



(12) 发明专利

(10) 授权公告号 CN 101091157 B

(45) 授权公告日 2010.09.01

(21) 申请号 200480043969.1

CN 1396519 A, 2003.02.12,

(22) 申请日 2004.07.12

WO 0219093 A1, 2002.03.07,

(85) PCT申请进入国家阶段日
2007.03.12

Tony R. Kuphaldt. principles of digital
computing: Look-Up Tables. [http://www.
allaboutcircuits.com/vol_4/chpt_16/2.
html](http://www.allaboutcircuits.com/vol_4/chpt_16/2.html). 2003, 4

(86) PCT申请的申请数据
PCT/NL2004/000505 2004.07.12

审查员 宋朝

(87) PCT申请的公布数据
W02006/006842 EN 2006.01.19

(73) 专利权人 哈利尔·基利克
地址 荷兰海牙

(72) 发明人 哈利尔·基利克

(74) 专利代理机构 中国专利代理(香港)有限公
司 72001

代理人 李亚非 陈景峻

(51) Int. Cl.
G06F 7/60 (2006.01)

(56) 对比文件
CN 1468396 A, 2004.01.14,

权利要求书 2 页 说明书 10 页 附图 9 页

(54) 发明名称

用于处理数字数据的数字处理器和方法

(57) 摘要

本发明描述了一种新型计算机体系结构。数字处理器包括处理单元,所述处理单元包括用于把 N- 位数据字的第一部分解码成仅具有一位高度的数字代码。这些位在算术模块中被处理,所述算术模块接收来自两个处理单元的输出、以及由指令电路所生成的触发信号。该指令电路通过读取 N- 位数据字的第二部分而接收关于要处理的数据的经验。该第二部分可以被分成几个所谓的 DNA 组。所述 DNA 组包含有关要处理的数据的符号/值的类型的消息。利用关于数据的该经验,可以执行非常快速且简单的并行处理。

1. 一种用于处理数据字的数字处理器,数据字被构成为信元,所述信元具有:作为数据字的第一部分的表示符号的信元主体,所述符号表示操作或数据;具有三个单环的DNA-A组,单环被定义为一组位,第一单环指示DNA-B组的长度,第二单环指示DNA-C组的长度,以及第三单环指示由所述信元主体所表示的符号的开始和结束;包括一组位并且包含关于由所述信元所表示的符号的种类的信息的所述DNA-B组;以及包括一组位并且指示相应符号在信元链中的位置的所述DNA-C组,其中信元链表示由多个信元表示的、大于一个十进制的十进制数,所述数字处理器包括:

- 第一处理单元(1),用于接收第一N-位信元(6),以及包括:
 - 第一解码器(3),用于接收所述第一N-位信元(6)的第一部分并且把所述第一N-位信元(6)的所述第一部分解码成第一D-位信元主体(5),其中N、D是正整数,以及
 - 第一读出电路(9),用于读出所述第一N-位信元(6)的第二部分的所述DNA-A组、所述DNA-B组以及所述DNA-C组,以便产生第一经验信号(13);
 - 第二处理单元(2),用于接收第二N-位信元(8),以及包括:
 - 第二解码器(4),用于接收所述第二N-位信元(8)的第一部分,以及把所述第二N-位信元(8)的所述第一部分解码成第二D-位信元主体(7),以及
 - 第二读出电路(11),用于读出所述第二N-位信元(8)的第二部分的所述DNA-A组、所述DNA-B组以及所述DNA-C组,以便产生第二经验信号;
 - 多个算术模块(25,27,29,31),每个算术模块(25,27,29,31)用于接收所述第一和所述第二D-位信元主体(5,7),并且用于对所述第一和所述第二D-位信元主体(5,7)执行预定算术运算;以及
 - 至少一个指令电路(17),用于接收输入指令信号(19),以及用于利用所述第一和所述第二经验信号(13,15)把所述输入指令信号(19)转换成触发信号(33),以及用于触发所述多个算术模块(25,27,29,31)之中的一个算术模块(25,27,29,31),以便产生输出数据字(35)。

2. 根据权利要求1的数字处理器,其中所述解码器(3,4)用于把二进制代码转换成其中每次仅一位有效的输出代码。

3. 根据权利要求1的数字处理器,其中每个所述解码器都包括十个输出门,每个输出门表示一个十进制数值。

4. 根据权利要求1的数字处理器,其中所述处理器包括多个处理单元、多个算术模块和至少一个指令电路,

其中所述多个处理单元的每一个都用于把N-位数据字的第一部分解码成D-位数据字,以及用于把所述D-位数据字输出到所述多个算术模块的至少一个,所述多个算术模块根据由包含在所述多个处理单元中的读出电路所产生的经验信号,而由所述至少一个指令电路所产生的触发信号来触发。

5. 根据权利要求1的数字处理器,其中所述第一和第二N-位数据字的所述第二部分包括位的三个操作组。

6. 根据权利要求1的数字处理器,其中N至少是26位。

7. 一种计算机,包括根据权利要求1的处理器。

8. 一种用于处理数字数据的方法,包括下述步骤:

- 读取第一 N- 位数据字；
- 把所述第一 N- 位数据字的第一部分解码成第一 D- 位数据字，其中 N, D 是正整数；
- 读出所述第一 N- 位数据字的第二部分的特定位置，以便产生第一经验信号；
- 读取第二 N- 位数据字；
- 把所述第二 N- 位数据字的第一部分解码成第二 D- 位数据字；
- 读出所述第二 N- 位数据字的第二部分的特定位置，以便产生第二经验信号；
- 利用所述第一和所述第二经验信号而把输入指令信号转换成触发信号；
- 根据所述触发信号，对所述第一和所述第二 D- 位数据字执行特定算术运算以便产生结果，其中所述一个特定算术运算是多个预定算术运算中的一个。

用于处理数字数据的数字处理器和方法

技术领域

[0001] 本发明涉及数字处理。本发明尤其涉及一种用于处理以基于特定遗传学的方式来编码的数据字的数字处理器。

背景技术

[0002] 现在在数字计算领域存在三种主要的发展。

[0003] 1. VLSI 技术的改进,例如降低 t 周期并且提高每个芯片上晶体管的有效数目;

[0004] 2. 体系结构的发展,象获取、影响 t 周期、指令数目 (Ninstr) 以及 CPI 等等的流水线指令、并行指令;

[0005] 3. 编译器和或汇编程序的发展,尤其是指令级并行性的开发,这同样影响了 Ninstr 和 CPI。

[0006] 现在我们到达了 VLSI 技术的极限能力,10nm 以下电子刚好经由芯片内线路之间的绝缘而穿过(量子动力学效应)。增大的功耗强制使得较低电源电压 (3.3V) 和具有较低电压的 VLSI 处理的使用成为可能。然而,设计复杂性以及二进制不兼容性正在按指数规律增加。这就是为什么发展的可扩展性到达了其极限的原因之一。除那些因素之外,不同级别的兼容性以及现有软件和硬件的可扩展性的问题使得新产品类型的实现按指数规律地复杂和困难。此外,需要执行简单的小数运算的逻辑运算的数量相对较高。

[0007] 从十进制数据向二进制数据的翻译次数及相反进行的翻译次数需要一定量的有效计算能力输出 (ECO)。与有效计算能力输入 (ECI) 的数量相比,ECO 是非常低的,也就是说,关于计算能力的回报 (Return On Computing power, ROC) (其中 $ROC = ECO/ECI$) 是非常小的。上个例子中计算的 ROC 小于 $1/256$ 。当把第 4 或第 5 级计算机语言编译成机器语言(例如十六进制或 BCD 或 ASCII 码)时,二进制位的数目甚至会增大,这将进一步导致 ROC 的值甚至变得更低(即更糟)。

发明内容

[0008] 本发明的一个目的是引入提供较高计算速度的另一种类型的计算机体系结构。该目的是通过提供数字处理器而实现的,所述数字处理器包括:

[0009] - 第一处理单元,用于接收第一 N- 位数据字,以及包括:

[0010] - 第一解码器,用于接收所述第一 N- 位数据字的第一部分并且把所述第一 N- 位数据字的所述第一部分解码成第一 D- 位数据字,其中 N, D 是正整数,以及

[0011] - 第一读出电路,用于读出所述第一 N- 位数据字的第二部分的特定位置以便产生第一经验信号;

[0012] - 第二处理单元,用于接收第二 N- 位数据字,以及包括:

[0013] - 第二解码器,用于接收所述第二 N- 位数据字的第一部分以及把所述第二 N- 位数据字的所述第一部分解码成第二 D- 位数据字,以及

[0014] - 第二读出电路,用于读出所述第二 N- 位数据字的第二部分的特定位置以便产生第

二经验信号；

[0015] - 多个算术模块,每个算术模块用于接收所述第一和所述第二 D- 位数据字,并且用于对所述第一和所述第二 D- 位数据字执行预定算术运算；

[0016] - 至少一个指令电路,用于接收输入指令信号,以及用于利用所述第一和所述第二经验信号而把所述输入指令信号转换成触发信号,以及用于触发所述多个算术模块之中的一个算术模块,以便产生输出数据字。根据本发明的处理器具有允许快速且并行地处理数据的新型体系结构,其中每次计算具有最少数目的逻辑门操作。

[0017] 这个新型体系结构需要考虑到计算机体系结构的设计的另一方式。利用本发明,有可能制造出超级标度和超级流水线处理器 (VLIW),其利用新型计算机体系结构来运算。本发明是基于下述理解之上的,即染色 (dye) 的密度 (其可能是 MOS/CMOS) 不是前文中所提及的主要发展的最重要问题。计算能力的限制 (也就是每秒逻辑运算次数) 是要解决的关键问题。如在上述例子中所示的那样,当前处理器中的 ROC 值是很低的。在根据本发明的处理器中,ROC 值被显著地提高了,如根据所示例子而变得显而易见的那样。

[0018] 在本发明的另一个方面中,提供了一种计算机,其包括如上所述的数字处理器。

[0019] 在又一个方面中,提供了一种如权利要求 8 所述的用于处理数字数据的方法。

[0020] 附图的简要说明

[0021] 下面,将参考一部分附图而阐明本发明,其仅是用于说明的目的而不是要限制如在所附权利要求中所定义的保护范围,其中

[0022] 图 1 显示了根据本发明实施例的数字处理器的一部分,

[0023] 图 2 是由根据实施例的处理器所执行的计算的简单例子,

[0024] 图 3 是计算的另一个例子,

[0025] 图 4 显示了乘法 254×2 的例子,

[0026] 图 5 示意地描述了 I/O 总线和数字处理器之间的关系,

[0027] 图 6 显示了用于乘以十进制数的原理的例子,

[0028] 图 7 是列出在本发明的一个实施例中所使用的可能编码的表格,以及

[0029] 图 8 是包含在本发明的实施例中的解码器的硬件实现的例子。

[0030] 具体实施例详述

[0031] 在图 1 中,显示了根据本发明的实施例的数字处理器的一部分。数字处理器包括至少第一处理单元 1 和第二处理单元 2。每个处理单元 1、2 都包括解码器 3、4,其用于把包括 N 位的输入数据字的第一部分 (参见箭头 6、8) 解码成分别描述为 5、7 的 D- 位数据字。处理单元 1、2 还包括读出电路 9、11 (也称为 DNA 解码器),其用于接收输入数据字的第二部分。读出电路 9、11 把 N- 位数据字的第二部分转换成所谓的经验信号 13、15,所述经验信号 13、15 被导向指令电路 17。指令电路用于接收输入指令信号 19 并且根据所述经验信号 13、15 把该输入指令信号 19 转换成触发信号。数字处理器还包括多个算术模块 25、27、29、31,其用于对 D- 位数据字 5、7 执行算术运算,所述 D- 位数据字 5、7 是从第一和第二处理单元的解码器 3、4 中接收的。每个算术模块 25、27、29、31 可以实施不同的算术运算,诸如乘、加、减、除等等。指令电路 17 用于产生触发信号 33,所述触发信号被发送到不同的算术模块 25、27、29、31。只有当算术模块 25、27、29、31 接收到来自指令电路 17 的触发信号时,所述算术模块 25、27、29、31 才执行它们的特定算术运算。以利用与用于编码输入数据字的编码

相同的编码的格式来表示特定算术运算的结果 35。采用这种方式,可以将算术模块的输出输入到其他处理单元中,以进行进一步的处理。

[0032] 在本发明的实施例中,解码器 3、4 用于把二进制代码转换成 D- 位数据字,其是以其中每次仅一位有效的格式来编码的。解码器 3、4 例如包括 10 个输出门,每个输出门表示一个十进制数。解码器的每个输出门可传递给特定算术模块,所述算术模块然后只须执行关于两个位的计算。这就实现了算术模块 25、27、29、31 的非常简单的体系结构,随后其将对数字处理器的速度十分有利。

[0033] 在图 2 中显示了计算的一个简单例子。在这个例子中描述了一个非常简单的乘法 1×3 (十进制)。值 1 被存储在寄存器 40 中,寄存器 40 是第一处理单元 1 的一部分。应当注意到这个寄存器可能是 I/O 总线的一部分,向处理单元 1、2 提供数据。十进制数 3 被存储在寄存器 41 中。两个十进制数都是二进制编码的。第一 N- 位数据字 40 的第一部分被输入到解码器 3,解码器 3 把该数字值转换成第一 D- 位数据字,其中仅一个输出门为高,在这种情况下符号为“1”的输出门为高。其他输出门为低并且不是这个例子所关心的。第二 N- 位数据字 41 的第一部分被输入到解码器 4 中,解码器 4 把它转换成第二 D- 位数据字,其中仅输出门“3”为高。解码器 3、4 的输出被发送到乘法器 43,乘法器 43 包括“与”门和两个“或”门 46、47。图 2 还显示了指令电路 17 的例子,指令电路 17 接收指令信号 19,指令信号 19 的值为 $\langle 0 \rangle$ 。在这种情况下指令电路 17 包括一个 NAND51,NAND51 接收指令信号 19 和来自两个 N- 位数据字的经验数据部分中的特定位置的数据。两个位表示该取决于所述经验信号 13、15 信元主体是否包含数值。此时,它们都包含 $\langle 0 \rangle$,其在这种情况下意味着两个值都是数值。指令电路 17 把触发信号 53 发送给乘法器 43,以便触发乘法器。这个乘法的结果是产生了数字代码 00011,参见输出缓冲器 54。应当注意的是输出缓冲器 54 可能是另一个处理单元的输入缓冲器,未显示。

[0034] 在图 2 的例子中,乘法器 43 还包括反相器 57,反相器 57 连接到输出缓冲器 54 的输入门以填充与涉及输出值是数值的信息有关的位。在这种情况下 $\langle 0 \rangle$ 被存储在输出缓冲器 54 的第二部分 55 的适当位置中。应当注意到经验数据向输出缓冲器 54 中的这种填充也可以由指令电路 17 来执行。

[0035] 参考图 3 来阐明另一个例子。在这个例子中,以被称为 ABGED 编码的特定格式来编码所处理的 N- 位数据字。稍后在说明书中将参考图 5 和 7 更详细地阐明这个 ABGED 编码的细节。

[0036] 图 3 显示了 102×103 的计算 (即 100×1000 十进制)。值 102 以 ABGED 编码格式被存储在寄存器 60 中,这意味着该信元主体包含 $\langle 00010 \rangle$ 并且经验数据中的特定位置 (即数据字的第二部分) 是 $\langle 1 \rangle$,表示信元主体包含指数值这样的事实。值 103 被存储在寄存器 61 中,参见图 3。每个解码器 3、4 都把数字信元主体数据解码成 D- 位数据字,所述 D- 位数据字被输入到乘法器 63 和加法器 65 中。指令电路 17 经由读出电路接收来自输入 N- 位数据字的第二部分的经验信息,读出电路没有被显示。指令电路 17 接收乘法指令信号 19 并且把它转换成触发信号 75。在这种情况下,触发信号 75 被发送到加法器 65,而不是发送到多路复用器 63。这是存在于输入数据字的第二部分中的经验的结果。这个经验是由特定位置来表示的,其接着触发指令电路中的适当的逻辑门。这将产生触发信号,其间接地体现出可以通过把指数相加并保持底数不变而把具有相同底数的两个指数值相乘的算术经验。加法器

65 将填充输出寄存器 77, 其中利用值 <00001> 和数据字的第二部分来填充信元主体, 表示将以指数位 E1 为 <1> 的方式来填充输出数据字的经验 (例如结果具有底数 10 (十进制))。

[0037] 根据本发明, 可以由多个信元 (称为 '信元链') 来表示大于一个十进制的十进制数。例如, 十进制数 254 可以由三个信元的信元链来表示。三个十进制数中的每一个都被编码成二进制代码并且存储在数据字的第一部分中。最好是, N- 位数据字包含至少 26 位。信元的第二部分包含表示关于存在于信元主体之中的值的其他经验 (即, 数据字的第一部分) 的位。根据实施例, 每个信元是由一个有效处理单元来表示 (即处理) 的, 因此, 信元链是由许多处理单元表示的。这些处理单元可在单个的 IC (集成电路) 或多个 IC 中被实现。

[0038] 根据本发明的实施例, 数字处理器包括多个处理单元、以及多个算术模块和至少一个指令电路, 其中所述多个处理单元中的每一个用于把 N- 位数据字的第一部分解码成 D- 位数据字, 并且用于把所述 D- 位数据字输出到所述多个算术模块的至少一个之中, 取决于由包含在所述多个处理单元中的读出电路所产生的经验信号, 所述多个算术模块用于由所述至少一个指令电路所产生的触发信号来触发。

[0039] 参考图 4 利用一个简单的例子来阐明这个实施例。在图 4 中显示了乘法 254×2 的例子。这个值 254 是由三个信元来表示的, 而十进制数 2 是由一个信元来表示的。每一个信元是由一个处理单元来处理的。在图 4 中仅讨论了存储在信元主体中的信息。在图 4 中, 信元主体 80<00010> 被输入到解码器 85。具有值 <00101> 的信元主体 81 被输入到解码器 86, 而具有值 <00100> 的信元主体 82 被输入到解码器 87 中。另外, 具有值 <00010> 的信元主体 90 被输入到解码器 91。如从图 4 中可见的那样, 解码器 85、86、87、91 连接到算术模块。解码器 85 和解码器 91 连接到算术模块 93, 解码器 86 和解码器 91 连接到算术模块 95。解码器 87 和解码器 91 连接到算术模块 97。如图 2 中所阐明的那样, 算术模块会产生结果, 所述结果会被发送到下列处理单元 100、104、101、102 的输入端。图 4 仅显示了这些下列处理单元的一部分。算术模块 93 把它的数据输出到处理单元 100, 算术模块 95 把它的值输出到处理单元 101 以及算术模块 97 把它的值输出到处理单元 102。算术模块 95 正在处理乘法 5×2 , 这产生十进制数 10。由于这个十进制数不能由一个处理单元来表示 (注释: 一个处理单元只能表示一个符号, 即一个十进位数), 所以另一个算术模块 94 和处理单元 104 被触发。

[0040] 这个触发信号是由某种来自算术模块 95 的 '溢出' 信号来实现的, 可能经由独立的指令电路而来的, 其中所述独立的指令电路未显示。由于算术模块 94 和处理单元 104 的特定结构, 所以在处理单元 104 的输出端实现了输出值 <0001>。处理单元 104 经由加法器而连接到另一个处理单元 107。要注意在这种情况下加法器包括连接到处理单元 107 的适当输入门的两条线。这个加法器还接收来自处理单元 100 的输入。现在, 作为处理单元 104 的触发并且来自指令电路的触发信号具有关于处理单元 101 无法表示 (即包括) 信元链的最左数字的经验的结果, 加法器将被触发。加法器进行相加的结果 (即十进制数 5) 是由处理单元 107 表示的。这个乘法的最终结果是三个有效的处理单元 107、101 和 102, 包括表示十进制数 508 的 ABGED 编码。

[0041] 根据本发明的优选实施例, 第二部分包括三个组, 即, DNA-A 组, DNA-B 组以及 DNA-C 组。在表 1 中, 显示了一个示意图, 其中阐明了信元的结构, 尤其是不同 DNA 组和信元

主体之间的关系。

[0042] 表 1：

[0043]

信元主体 5 位<00000>	[DNA-A] 3 位的第一个集合表示 DNA-B 组的长度, 而 5 位的第二个组表示 DNA-A 组的长度, 3 位的第三个集合包含了数据或运算的开始和结束 <010>, <00010>, <001>
[DNA-C] 可变的#的位集合, 每个集合表示符号在链中的位置, 如果为正: 数字在逗点之前 如果为负: 数字在逗点之后 ←.....<00000> <00000> 计算 (accounting): ← 2 1	[DNA-B] 它是哪种信元 这个 DNA 组描述了信元主体 字符 <0000> <0000> ←2 1

[0044] 如根据表 1 而看见的那样, 在优选实施例中, 信元主体包括表示字母数字符号的五个位。DNA-A 组包含表示 DNA-C 组和 DNA-B 组的位串的长度的值。除了这个长度信息之外, DNA-A 组还包括用于表示要处理的特定数量的数据、或特定操作的开始或结束的一组位。DNA-A 组中的这个第三单环的可能编码是：

[0045] 第一位等于 0, 那么信元主体包含数据, 如果等于 1, 那么信元主体是操作码：

[0046] 001 数据的开始

[0047] 010 数据的结束

[0048] 101 操作的开始

[0049] 110 操作的结束。

[0050] DNA-B 组包含有关信元主体中包含哪种数据的消息。

[0051] 在优选实施例中, DNA-C 组具有可变长度。因为信元具有可变长度, 所以大于 26 位, 需要不止一个单元来存储信元。有几个寄存器单元 (我把这个称作 DNA- 映射寄存器), 其包含关于 DNA-A、DNA-B 和 DNA-C 的长度的信息以及每个信元的位置。数据的映射可以是串行和 / 或平行地执行的。

[0052] 图 1 所示处理电路 25、27、29、31 都执行某些算术运算。应当注意到处理电路的数目不局限于四个, 而可以是更多。

[0053] 图 5 显示了可被用于根据本发明的处理器中的 I/O 总线 600 的配置的例子。I/O 总线管理器 (参见 601) 的功能是通过读取 DNA-A 组和保存这个信息来分析输入数据的结构。例如,特定信元从第 1 位开始并且它在从右到左的第 26 结束。I/O 总线 600 的大小是相对灵活的。外部 I/O 总线 602 具有均一大小 (这取决于所使用的插脚数目)。I/O 总线 600 的灵活性是由 I/O 总线管理器 601 来实现的,I/O 总线管理器 601 读取并计算信元链内下一个信元的下一个开始位置。这个信息被存储在位置寄存器中,未显示。当所有位置都被存储在位置寄存器中时,由 I/O 总线 600 来创建信元和信元链的映射。

[0054] 图 6 显示了用于乘以十进制数的原理的例子,这是由数字处理器中的处理电路来实现的。通过经由逻辑门来把某些输入门连接到处理电路的输出门而在处理电路中实现该原理。这意味着具有乘法器和多个值的图 6 所示的表被硬连接在处理电路中。技术人员应当清楚如何实现加法器、除法器、比较器。

[0055] 利用根据本发明的数字处理器,有可能在机器级上处理字母。这意味着像 MOV 这样的字可以采用例如 ABGED 编码来进行编码并且由处理器进行处理,其中处理器将处理包含信元“M,0,V”的信元链。正在处理的字符由处理单元存储为高级计算机语言 (诸如汇编程序或 C) 的字符,并且它们可直接由数字处理器的硬件进行解释。诸如英语或土耳其语之类的自然语言可用于执行数字处理器。一个或多个编程字符的组合可由一个或多个处理单元来执行。处理字母可以像例如主语、动词、时间、地点以及时间等等这样的句子结构那样被组织。例如,利用第一个 3 位,处理命令“contr-p”的符号由 DNA-A 组进行存储,这是单环 1 (继之以单环 2:length_A 以及继之以单环 3:length_B)。

[0056] 图 7 显示在本发明的实施例中所使用的可能编码的表。该表显示了从数值到大写字母进行变化的符号的数字编码。该表的每行都包含具有固定长度的简单信元。信元的长度可以变化,如上所述。信元主体的位称为 C(a)、C(e)、C(d)、C(c)、C(b)。DNA-C 组包含关于由信元主体所表示的符号的位置。DNA-C 中的位被称为 DNA-C(a)、DNA-C(d)、DNA-C(c)、DNA-C(b)。此外,图 7 的表包含 DNA-B 组和 DNA-A 组,它们包括位 DNA-B(a)、DNA-B(d)、DNA-B(c)、DNA-B(b) 以及 DNA-A(a)、DNA-A(c)、DNA-A(b)。下面,将更详细地阐明该位的含义。

[0057] C(a) 关键字位,这是一个正负号位,用于提供关于信元链中的信元的位置的 DNA-C 处理器信息,(当信元主体是根据 DNA-B 的数字时)其可以在逗号或分数的前面或者后面。这个表示法允许 10^{30} 的位置 (这意味着我们可以建立例如像“123..... (其具有 10^{30} 个数字),1233.....08 (在逗号后有 10^{30} 个数字)”)。

[0058] C(e), C(d), C(c), C(b) 二进制代码,用于表示符号或指数值,

[0059] 例子:Core = <00011> 表示十进制数 3,

[0060] Core = <01101> 表示小数点,Core = <>

[0061] DNA-B 组通知算术模块信元在信元链内具有哪个位置。信元链例如是 543,表示十进制数。在 ABGED 编码中,这个十进制数可以由信元链中的三个信元来表示。信元链内的位置被编码于 DNA-B 组中。这个组 DNA-B(a) 的第一位是关键字位。如果这个关键字位等于 0,那么这个信元在信元链中的位置是在十进制逗号之前。请注意在英语表示法中,采用的是十进制小数点(decimal point)而不是逗号(comma)。因此,如果在本说明书中采用了术语十进制逗号,那么它还可以被读作是十进制小数点。如果信元主体的关键字位,即 C(a),

等于 1,那么信元在信元链中的位置是在十进制逗点之后。如果在十进制数中没有逗点,取决于要完成哪种处理,则 DNA-B 管理器创建一个逗点以便能够执行算术指令读取。利用例子来阐明这一点:

[0062] 25,5[输入 A] 加上 1[输入 B](两个数字都是十进制计数法)。

[0063] 输入 B 没有逗点;但是能够进行正确的计算,数字处理器在输入 B 引入两个新信元并且这些信元具有下列信元链编码:

[0064] DNA-A:helix1:<001>helix2:<001>helix3<001>;

[0065] DNA-B:<000>;

[0066] DNA-C:<00000>;

[0067] Cell-core:<01101>这意味着逗点,

[0068] DNA-A:helix1:<001>helix2:<001>helix3<001>;

[0069] DNA-B:<000>;>;DNA-C:<10001>;

[0070] Cell-core:<00000>这意味着零。

[0071] 所需要的零的数目取决于输入 A 具有的位数。这意味着如果逗点之后的位数是二,则 DNA-B 必须引入相同数目的零(即,两个零)。在创建附加信元(即触发附加处理单元)之后,现在有可能执行正确的指令,在这个例子中:

[0072] 输入 1 25,5

[0073] 输入 2 1,0 +

[0074] 和 = 26,5

[0075] 在 ABGED 编码中,N-位数据字的经验部分还包括称作 DNA-B 组的组,包括对关于该主体中所编码的符号的类型的经验进行编码。这个 DNA-B 组例如包括四个位,如图 7 的表中所示。第一位 DNA-B(a) 是关键字位。如果这个关键字位等于 0,那么信元主体包含数值。

[0076] DNA-B 组可包括几个单环。单环被定义为 DNA-组内的一组基因(即位),其彼此相关并且表示信元的 DNA 部分的特定解释。出于简单性考虑,在下面的例子中,DNA-B 组仅包括具有四位字节的一个单环。

[0077] DNA-B 组位的第一集合(即单环)意味着正数<0000> = + 数。

[0078] DNA-B 第一单环

[0079] 关键字位 a = 0 数值

[0080] <adcb> 遵循基因的顺序

[0081] 0000 = + 数

[0082] 0001 = 恒定值底数 10(系数 = 10) 指数 = 10^n [n = 正数]

[0083] 0011 = 恒定值底数 10(系数 = 10) 指数 = 10^{-n} [n = 负数]

[0084] 0100 = 可变底数 x^n (BCD) [x = 正数] { 在这个 DNA 中存储了 x- 值 }

[0085] 0101 = 可变底数 $-x^n$ (BCD) [x = 负数] { 在这个 DNA 中存储了 x- 值 }

[0086] 0110 = 可变底数 x^n (BCD) [x = 正数] { 在这个 DNA 中存储了 n- 值 }

[0087] 0111- 可变底数 x^{-n} (BCD) [n = 负数] { 在这个 DNA 中存储了 n- 值 }

[0088] 如果关键字位 DNA-B(a) 等于 1,那么信元主体包含字母、字符或逻辑函数。

[0089] 关键字位 a = 1 字母 / 字符 / 逻辑函数

[0090] <adcb> 遵循基因的顺序

[0091] 1000 = 大写字母

[0092] 1001 = 小写字母

[0093] 1010 = 阿拉伯字母

[0094] 1011 = 特殊阿拉伯字母

[0095] 1100 = 运算函数

[0096] 1101 = 字符

[0097] 第三 DNA 组, 称作 DNA-A 组, 包括三个单环, 见表 1。在第一个单环中, 存储了关于 DNA-B 组中的单环数目的信息。在第二个单环中, 存储了关于 DNA-C 组中的单环数目的信息。在第三个单环中, 信息被存储在操作或数据的开始和结束, 如上所述。利用这一点, DNA-A 组创建了处理具有灵活长度的编码数据字的可能性, 使得有可能表示非常大量的值和 / 或符号串。

[0098] 如果 DNA-A 的第二个单环是 <0010>, 参见表 1, 这指的是信元 - 主体在信元链中的位置是由两位数字表示的, 因此在这种情况下它能在 10 到 99 之间变化。

[0099] 在图 9 中, 显示了解码器的硬件实施的例子。解码器包括用于从输入门和 / 或其他逻辑门接收数据的几个逻辑门。

[0100] ABGED 编码包含“位级经验”, 即哪种值或符号被存储在信元主体中。通过分析不同 DNA 组中的位, 可以使用关于要处理的数据的经验。这使得对相对短数据字执行数学运算成为可能。

[0101] 通过引入基因 (单环) 链和信元的概念, 人们可以在处理器内 ‘创建课本 (textbook)’。结果例如是存储器映射可以从 0-9 十进制、指数从 0 到 9 以及对数范围 0 到 9 进行寻址。那些存储器映射还可以是负值。

[0102] 同十六进制系统 (例如 000000H 或 FFFFFFFH) 比较起来, 在 ABGED 编码中数值范围可以是

[0103] 000...00000-999...99999 (64 个数目的 9) AAA...AAAAA-ZZZ...ZZZZZ (也是 64 个字符)

[0104] 寻址可以采用连续顺序来完成, 或者可以对寄存器地址赋予特定名称, 所述名称以动词定义了由所述寄存器所表示的所述信元的特定含义。那些动词可以像自然语言语法那样来组织, 以便有可能以动词的顺序 (即, 句子) 进行寻址。

[0105] 如果我们想使所有寄存器地址的使用都有效, 那么那些范围例如是从 0 到 9 ; A 到 Z ; a 到 z ; 0 到 -9 ; 指数标度的 0 到 9 ; “+” 或 “-” 对数标度的 0 到 9。

[0106] 在与扩展十进制相结合的字母词汇单词中可以完成寻址。甚至计算机操作指令本身都可以由机器级上的特定句子 (即字链) 来表示, 而不必需要从较高的计算机语言翻译到机器代码, 如利用例如汇编程序时所必需的那样。

[0107] 通过处理 ABGED 编码, 在执行实际计算之前利用关于操作数 (例如十进制数或指数值) 的经验。在实施例中, 信元的主体中的数字数据被解码成十进制数。利用那些值有可能在十进制系统对它进行处理。除十进制系统之外, 使用不同的标度, 例如像指数或对数标度。在根据本发明的处理器中, 基本代数规则被实现为硬连接。下面, 显示了十进制计算的一部分例子, 所述十进制计算是利用与人们通常所完成的相同的代数规则由本发明进

行处理的。

[0108] $300+1200 = 1500$

[0109] $3*10^2+12*10^2 = (3+12)*(10^2) = 15*10^2$

[0110] $5, 24*10^5-3, 12*10^2 = (5240)(10^2)-(3, 12)(10^2) = (5240-3, 12)*10^2 = 5236, 88*10^2 = 52366$, 另一种可能性是 $524000-312 = 52388$

[0111] $300*1200 = >(3*10^2)*(12*10^2) = (3*12)*(10^{(2+2)}) = 36*10^4 = 360.000$

[0112] 在根据本发明的数字处理器中,同技术现状比较起来,一次计算的逻辑运算的数目变小了。利用一个例子来阐明这一点。在这个例子中,利用 BCD 码来执行十进制运算 $32*234$ 。

[0113] $32 \qquad \qquad \qquad 00110010$

[0114] $234 \qquad \qquad \qquad 001000110100$

[0115] $\qquad \qquad \qquad \text{-----}$

[0116] $\qquad \qquad \qquad 00000000$

[0117] $\qquad \qquad \qquad 00000000$

[0118] $\qquad \qquad \qquad 00110010$

[0119] $\qquad \qquad \qquad 00000000$

[0120] $\qquad \qquad \qquad 00110010$

[0121] $\qquad \qquad \qquad 00110010$

[0122] $\qquad \qquad \qquad 00000000$

[0123] $\qquad \qquad \qquad 00000000$

[0124] $\qquad \qquad \qquad 00000000$

[0125] $\qquad \qquad \qquad 00110010$

[0126] $\qquad \qquad \qquad 00000000$

[0127] $\qquad \qquad \qquad 00000000$

[0128] -----

[0129] $7488 = 0110010001000100 \text{ (BCD)}$

[0130] 这个计算的 ROC 是 0,0641 (计算的方式 :5 数值 / 以上所计数的触发器 $780 = 0,0641$)。如果该值随着 5BCD- 数值而增大,则这个计算会增大指数。

[0131] 利用根据本发明的数字处理器,随着利用扩展的十进制系统 (ABGED) 对位级人工智能的发展,将把计算量减少到

[0132] 234

[0133] $32*$

[0134] 468

[0135] 702

[0136] $7488 \text{ (ABGED)} 10 \text{ 个触发器。ROC (ABGED)} = 5/10 = >0,5$

[0137] 和 BCD 相比的 ABGED 比现有的系统更好 $0,5/0,000641 = 780\%$ 。但是,如果包括在处理器从编译级翻译到二进制级的触发器运算量,则这个 ROC 值会增大。

[0138] 略语表

[0139] BCD 二 - 十进制

[0140] DNA 动态数值访问

[0141] ABGED 基于人工经验的二进制编码扩展指数十进制（以阿拉伯文写的 ABCD 被读作 AB-GED）。

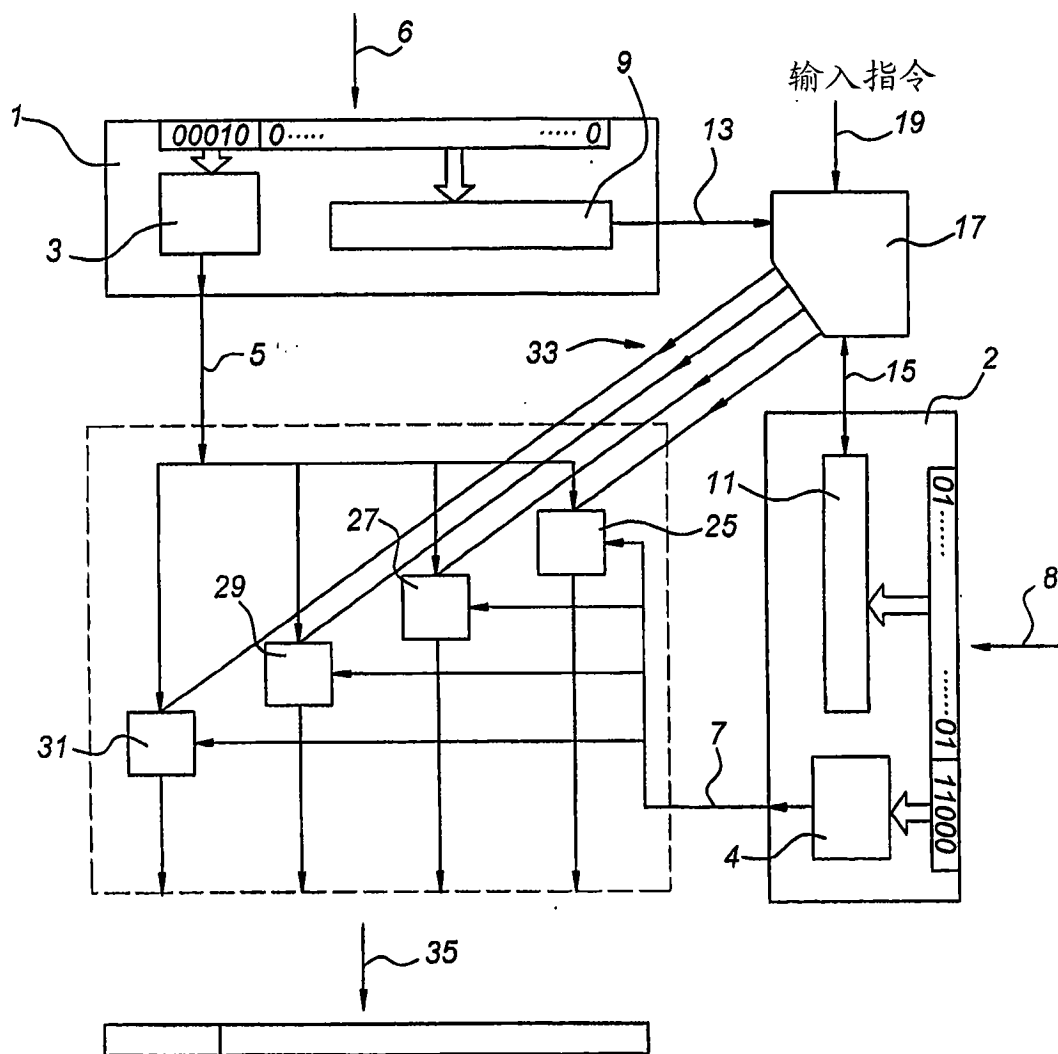


图 1

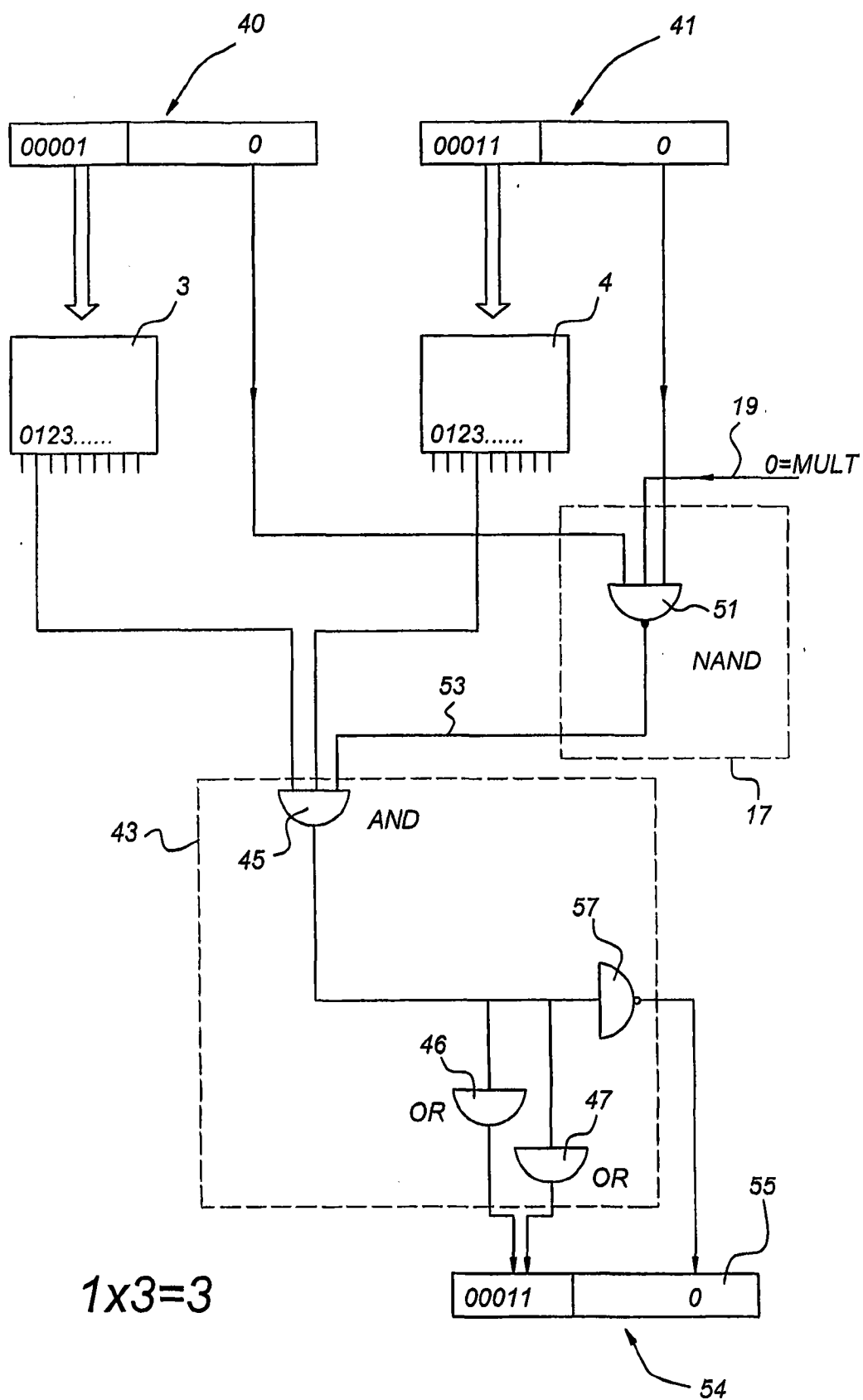


图 2

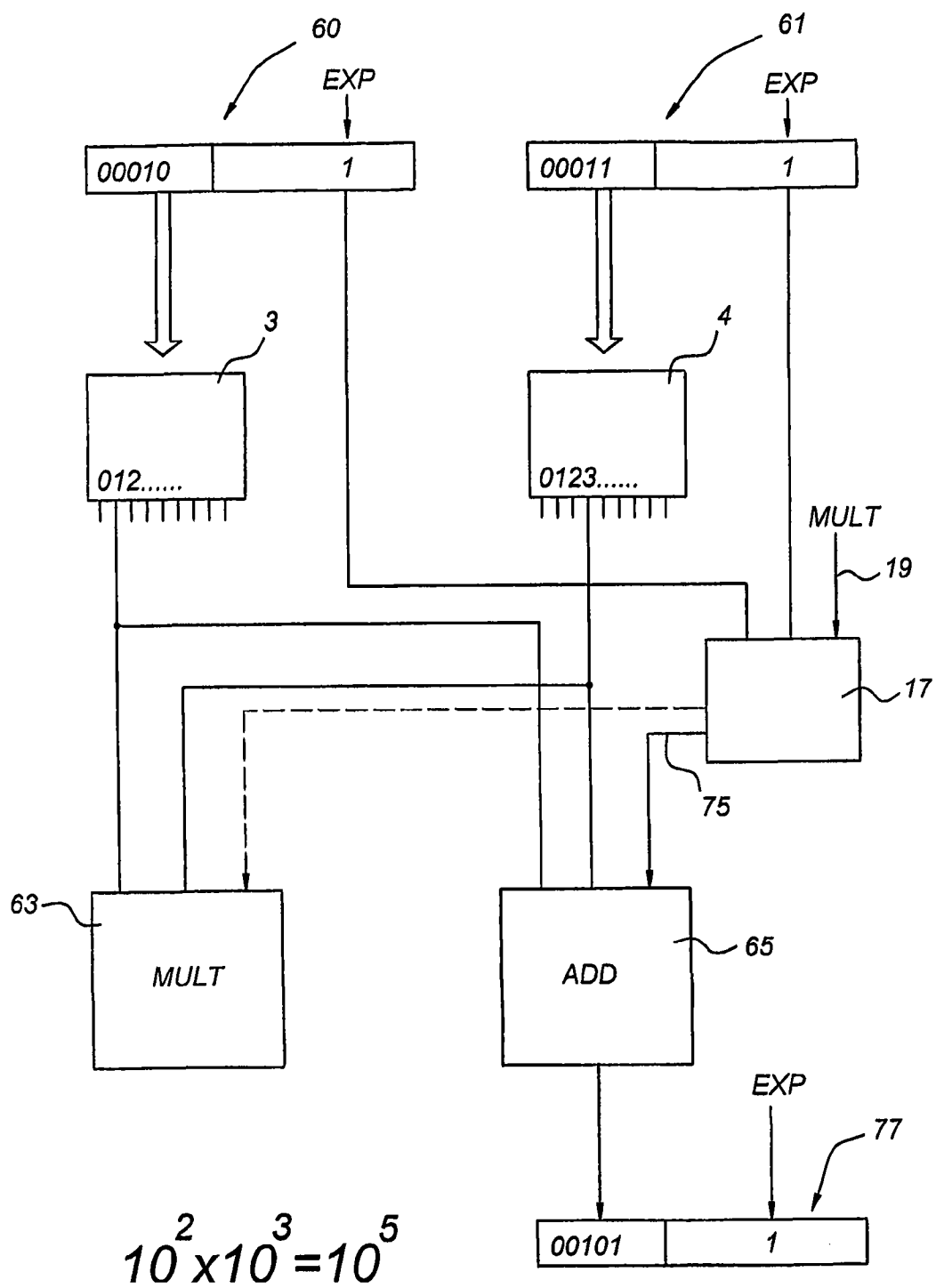


图 3

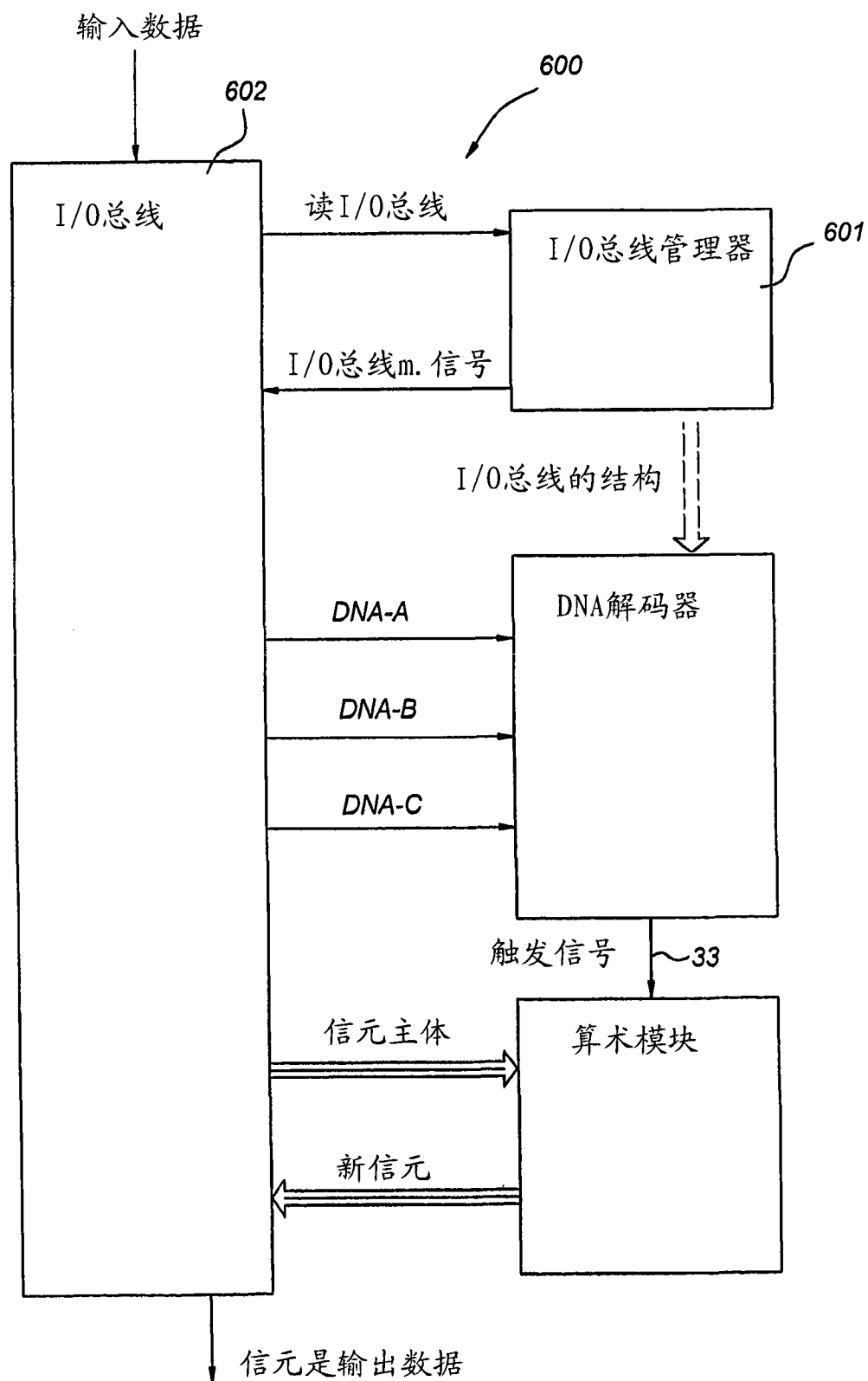


图 5

		cell-core					position				type				DNA group A		
		a	e	d	c	b	a	d	c	b	a	d	c	b	c	b	a
Binary Code + Decimal	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1
	2	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1
	3	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	1
	4	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1
	5	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0	1
	6	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	1
	7	0	0	1	1	1	0	0	0	0	0	0	0	1	0	0	1
	8	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1
	9	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	1
	not used	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	1
	not used	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0	1
	not used	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	1
	comma	0	1	1	0	1	0	0	0	0	0	0	0	1	0	0	1
	dot	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	1
	fraction	0	1	1	1	1	0	0	0	0	0	0	0	1	0	0	1
Binary Code - Decimal	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
	-1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1
	-2	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1
	-3	1	0	0	1	1	0	0	0	0	0	0	0	1	0	0	1
	-4	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1
	-5	1	0	1	0	1	0	0	0	0	0	0	0	1	0	0	1
	-6	1	0	1	1	0	0	0	0	0	0	0	0	1	0	0	1
	-7	1	0	1	1	1	0	0	0	0	0	0	0	1	0	0	1
	-8	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1
	-9	1	1	0	0	1	0	0	0	0	0	0	0	1	0	0	1
	not used	1	1	0	1	0	0	0	0	0	0	0	0	1	0	0	1
	not used	1	1	0	1	1	0	0	0	0	0	0	0	1	0	0	1
	not used	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	1
	comma	1	1	1	0	1	0	0	0	0	0	0	0	1	0	0	1
	dot	1	1	1	1	0	0	0	0	0	0	0	0	1	0	0	1
	fraction	1	1	1	1	1	0	0	0	0	0	0	0	1	0	0	1
Extended Binary Code exponenti al Decimal at + Log.	E0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	E1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
	E2	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
	E3	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1
	E4	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
	E5	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1
	E6	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1
	E7	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1
	E8	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	E9	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1
	not used	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1
	not used	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1
	not used	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1
	comma	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1
	dot	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1
	fraction	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1
Extended Binary Code exponenti al Decimal at - Log.	E-1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	E-2	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
	E-3	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
	E-4	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1
	E-5	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
	E-6	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1
	E-7	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1
	E-8	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1
	E-9	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	not used	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1
	not used	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1
	not used	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1
	not used	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1
	not used	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1
	not used	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1
	not used	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1



图 7



A	0	0	0	0	0
B	0	0	0	0	1
C	0	0	0	1	0
D	0	0	0	1	1
E	0	0	1	0	0
F	0	0	1	0	1
G	0	0	1	1	0
H	0	0	1	1	1
I	0	1	0	0	0
J	0	1	0	0	1
K	0	1	0	1	1
L	0	1	1	0	0
M	0	1	1	0	1
N	0	1	1	1	0
O	0	1	1	1	1
P	1	0	0	0	0
Q	1	0	0	0	1
R	1	0	0	1	0
S	1	0	0	1	1
T	1	0	1	0	0
U	1	0	1	0	1
V	1	0	1	1	0
W	1	0	1	1	1
X	1	1	0	0	0
Y	1	1	0	0	1
Z	1	1	1	0	1
a	1	1	1	1	0
b	1	1	1	1	1
c	1	1	1	1	1
d	1	1	1	1	1
e	1	1	1	1	1
f	1	1	1	1	1
g	1	1	1	1	1
h	1	1	1	1	1
i	1	1	1	1	1
j	1	1	1	1	1
k	1	1	1	1	1
l	1	1	1	1	1
m	1	1	1	1	1
n	1	1	1	1	1
o	1	1	1	1	1
p	1	1	1	1	1
q	1	1	1	1	1
r	1	1	1	1	1
s	1	1	1	1	1
t	1	1	1	1	1
u	1	1	1	1	1
v	1	1	1	1	1
w	1	1	1	1	1
x	1	1	1	1	1
y	1	1	1	1	1
z	1	1	1	1	1

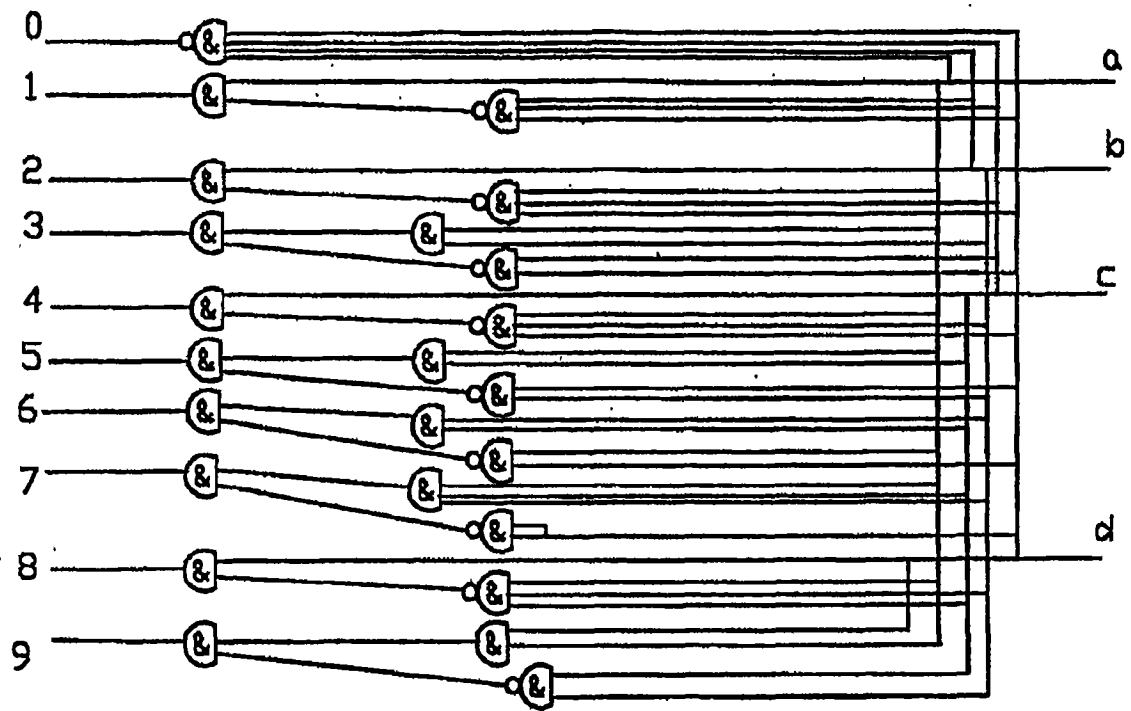


图 8