隔值从临时存储的解压缩值中获得数据代码；

(h) 处理所述数据代码，以便获得解压缩值；

(i) 临时存储所述解压缩值；

(j) 重复步骤(f)、(g)、(h)和(i)，直至计数器等于迭代计数；以及

(k) 输出被存储的解压缩值，用于代表原始数字数据。

7. 用于压缩数字数据的设备，其特征在于，包括：

输入端，用于接收要压缩的数字数据；

输出端，用于提供被压缩的数字数据；和

第一、第二和第三压缩电路，每个压缩电路包括：

输入电路，用于接收输入数据；

处理电路，它根据预定标准将接收到的输入数据分成序列；

二进制搜索处理器，它用二进制搜索过程对序列进行处理，通过删除低位位，获得与序列对应的压缩序列值；

代码发生器，它根据压缩序列值和删除代码的特性，产生表示被删除位数目的删除代码和分隔代码；以及

输出电路，用于提供压缩序列值、删除代码和分隔代码，将其作为输出数据；

第一压缩电路的输入电路与输入端相连，第一压缩电路的输出电路与第二压缩电路的输入电路相连，第二压缩电路的输出电路与第三压缩电路的输入电路相连，而第三压缩电路的输出电路与输出端相连。

8. 如权利要求 7 所述的设备，其特征在于，还包括多个第二压缩电路，它们串联连接在第一和第二压缩电路之间。

说　明　书

数字数据无损压缩的方法和设备

发明背景

本发明总体上涉及数字数据压缩，尤其涉及用于进行数字数据无损压缩和解压缩的方法和设备。

在许多重要的工业和商业应用中，使用用数字数据形式表示的信息。众所周知，数字数据与模拟数据不同，它是用一系列组成“字节”和“字”的1和0(称为二进制数字或“位”)表示的．例如，可以将文本和图像表示成数字数据组进行存储或传输。

有些类型的信息形成数字数据的交错量。例如，诸如脑瘤X射线等精细图像很容易需要几百万字节的数字数据。为了降低数据的存储和传输成本，已研究出“压缩”数字数据的方法，将原始信息转换成一组新的具有较小位数的数字数据。然后对压缩数据进行“解压缩”逆处理，以便恢复原始信息。

两种主要类型的数字数据压缩是“有损”压缩和“无损”压缩．有损数据压缩允许解压缩数据与原始数据不同，只要解压缩数据满足一些逼真标准。例如，当把图像转换成数字数据，然后对其压缩时，即使存储差异，也足以从解压缩数据中获得人眼看来与原始图像相同的图像．因此，有损压缩方法只能用于某些类型的数据，诸如视频和音频等。有损压缩的优点是，可以比无损压缩获得更高程度的压缩．压缩程度一般表示成一个比率．例如，如果原始数据的大小为压缩数据的二倍，则称该压缩方法具有2： 1压缩率．

与有损数据压缩相反，无损数据压缩在对数据进行压缩，然后解压缩时，提供了与原始数据完全相同的数据．无损压缩的优点是，可以用于要求完全恢复原始数据的各类信息．

无损数据压缩在20世纪50年代早期通过当时在贝尔实验室工作的David Huffman的工作，而广为人知． Huffman的方法事实上已成为无损压缩领域所有工作的基础。但是，尽管发展了40年，但目前关于无损数据压缩的方法和设备仍只能获得大约2： 1的平均压缩率．

许多已知的对文本数据进行无损压缩的现有技术依赖于预先已知(边工作时确定)字母符号在原始数据中出现的大概频率。例如，对于英文文本，目前的无损压缩方法通常依赖于这样的认识，即字母 E 和 N 出现的次数通常比字母 Q 和 Z 出现的次数多。于是，利用该信息开发一种代码结构，该代码结构将较短的代码分配给出现次数多的符号，而将较长的代码分配给出现次数较少的字母。尽管已知的方法在许多场合很有用，但希望提供这样的无损数字数据压缩方法和设备，这类方法和设备可以获得比现有技术高的压缩率，并且不要求预先知道有关源数据内容的信息。

本发明的目的是提供下述无损数字数据压缩/解压缩方法和设备，在这些方法和设备中，可以一个字节一个字节地恢复所有经编码的数据，原始信息不损失。本发明提供了可以获得很大压缩率的无损压缩方法和设备。

发明内容

本发明的优点和目的在以下说明中得到部分叙述，并且部分将从说明中显现出来，或者通过实施本发明而学习到。利用后附权利要求书中特别指出的元件及其组合，可以实现和获得本发明的优点和目的。

为了获得这些优点，并且依照本发明的目的，如这里所实施和概述的，本发明提供了无损数字数据压缩的方法和设备。具体地说，在第一方面，本发明是一种数字数据压缩方法。该方法包括以下步骤：接收要压缩的数字数据；将数字数据分成组，每组具有由 1 和 0 组成的预定序列；根据预定过程处理一序列，通过删除位获得具有可变长度的压缩序列值；并且将压缩序列值存储在临时存储区中。该方法还包括以下步骤：产生一删除代码，该代码表示被删除位的数目；根据预定标准产生一分隔代码；按预定关系将删除代码和分隔代码与压缩序列值一起存储在临时存储区中；以及对每个序列执行先前的步骤，获得中间数据组。该方法还包括以下步骤：将中间数据组分成序列；以及根据预定标准，通过多次迭代重复执行先前的步骤。最后的步骤是 提供最后一次迭代的中间数据组，将其作为接收数字数据的压缩形式。

在另一方面，本发明包括了用于无损数字数据压缩的设备，该设备包括：输入端，用于接收要压缩的数字数据；输出端，用于提供被压缩的数字数据；以及第一、第二和第三压缩电路。每个压缩电路包括：输入电路，用于接收输

入数据；处理电路，它根据预定标准将接收到的输入数据分成序列；二进制搜索处理器，它用二进制搜索过程对序列进行处理，通过删除低位位，获得与序列对应的压缩序列值；代码发生器，它根据压缩序列值和删除代码的特性，产生表示被删除位数目的删除代码和分隔代码；以及输出电路，用于提供压缩序列值、删除代码和分隔代码，将其作为输出数据。第一压缩电路的输入电路与输入端相连，第一压缩电路的输出电路与第二压缩电路的输入电路相连，第二压缩电路的输出电路与第三压缩电路的输入电路相连，而第三压缩电路的输出电路与输出端相连。

附图概述

说明书中包含的附图构成说明书的一部分，它们说明了本发明的实施例，并且与说明书一起用于说明本发明的原理。

图 1 是一逻辑流程图，示出了对数字数据进行无损压缩的方法。

图 2 是一逻辑流程图，示出了对经压缩的原始数字数据进行解压缩的方法。

图 3 是一逻辑流程图，示出了对图 2 方法中的分隔代码进行识别的方法。

图 4 是一逻辑流程图，示出了构成本发明另一实施例的对数字数据进行无损压缩的方法。

图 5 是一逻辑流程图，示出了对图 4 方法产生的数据进行解压缩的方法。

图 6 是 VLSI 定制集成电路，用图 3 所示的方法对数字数据进行无损压缩；以及

图 7 是一方框图，示出了用于形成图 5 中 VLSI 电路之组成电路的公共压缩电路。

较佳实施例的描述

本发明使用迭代技术，利用搜索过程在一次执行时获得一标称压缩率，然后在一系列执行或迭代后获得大得多的压缩。在本较佳实施例中，使用二进制搜索过程。例如利用该过程，可以在一次简单的迭代中将所有 8 位整数压缩为 7 位或更少。特别是，将所有奇整数压缩成 7 位，将所有偶整数压缩成 6 位或更少。对所有 8 位整数合起来获得的平均压缩为每个整数 2 位，在一次迭代后

对原始的 8 位整数产生平均为 6 位的压缩代码长度．这对应于每个整数 25 % 的总压缩．使用术语“总的”是因为额外开销将用掉一些压缩，致使每次迭代对每个整数的净压缩明显变小．但是，本发明使用连续迭代，最终获得非常大的压缩率。

现在就 8 位整数，概述众所周知的二进制搜索技术．首先，对所有 8 位整数选择数值中点．由于 8 位所能包含的最大整数是 225，所以中点是 128，或 2^7．接下来，判断目前正在编码的输入整数值是大于还是小于中点 128．如果输入值小于中点，则输出 0．如果输入值大于中点，则输出 1 位．如果输入值等于中点，则过程终止，不输出、

下一步是选择 2 的下一个较低次幂，在本例中，为 2^6，或 64．如果输入值大于先前的中点，则将该值与先前的中点相加，如果输入值小于先前的中点，则用先前的中点减去该值．此步骤产生 192 或 64．然后将输入值与这个新的中点比较．再一次，如果输入值小于该中点，则输出 0，如果大于中点，则输出 1 位．如果输入值等于中点，则过程终止．在用这种方式继续下去时，保留正在产生的输出位的数目．上述过程继续，直至正确定位了正在编码的当前值．该过程产生一个输出位流，称为二进制搜索代码，长度如上所述(即，如果输入值是奇数，为 7 位，如果输入值为偶数，为 6 位或更少)．因此，连续输入值的压缩输出由一系列二进制搜索代码组成，其包含各种长度的位序列．

为了对二进制搜索的压缩输出解压缩，解压缩方法必须能够确定它将解码的、构成二进制搜索代码的下一个位序列的长度．因此，本发明的较佳实施例提供了与二进制搜索代码相关的附加代码，称为“删除代码”和“分隔代码”．正是这些附加代码构成了上述额外开销．

在一较佳实施例中，本发明将输入数据分成“序列”．序列的定义是从一个或多个 1 开始，到一个或多个 0 结束(即，在下一个 1 位之前)的位串．

例如，假设以下位串是接收到的输入:

101101001100

这个串按顺序产生如下四个“序列”:

序列 1 = 10

序列 2 = 110

序列 3 = 100

序列 4-1100 .

对于随机数字，序列的平均长度为 4 位．用该方式定义的序列的最重要特点是，它们都以零结尾．其价值将在以下描述的序列压缩过程中体现出来，该过程能够产生一净压缩，足以使本发明较佳实施例成为一可行过程，呈现其迭代特性．也就是说，每次迭代或循环的压缩量相对较小，通过许多循环的累积，可以产生极大的压缩率，这是本发明的特征。

在本发明的较佳实施例中，用二进制搜索技术处理每个序列，产生经压缩的序列值，即由输入序列组成代码，但将某些位删除，具体地说，删除从低位(即，最右)的零位向上并包括第一个低位的 1 位．本较佳实施例在每个经压缩的序列值前面放一个删除代码，它表示用二进制技术从原始序列中删除的位数．下面，表 I 示出了删除代码．

表 I

删除位数	被删除的实际位	删除代码
2	10	0
3	100	10
4	1000	110
5	10000	1110

等等．

注意，在所有情况中，删除代码比被删除的位数小 1 位．这说明通过包括删除代码，压缩序列实现了一位压缩．

压缩序列值根据其特性分 4 种(称为“类”)．表 II 例示了这四类压缩序列，以下对每一类进行讨论．

表 II

源位序列	压缩序列*	类代码
110	(0)1	H
100	(10)	R
1100	(10)1	B
10	(0)	N

*删除代码示于括号中，以便与压缩序列中的其它位相区别。

类代码 H

将两个低位位“ 10 ”删除，并用删除代码“ 0 ”代替(参见上节)．注意，删除两个低位位留下 1 位，在压缩序列中跟在删除代码的后面．由于这一位留下来挂着，所以称为“挂位”．这里描述了称为 H 类的压缩序列，即具有挂位的压缩序列。

类代码 R

删除源序列“ 100 ”中三个低位位，并用删除代码“ 10 ”代替．该压缩序列没有挂位．但是，它在删除代码中有一高位 1 位．该特征可表征此类(即，删除代码 1 位)，并说明类代码“ R ”．

类代码 B

删除源位序列“ 100 ”中的三个低位位，并用删除代码“ 10 ”代替．这时，存在一个挂位和一个删除代码 1 位．由于它具有挂位和删除代码 1 位两者，所以类代码为“ B ”

类代码 N

删除“ 10 ”，产生删除代码“ 0 ”，并且没有挂位．由于该压缩序列既没有挂位也没有删除代码 1 位，所以其类代码为“ N ”．

上述例子描述了所有可能发生的压缩序列类型．显然， H 类可以有不止一个挂位， R 类可以有不止一个删除 1 位． B 类可以有多个挂位和多个删除代码 1 位．但类型名称不变。

如果将前一节中的四个压缩序列串在一起，形成一个压缩位流，那么它们将如下所示，这里同样将删除代码放在括号内:

(0)1(10)(10)1(0)

当对该压缩流解压缩时，当然没有括号帮助．因为根据本发明压缩的数据中的第一信息必须是删除代码，所以从左向右移，显然第一个零位构成删除代

码零．但是，下一个 1 位可以是挂位，或者是删除代码 1 位。在许多情况下，相邻的压缩序列在它们的边界处“掺混(bleed)”在一起。为了解决该问题，本发明的较佳实施例使用一个最终类型的附加代码，称为“分隔”代码。

根据以下规则应用分隔代码:

-当存在挂位(即，有一个或多个 1 位紧跟在删除代码后面)时，在最后一个悬挂的 1 位后面插入分隔代码零位。

-当不存在挂位而删除代码有一个或多个高位 1 位时，在下一个删除代码开始之前插入一个分隔代码零位。

以下是经修改的压缩代码流，分隔代码带下划线:

(0)1 0(10)0(10)1 0(0)

对压缩序列附加的分隔位牺牲了对序列的压缩，但不会引起序列的扩张。因为 75 % 的序列需要分隔位，净压缩达到 25 % 来自不需要分隔码。

现在举例描述此技术。考虑下述 32 位随机的源输入位流:

10100101100110100011100011101010(= 32 位)

表 III

	源序列	压缩序列(带删除代码)	类代码
1	10	(0)	N
2	100	(10)	R
3	10	(0)	N
4	1100	(10)1	B
5	110	(0)1	H
6	1000	(110)	R
7	111000	(110)11	B
8	1110	(0)11	H
9	10	(0)	N
10	10	(0)	N

用上述方式处理此输入位流，如上述表 III 中左列所示，输入首先被分隔成序列。然后，用二进制搜索技术对序列压缩，以获得表 III 中列所示的压缩

序列值，其中删除代码位于括号中。然后，在表 III 的右列示出了每个序列的类代码。

然后根据上述两个规则，对表 3 中列的压缩序列值进行分析，产生分隔代码。应用这些规则并添加分隔代码，产生以下压缩序列位流:

(0)0(10)(0)0(10)10(0)10(110)0(110)110(0)110(0)(0)　(=29 位，压缩 9.375 %)

同样，删除代码表示在括号中。分隔代码带下划线。可以看出，应用上述技术可以由 32 位的原始输入位流产生 29 位的压缩位流。这表示对于此特列，压缩了 9.35 %。

因此，

25 % 压缩(用删除代码)

× 25 % 对所有序列

每个循环 6.25 % 的净压缩。

在使用“节律流水式”结构的 VLSI 上执行本发明的一个实施例，在该设计中，将相同的电路串联在构成节律流水式处理器的顺序链中。输入数据进入流水线的第一电路。当第一电路(对应于第一“循环”)产生压缩输出数据时，顺序链中的第二电路将其接收。该过程沿整个链路继续下去，直至序列中的最后一个电路提供构成压缩数据的输出位。

此作用是使大量的处理时间重叠。也就是说，当新数据正在进入第一电路时，先前已进入的数据同时被链路下游的其它电路处理。系统所表现的延迟称为“等待”时间，它是数据开始进入流水线顶部(即，输入电路)与流水线底部(即，序列中的最后一个电路)开始产生输出之间 的时间。以目前半导体设备的速度，即便是很长的流水线，总的等待时间也非常短。本发明甚至可以跟得上实时要求最高的、具有很高位速率的应用。

流水线的长度，即链路中顺序电路的个数，决定了总流水线将产生的压缩量。因此，可以生成包含不同程度压缩的不同芯片，为特殊应用提供硬件支持。

现在参照图 1，逻辑流程图示出了构成本发明第一实施例的方法。在第一步骤 10 中，接收要压缩的数字数据，并将其存储在输入缓冲器中。在步骤 12，将数据分隔成许多序列。

接下来，在步骤 14，依照一预定过程处理被存储的数据，以获得长度可

变的压缩数据值．在本较佳实施例中，如上所述，使用二进制搜索过程．在步骤 16，将压缩数据值存储在临时存储区中，并在步骤 18，产生一删除代码，它表示在压缩步骤中从序列中删除的位数．然后，在步骤 20，将删除代码按一预定关系与压缩字值一起存储在临时存储区中．

在步骤 22，判断是否需要分隔代码．如果需要，在步骤 24，根据一组预定的规则产生一分隔代码，并在步骤 26，将分隔代码添加到压缩序列值中．在本较佳实施例中根据上述规则产生分隔代码．

在步骤 28，判断是否已经处理了输入缓冲器中的所有数据．如果结论肯定，那么存储在临时存储区中数据构成一组中间数据，方法行至步骤 30．如果还没有处理完所有输入数据字，那么方法返回步骤 14，并对存储在输入缓冲器中的每个输入数据字节执行步骤 12-22．在步骤 30，根据预定标准，判断是否已达到了所需的数据压缩程度．在本较佳实施例中，预定标准指定了时限．还可以使用其它标准，例如迭代的预定数目，预定的期望压缩率，或其它等价的标准．

如果还没有获得所需的数据压缩，方法行至步骤 32，将中间数据组存储在输入缓冲器中．方法返回步骤 12，并将输入缓冲器的数据分离成序列．然后，方法执行步骤 14 至 28，直到处理完输入缓冲器中的所有数据，然后行至步骤 30，在那里再次判断是否已达到所需的数据压缩程度．如果达到了所需的压缩程度，那么在步骤 34 对存储在临时存储区内的中间数据组添加迭代计数，该计数表示已执行的压缩循环次数．在步骤 36，提供该数据，作为编码器的输出，并为步骤 10 中接收到的数字数据指定压缩形式．

现在参照图 2，逻辑流程图示出了构成本发明第一实施例另一方面的用于对压缩后的原始数字数据进行解压缩的方法．如图 2 所示，方法从步骤 50 开始，接收由图 1 所示方法产生的经压缩的原始数字数据．接收数据包括各种长度的压缩数据代码，其中包括删除代码，并由分隔值分隔，还包括迭代计数．在步骤 52，利用分隔值从接收数据中获得下一个数据代码．下面会更详细地描述分隔代码的识别方法．然后，在步骤 54，处理下一个压缩序列值，以获得解压缩的值．在本较佳实施例中，该过程基于二进制搜索技术，它用删除代码和 1 位来获得解压缩的序列．在步骤 56，临时存储解压缩序列．在步骤 58，判断是否已处理了所有输入数据．如果没有，则返回并重复步骤 52 至 56．如

果已处理了所有数据，那么方法行至步骤 60，使计数器增值。

接下来，在步骤 62，由临时存储的解压缩值获得一数据代码，用下述方式从临时存储的数据中识别出分隔代码。

然后，在步骤 64，处理数据代码，得到解压缩序列。在步骤 66，临时存储解压缩序列，并在步骤 68 判断计数器的值是否等于迭代计数。如果不相等，那么返回并重复步骤 60、62、64 和 66。如果计数器的值等于迭代计数，那么解压缩结束。然后，提供临时存储的压缩值，作为输出。此输出数据是原始数字数据。

现在参照图 3，逻辑流程图示出了在对压缩输入数据解压缩时用于识别分隔代码的方法。

方法从步骤 70 开始。如步骤 71 所示，识别位流中的下一个删除代码。这包括识别位流中的最高位 0 位。所有高位位一直到且包括该 0 位构成了删除代码。接下来，在步骤 72，判断删除代码后的下一位是 0 还是 1。如果是 1，则此位和所有后续的 1 位构成与前一个删除代码相关的挂位。于是如步骤 74 所示，明确地将后面一个 0 位识别为分隔代码。然后，返回步骤 70。

另一方面，如果在步骤 72 判断出下一位是 0 位，那么该 0 位可以是分隔代码，也可以是删除代码。在步骤 76，通过判断下一位(在步骤 70 识别出的删除代码后的第二位)是 1 还是 0 来弄清楚。如果该位是 1，那么在步骤 78 明确地将前一位(在步骤 70 识别出的删除代码后面的第一位)识别为分隔代码，并且程序循环返回步骤 70。

另一方面，如果步骤 76 判断出在步骤 71 识别出的删除代码后面的第二位是 0，那么步骤 80 将步骤 72 得出的 0 位识别了删除代码。用这种方法，可以明确地识别在压缩位流中的分隔位，从而有助于图 1 的方法。

图 4 是一逻辑流程图，更详细地示出了构成本发明第二实施例一个方面的数据压缩方法，它可以用为应用程序在可编程的数字计算机上执行。工作中，程序要填充多个循环缓冲器，每个缓冲器对应于一个附加的压缩程度。用户确定所需的压缩程度，从而确定程序执行的循环次数。在工作期间，第一时间段包括读入输入数据并填充程序的所有缓冲器。从概念上可以将此看作“填充流水线”。在此期间，程序接受输入数据，但不产生输出。

所有缓冲器充满后，程序继续接收输入数据，并且还产生压缩数据输出。

此第二时段一直继续到用完所有输入数据。

当所有输入数据用完并且没有附加输入数据要压缩时，程序继续产生压缩数据输出，因为它要排空在第一时段充满的缓冲器。从概念上可以将此第三时段看作“排空流水线”。现在对该实施例作更详细的描述。

在步骤 100，程序开始执行。在步骤 102，进行初始化，在此期间，程序与用户交流，确定对当前运行的要求。包括诸如要压缩文件的名称，将存储压缩数据的输出文件的名称、以及要执行的最大循环次数(即，希望达到的压缩程度)等信息。然后。程序继续，打开用户为输入和输出指定的文件，并将所需的输入数据存储在零循环输入缓冲器中。循环次数初始设为零。

在步骤 104，当存储在零循环输入缓冲器中时，程序查看是否存在要压缩的附加输入字节。如果存在，那么程序行至步骤 106，取得该输入，并在步骤 108 将其构成序列。在步骤 110，如果没有足够的输入形成序列，那么程序返回步骤 104。

在步骤 112，通过删除低位 0 位和第一低位 1 位并增加删除代码来压缩序列。在步骤 114，如果必要，增加分隔代码。

在步骤 116，将经压缩的序列代码、其删除代码和分隔代码存储在输入缓冲器中，供下一循环使用。“循环次数”计数器用于跟踪当前正在执行的压缩循环的次数。在步骤 118，对该计数器增 1。

在步骤 120，程序判断当前循环输入缓冲器是否具有剩余的输入。如果有，那么在步骤 122，程序试图形成一个完整的序列。如果序列完整(步骤 124)，那么在步骤 126 用删除代码形成一个新的压缩序列代码，并在步骤 128 加入一个分隔代码。在步骤 130，将新压缩的序列代码存储在下一个循环输入缓冲器中。

在步骤 132，判断是否已经达到了预定的最大循环次数。如果已经达到，那么已经将提供给程序的第一数据压缩到此特定程序的最大程度，对该序列的压缩结束。将循环次数计数器复位到零，且程序返回步骤 104，从零循环输入缓冲器中获取新的输入数据。程序继续循环上述步骤，直至输入文件中没有数据。然后，程序行至步骤 140。

这时，程序用相应的方法“排空流水线”，即处理各循环输入缓冲器中的所有数据，直至没有位可用来形成序列。然后，程序执行步骤 144，用未压缩

形式将剩余的位存储在最大循环缓冲器的最后．然后，将最大循环缓冲器中的内容写入指定的输出文件中．程序还将最大循环次数写在输出文件的前面，并在步骤 146 结束。

现在参照图 5，逻辑流程图示出了包含本发明一较佳实施例另一方面的、对压缩数字数据解压缩的方法．用个人计算机上运行的一个程序来实施本方法．在步骤 200，程序开始执行．在初始化步骤期间，程序与用户交流，确定对当前运行的要求．包括诸如被压缩数据文件的名称和被解压缩输出文件的名称(即，被恢复的源文件)等信息．另一种方式是，在压缩文件内包含被压缩文件的名称．然后，程序继续下去，打开用户为压缩输入和和解码输出而指定的文件．在打开压缩输入文件后，程序从该文件的前面抽取出要在编码运行期间达到的最大循环次数．然后，在步骤 202，使用该数值对循环计数器初始化．这时，我们处于迭代的底部(即，最低)循环．在步骤 204，程序查看是否存在要解码的压缩输入附加字节．如果存在，那么程序行至步骤 206，获取该压缩输入，并处理它．如果不存在，那么程序行至步骤 236，对早已输入的剩余数据进行解压缩(在下文中描述)，并结束当前运行．

在步骤 206，程序检索下一个删除代码．

在步骤 208，如图 2A 所述，识别下一个删除代码和分隔代码．然后，在步骤 210，用删除代码和任何挂位获得解压缩序列．

在步骤 214，将解码序列写入下一个较低循环的输入缓冲器中．

在步骤 216，程序对循环次数减 1，并且开始新的循环处理．在步骤 218，处理新循环的第一步骤是判断当前循环输入缓冲器中是否存在一个完整的序列等待处理．如果有，那么程序检索该信息，并行至步骤 220 进行处理．如果没有足够的位，那么程序行至步骤 232．程序通过位检索一直到下一个删除代码的开始，来确定它具有完整的序列．这可以包括被抛弃的分隔代码．

在步骤 220，与步骤 208 一样，程序识别当前循环中的删除代码和分隔代码．在步骤 222，程序用同样的方式判断是否有完整的压缩序列等在当前循环输入缓冲器中．如果有，那么程序行至步骤 224．否则，程序行至步骤 223．

在步骤 224，程序用与步骤 210 相同的方法获得下一个压缩序列输入位，并对其解码．然后，用与步骤 214 相同的方法，将经解码的序列写到下一个较低的循环输入缓冲器中。

在步骤 230，程序判断当前循环是否为 0 循环．如果是，则结束了对数据的解压缩，并且位于该循环输出缓冲器中的数据与原始源数据相同．现在，将该数据写到指定的输出文件中，并且程序行至步骤 232．如果程序不处于 0 循环，则程序行至步骤 216，开始下一循环处理．

在步骤 232，系统将循环次数复位到最大循环，其中最大循环在位于当前压缩输入文件开始的标题信息中有规定．这使得程序返回流水线的底部，如前所述继续处理．并且程序在步骤 234 返回．步骤 204 判断被解码的当前文件是否有附加的压缩输入．

如果输入文件中没有更多的数据，那么程序执行“排空流水线”，并结束对现在包含在各种输入缓冲器中的数据的解压缩．当所有输入缓冲器都是空的时，程序行至步骤 238．在步骤 238 执行各种“整理”任务，准备结束，并且在步骤 240，解码程序结束．包含在指定输出文件中的数字数据是对输入解码器中的原始数据的精确复制．

图 4 和图 5 描述的程序可以在具有 80486 处理器的 IBM 兼容个人计算机上执行，程序在 MS-DOS 操作系统下运行．当然，更高级的处理器可以获得更快的压缩和解压缩时间．

现在参照图 6，一较佳实施例示出了本发明的另一方面，该实施例在 VLSI 定制集成电路(“IC”)300 上执行．电路 300 用第一、第二和第三压缩电路实现联系图 3 程序描述的编码功能．特别是，第一压缩电路 302 实现在步骤 106、108、110、112、114 和 116 中描述的 0 循环的功能．但是，IC300 的电路 302 不象步骤 116 所述的那样将输出写入“下一个循环输入缓冲器”中，而是将输出提供给第二电路 304 的输入端．第二压缩电路 304 实现步骤 122、124、126、128 和 130 中所示的功能．但是，电路 304 不象步骤 130 所述的那样将压缩输出写入下一个循环输入缓冲器中，而是将压缩输出作为输入提供给 IC300 所含顺序中的下一个电路．

IC 300 的不同实施可以包括不同数量的电路 302-306．对于总压缩率的每个 10 次幂，大致需要 75 个电路．例如，IC 300 中的 75 个电路将产生大约 10：1 的压缩率，150 个电路将产生大约 100：1 的压缩率，225 个电路将产生大约 1000：1 的压缩率，等等．

第三压缩电路 306 实现步骤 122、124、126、128 和 130 的功能．但是，

电路 306 不象步骤 130 所述的那样把数据写入下一循环的输入缓冲器中，而是提供一个由位流组成的输出信号，表示提供给电路 302 的输入的压缩形式．另外，当电路 306 判定没有足够的附加数据位可以形成序列时，该电路提供剩余的位，将其作为未压缩的“下脚料(footer)”，终止输入数据流。

图 7 是一方框图，示出了形成电路 302 、 304 和 306 的公共压缩电路 350 。在较佳实施例中，这些电路 302 、 304 、 306 是相同的，有利于在 VLSI 芯片上实现大量的这种电路。

电路 350 包括输入电路 352，用于接收来自输入端 308(对于电路 302)或前一个紧接的上游压缩电路(对于电路 304 和 306)的数据．输入电路 352 与二进制搜索处理器 354 耦合，而处理器根据图 3 所示的方法进行数字数据压缩．用存储电路 356 来实现处理 354 的临时存储．处理器 354 产生分隔值，并且提供给输出电路 358，而输出电路 358 又将这些信号提供给输出端 310(对于电路 306)或下一个紧接的下游压缩电路的输入电路(对于电路 302 和 304)．

在本发明的另一方面，包含多种相同解码电路的 VLSI 集成电路用类似于上述图 6 所示的关于编码器硬件形式的方法，实现图 5 程序的功能．

本领域的技术人员应该理解，不脱离本发明的范围或精神可以对本发明的方法和较佳实施例的结构进行各种变化和改变．通过参考说明书以及实施这里所揭示的发明，本发明的其它实施例对本领域的技术人员来说是显而易见的．说明和例子只能视为举例，本发明的真实范围和精神由以下权利要求书确定．

说 明 书 附 图

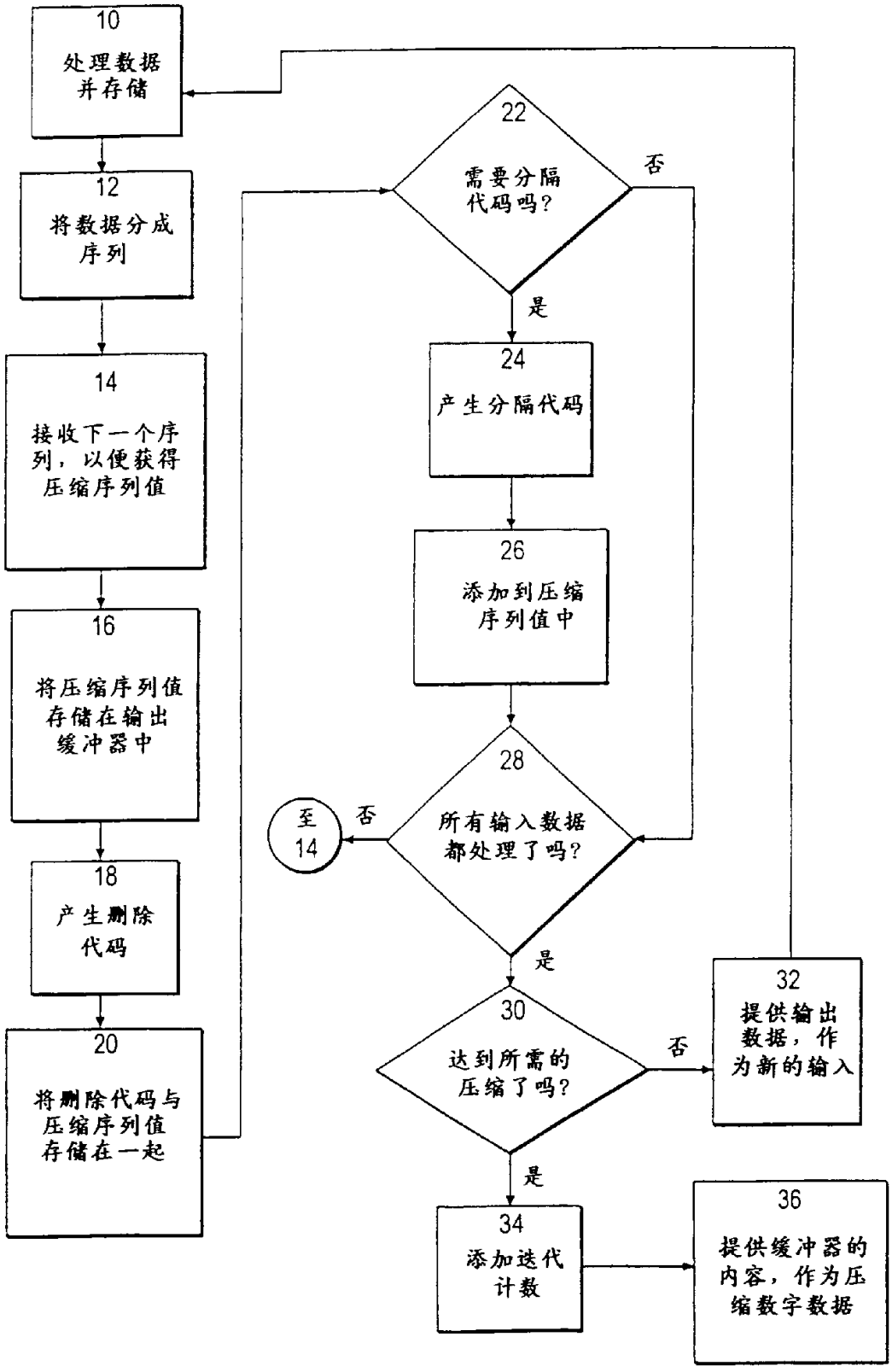

10
处理数据并存储
12
将数据分成序列
14
接收下一个序列，以便获得压缩序列值
16
将压缩序列值存储在输出缓冲器中
18
产生删除代码
20
将删除代码与压缩序列值存储在一起
22
需要分隔代码吗？
否
是
24
产生分隔代码
26
添加到压缩序列值中
28
所有输入数据都处理了吗？
至14
否
是
30
达到所需的压缩了吗？
否
32
提供输出数据，作为新的输入
是
34
添加迭代计数
36
提供缓冲器的内容，作为压缩数字数据

图 1

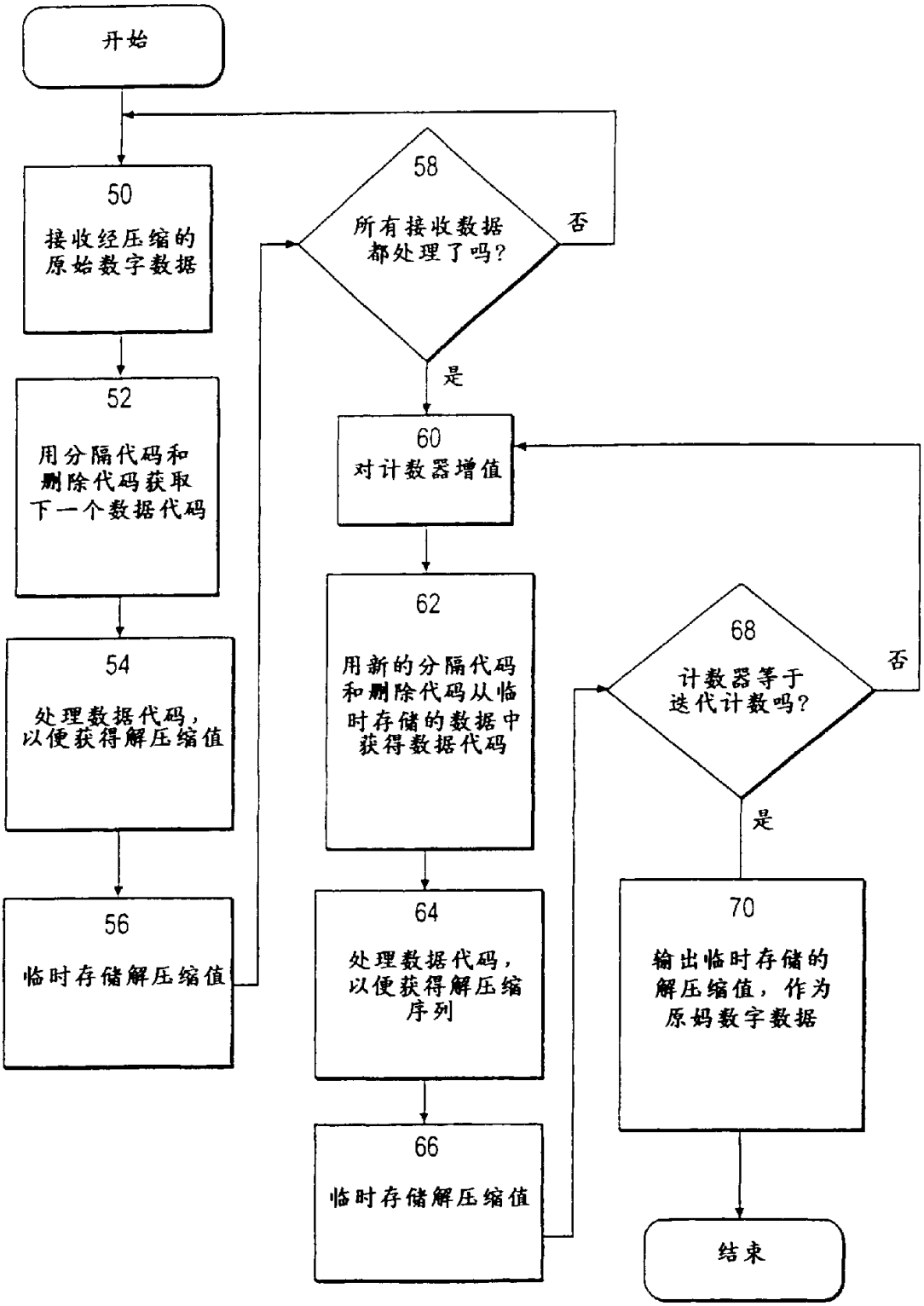

开始
50
接收经压缩的原始数字数据
52
用分隔代码和删除代码获取下一个数据代码
54
处理数据代码，以便获得解压缩值
56
临时存储解压缩值
58
所有接收数据都处理了吗?
否
是
60
对计数器增值
62
用新的分隔代码和删除代码从临时存储的数据中获得数据代码
64
处理数据代码，以便获得解压缩序列
66
临时存储解压缩值
68
计数器等于迭代计数吗?
否
是
70
输出临时存储的解压缩值，作为原始数字数据
结束

图 2

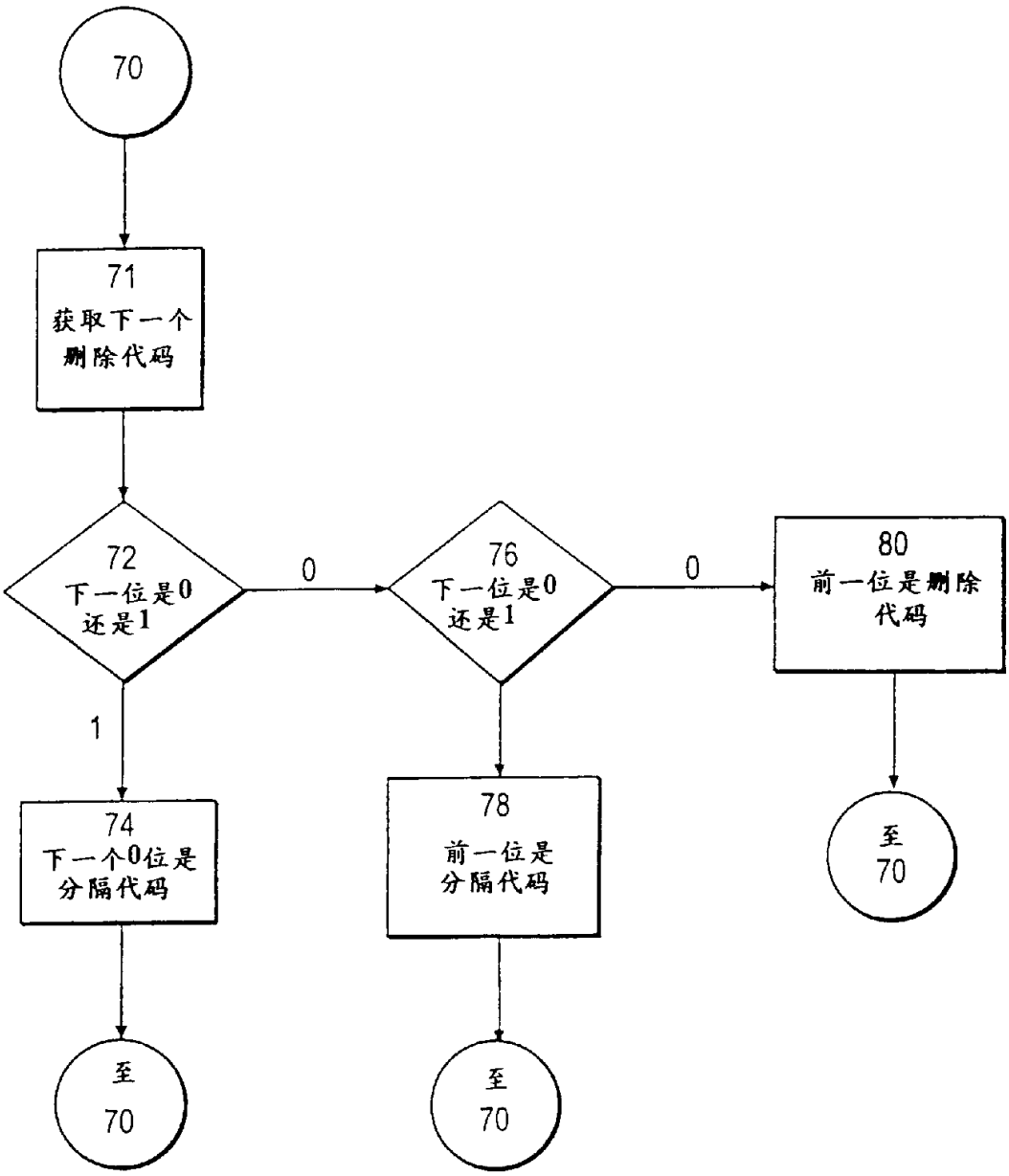

70
71
获取下一个
删除代码
72
下一位是0
还是1
0
76
下一位是0
还是1
0
80
前一位是删除
代码
1
74
下一个0位是
分隔代码
78
前一位是
分隔代码
至
70
至
70
至
70

图 3

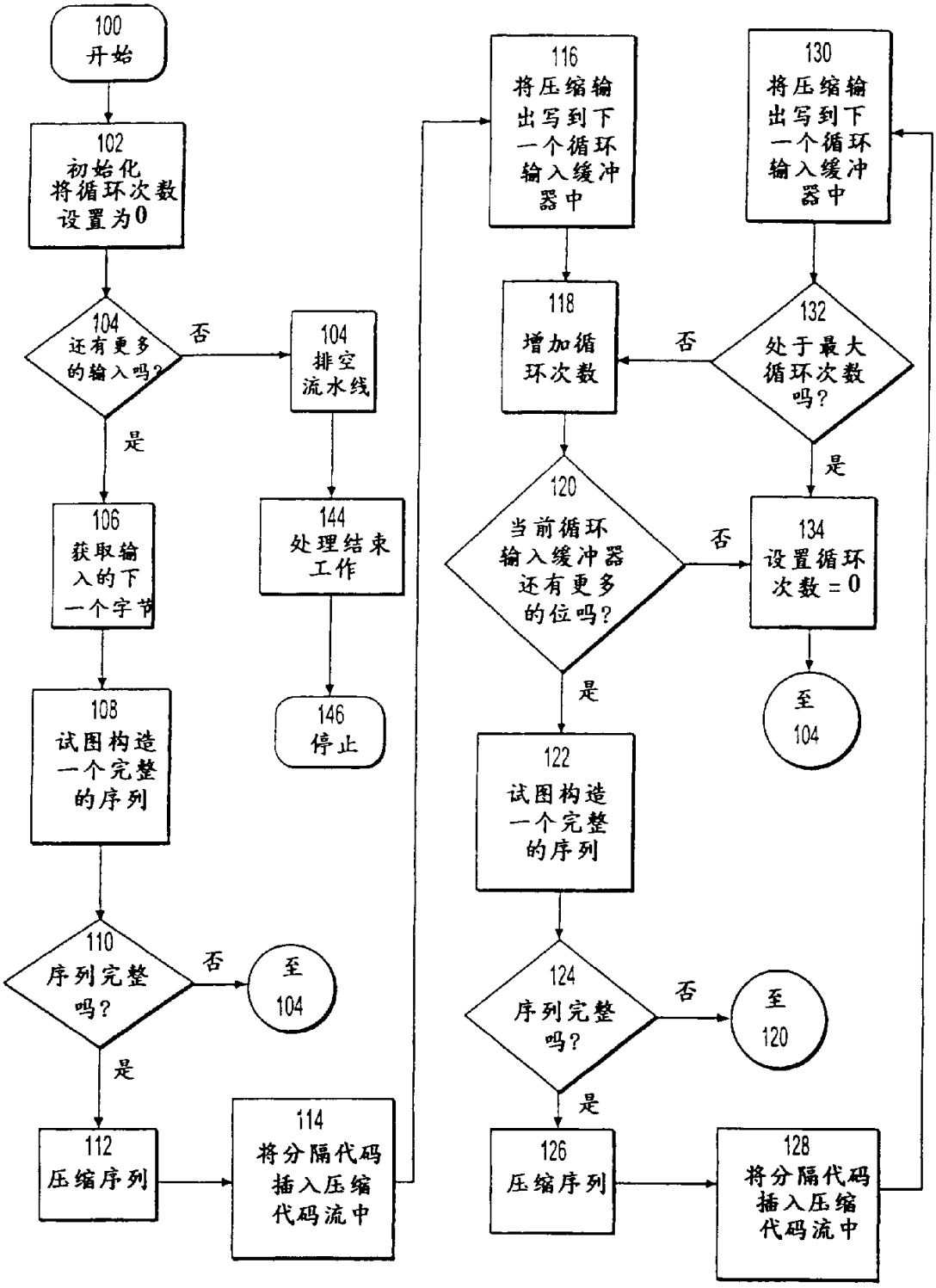

100
开始
102
初始化
将循环次数
设置为0
104
还有更多
的输入吗?
否
104
排空
流水线
是
106
获取输
入的下
一个字节
144
处理结束
工作
108
试图构造
一个完整
的序列
146
停止
110
序列完整
吗?
否
至
104
是
112
压缩序列
114
将分隔代码
插入压缩
代码流中
116
将压缩输
出写到下
一个循环
输入缓冲
器中
118
增加循
环次数
否
120
当前循环
输入缓冲器
还有更多
的位吗?
否
是
122
试图构造
一个完整
的序列
124
序列完整
吗?
否
至
120
是
126
压缩序列
128
将分隔代码
插入压缩
代码流中
130
将压缩输
出写到下
一个循环
输入缓冲
器中
132
处于最大
循环次数
吗?
是
134
设置循环
次数=0
至
104

图 4

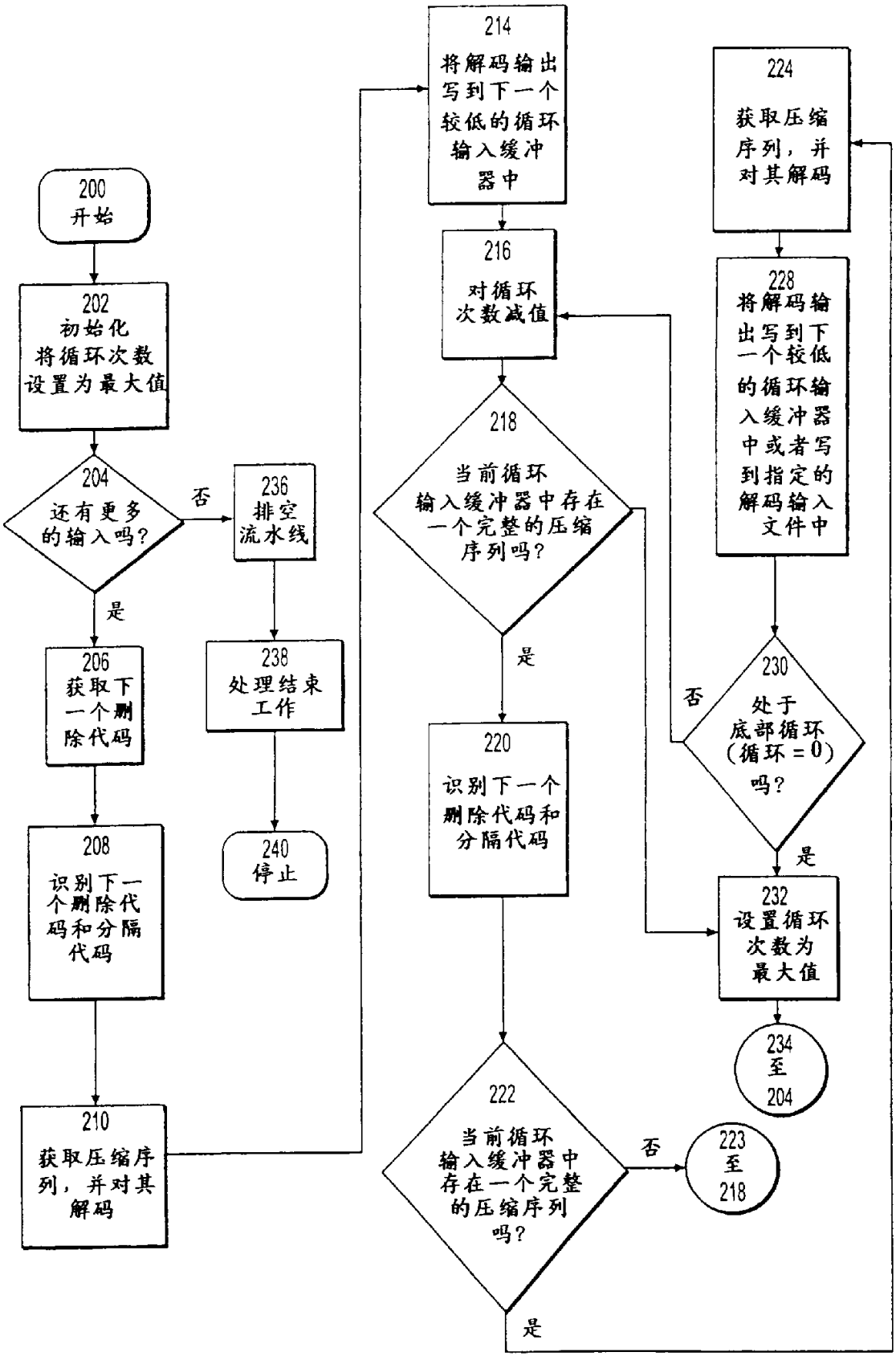

200
开始
202
初始化
将循环次数
设置为最大值
204
还有更多
的输入吗?
否
236
排空
流水线
是
206
获取下
一个删
除代码
238
处理结束
工作
208
识别下一
个删除代
码和分隔
代码
240
停止
210
获取压缩序
列，并对其
解码
214
将解码输出
写到下一个
较低的循环
输入缓冲
器中
216
对循环
次数减值
218
当前循环
输入缓冲器中存在
一个完整的压缩
序列吗?
是
220
识别下一个
删除代码和
分隔代码
222
当前循环
输入缓冲器中
存在一个完整
的压缩序列
吗?
否
223
至
218
是
224
获取压缩
序列，并
对其解码
228
将解码输
出写到下
一个较低
的循环输
入缓冲器
中或者写
到指定的
解码输入
文件中
230
处于
底部循环
(循环 = 0)
吗?
否
是
232
设置循环
次数为
最大值
234
至
204

图 5

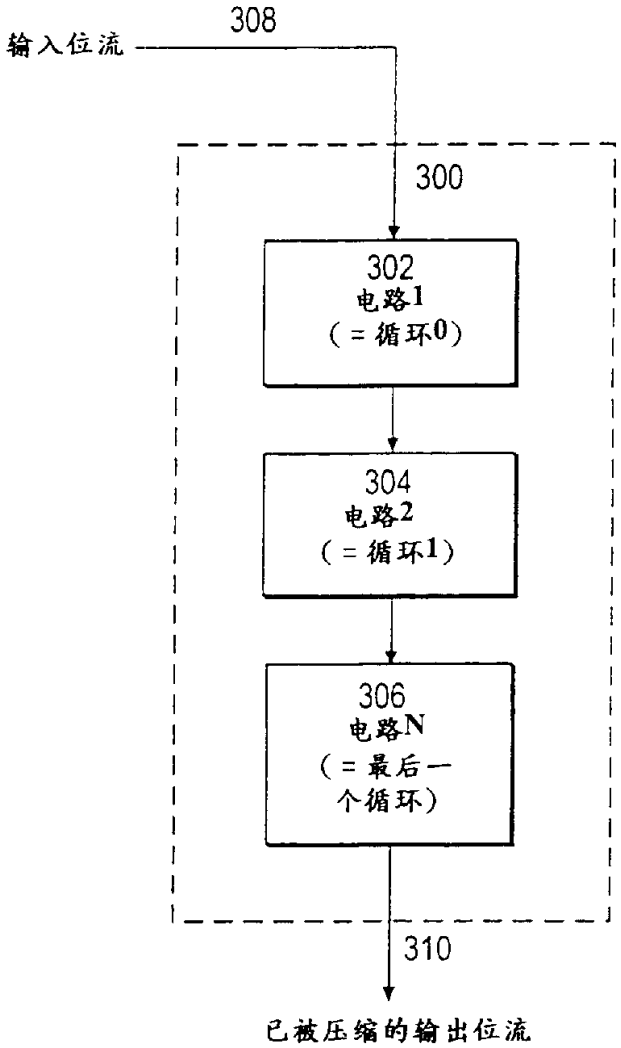

输入位流
308
300
302
电路1
(=循环0)
304
电路2
(=循环1)
306
电路N
(=最后一
个循环)
310
已被压缩的输出位流

图 6

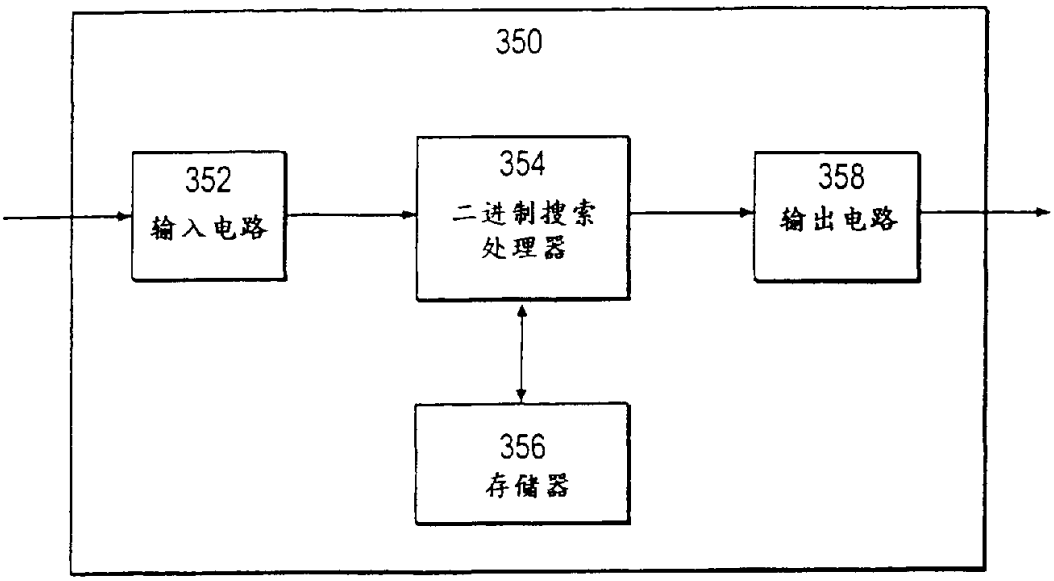


图 7