(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 103064650 B
(45)授权公告日 2016.02.24

(21)申请号 201210574820.2
(22)申请日 1996.07.17
(30)优先权数据
08/521360 1995.08.31 US
(62)分案原申请数据
96197839.2 1996.07.17
(73)专利权人 英特尔公司
地址 美国加利福尼亚州
(72)发明人 A.D.佩勒格 Y.雅里 M.米塔尔
L.M.门内梅尔 B.艾坦 A.F.格卢
C.杜龙 E.科瓦施 W.维特
(74)专利代理机构 中国专利代理(香港)有限公司 72001
代理人 徐予红 朱海煜
(51)Int.Cl.
G06F 7/57(2006.01)
G06F 7/544(2006.01)
G06F 7/60(2006.01)
G06F 9/30(2006.01)
G06F 15/78(2006.01)

(56)对比文件
CN 1106152 A,1995.08.02,
US 5268855 A,1993.12.07,
US 5257214 A,1993.10.26,

审查员 贾云杰

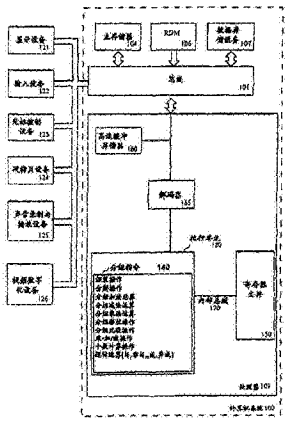
权利要求书2页 说明书57页 附图29页

(54)发明名称

控制移位分组数据的位校正的装置

(57)摘要

一种在处理器中加入支持典型的多媒体应用所要求的分组数据上的操作的指令集的装置。在一个实施例中,本发明包括具有存储区(150)、解码器(165)及多个电路(130)的处理器。该多个电路提供若干指令的执行来操作分组数据。在这一实施例中,这些指令包含组装、分解、分组乘法、分组加法、分组减法、分组比较及分组移位。



CN 103064650 B

1. 一种处理装置，包括：

高速缓冲存储器（206）；

解码器（165），可操作以解码分组指令集（140）；

一组寄存器（209），可操作以存储包括第一多个数据元素的第一分组数据和包括第二多个数据元素的第二分组数据；

执行单元（130），耦合至所述高速缓冲存储器、所述解码器和所述一组寄存器，所述执行单元可操作以执行所述分组指令集的解码的指令，包括指定下列操作的指令：

组装操作，用于组装来自至少两个分组数据中的数据元素的一部分位以形成第三分组数据；

分解操作，用于生成包含来自所述第一分组数据的至少一个数据元素和来自所述第二分组数据的至少一个相对应的数据元素的第四分组数据。

2. 如权利要求 1 所述的处理装置，其中所述分组指令集还包括指定下列操作的指令：

分组加法操作，用于将来自至少两个分组数据的相对应的数据元素并行地加在一起。

3. 如权利要求 1 所述的处理装置，其中所述分组指令集还包括指定下列操作的指令：

分组减法操作，用于将来自至少两个分组数据的相对应的数据元素并行地相减。

4. 如权利要求 1 所述的处理装置，其中所述分组指令集还包括指定下列操作的指令：

分组乘法操作，用于将来自至少两个分组数据的相对应的数据元素相乘。

5. 如权利要求 1 所述的处理装置，其中所述分组指令集还包括指定下列操作的指令：

分组移位操作，用于将分组数据操作数中的各数据元素移位所指示的计数。

6. 如权利要求 1 到 5 中的任一项所述的处理装置，其中所述分组指令集还包括指定下列操作的指令：

分组比较操作，用于根据指示的关系将来自至少两个分组数据的相对应的数据元素进行比较并且作为结果将分组掩码存储在第一寄存器中，其中所述分组掩码将至少包含第一掩码元素和第二掩码元素，所述第一掩码元素中的每个位指示比较一组相对应的数据元素的结果，而所述第二掩码元素中的每个位指示比较第二组数据元素的结果。

7. 如权利要求 1 到 5 中的任一项所述的处理装置，其中所述分组指令集还包括指定下列操作的指令：

乘 - 加操作，用于将来自两个分组数据的相对应的数据元素相乘以生成中间结果数据元素，并且将所述中间结果数据元素相加。

8. 一种计算系统，包括：

显示装置（121），用于支持触摸界面；

高速缓冲存储器（206）；

解码器（165），可操作以解码分组指令集（140）；

一组寄存器（209），可操作以存储包括第一多个数据元素的第一分组数据和包括第二多个数据元素的第二分组数据；以及

执行单元（130），耦合至所述高速缓冲存储器、所述解码器和所述一组寄存器，所述执行单元可操作以执行所述分组指令集的解码的指令，包括指定下列操作的指令：

组装操作，用于组装来自至少两个分组数据中的数据元素的一部分位以形成第三分组数据；

分解操作，用于生成包含来自所述第一分组数据的至少一个数据元素和来自所述第二分组数据的至少一个相对应的数据元素的第四分组数据；

分组加法操作，用于将来自至少两个分组数据的相对应的数据元素并行地相加；

分组减法操作，用于将来自至少两个分组数据的相对应的数据元素并行地相减；

分组乘法操作，用于将来自至少两个分组数据的相对应的数据元素相乘；

分组移位操作，用于将分组数据操作数中的各数据元素移位所指示的计数；

分组比较操作，用于根据指示的关系将来自至少两个分组数据的相对应的数据元素进行比较并且作为结果将分组掩码存储在第一寄存器中，其中所述分组掩码将至少包含第一掩码元素和第二掩码元素，所述第一掩码元素中的每个位指示比较一组相对应的数据元素的结果，而所述第二掩码元素中的每个位指示比较第二组数据元素的结果；以及

乘－加操作，用于将来自两个分组数据的相对应的数据元素相乘以生成中间结果数据元素，并且将所述中间结果数据元素相加。

控制移位分组数据的位校正的装置

[0001] 本申请是申请日为1996年7月17日、申请号为200610101459.6、发明名称为“控制移位分组数据的位校正的装置”的专利申请的分案申请。

技术领域

[0002] 本发明具体涉及计算机系统领域。更具体地，本发明涉及分组数据操作领域。

背景技术

[0003] 在典型的计算机系统中，将处理器实现为利用产生一种结果的指令在由大量的位（如64）表示的值上操作。例如，执行加法指令将第一个64位值与第二个64位值相加并作为第三个64位值存储该结果。然而，多媒体应用（诸如以计算机支持的协作为目的的应用（CSC-电话会议与混合媒体数据处理的集成）、2D/3D图形、图象处理、视频压缩/解压、识别算法与音频处理）要求处理可以用少量的位表示的大量数据。例如，图形数据通常需要8或16位，声音数据通常需要8或16位。这些多媒体应用的各个需要一种或多种算法，各需要若干操作。例如，算法可能需要加法、比较及移位操作。

[0004] 为了改进多媒体应用（以及具有相同特征的其它应用），先有技术处理器提供分组数据格式。分组数据格式中通常用来表示单个值的位被分成若干固定长度的数据元素，各元素表示单独的值。例如，可将一个64位寄存器分成两个32位元素，各元素表示一个单独的32位值。此外，这些先有技术处理器提供并行分开处理这些分组数据类型中各元素的指令。例如，分组的加法指令将来自第一分组数据与第二分组数据的对应数据元素相加。从而，如果多媒体算法需要包含必须在大量数据元素上执行的五种操作的循环，总是希望组装该数据并利用分组数据指令并行执行这些操作。以这一方式，这些处理器便能更高效地处理多媒体应用。

[0005] 然而，如果该操作循环中包含处理器不能在分组数据上执行的操作（即处理器缺少适当的指令），则必须分解该数据来执行该操作。例如，如果多媒体算法要求加法运算而不能获得上述分组加法指令，则程序员必须分解第一分组数据与第二分组数据（即分开包含第一分组数据与第二分组数据的元素），将各个分开的单独的元素相加，然后将结果组装成分组的结果供进一步分组处理。执行这种组装与分解所需的处理时间通常抵消了提供分组数据格式的性能优点。因此，希望在通用处理器上包含提供典型多媒体算法所需的所有操作的分组数据指令集。然而，由于当今微处理器上的有限芯片面积，可以增加的指令数目是有限的。

[0006] 包含分组数据指令的一种通用处理器便是加州Santa Clara的Intel公司制造的i860XP™处理器。i860XP处理器包含具有不同元素大小的若干分组数据类型。此外，i860XP处理器包含分组加法与分组比较指令。然而，分组加法指令并不断开进位链，因此程序员必须保证软件正在执行的运算不会导致溢出，即运算不会导致来自分组数据中一个元素的位溢出到该分组数据的下一元素中。例如，如果将值1加到存储“11111111”的8位分组数据元素上，便出现溢出而结果为“100000000”。此外，i860XP所支持的分组数据类型中的小数

点位置是固定的（即 i860XP 处理器支持数 8.8、6.10 与 8.24，其中数 i.j 包含 i 个最高位及小数点后的 j 位）。从而限制了程序员可以表示的值。由于 i860XP 处理器只支持这两条指令，它不能执行采用分组数据的多媒体算法所要求的许多运算。

[0007] 另一种支持分组数据的通用处理器便是 Motorala 公司制造的 MC88110™处理器。MC88110 处理器支持具有不同长度元素的若干种不同的分组数据格式。此外，MC88110 处理器所支持的分组指令集中包括组装、分解、分组加法、分组减法、分组乘法、分组比较与分组旋转。

[0008] MC88110 处理器分组命令通过连接第一寄存器对中的各元素的 (t*r)/64（其中 t 为该分组数据的元素中的位数）个最高有效位进行操作来生成宽度为 r 的一个字段。该字段取代存储在第二寄存器对中的分组数据的最高有效位。然后将这一分组数据存储在第三寄存器对中并左旋 r 位。下面在表 1 与 2 中示出所支持的 t 与 r 值，以及这一指令的运算实例。

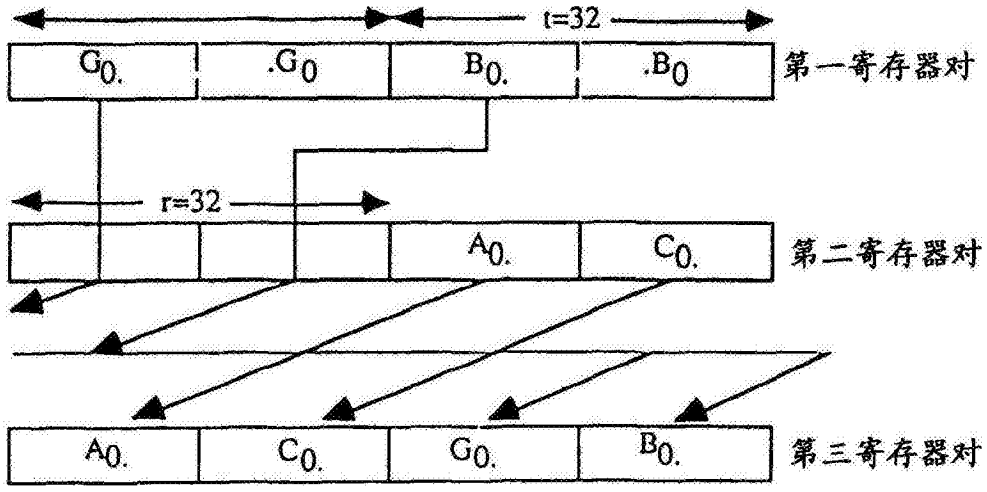
[0009]

		r		
		8	16	32
t	8	x	x	4
	16	x	4	8
	32	4	8	16

[0010] X ＝未定义的操作

[0011] 表 1

[0012]



[0013] 表 2

[0014] 分组指令的这一实现具有两个缺点。第一是需要附加的逻辑在指令结束时执行旋转。第二是生成分组数据结果所需的指令数目。例如，如果希望使用 4 个 32 位值来生成第三寄存器（以上所示）中的结果，便需要两条具有 t ＝ 32 与 r ＝ 32 的指令，如下面表 3 中所示。

ppack Source1,Source2

A0.	.A0	C0.	.C0	(源1)
x	x	x	x	(源2)
=				
x	x	A0.	C0.	(结果1)

[0015]

ppack Result1,Source3

G0.	.G0	B0.	.B0	(结果1)
x	x	A0.	C0.	(源3)
=				
A0.	C0.	G0.	B0.	(结果2)

[0016] 表 3

[0017] MC88110 处理器分解命令通过将来自分组数据的 4、8 或 16 位数据元素放入两倍长 (8、16 或 32 位) 的数据元素的低位一半中，并填充以零，即将得出的数据元素的较高位设定为零进行操作。下面表 4 中示出了这一分解命令的操作的一个例子。

[0018]

第一寄存器对

00101010	01010101	01010101	11111111	10000000	01110000	10001111	10001000
7	6	5	4	3	2	1	0

解开

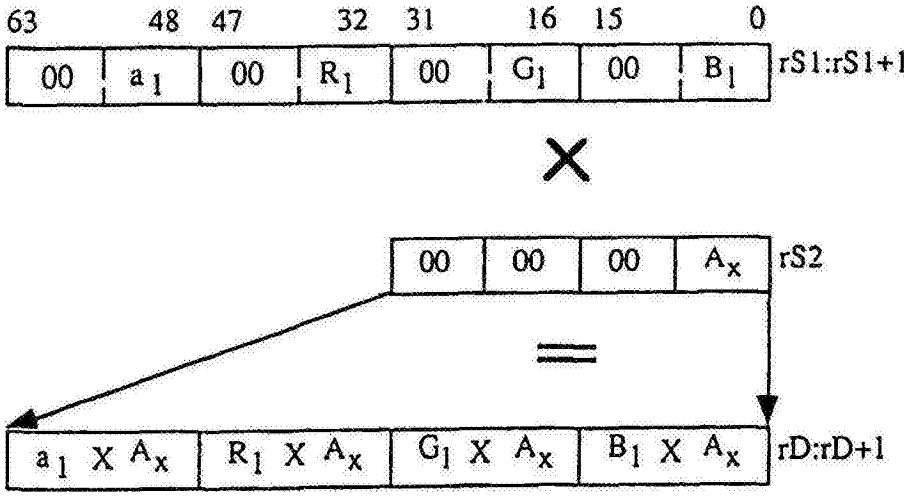
第二寄存器对

00000000 10000000	00000000 01110000	00000000 10001111	00000000 10001000
3	2	1	0

[0019] 表 4

[0020] MC88110 处理器分组乘法指令将 64 位分组数据的各元素乘以一个 32 位值，如同该分组数据表示单一的值一样，如下面表 5 中所示。

[0021]



[0022] 表 5

[0023] 这一乘法指令具有两个缺点。首先这一乘法指令并不断开进位链，从而程序员必须保证在分组数据上执行的运算并不导致溢出。结果，程序员有时必须加入附加的指令来防止这一溢出。第二，这一乘法指令将分组数据中的各元素乘以单一的值（即该 32 位值）。结果，用户没有选择分组数据中哪些元素乘以该 32 位值的灵活性。因此，程序员必须制备数据使得分组数据中的每一个元素上都需要相同的乘法或者每当需要对该数据中少于全部元素进行乘法时浪费处理时间来分解数据。因此程序员不能并行利用多个乘数来执行多个乘法。例如，将 8 个不同的数据片相乘，每一数据片一个字长，需要四次单独的乘法运算。各运算每次乘两个字，实际上浪费了用于位 16 以上的位的数据线与电路。

[0024] MC88110 处理器分组比较指令比较来自第一分组数据与第二分组数据的对应的 32 位数据元素。两个比较中各个可能返回小于（＜）或大于等于（≥）之一，得出四种可能的组合。该指令返回一个 8 位结果串；四位表示符合四种可能条件中哪一种，四位表示这些位的补码。根据这一指令的结果的条件转移能以两种方式实现：1) 用一序列条件转移；或 2) 用跳转表。该指令的问题在于它需要根据数据的条件转移来执行函数的事实，诸如：ifY ＞ A then X ＝ X+B else X ＝ X。这一函数的伪码编译表示将是：

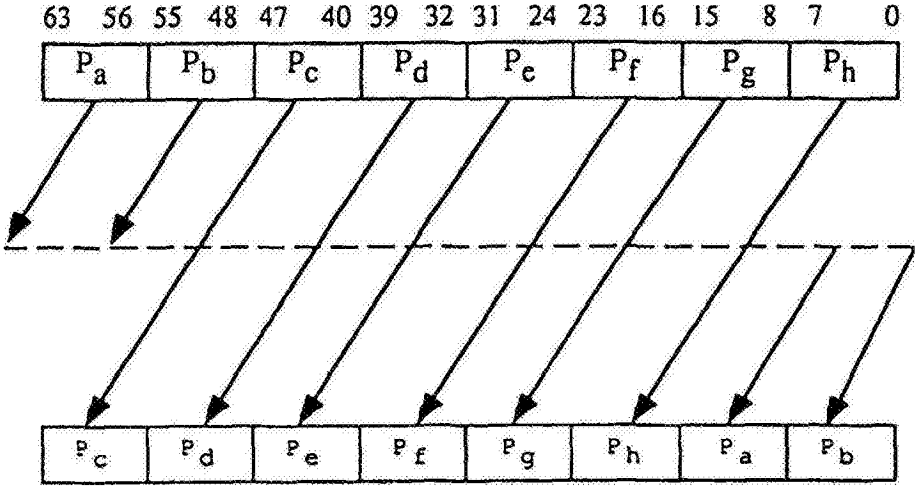
[0025]

```
比较 Y，A
如果条件标志指示 Y≤A 则转移到
加 X，B
...
```

[0026] 新的微处理器试图通过推测转移到哪里来加快执行。如果预测正确，便不损失性能并且存在着提高性能的潜力。然而如果预测错误，便损失性能。因此，预测得好的鼓励是巨大的。然而根据数据的转移（诸如上面的）呈现为不可预测的方式，这破坏了预测算法并得出更多的错误预测。结果，使用这一比较指令来建立根据数据的条件转移要付出性能上的高昂代价。

[0027] MC88110 处理器旋转指令旋转一个 64 位值到 0 与 60 位之间的任一模 4 边界上（见下面表 6 的示例）。

[0028]



[0029] 表 6

[0030] 由于旋转指令使移出寄存器的高位移入寄存器的低位，MC88110 处理器并不支持

单个地移位分组数据中的各元素。结果，要求单独移位分组数据类型中各元素的编程算法需要：1）分解数据，2）单独地在各元素上执行移位，及 3）将结果组装成结果分组数据供进一步分组数据处理。

发明内容

[0031] 本发明描述了在处理器中加入支持典型的多媒体应用所要求的操作的分组数据指令集的方法与装置。在一个实施例中，本发明包括一个处理器及一个存储区。存储区中包含若干指令供处理器执行以操作分组数据。在这一实施例中，这些指令包括组装、分解、分组加法、分组减法、分组乘法、分组移位及分组比较。

[0032] 处理器对接收组装指令作出响应，组装来自至少两个分组数据中的数据元素的一部分位以构成第三分组数据。作为比较，处理器对接收该分解指令作出响应，生成包含来自第一分组数据操作数的至少一个数据元素及来自第二分组数据操作数的至少一个对应数据元素的第四分组数据。

[0033] 处理器响应接收该分组加法指令单独地将来自至少两个分组数据的对应数据元素并行加在一起。作为对比，处理器响应接收该分组减法指令单独地将来自至少两个分组数据的对应数据元素并行相减。

[0034] 处理器响应接收分组乘法指令单独地将来自至少两个分组数据的对应数据元素并行相乘。

[0035] 处理器响应接收分组移位指令单独地将分组数据操作数中的各数据元素并行移位所指示的计数值。

[0036] 处理器响应接收分组比较指令按照指示的关系单独地将来自至少两个分组数据的对应数据元素并行比较，并作为结果将一个分组掩码存储在第一寄存器中。分组掩码至少包含第一掩码元素与第二掩码元素。第一掩码元素中的各位表示比较一组对应数据元素的结果，而第二掩码元素中的各位表示第二组数据元素的比较结果。

附图说明

[0037] 本发明是在附图中用示例而非限制方式说明的。相同的参照指示相同的元素。

[0038] 图 1 说明按照本发明的一个实施例的示例性计算机系统。

[0039] 图 2 说明按照本发明的一个实施例的处理器的寄存器文件。

[0040] 图 3 为说明按照本发明的一个实施例的处理器用来处理数据的通用步骤的流程图。

[0041] 图 4 说明按照本发明的一个实施例的分组数据类型。

[0042] 图 5a 表示按照本发明的一个实施例的寄存器中分组数据。

[0043] 图 5b 表示按照本发明的一个实施例的寄存器中分组数据。

[0044] 图 5c 表示按照本发明的一个实施例的寄存器中分组数据。

[0045] 图 6a 表示按照本发明的一个实施例指示分组数据的使用的控制信号格式。

[0046] 图 6b 说明按照本发明的一个实施例的指示分组数据的使用的第二控制信号格式。

[0047] 分组加法 / 减法

[0048] 图 7a 说明按照本发明的一个实施例执行分组加法的方法。

[0049] 图 7b 说明按照本发明的一个实施例执行分组减法的方法。

[0050] 图 8 说明按照本发明的一个实施例在分组数据的各个位上执行分组加法与分组减法的电路。

[0051] 图 9 说明按照本发明的一个实施例在分组字节数据上执行分组加法与分组减法的电路。

[0052] 图 10 为按照本发明的一个实施例在分组字数据上执行分组加法与分组减法的电路的逻辑视图。

[0053] 图 11 为按照本发明的一个实施例在分组双字数据上执行分组加法与分组减法的电路的逻辑视图。

[0054] 分组乘法

[0055] 图 12 为说明按照本发明的一个实施例在分组数据上执行分组乘法运算的方法的流程图。

[0056] 图 13 说明按照本发明的一个实施例执行分组乘法的电路。

[0057] 乘 - 加 / 减

[0058] 图 14 为说明按照本发明的一个实施例在分组数据上执行乘 - 加与乘 - 减运算的方法的流程图。

[0059] 图 15 说明按照本发明的一个实施例在分组数据上执行乘 - 加与 / 或乘 - 减运算的电路。

[0060] 分组移位

[0061] 图 16 为说明按照本发明的一个实施例在分组数据上执行分组移位操作的方法的流程图。

[0062] 图 17 说明按照本发明的一个实施例在分组数据的各个字节上执行分组移位的电路。

[0063] 组装

[0064] 图 18 为说明按照本发明的一个实施例在分组数据上执行组装操作的方法的流程图。

[0065] 图 19a 说明按照本发明的一个实施例在分组字节数据上执行组装操作的电路。

[0066] 图 19b 说明按照本发明的一个实施例在分组字数据上执行组装操作的电路。

[0067] 分解

[0068] 图 20 为说明按照本发明的一个实施例在分组数据上执行分解操作的方法的流程图。

[0069] 图 21 说明按照本发明的一个实施例在分组数据上执行分解操作的电路。

[0070] 个数计数

[0071] 图 22 为说明按照本发明的一个实施例在分组数据上执行个数计数操作的方法的流程图。

[0072] 图 23 为说明按照本发明的一个实施例在分组数据的一个数据元素上执行个数计数操作及为结果分组数据生成单一结果数据元素的方法的流程图。

[0073] 图 24 说明按照本发明的一个实施例在具有四个字数据元素的分组数据上执行个

数计数操作的电路。

[0074] 图 25 说明按照本发明的一个实施例在分组数据的一个字数据元素上执行个数计数操作的详细电路。

[0075] 分组逻辑运算。

[0076] 图 26 为说明按照本发明的一个实施例在分组数据上执行若干逻辑运算的方法的流程图。

[0077] 图 27 说明按照本发明的一个实施例在分组数据上执行逻辑运算的电路。

[0078] 分组比较

[0079] 图 28 为说明按照本发明的一个实施例在分组数据上执行分组比较操作的方法的流程图。

[0080] 图 29 说明按照本发明的一个实施例的分组数据的单个字节上执行分组比较操作的电路。

具体实施方式

[0081] 本申请描述在处理器中包括支持典型的多媒体应用所要求的分组数据上的操作的指令集的方法与装置。在下面的描述中，陈述了许多特定细节以提供对本发明的全面理解。然而，应理解本发明可以不用这些特定细节实现。在其它实例中，为了避免使本发明不必要地冲淡，不详细示出众所周知的电路、结构与技术。

[0082] 定义

[0083] 为了提供理解本发明的实施例的描述的基础，提出以下定义。

[0084] 位 X 至位 Y ；

[0085] 定义二进制数的子字段。例如，字节 00111010_2（以基 2 表示）的位 6 至位 0 表示子字段 111010_2。二进制数后面的‘2’表示基 2。因此，1000_2等于 8_{10}，而 F_{16}等于 15_{10}。

[0086] R_X：为寄存器。寄存器为能存储与提供数据的任何器件。寄存器的进一步功能在下面描述。寄存器不是处理器组件的必要部件。

[0087] SRC1，SRC2 与 DEST ：

[0088] 标识存储区（诸如存储器地址、寄存器等）

[0089] Source1-i 与 Result1-i ：表示数据

[0090] 计算机系统

[0091] 图 1 说明按照本发明的一个实施例的示范性计算机系统 100。计算机系统 100 包括用于传递信息的总线 101 或其它通信硬件与软件，及用于处理信息的与总线 101 耦合的处理器 109。处理器 109 表示包含 CISC（复杂指令集计算）或 RISC（精减指令集计算）类型体系结构在内的任何类型体系结构的中央处理单元。计算机系统 100 还包括耦合在总线 101 上用于存储信息及要由处理器 109 执行的指令的随机存取存储器 (RAM) 或其它动态存储设备（称作主存储器 104）。在处理器 109 执行指令期间，主存储器 104 也可用来存储临时变量或其它中间信息。计算机系统 100 还包括耦合在总线 101 上用于存储静态信息及处理器 109 的指令的只读存储器 (ROM)106 与 / 或其它静态存储设备。数据存储设备 107 耦合在总线 101 上用于存储信息与指令。

[0092] 图 1 还示出处理器 109 包括执行单元 130、寄存器文件 150、高速缓冲存储器 160、

解码器 165 及内部总线 170。当然，处理器 109 还包含其它电路，为了不冲淡本发明而未示出它们。

[0093] 执行单元 130 用于执行处理器 109 所接收的指令。除了识别通常在通用处理器实现的指令，执行单元 130 还识别在分组数据格式上执行操作的分组指令集 140 中的指令。在一个实施例中，分组指令集 140 包含以此后描述的方式支持组装操作、分解操作、分组加法运算、分组减法运算、分组乘法运算、分组移位操作、分组比较操作、乘－加运算、乘－减运算、个数计算操作及一组分组逻辑运算（包含分组“与”、分组“与非”、分组“或”及分组“异或”）的指令。虽然描述了包含这些指令的分组指令集 140 的一个实施例，其它实施例可包含这些指令的子集或超集。

[0094] 通过包含这些指令，可以用分组数据执行多媒体应用中所使用的许多算法所需要的操作。从而，可以编写这些算法来组装必要的数据及在分组数据上执行必要的操作，而无须分解这些分组数据来一次在一个数据元素上执行一个或多个操作。如上所述，这比不支持某些多媒体算法所要求的分组数据操作的先有技术通用处理器（即，如果多媒体算法要求不能在分组数据上执行的操作，则程序必须分解该数据，单独地在分开的元素上执行操作，然后将结果组装成分组结果供进一步分组处理）具有性能上的优势。此外，所公开的在其中执行若干条这些指令的方式改进了许多多媒体应用的性能。

[0095] 执行单元 130 由内部总线 170 耦合在寄存器文件 150 上。寄存器文件 150 表示处理器 109 上用于存储包含数据在内的信息的存储区。应理解本发明的一个方面是在分组数据上操作的所描述的指令集。按照本发明的这一方面，用来存储分组数据的存储区不是关键的。但是，稍后参照图 2 描述寄存器文件 150 的一个实施例。执行单元 130 耦合在高速缓冲存储器 160 及解码器 165 上。高速缓冲存储器 160 用来高速缓冲来自诸如主存储器 104 的数据与 / 或控制信号、解码器 165 用来将处理器 109 所接收的指令解码成控制信号与 / 或微代码入口点。响应这些控制信号与 / 或微代码入口点，执行单元 130 执行适当的操作。例如，如果接收到一个加法指令，解码器 165 便令执行单元 130 执行要求的加法；如果接收到一个减法指令，解码器 165 便令执行单元 130 执行要求的减法；等。解码器 165 可用任何数目的不同机构实现（诸如，查找表、硬件实现、PLA 等）。从而，尽管解码器与执行单元执行各种指令是由一系列 if/then 语句表示的，但应理解指令的执行并不需要这些 if/then 语句的一系列处理。而是将任何用于逻辑执行这一 if/then 处理的机构认为是在本发明的范围之内。

[0096] 图 1 还示出了数据存储设备 107，诸如磁盘或光盘，及其对应的盘驱动器。计算机系统 100 也能通过总线 101 耦合在将信息显示给计算机用户的显示设备 121 上。显示设备 121 可包含帧缓冲器、专用的图形描绘设备 (graphics rendering device)、阴极射线管 (CRT) 与 / 或平板显示器。包含字母数字与其它键的字母数字输入设备 122 通常耦合在总线 101 上，用于向处理器 109 传递信息及命令选择。另一种用户输入设备为光标控制设备 123，诸如用于传递方向信息及命令选择给处理器 109 及用于控制光标在显示设备 121 上运动的鼠标器、轨迹球、笔、触摸屏或光标方向键。这一输入设备通常具有两根轴上的两个自由度，第一轴（如 X）及第二轴（如 Y），它允许设备指定平面上的位置。然而，本发明不应限制在只有两个自由度的输入设备上。

[0097] 另一可以耦合到总线 101 上的设备为可用来在诸如纸、胶片等介质或类似类型的

介质上打印指令、数据或其它信息的硬拷贝设备 124。此外，可将计算机系统 100 耦合在用于声音录制与 / 或播放的设备 125 上，诸如耦合在用于录制信息的麦克风上的音频数字化器。此外，该设备可包含耦合在数模 (D/A) 转换器上的扬声器，用于播放数字化声音。

[0098] 计算机系统 100 也可以是计算机网络（诸如 LAN）中的终端。计算机系统 100 这时便是计算机网络的一个计算机子系统。计算机系统 100 可选地包含视频数字化设备 126。视频数字化设备 126 能用来捕捉能在计算机网络上传输给其它计算机的视频图象。

[0099] 在一个实施例中，处理器 109 附加支持与 X86 指令集（诸如加州 Santa Clara 的 Intel 公司制造的Pentium®处理器等现有微处理器所使用的指令集）兼容的指令集。从而，在一个实施例中，处理器 109 支持加州 Santa Clara 的 Intel 公司所定义的 IA™-Intel 体系结构所支持的所有操作（见“微处理器”，Intel 资料集卷 1 与卷 2，1992 与 1993，可从加州 Santa Clara 的 Intel 购得）。结果，除了本发明的操作外，处理器 109 还能支持现有的 X86 操作。虽然本发明是描述为包含在基于 X86 指令集中的，替代实施例可将本发明包含在其它指令集中。例如，可将本发明包含在采用新指令集的 64 位处理器中。

[0100] 图 2 说明按照本发明的一个实施例的处理器的寄存器文件。寄存器文件 150 用来存储信息，包括控制 / 状态信息、整数数据、浮点数据及分组数据。在图 2 所示的实施例中，寄存器文件 150 包括整数寄存器 201、寄存器 209、状态寄存器 208 及指令指针寄存器 211。状态寄存器 208 指示处理器 109 的状态。指令指针寄存器 211 存储要执行的下一条指令的地址。整数寄存器 201、寄存器 209、状态寄存器 208 及指令指针寄存器 211 全都耦合在内部总线 170 上。任何附加的寄存器也耦合在内部总线 170 上。

[0101] 在一个实施例中，寄存器 209 既用于分组数据又用于浮点数据。在这一实施例中，处理器 109 在任何给定时刻都必须将寄存器 209 作为栈定位的浮点寄存器或作为非栈定位的分组数据寄存器对待。在本实施例中，包括了一种机制允许处理器 109 在作为栈定位的浮点寄存器与非栈定位的分组数据寄存器的寄存器 209 上操作之间切换。在另一实施例中，处理器 109 可同时在作为非栈定位的浮点与分组数据寄存器的寄存器 209 上操作。作为另一实例，在另一实施例中，这些相同的寄存器可用来存储整数数据。

[0102] 当然，可以实现替代的实施例来包括或多或少的寄存器组。例如，替代实施例可包含独立的浮点寄存器组用于存储浮点数据。作为另一实例，替代实施例可包含第一组寄存器，各用于存储控制 / 状态信息，及第二组寄存器，各能存储整数、浮点及分组数据。为了清楚起见，不应将一个实施例的寄存器的意义限制在特定类型的电路上。而是，一个实施例的寄存器只需要能存储与提供数据，并执行这里所描述的功能。

[0103] 可将各种寄存器组（诸如整数寄存器 201、寄存器 209）实现成包含不同数目的寄存器与 / 或不同大小的寄存器。例如，在一个实施例中，将整数寄存器 201 实现为存储 32 位，而将寄存器 209 实现为存储 80 位（全部 80 位用来存储浮点数据而只用 64 位存储分组数据）。此外，寄存器 209 包含 8 个寄存器，$R_0$212a 至 $R_7$212h。$R_1$212a、$R_2$212b 及 $R_3$212c 为寄存器 209 中各个寄存器的实例。可将寄存器 209 中的一个寄存器的 32 位移到整数寄存器 201 中的一个整数寄存器中。类似地，可将整数寄存器中的值移入寄存器 209 中的一个寄存器 32 位中。在另一实施例中，整数寄存器 201 各包含 64 位，并且 64 位数据可在整数寄存器 201 与寄存器 209 之间传送。

[0104] 图 3 为说明按照本发明的一个实施例的处理器用来处理数据的通用步骤的流程

图。例如，这些操作中包含加载操作，将来自高速缓冲存储器 160、主存储器 104、只读存储器 (ROM)104 或数据存储设备 107 的数据加载到寄存器文件 150 中的寄存器。

[0105] 在步骤 301，解码器 202 接收来自高速缓冲存储器 160 或总线 101 的控制信号 207。解码器 202 解码该控制信号来确定要执行的操作。

[0106] 在步骤 302，解码器 202 存取寄存器文件 150 或存储器中的单元。取决于控制信号 207 中指定的寄存器地址存取寄存器文件 150 中的寄存器或存储器中的存储器单元。例如，对于分组数据上的操作，控制信号 207 可包含 SRC1、SRC2 及 DEST 寄存器地址。SRC1 是第一源寄存器的地址。SRC2 是第二源寄存器的地址。由于不是所有操作都需要两个源地址，在某些情况中 SRC2 地址是可选的。如果一种操作不需要 SRC2 地址，便只使用 SRC1 地址。DEST 是存储结果数据的目的地寄存器的地址。在一个实施例中，SRC1 或 SRC2 也用作 DEST。相对于图 6a 与图 6b 更全面地描述 SRC1、SRC2 及 DEST。存储在对应寄存器中的数据分别称作源 1(Source1)、源 2(Source2) 与结果 (Result)。每一个这种数据的长度都是 64 位。

[0107] 在本发明的另一实施例中，SRC1、SRC2 及 DEST 中任何一个或全部能定义处理器 109 的可寻址存储器空间中的一个存储器单元。例如，SRC1 可标识主存储器 104 中的存储器单元，而 SRC2 标识整数寄存器 201 中的第一寄存器及 DEST 标识寄存器 209 中的第二寄存器。这里为了简化描述，本发明将相对于存取寄存器文件 150 描述。然而，这些存取能对存储器进行。

[0108] 在步骤 303，启动执行单元 130 在存取的数据上执行操作。在步骤 304，按照控制信号 207 的要求将结果存储回寄存器文件 150。

[0109] 数据与存储格式

[0110] 图 4 说明按照本发明的一个实施例的分组数据类型。示出了三种分组数据格式：分组字节 401、分组字 402 及分组双字 403。在本发明的一个实施例中，分组字节为包含 8 个数据元素的 64 位长。各数据元素为一个字节长。通常，数据元素是与其它相同长度的数据元素一起存储在单一寄存器（或存储器单元）中的一个单独的数据片段。在本发明的一个实施例中，存储在寄存器中的数据元素的数目是 64 位除以一个数据元素的位长度。

[0111] 分组字 402 为 64 位长并包含 4 个字 402 数据元素。各字 402 数据元素包含 16 位信息。

[0112] 分组双字 403 为 64 位长并包含两个双字 403 数据元素。各双字 403 数据元素包含 32 位信息。

[0113] 图 5a 至 5c 说明按照本发明的一个实施例的寄存器中分组数据存储表示。无符号分组字节的寄存器表示 510 示出在寄存器 R0212a 至 R7212h 之一中无符号分组字节 401 的存储。各字节数据元素的信息存储在字节 0 的位 7 至位 0、字节 1 的位 15 至位 8、字节 2 的位 23 至位 16、字节 3 的位 31 至位 24、字节 4 的位 39 至位 32、字节 5 的位 47 至位 40、字节 6 的位 55 至位 48 及字节 7 的位 63 至位 56 中。从而，寄存器中使用了所有可利用的位。这一存储布置提高了处理器的存储效率。同时，通过存取 8 个数据元素，便可同时在 8 个数据元素上执行一种操作。带符号分组字节寄存器表示 511 示出带符号分组字节 401 的存储。注意只需要每一个字节数据元素的第八位用于符号指示。

[0114] 无符号分组字寄存器表示 512 示出如何将字 3 至字 0 存储在寄存器 209 的一个寄

存器中。位 15 至位 0 包含字 0 的数据元素信息，位 31 至位 16 包含字 1 的数据元素信息，位 47 至位 32 包含数据元素字 2 的信息而位 63 至位 48 包含数据元素字 3 的信息。带符号分组字寄存器表示 513 类似于无符号分组字寄存器表示 512。注意只需要各字数据元素的第 16 位用作符号指示。

[0115] 无符号分组双字寄存器表示 514 示出寄存器 209 如何存储两个双字数据元素。双字 0 存储在寄存器的位 31 至位 0 中。双字 1 存储在寄存器的位 63 至位 32 中。带符号分组双字寄存器表示 515 类似于无符号分组双字寄存器表示 514。注意必要的符号位是双字数据元素的第 32 位。

[0116] 如上所述，寄存器 209 既可用于分组数据又可用于浮点数据。在本发明的这一实施例中，可能要求该单个编程处理器 109 来跟踪诸如 R0212a 的所寻址的寄存器是存储分组数据还是浮点数据。在一个替代实施例中，处理器 109 能跟踪存储在寄存器 209 的各个寄存器中的数据的类型。然后如果例如在浮点数据上试图进行分组加法运算时，这一替代实施例便能产生出错。

[0117] 控制信号格式

[0118] 下面描述处理器 109 用来操作分组数据的控制信号格式的一个实施例。在本发明的一个实施例中，控制信号是表示为 32 位的。解码器 202 可从总线 101 接收控制信号 207。在另一实施例中，解码器 202 也能从高速缓冲存储器 160 接收这种控制信号。

[0119] 图 6a 说明按照本发明的一个实施例指示使用分组数据的控制信号格式。操作字段 OP 601（位 31 至位 26）提供关于由处理器 109 执行的操作的信息，例如分组加法、分组减法等。SRC1602（位 25 至位 20）提供寄存器 209 中的寄存器的源寄存器地址。这一源寄存器包含在控制信号执行中要使用的第一分组数据 Source1。类似地，SRC2603（位 19 至位 14）包含寄存器 209 中的寄存器的地址。这一第二源寄存器包含在该操作执行期间要使用的分组数据 Source2。DEST 605（位 5 至位 0）包含寄存器 209 中的寄存器地址。这一目的地寄存器将存储分组数据操作的结果分组数据 Result。

[0120] 控制位 SZ 610（位 12 与位 13）指示在第一与第二分组数据源寄存器中的数据元素的长度。如果 SZ 610 等于 01_2，则将分组数据格式化为分组字节 401。如果 SZ 610 等于 10_2，则将分组数据格式化为分组字 402。SZ 610 等于 00_2 或 11_2 保留不用，然而在另一实施例中，这些值之一可用来指示分组双字 403。

[0121] 控制位 T 611（位 11）指示是否要以饱和模式进行该操作。如果 T 611 等于 1，则执行饱和操作。如果 T 611 等于 0，则执行非饱和操作。稍后将描述饱和操作。

[0122] 控制位 S 612（位 10）指示使用带符号操作。如果 S 612 等于 1，则执行带符号操作。如果 S 612 等于 0，则执行无符号操作。

[0123] 图 6b 说明按照本发明的一个实施例指示采用分组数据的第二种控制信号格式。这一格式对应于可从 Intel 公司，文献销售处 (P.O.Box7641，Mt.Prospect，IL，60056-7641) 购得的“Pentium 处理器系列用户手册”中描述的通用整数操作码格式。注意将 OP 601、SZ 610、T 611 与 S 612 全部合并成一个大字段。对于某些控制信号，位 3 至 5 为 SRC1602。在一个实施例中，当存在一个 SRC1602 地址时，则位 3 至 5 也对应于 DEST 605。在一个替代实施例中，当存在 SRC2603 地址时，则位 0 至 2 也对应于 DEST 605。对于其它控制信号，如分组移位立即操作，位 3 至 5 表示对操作码字段的扩展。在一个实施例中，这一

扩展允许程序员包含一个具有控制信号的立即值，诸如移位计数值。在一个实施例中，立即值在控制信号后面。这在“Pentium 处理器系列用户手册”附录 F，页 F-1 至 F-3 中有更详细的描述。位 0 至 2 表示 SRC2603。这一通用格式允许寄存器到寄存器、存储器到寄存器、寄存器被存储器、寄存器被寄存器、寄存器被立即数、寄存器到存储器的寻址。同时，在一个实施例中，这一通用格式能支持整数寄存器到寄存器及寄存器到整数寄存器寻址。

[0124] 饱和 / 不饱和的说明

[0125] 如上所述，T 611 指示操作是否可选择地饱和。在允许饱和时，当操作的结果上溢或下溢出数据范围时，结果将被箝位。箝位的意思是如果结果超出范围的最大或最小值便将该结果设定在最大或最小值上。下溢的情况中，饱和将结果箝位到范围中的最低值上，而在上溢的情况中，则到最高值上。表 7 中示出各数据格式的允许范围。

[0126]

数据格式	最小值	最大值
无符号字节	0	255
带符号字节	-128	127
无符号字	0	65535
带符号字	-32768	32767
无符号双字	0	$2^{64}-1$
带符号双字	-2^{63}	$2^{63}-1$

[0127] 表 7

[0128] 如上所述，T 611 指示是否正在执行饱和操作。因此，采用无符号字节数据格式，如果操作结果＝ 258 且允许饱和，则在将结果存储进该操作的目的地寄存器之前将该结果箝位到 255。类似地，如果操作结果＝ -32999 且处理器 109 采用允许饱和的带符号字数据格式，则在将结果存储进操作的目的地寄存器之前将其箝位到 -32768。

[0129] 分组加法

[0130] 分组加法运算

[0131] 本发明的一个实施例能够在执行单元 130 中执行分组加法运算。即，本发明使第一分组数据的各数据元素能单个地加在第二分组数据的各数据元素上。

[0132] 图 7a 说明按照本发明的一个实施例执行分组加法的方法。在步骤 701，解码器 202 解码处理器 109 接收的控制信号 207。从而，解码器 202 解码出 ：分组加法的操作码 ；寄存器 209 中的 SRC1602、SRC2603 及 DEST 605 地址 ；饱和 / 不饱和、带符号 / 无符号及分组数据中的数据元素的长度。在步骤 702，解码器 202 通过内部总线 170 存取寄存器文件 150 中给出 SRC1602 与 SRC2603 地址的寄存器 209。寄存器 209 向执行单元 130 提供分别存储在这些地址上的寄存器中的分组数据 Source1 与 Source2。即，寄存器 209 通过内部总线 170 将分组数据传递给执行单元 130。

[0133] 在步骤 703，解码器 202 启动执行单元 130 去执行分组加法运算。解码器 202 还通过内部总线 170 传递分组数据元素的长度、是否采用饱和及是否采用带符号算术运算。在步骤 704，数据元素的长度确定下面执行哪一步骤。如果分组数据中的数据元素长度为 8 位（字节数据），则执行单元 130 执行步骤 705a。然而，如果分组数据中的数据元素长度为 16

位（字数据），则执行单元 130 执行步骤 705b。在本发明的一个实施例中，只支持 8 位与 16 位数据元素长度分组加法。然而，其它实施例能支持不同的与 / 或其它长度。例如，在一个替代实施例中能附加支持 32 位数据元素长度分组加法。

[0134] 假定数据元素长度为 8 位，则执行步骤 705a。执行单元 130 将 Source1 的位 7 至位 0 加在 SRC2 的位 7 至位 0 上，生成 Result 分组数据的位 7 至位 0。与这一加法并行，执行单元 130 将 Source1 的位 15 至位 8 加在 Source2 的位 15 至位 8 上，产生 Result 分组数据的位 15 至位 8。与这些加法并行，执行单元 130 将 Source1 的位 23 至位 16 加在 Source2 的位 23 至位 16 上，产生 Result 分组数据的位 23 至位 16。与这些加法并行，执行单元 130 将 Source1 的位 31 至位 24 加在 Source2 的位 31 至位 24 上，产生 Result 分组数据的位 31 至位 24。与这些加法并行，执行单元将 Source1 的位 39 至位 32 加在 Source2 的位 39 至位 32 上，产生 Result 分组数据的位 39 至位 32。与这些加法并行，执行单元 130 将 Source1 的位 47 至 40 加在 Source2 的位 47 至位 40 上，产生 Result 分组数据的位 47 至位 40。与这些加法并行，执行单元 130 将 Source1 的位 55 至位 48 加在 Source2 的位 55 至位 48 上，产生 Result 分组数据的位 55 至位 48。与这些加法并行，执行单元 130 将 Source1 的位 63 至位 56 加在 Source2 的位 63 至位 56 上，产生 Result 分组数据的位 63 至位 56。

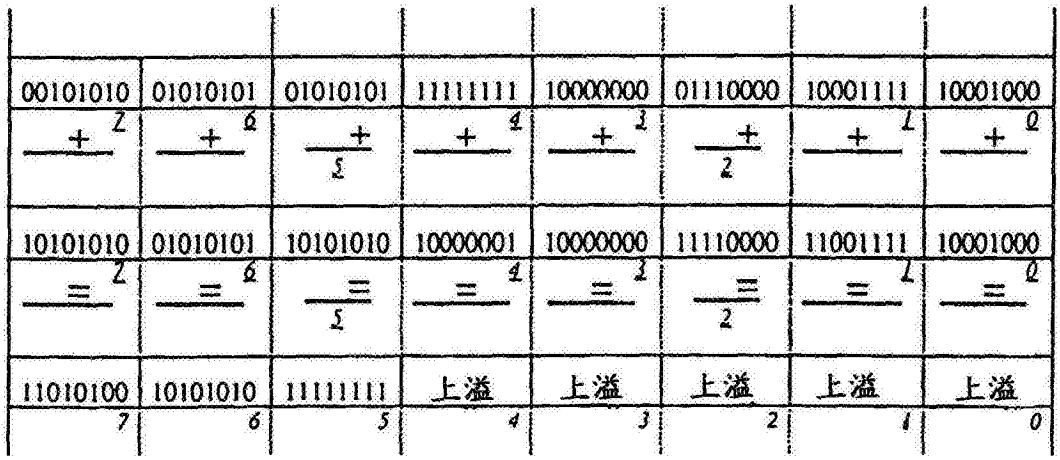
[0135] 假定数据元素长度为 16 位，则执行步骤 705b。执行单元 130 将 Source1 的位 15 至位 0 加在 SRC2 的位 15 至位 0 上，产生 Result 分组数据的位 15 至位 0。与这一加法并行，执行单元 130 将 Source1 的位 31 至位 16 加在 Source2 的位 31 至位 16 上，产生 Result 分组数据的位 31 至位 16。与这些加法并行，执行单元 130 将 Source1 的位 47 至位 32 加在 Source2 的位 47 至位 32 上，产生 Result 分组数据的位 47 至位 32。与这些加法并行，执行单元 130 将 Source1 的位 63 至位 48 加在 Source2 的位 63 至位 48 上，产生 Result 分组数据的位 63 至位 48。

[0136] 在步骤 706，解码器 202 用目的地寄存器的 DEST605 地址启动寄存器 209 中的一个寄存器。从而，将 Result 存储在 DEST605 寻址的寄存器中。

[0137] 表 8a 说明分组加法运算的寄存器表示。第一行的位是 Source1 分组数据的分组数据表示。第二行的位是 Source2 分组数据的分组数据表示。第三行的位是 Result 分组数据的分组数据表示。各数据元素位下面的号码是数据元素号码。例如，Source1 数据元素 0 为 10001000_2。因此，如果该数据元素是 8 位长度（字节数据）且执行的是无符号不饱和的加法，则执行单元 130 产生所示的 Result 分组数据。

[0138] 注意在本发明的一个实施例中，当结果上溢或下溢且运算采用不饱和时，简单地截位结果。即忽略进位位。例如，在表 8a 中，结果数据元素 1 的寄存器表示将是：$10001000_2+10001000_2= 00001000_2$。类似地，对于下溢也截位其结果。这一截位形式使程序员能容易地执行模运算。例如，结果数据元素 1 的公式可表示为：(Source1 数据元素 1+Source2 数据元素 1)mod 256 ＝结果数据元素 1。此外，熟悉本技术的人员会从这一描述理解上溢与下溢可通过在状态寄存器中设置出错位来检测。

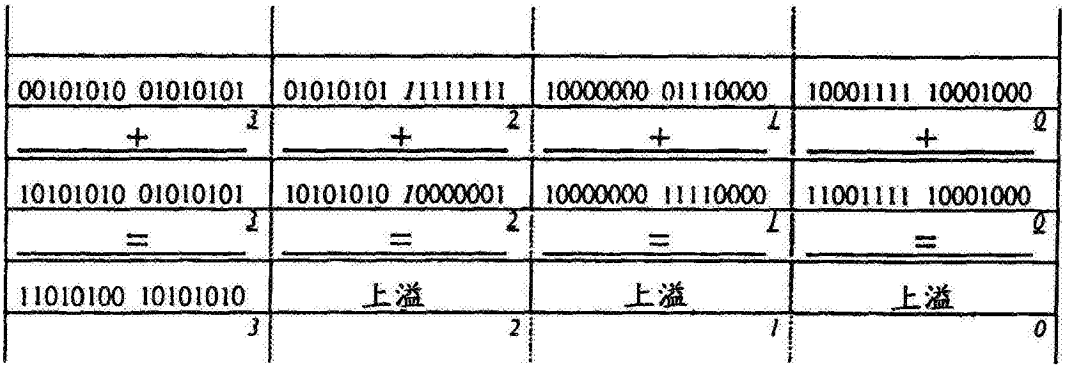
[0139]



[0140] 表 8a

[0141] 表 8b 说明分组字数据加法运算的寄存器表示。因此，如果该数据元素是 16 位长度（字数据）且所执行的是无符号不饱和加法，执行单元 130 产生所示的 Result 分组数据。注意在字数据元素 2 中，来自位 7（见下面强调的位 1）的进位传播到位 8 中，导致数据元素 2 上溢（见下面强调的“上溢”）。

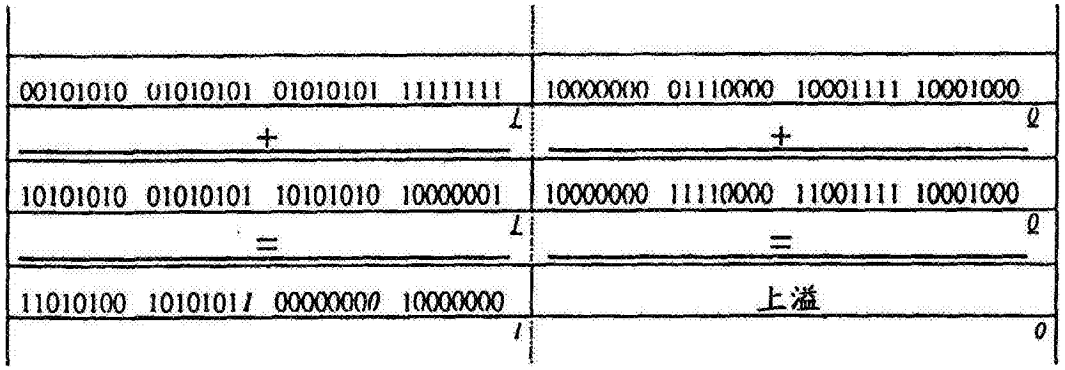
[0142]



[0143] 表 8b

[0144] 表 8c 说明分组双字数据加法运算的寄存器表示。本发明的一个替代实施例支持这一运算。因此，如果该数据元素是 32 位长度（即双字数据）且执行的是无符号不饱和加法，执行单元 130 产生所示的 Result 分组数据。注意来自双字数据元素 1 的位 7 与位 15 的进位分别传播到位 8 与位 16 中。

[0145]



[0146] 表 8c

[0147] 为了更好地说明分组加法与普通加法之间的差别，表 9 中复制了来自上例的数据。然而，在这一情况中，在数据上执行普通加法 (64 位)。注意来自位 7、位 15、位 23、位 31、位 39 及位 47 的进位已分别带到位 8、位 16、位 24、位 32、位 40 及位 48 中。

[0148]

00101010	01010101	01010101	11111111	10000000	01110000	10001111	10001000
±							
10101010	01010101	10101010	10000001	10000000	11110000	11001111	10001000
=							
11010100	10101011	00000000	10000001	00000001	01100001	01011111	00010000

[0149] 表 9

[0150] 带符号 / 不饱和分组加法

[0151] 表 10 说明带符号分组加法的示例，其中的分组数据的数据元素长度是 8 位。不使用饱和。因此，结果能上溢与下溢。表 10 利用与表 8a-8c 及表 9 不同的数据。

[0152]

00101010	01010101	01010101	01111111	00000000	11110000	00001111	10001000
+ 7	+ 6	+ 5	+ 4	+ 3	+ 2	+ 1	+ 0
10101010	01010101	10101010	00000001	00000000	11110000	00001111	10001000
= 7	= 6	= 5	= 4	= 3	= 2	= 1	= 0
11010100	上溢	11111111	上溢	00000000	下溢	00011110	下溢
7	6	5	4	3	2	1	0

[0153] 表 10

[0154] 带符号 / 饱和的分组加法

[0155] 表 11 说明带符号分组加法的示例，其中的分组数据的数据元素长度是 8 位。采用了饱和，因此将上溢箝位到最大值及将下溢箝位到最小值。表 11 使用与表 10 相同的数据。这里将数据元素 0 与数据元素 2 箝位到最小值，而将数据元素 4 与数据元素 6 箝位到最大值。

[0156]

00101010	01010101	01010101	01111111	00000000	11110000	00001111	10001000
+ 7	+ 6	+ 5	+ 4	+ 3	+ 2	+ 1	+ 0
10101010	01010101	10101010	00000001	00000000	11110000	00001111	10001000
= 7	= 6	= 5	= 4	= 3	= 2	= 1	= 0
11010100	01111111	11111111	01111111	00000000	10000000	00011110	10000000
7	6	5	4	3	2	1	0

[0157] 表 11

[0158] 分组减法

[0159] 分组减法运算

[0160] 本发明的一个实施例使在执行单元 130 中能执行分组减法运算。即，本发明使第二分组数据的各数据元素能从第一分组数据的各数据元素中分别地减去。

[0161] 图 7b 说明按照本发明的一个实施例执行分组减法的方法。注意，步骤 710-713 类似于步骤 701-704。

[0162] 在本发明的当前实施例中，只支持 8 位与 16 位数据元素长度分组减法。然而，替代实施例能支持不同的与 / 或其它长度。例如，一个替代实施例能附加支持 32 位数据元素长度分组减法。

[0163] 假定数据元素长度是 8 位，便执行步骤 714a 与 715a。执行单元 130 求 Source2 的位 7 至位 0 的 2 的补码。与求 2 的补码并行，执行单元 130 求 Source2 的位 15 至位 8 的 2 的补码。与这些求 2 的补码并行，执行单元 130 求 Source2 的位 23 至位 16 的 2 的补码。与这些求 2 的补码并行，执行单元 130 求 Source2 的位 31 至位 24 的 2 的补码。与这些求 2 的补码并行，执行单元 130 求 Source2 的位 39 至位 32 的 2 的补码。与这些求 2 的补码并行，执行单元 130 求 Source2 的位 47 至位 40 的 2 的补码。与这些求 2 的补码并行，执行单元 130 求 Source2 的位 55 至位 48 的 2 的补码。与这些求 2 的补码并行，执行单元 130 求 Source2 的位 63 至位 56 的 2 的补码。在步骤 715a，执行单元 130 执行 Source2 的 2 的补码位与 Source1 的位的加法，如对步骤 705a 总的描述。

[0164] 假定数据元素长度是 16 位，则执行步骤 714b 与 715b。执行单元 130 求 Source2 的位 15 至位 0 的 2 的补码。与这一求 2 的补码并行，执行单元 130 求 Source2 的位 31 至位 16 的 2 的补码。与这些求 2 的补码并行，执行单元 130 求 Source2 的位 47 至位 32 的 2 的补码。与这些求 2 的补码并行，执行单元 130 求 Source2 的位 63 至位 48 的 2 的补码。在步骤 715b，执行单元 130 执行 Source2 的 2 的补码位与 Source1 的位的加法，如对步骤 705b 总的描述。

[0165] 注意步骤 714 与 715 为用在本发明的一个实施例中从第二个数减去第一个数的方法。然而，其它形式的减法在本技术中是已知的，不应认为本发明限于采用 2 的求补算术运算。

[0166] 在步骤 716，解码器 202 用目的地寄存器的目的地地址启动寄存器 209。从而，将结果分组数据存储在寄存器 209 的 DEST 寄存器中。

[0167] 表 12 说明分组减法运算的寄存器表示。假定数据元素为 8 位长度（字节数据）且所执行的是无符号不饱和减法，则执行单元 130 产生所示的结果分组数据。

[0168]

00101010	01010101	01010101	01111111	00000000	11110000	00001111	10001000
7	6	5	4	3	2	1	0
10101010	01010101	10101010	00000001	00000000	11110000	00001111	10001000
7	6	5	4	3	2	1	0
下溢	00000000	下溢	01111110	00000000	00000000	00000000	00000000
7	6	5	4	3	2	1	0

[0169] 表 12

[0170] 分组数据加法 / 减法电路

[0171] 图 8 说明按照本发明的一个实施例在分组数据的各个位上执行分组加法与分组减法的电路。图 8 示出修改过的位片加法器 / 减法器 800。加法器 / 减法器 801a-b 能在 Source1 加上或减去来自 Source2 的两位。运算与进位控制 803 向控制线路 809a 传输控制信号启动加法或减法运算。从而，加法器 / 减法器 801a 在 $Source1_i$ 804a 上所接收的位 i 上加上或减去 $Source2_i$ 805a 上接收的位 i，产生在 $Result_i$ 806a 上传输的结果位。Cin 807a-b 与 Cout 808a-b 表示在加法器 / 减法器上经常见到的进位控制电路。

[0172] 从运算与进位控制 803 通过分组数据使能 811 启动位控制 802 来控制 Cin_{i+1}807b 及 $Cout_i$。例如，在表 13a 中，执行无符号分组字节加法。如果加法器 / 减法器 801a 相加 Source1 位 7 与 Source2 位 7，则运算与进位控制 803 将启动位控制 802，停止将进位从位 7 传播到位 8。

[0173]

...	...	...	...	...	...	00001111	10001000
+	+	+	+	+	+	+	+
...	...	...	...	...	...	00001111	10001000
=	=	=	=	=	=	=	=
...	...	...	...	...	...	00011110	上溢
7	6	5	4	3	2	1	0

[0174] 表 13a

[0175] 然而，如果执行的是无符号分组字加法，并且类似地用加法器 / 减法器 801a 将 Source1 的位 7 加在 Source2 的位 7 上，则位控制 802 传播该进位到位 8。表 13b 说明这一结果。对于分组双字加法及非分组加法都允许这一传播。

[0176]

...	...	...	00001111 10001000
+	+	+	+
...	...	...	00001111 10001000
=	=	=	=
...	...	...	00011111 00010000
3	2	1	0

[0177] 表 13b

[0178] 加法器 / 减法器 801a 通过首先反相 Source2i 805a 及加 1 形成 $Source2_i$ 805a 的 2 的补码，从 Source 1_i 804a 减去位 $Source2_i$ 805a。然后，加法器 / 减法器 801a 将这一结果加在 $Source1_i$ 804a 上。位片的 2 的补码运算技术是本技术中众所周知的，熟悉本技术的人员会理解如何设计这一位片的 2 的补码运算电路。注意进位的传播是受位控制 802 及

运算与进位控制 803 控制的。

[0179] 图 9 说明按照本发明一个实施例在分组字节数据上执行分组加法与分组减法的电路。Source1 总线 901 与 Source2 总线 902 分别通过 $Source1_{in}$ 906a-h 与 $Source2_{in}$ 905a-h 将信息信号带到加法器 / 减法器 908a-h 中。从而，加法器 / 减法器 908a 在 Source1 位 7 至位 0 上加上 / 减去 Source2 位 7 至位 0 ；加法器 / 减法器 908b 在 Source1 位 15 至位 8 上加上 / 减去 Source2 位 15 至位 8，等等。CTRL 904a-h 通过分组控制 911 接收来自运算控制 903 的禁止进位传播、允许 / 禁止饱和以及允许 / 禁止带符号 / 无符号算术运算的控制信号。运算控制 903 通过从 CTRL904a-h 接收进位信息并且不将它传播给次最高位加法器 / 减法器 908a-h 而禁止进位传播。从而，运算控制 903 执行运算与进位控制 803 及 64 位分组数据的位控制 802 的运算。给出了图 1-9 中的示例及上述描述，熟悉本技术的人员能建立这一电路。

[0180] 加法器 / 减法器 908a-h 通过结果输出 907a-h 将各种分组加法的结果信息传递给结果寄存器 910a-h。各结果寄存器 910a-h 存储及随后将结果信息传输到 Result 总线 909 上。然后将这一结果信息存储在 DEST605 寄存器地址指定的整数寄存器中。

[0181] 图 10 为按照本发明的一个实施例在分组字数据上执行分组加法与分组减法的电路的逻辑视图。这里，正在执行分组字运算。运算控制 903 启动位 8 与位 7、位 24 与位 23、位 40 与位 39 以及位 56 与位 55 之间的进位传播。从而，示出为虚拟加法器 / 减法器 1008a 的加法器 / 减法器 908a 与 908b 一起工作在分组字数据 Source1 的第一个字 (位 15 至位 0) 上加上 / 减去分组字数据 Source2 的第一个字 (位 15 至位 0) ；示出为虚拟加法器 / 减法器 1008b 的加法器 / 减法器 908c 与 908d 一起工作在分组字数据 Source1 的第二个字 (位 31 至位 16) 上加 / 减分组字数据 Source2 的第二个字 (位 31 至位 16)，等等。

[0182] 虚拟加法器 / 减法器 1008a-d 通过结果输出 1007a-d(组合的结果输出 907a-b、907c-d、907e-f 及 907g-h) 将结果信息传递给虚拟结果寄存器 1010a-d。各虚拟结果寄存器 1010a-d(组合的结果寄存器 910a-b、910c-d、910e-f 及 910g-h) 存储要传递到 Result 总线 909 上的 16 位结果数据元素。

[0183] 图 11 为按照本发明的一个实施例在分组双字数据上执行分组加法与分组减法的电路的逻辑图。运算控制 903 启动位 8 与位 7、位 16 与位 15、位 24 与位 23、位 40 与位 39、位 48 与位 47 及位 56 与位 55 之间的进位传播。从而，示出为虚拟加法器 / 减法器 1108a 的加法器 / 减法器 908a-d 一起工作在组合字数据 Source1 的第一个双字 (位 31 至位 0) 上加上 / 减去组合双字数据 Source2 的第一个双字 (位 31 至位 0) ；示出为虚拟加法器 / 减法器 1108b 的加法器 / 减法器 908e-h 一起工作在组合双字数据 Source1 的第二个双字 (位 63 至位 32) 上加上 / 减去组合双字数据 Source2 的第二个双字 (位 63 至位 32)。

[0184] 虚拟加法器 / 减法器 1108a-b 通过结果输出 1107a-b(组合的结果输出 907a-d 与 907e-h) 将结果信息传递给虚拟结果寄存器 1110a-b。各虚拟结果寄存器 1110a-b(组合的结果寄存器 910a-d 与 910e-h) 存储要传递到 Result 总线 909 上的 32 位结果数据元素。

[0185] 分组乘法

[0186] 分组乘法运算

[0187] 在本发明的一个实施例中，SRC1 寄存器中包含被乘数数据 (Source1)，SRC2 寄存器中包含乘数数据 (Source2)，而 DEST 寄存器中则包含乘积 (结果) 的一部分。即 Source1

的各数据元素独立地乘以 Source2 的相应数据元素。取决于乘法的类型，Result 中将包含积的高阶位或低阶位。

[0188] 在本发明的一个实施例中，支持下述乘法运算：乘法高无符号分组、乘法高带符号分组及乘法低分组。高 / 低表示在 Result 中要包含来自乘积的哪些位。这是必要的，因为两个 N 位数相乘得出具有 2N 位的积。由于各结果数据元素与被乘数及乘数数据元素大小相同，结果只能表示积的一半。高导致较高阶位被作为结果输出。低导致低阶位被作为结果输出。例如，Source1[7:0]×Source2[7:0] 的无符号高分组乘法，在 Result[7:0] 中存储积的高阶位。

[0189] 在本发明的一个实施例中，高 / 低运算修饰符的使用消除了从一个数据元素上溢到下一个较高数据元素的可能性。即，这一修饰符允许程序员选择积中哪些位要在结果中而不考虑上溢。程序员能用分组乘法运算的组合生成完整的 2N 位积。例如，程序员能用 Source1 和 Source2 用乘法高无符号分组运算然后再用相同的 Source1 与 Source2 用乘法低分组运算得出完整的 (2N) 积。提供了乘法高运算因为通常积的高阶位是积的仅有的重要部分。程序员可以不必首先执行任何截位便得到积的高阶位，这种截位对于非分组数据运算通常是需要的。

[0190] 在本发明的一个实施例中，Source2 中的各数据元素可具有一个不同的值。这向程序员提供了对于 Source1 中的各被乘数可具有不同的值作为乘数的灵活性。

[0191] 图 12 为说明按照本发明的一个实施例在分组数据上执行分组乘法运算的方法的流程图。

[0192] 在步骤 1201，解码器 202 解码处理器 109 接收的控制信号 207。从而，解码器 202 解码出：适当乘法运算的运算码；寄存器 209 中的 SRC1602、SRC2603 及 DEST 604 地址；带符号 / 无符号、高 / 低及分组数据中的数据元素的长度。

[0193] 在步骤 1202，解码器 202 通过内部总线 170 存取寄存器文件 150 中给定 SRC1602 与 SRC2603 地址的寄存器 209。寄存器 209 向执行单元 130 提供存储在 SRC1603 寄存器中的分组数据 (Source1) 及存储在 SRC2603 寄存器中的分组数据 (Source2)。即，寄存器 209 通过内部总线 170 将分组数据传递给执行单元 130。

[0194] 在步骤 1130 解码器 202 启动执行单元 130 执行适当的分组乘法运算。解码器 202 还通过内部总线 170 传递用于乘法运算的数据元素的长度及高 / 低。

[0195] 在步骤 1210，数据元素的长度确定下面执行哪一步骤。如果数据元素的长度是 8 位（字节数据），则执行单元 130 执行步骤 1212。然而，如果分组数据中的数据元素长度为 16 位（字数据），则执行单元 130 执行步骤 1214。在一个实施例中，只支持 16 位数据元素长度的分组乘法。在另一实施例中，支持 8 位与 16 位数据元素长度分组乘法。然而，在另一实施例中，还支持 32 位数据元素长度分组乘法。

[0196] 假定数据元素的长度为 8 位，便执行步骤 1212。在步骤 1212 中，执行下述各操作，将 Source1 位 7 至位 0 乘以 Source2 位 7 至位 0 生成 Result 位 7 至位 0。将 Source1 位 15 至 8 乘以 Source2 位 15 至 8 生成 Result 位 15 至 8。将 Source1 位 23 至 16 乘以 Source2 位 23 至 16 生成 Result 位 23 至 16。将 Source1 位 31 至 24 乘以 Source2 位 31 至 24 生成 Result 位 31 至 24。将 Source1 位 39 至 32 乘以 Source2 位 39 至 32 生成 Result 位 39 至 32。将 Source1 位 47 至 40 乘以 Source2 位 47 至 40 生成 REsult 位 47 至 40。将 Source1

位 55 至 48 乘以 Source2 位 55 至 48 生成 Result 位 55 至 48。将 Source1 位 63 至 56 乘以 Source2 位 63 至 56 生成 Result 位 63 至 56。

[0197] 假定数据元素的长度为 16 位，则执行步骤 1214。在步骤 1214 中，执行下述操作。将 Source1 位 15 至 0 乘以 Source2 位 15 至 0 生成 Result 位 15 至 0。将 Source1 位 31 至 16 乘以 Source2 位 31 至 16 生成 Result 位 31 至 16。将 Source1 位 47 至 32 乘以 Source2 位 47 至 32 生成 Result 位 47 至 32。将 Source1 位 63 至 48 乘以 Source2 位 63 至 48 生成 Result 位 63 至 48。

[0198] 在一个实施例中，同时执行步骤 1212 的乘法。然而在另一实施例中，这些乘法是串行执行的。在另一实施例中，这些乘法中一些是同时执行的而一些是串行执行的。这一讨论也同样适用于步骤 1214 的乘法。

[0199] 在步骤 1220 将 Result 存储在 DEST 寄存器中。

[0200] 表 14 示出在分组字数据上的分组乘法无符号高运算的寄存器表示。第一行的位为 Source1 的分组数据表示。第二行的位为 Source2 的数据表示。第三行的位为 Result 的分组数据表示。各数据元素位下面的号码为数据元素号。例如，Source1 数据元素 2 为 1111111100000000_2。

[0201]

11111111 11111111	11111111 00000000	11111111 00000000	00001110 00001000
乘 3	乘 2	乘 1	乘 0
00000000 00000000	00000000 00000001	10000000 00000000	00001110 10000001
=	=	=	=
00000000 00000000	00000000 00000000	01111111 10000000	00000000 11001011
3	2	1	0

[0202] 表 14

[0203] 表 15 示出分组字数据上的乘法高带符号分组运算的寄存器表示。

[0204]

11111111 11111111	11111111 00000000	11111111 00000000	00001110 00001000
乘 3	乘 2	乘 1	乘 0
00000000 00000000	00000000 00000001	10000000 00000000	00001110 10000001
=	=	=	=
00000000 00000000	11111111 11111111	00000000 10000000	00000000 11001011
3	2	1	0

[0205] 表 15

[0206] 表 16 示出分组字数据上的分组乘法低运算的寄存器表示。

[0207]

11111111 11111111	11111111 00000000	11111111 00000000	00001110 00001000
乘 3	乘 2	乘 1	乘 0
00000000 00000000	00000000 00000001	10000000 00000000	00001110 10000001
=	=	=	=
00000000 00000000	11111111 00000000	00000000 00000000	10000010 00001000
3	2	1	0

[0208] 表 16

[0209] 分组数据乘法电路

[0210] 在一个实施例中，可以在与分解的数据上的单一乘法运算相同数目的时钟周期中在多个数据元素上出现乘法运算。为了达到在相同数目的时钟周期中执行，采用了并行性。即同时指示寄存器在数据元素上执行乘法运行。在下面更详细地讨论这一点。

[0211] 图 13 说明用于按照本发明的一个实施例执行分组乘法的电路。运算控制 1300 控制执行乘法的电路。运算控制 1300 处理乘法运算的控制信号并具有下述输出 :高 / 低使能 1380 ;字节 / 字使能 1381 及符号使能 1382，高 / 低使能 1380 标识结果中要包含积的高阶位还是低阶位。字节 / 字使能 1381 标识要执行的是字节分组数据还是字分组数据乘法运算。符号使能 1382 指示是否应采用带符号乘法。

[0212] 分组字乘法器 1301 同时乘四个字数据元素。分组字节乘法器 1302 乘 8 个字节数据元素。分组字乘法器 1301 及分组字节乘法器都具有下述输入 :Source1[63:0]1331、Source[63:0]1333、符号使能 1382 及高 / 低使能 1380。

[0213] 分组字乘法器 1301 包含 4 个 16×16 乘法器电路 :16×16 乘法器 A1310、16×16 乘法器 B1311、16×16 乘法器 C 1312 及 16×16 乘法器 D1313。16×16 乘法器 A 1310 具有输入 Source1[15:0] 与 Source2[15:0]，16×16 乘法器 B 1311 具有输入 Source1[31:16] 与 Source2[31:16]，16×16 乘法器 C 1312 具有输入 Source1[47:32] 与 Source2[47:32]，16×16 乘法器 D 1313 具有输入 Source1[63:48] 与 Source2[63:48]。各 16×16 乘法器耦合在符号使能 1382 上。各 16×16 乘法器产生 32 位积。对于各乘法器，多路复用器 (分别为 Mx01350、Mx11351、Mx21352 及 Mx31353) 接收 32 位结果。取决于高 / 低使能 1380 的值，各多路复用器输出积的 16 个高阶位或 16 个低阶位。将四个多路复用器的输出组合成一个 64 位结果。这一结果可选地存储在结果寄存器 11371 中。

[0214] 分组字节乘法器 1302 包含 8 个 8×8 乘法器电路 :8×8 乘法器 A1320 至 8×8 乘法器 H 1337。各 8×8 乘法器具有来自各 Source1[63:0]1331 及 Source2[63:0]1333 的 8 位输入。例如 8×8 乘法器 A 1320 具有输入 Source1[7:0] 与 Source2[7:0]，而 8×8 乘法器 H 1327 具有输入 Source1[63:56] 与 Source2[63:56]。各 8×8 乘法器耦合在符号使能 1382 上。各 8×8 乘法器产生一个 16 位积。对于各乘法器，多路复用器 (诸如 Mx41360 与 Mx111367) 接收 16 位结果。取决于高 / 低使能 1380 的值。各多路复用器输出积的 8 个高阶位或 8 个低阶位。将 8 个多路复用器的输出组合成一个 64 位结果。可选地将这一结果存储在结果寄存器 21372 中。取决于该运算要求的数据元素的长度，字节 / 字使能 1381 启动特定的结果寄存器。

[0215] 在一个实施例中，通过制造能乘两个8×8数或一个16×16数两者的电路，减少用于实现乘法的面积。即将两个8×8乘法器及一个16×16乘法器组合成一个8×8与16×16乘法器。运算控制1300将允许乘法的适当长度。在这一实施例中，可以缩小乘法器所使用的物理面积，然而它将难于执行分组字节乘法及分组字乘法。在另一支持分组双字乘法的实施例中，一个乘法器能执行四个8×8乘法、两个16×16乘法或一个32×32乘法。

[0216] 在一个实施例中，只提供分组字乘法运算。在这一实施例中，不包含分组字节乘法器1302及结果寄存器21372。

[0217] 在指令集中包含上述分组乘法运算的优点

[0218] 从而上述分组乘法指令提供了Source1中各数据元素乘以Source2中其对应数据元素的独立乘法。当然，要求Source1各元素乘以同一个数算法可通过将该相同的数存储在Source2的各元素中来执行。此外，这一乘法指令通过断开进位链防止上溢；借此解除程序员的这一责任，不再需要准备数据来防止上溢的指令，并得出更健壮的代码。

[0219] 反之，不支持这一指令的先有技术通用处理器需要通过分解数据元素、执行乘法及随后组装结果供进一步分组处理，来执行这一运算。这样，处理器109便能利用一条指令并行地将分组数据的不同数据元素乘以不同的乘数。

[0220] 典型的多媒体算法执行大量的乘法运算。从而，通过减少执行这些乘法运算所需的指令数，便能提高这些多媒体算法的性能。从而，通过在处理器109所支持的指令集中提供这一乘法指令，处理器109便能在较高的性能级上执行需要这一功能的算法。

[0221] 乘－加／减

[0222] 乘－加／减运算

[0223] 在一个实施例中，采用下面表17a与表17b中所示的单一乘－加指令执行两个乘－加运算。表17a示出所公开的乘－加指令的简化表示，而表17b示出公开的乘－加指令的位级示例。

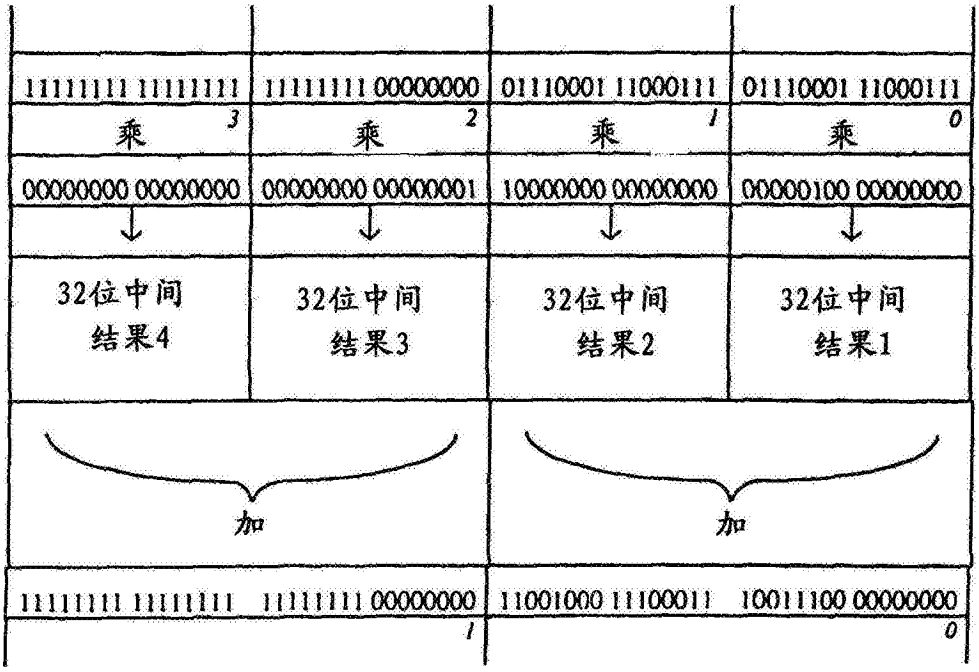
[0224]

Multiply-Add Source1, Source2

A_1	A_2	A_3	A_4	(源1)
B_1	B_2	B_3	B_4	(源2)
		=		
$A_1B_1+A_2B_2$		$A_3B_3+A_4B_4$		(结果1)

[0225] 表17a

[0226]



[0227] 表 17b

[0228] 除了用“减”替换“加”之外，乘 - 减运算与乘 - 加运算相同。表 12 中示出执行两个乘 - 减运算的示例乘 - 减指令的运算。

[0229]

Multiply-Subtract Source1, Source2

A_1	A_2	A_3	A_4	(源1)
B_1	B_2	B_3	B_4	(源2)
=				
$A_1B_1-A_2B_2$		$A_3B_3-A_4B_4$		(结果1)

[0230] 表 12

[0231] 在本发明的一个实施例中，SRC1 寄存器包含分组数据 (Source1)，SRC2 寄存器包含分组数据 (Source2)，而 DEST 寄存器将包含在 Source 1 与 Source2 上执行乘 - 加或乘 - 减指令的结果 (Result)。在乘 - 加或乘 - 减指令的第一步中，Source1 的各数据元素独立地乘以 Source2 的对应数据元素以生成一组对应的中间结果。在执行乘 - 加指令时，将这些中间结果成对相加，生成两个数据元素，将它们作为 Result 的数据元素存储。相反，在执行乘 - 减指令时，成对相减这些中间结果生成两个数据元素，将它们作为 Result 的数据元素存储。

[0232] 替代实施例可改变中间结果的数据元素与 / 或 Result 中的数据元素中的位数。此外，替代实施例可改变 Source1、Source2 及 Result 中的数据元素数。例如，如果 Source1 与 Source2 各有 8 个数据元素，可将乘 - 加 / 减指令实现为产生带有 4 个数据元素 (Result 中的各数据元素表示两个中间结果的相加)、两个数据元素 (结果中的各数据元素表示四个中间结果的相加) 等的 Result。

[0233] 图 14 为说明按照本发明的一个实施例在分组数据上执行乘 - 加与乘 - 减的方法的流程图。

[0234] 在步骤 1401，解码器 202 解码处理器 109 接收的控制信号 207。从而，解码器 202

解码出：乘－加或乘－减指令的操作码。

[0235] 在步骤1402，解码器202通过内部总线170存取给出SRC1602与SRC2603地址的寄存器文件150中的寄存器209。寄存器209向执行单元130提供存储在SRC1602寄存器中的分组数据(Source1)及存储在SRC2603寄存器中的分组数据(Source2)。即寄存器209通过内部总线170将分组数据传递给执行单元130。

[0236] 在步骤1403，解码器202启动执行单元130去执行指令。如果该指令为乘－加指令，流程进到步骤1414。然而，如果该指令为乘－减指令，流程便进到步骤1415。

[0237] 在步骤1414中，执行以下运算。将Source1位15至0乘以Source2位15至0生成第一32位中间结果（中间结果1）。将Source1位31至16乘以Source2位31至16生成第二32位中间结果（中间结果2）。将Source1位47至32乘以Source2位47至32生成第三32位中间结果（中间结果3）。将Source1位63至48乘以Source2位63至48生成第四32位中间结果（中间结果4）。将中间结果1加到中间结果2上生成Result的位31至0，并将中间结果3加到中间结果4上生成Result的位63至32。

[0238] 步骤1415与步骤1414相同，除了将中间结果1与中间结果2相减以生成Result的位31至0，中间结果3与中间结果4相减以生成Result的位63至32。

[0239] 不同实施例可执行串行、并行或串行与并行运算的某种组合的乘与加／减。

[0240] 在步骤1420，将Result存储在DEST寄存器中。

[0241] 分组数据乘－加／减电路

[0242] 在一个实施例中，能和在分解的数据上的单个乘法一样的相同数目的时钟周期中在多个数据元素上出现各乘－加与乘－减指令。为了达到在相同数目的时钟周期中执行，采用了并行性。即指示寄存器在数据元素上同时执行乘－加或乘－减运算。下面更详细地讨论这一点。

[0243] 图15说明按照本发明的一个实施例在分组数据上执行乘－加与／或乘－减运算的电路。操作控制1500处理乘－加与乘－减指令的控制信号。操作控制1500在使能1580上输出信号来控制分组乘－加法器／减法器1501。

[0244] 分组乘－加法器／减法器1501具有下列输入：Source1[63:0]1531、Source2[63:0]1533及使能1580。分组乘－加法器／减法器1501包含4个16×16乘法器电路：16×16乘法器A 1510、16×16乘法器B 1511、16×16乘法器C 1512及16×16乘法器D 1513。16×16乘法器A 1510具有输入Source1[15:0]及Source2[15:0]。16×16乘法器B 1511具有输入Source1[31:16]及Source2[31:16]。16×16乘法器C 1512具有输入Source1[47:32]及Source2[47:32]。16×16乘法器D 1513具有输入Source1[63:48]及Source2[63:48]，16×16乘法器A 1510及16×16乘法器B 1511生成的32位中间结果由虚拟加法器／减法器1550接收，而16×16乘法器C 1512及16×16乘法器D 1513生成的32位中间结果则由虚拟加法器／减法器1551接收。

[0245] 基于当前指令是乘－加还是乘－减指令，虚拟加法器／减法器1550与1551或加或减它们各自的32位输入。虚拟加法器／减法器1550的输出（即Result的位31至0）及虚拟加法器／减法器1551的输出（即Result的位63至32）组合成64位Result并被传递给结果寄存器1571。

[0246] 在一个实施例中，虚拟加法器／减法器1551及1550是以类似虚拟加法器／减法

器 1108b 及 1108a 的方式实现的（即各虚拟加法器 / 减法器 1551 及 1550 是由带有适当传播延时的 4 个 8 位加法器组成的）。然而，替代实施例能用各种方式实现虚拟加法器 / 减法器 1551 及 1550。

[0247] 为了在分解数据上操作的先有技术处理器上执行这些乘 - 加或乘 - 减指令的等效指令，将需要四次独立的 64 位乘法运算与两次 64 位加或减法运算以及必要的加载与存储操作。这浪费用于 Source1 与 Source2 的高于位 16 及 Result 的高于位 32 的数据线及电路。并且，这种先有技术处理器生成的整个 64 位结果对于程序员可能是无用的。因此，程序员将必须截断各结果。

[0248] 在指令集中包含上述乘 - 加运算的优点

[0249] 上述乘 - 加 / 减指令可用于若干目的。例如，乘 - 加指令可用在复数乘法及值的相乘与累加。稍后要描述利用乘 - 加指令的若干算法。

[0250] 从而通过在处理器 109 支持的指令集中加入上述乘 - 加与 / 或乘 - 减指令，便能用比缺少这些指令的先有技术通用处理器较少的指令执行许多功能。

[0251] 分组移位

[0252] 分组移位操作

[0253] 在本发明的一个实施例中，SCR1 寄存器中包含要移位的数据 (Source1)，SRC2 寄存器中包含表示移位计数的数据 (Source2)，而 DEST 寄存器中将包含移位的结果 (Result)。即 Source1 中的各数据元素独立地移位该移位计数。在一个实施例中，将 Source2 解释为一个无符号的 64 位标量。在另一实施例中，Source2 为分组数据并包含 Source1 中各对应数据元素的移位计数。

[0254] 在本发明的一个实施例中，支持算术移位与逻辑移位两者。算术移位将各数据元素的位向下移位指定的数目，并用符号位的初始值填充各数据元素的高阶位。对于分组字节数据大于 7 的移位计数、对于分组字数据，大于 15 的移位计数、或对于分组双字大于 31 的移位计数导致用符号位的初始值来填充各 Result 数据元素。逻辑移位可用向上或向下移位来操作。在逻辑向右移位中，用 0 来填充各数据元素的高阶位。在逻辑向左移位中，用零来填充各数据元素的最低位。

[0255] 在本发明的一个实施例中，对分组字节与分组字支持算术向右移位、逻辑向右移位及逻辑向左移位。在本发明的另一实施例中，对分组双字也支持这些操作。

[0256] 图 16 为说明按照本发明的一个实施例在分组数据上执行分组移位操作的方法的流程图。

[0257] 在步骤 1601，解码器 202 解码处理器 109 所接收的控制信号 207。从而，解码器 202 解码出 ：用于适当移位操作的操作码 ；寄存器 209 中的 SRC1602、SRC2603 及 DEST 605 地址 ；饱和 / 不饱和（对移位操作不一定需要）、带符号 / 无符号（也不一定需要）及分组数据中的数据元素的长度。

[0258] 在步骤 1602，解码器 202 通过内部总线 170 存取寄存器文件 150 中给出 SRC1602 及 SRC2603 地址的寄存器 209。寄存器 209 向执行单元 130 提供 SRC1602 寄存器中所存储的分组数据 (Source1) 及存储在 SRC2603 寄存器中的标量移位计数 (Source2)。即寄存器 209 通过内部总线 170 将分组数据传递给执行单元 130。

[0259] 在步骤 1603，解码器 202 启动执行单元 130 去执行适当的分组移位操作。解码器

202 还通过内部总线 170 传递数据元素长度、移位操作类型及移位方向（对于逻辑移位）。

[0260] 在步骤 1610，数据元素的长度确定下面要执行哪一步骤。如果数据元素的长度是 8 位（字节数据），则执行单元 130 执行步骤 1612。然而，如果分组数据中的数据元素的长度为 16 位（字数据），则执行单元 130 执行步骤 1614。在一个实施例中，只支持 8 位与 16 位数据元素长度的分组移位。然而，在另一实施例中，也支持 32 位数据元素长度的分组移位。

[0261] 假定数据元素的长度为 8 位，便执行步骤 1612。在步骤 1612 中，执行下述操作。将 Source1 位 7 至 0 移位移位计数 (Source2 位 63 至 0) 生成 Result 位 7 至 0。Source1 位 15 至 8 移位移位计数生成 Result 位 15 至 8。将 Source1 位 23 至 16 移位移位计数生成 Result 位 23 至 16。将 Source1 位 31 至 24 移位移位计数生成 Result 位 31 至 24。将 Source1 位 39 至 32 移位移位计数生成 Result 位 39 至 32。将 Source1 位 47 至 40 移位移位计数生成 Result 位 47 至 40。将 Source1 位 55 至 48 移位移位计数生成 Result 位 55 至 48。将 Source1 位 63 至 56 移位移位计数生成 Result 位 63 至 56。

[0262] 假定数据元素的长度为 16 位，便执行步骤 1614。在步骤 1614 中执行下述操作。将 Source1 位 15 至 0 移位移位计数生成 Result 位 15 至 0。将 Source1 位 31 至 16 移位移位计数生成 Result 位 31 至 16。将 Source1 位 47 至 32 移位移位计数生成 Result 位 47 至 32。将 Source1 位 63 至 48 移位移位计数生成 Result 位 63 至 48。

[0263] 在一个实施例中，步骤 1612 的移位是同时执行的。然而，在另一实施例中，这些移位是串行执行的。在另一实施例中这些移位中一些是同时执行的而一些是串行执行的。这一讨论同样适用于步骤 1614 的移位。

[0264] 在步骤 1620，将 Result 存储在 DEST 寄存器中。

[0265] 表 19 说明字节分组算术向右移位操作的寄存器表示。第一行的位为 Source1 的分组数据表示。第二行的位为 Source2 的数据表示。第三行的位为 Result 的分组数据表示。各数据元素位下面的数字为数据元素号。例如，Source1 数据元素 3 为 10000000_2。

[0266]

00101010	01010101	01010101	11111111	10000000	01110000	10001111	10001000
移位 7	移位 6	移位 5	移位 4	移位 3	移位 2	移位 1	移位 0
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000100
=	=	=	=	=	=	=	=
00000010	00000101	00000101	11111111	11110000	00000111	11111000	11111000
7	6	5	4	3	2	1	0

[0267] 表 19

[0268] 表 20 说明分组字节数据上的分组逻辑向右移位操作的寄存器表示

[0269]

00101010	01010101	01010101	11111111	10000000	01110000	10001111	10001000
移位 7	移位 6	移位 5	移位 4	移位 3	移位 2	移位 1	移位 0
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000011
=	=	=	=	=	=	=	=
00000101	00001010	00001010	00011111	00010000	00001110	00010001	00010001
7	6	5	4	3	2	1	0

[0270] 表 20

[0271] 表 21 说明分组字节数据上的分组逻辑向左移位操作的寄存器表示。

[0272]

00101010	01010101	01010101	11111111	10000000	01110000	10001111	10001000
移位 7	移位 6	移位 5	移位 4	移位 3	移位 2	移位 1	移位 0
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000011
=	=	=	=	=	=	=	=
01010000	10101000	10101000	11111000	00000000	10000000	01111000	01000000
7	6	5	4	3	2	1	0

[0273] 表 21

[0274] 分组数据移位电路

[0275] 在一个实施例中，在与分解的数据上的单个移位操作相同数目的时钟周期中可在多个数据元素上出现移位操作。为了达到在相同数目的时钟周期中执行，采用了并行性。即指示寄存器同时在数据元素上执行移位操作，下面对此作更详细的讨论。

[0276] 图 17 说明按照本发明的一个实施例在分组数据的各个字节上执行分组移位的电路。图 17 示出经修改的字节片移位电路、字节片级 i1799 的使用。各字节片（除最高位数据元素字节片外）包含一个移位单元及位控制。最高位数据元素字节片只需一个移位单元。

[0277] 移位单元 i1711 及移位单元 i+11771 各允许将来自 Source1 的 8 位移位该移位计数。在一个实施例中，各移位单元象已知的 8 位移位电路那样操作。各移位单元具有一个 Source1 输入、一个 Source2 输入、一个控制输入、一个下一级信号、一个上一级信号及一个结果输出。因此，移位单元 i1711 具有 $Source1_i$ 1731 输入、Source2[63:0]1733 输入、控制 i1701 输入、下一级 i1713 信号、上一级 i1712 输入及存储在结果寄存器 i 1751 中的结果。因此，移位单元 i+11771 具有 $Source1_{i+1}$1732 输入、Source2[63:0]1733 输入、控制 i+11702 输入、下一级 i+11773 信号、上一级 i+11772 输入及存储在结果寄存器 i+11752 中的结果。

[0278] Source1 输入通常是 Source1 的一个 8 位部分。8 位表示数据元素的最小类型，一个分组字节数据元素。Source2 输入表示移位计数。在一个实施例中，各移位单元从 Source2[63:0]1733 接收相同的移位计数。操作控制 1700 传输控制信号启动各移位单元执行要求的移位。控制信号是从移位类型（算术 / 逻辑）及移位方向确定的。下一级信号是从该移位单元的位控制接收的。取决于移位的方向（左 / 右），在下一级信号上移位单元将最高位移出 / 进。类似地，取决于移位的方向（右 / 左），各移位单元在上一级信号上将最

低位移出 / 进。上一级信号是从前一级的位控制单元接收的。结果输出表示移位单元在其上操作的 Source1 的部分上的移位操作的结果。

[0279] 位控制 i1720 是从操作控制 1700 通过分组数据启动 i1706 启动的。位控制 i1720 控制下一级 i1713 及上一级 i+11772。例如，假定移位单元 i1711 负责 Source1 的 8 个最低位，而移位单元 i+11771 负责 Source1 的下一个 8 位。如果执行在分组字节上的移位，位控制 i1720 将不允许来自移位单元 i+11771 的最低位与移位单元 i1711 的最高位连通。然而，执行分组字上的移位时，则位控制 i1720 将允许来自移位单元 i+11771 的最低位与移位单元 i1711 的最高位连通。

[0280] 例如，在表 22 中，执行分组字节算术向右移位。假定移位单元 i+11771 在数据元素 1 上操作，而移位单元 i1711 在数据元素 0 上操作。移位单元 i+11771 将其最低位移出。然而操作控制 1700 将导致位控制 i1720 停止从上一级 i+11721 接收的该位传播到下一级 i1713。反之，移位单元 i1711 以符号位填充最高阶位 Source1[7]。

[0281]

...	...	...	...	...	...	00001110	10001000
Shift 7	Shift 6	Shift 5	Shift 4	Shift 3	Shift 2	移位 1	移位 0
...	...	...	...	...	...	...	00000001
=	=	=	=	=	=	=	=
...	...	...	...	...	...	00001111	01000100
7	6	5	4	3	2	1	0

[0282] 表 22

[0283] 然而，如果执行分组字算术移位，则将移位单元 i+11771 的最低位传递给移位单元 i 1711 的最高位。表 23 示出这一结果。这一传递对于分组双字移位同样允许。

[0284]

...	...	...	00001110 10001000
Shift 3	Shift 2	Shift 1	移位 0
...	...	...	00000001
=	=	=	=
...	...	...	00000111 01000100
3	2	1	0

[0285] 表 23

[0286] 各移位单元可选地耦合在结果寄存器上。结果寄存器临时存储移位操作的结果直到可将整个结果 Result[63:0]1760 传输给 DEST 寄存器为止。

[0287] 对于一个完整的 64 位分组移位电路，使用 8 个移位单元及 7 个位控制单元。这一电路也能用来执行 64 位非分组数据上的移位，从而使用同一电路来执行非分组移位操作与分组移位操作。

[0288] 在指令集中包含上述移位操作的优点

[0289] 上述分组移位指令导致 Source1 的各元素移位指定的移位计数。通过在指令集中加入这一指令，便可使用单一指令移位一个分组数据的各元素。反之，不支持这一操作的先

有技术通用处理器必须执行许多指令来分解 Source1，单个地移位各分解的数据元素，然后将结果组装成分组数据格式供进一步分组处理。

[0290] 传送操作

[0291] 传送操作向或从寄存器 209 传送数据。在一个实施例中，SRC2603 为包含源数据的地址而 DEST 605 则是数据要传送到的地址。在这一实施例中，不使用 SRC1602。在另一实施例中，SRC1602 等于 DEST 605。

[0292] 为了说明传送操作的目的，将寄存器与存储单元区分开。寄存器在寄存器文件 150 中而存储器则可以是诸如在高速缓冲存储器 160、主存储器 104、ROM 106、数据存储设备 107 中。

[0293] 传送操作可将数据从存储器传送到寄存器 209，从寄存器 209 到存储器，及从寄存器 209 中的一个寄存器到寄存器 209 中的第二寄存器。在一个实施例中，分组数据是存储在与存储整数数据不同的寄存器中的。在这一实施例中，传送操作能将数据从整数寄存器 201 传送到寄存器 209。例如，在处理器 109 中，如果分组数据存储在寄存器 209 中而整数数据存储在整数寄存器 201 中，则传送指令能用来将数据从整数寄存器 201 传送到寄存器 209，反之亦然。

[0294] 在一个实施例中，当为传送指定了一个存储器地址时，在该存储单元的 8 个字节数据（该存储单元包含最低字节）被加载到寄存器 209 中的一个寄存器或从该寄存器存储到指定存储器单元中。当指定寄存器 209 中的一个寄存器时，便将该寄存器的内容传送到或加载自寄存器 209 中的第二寄存器或者从第二寄存器向指定寄存器加载。如果整数寄存器 201 的长度为 64 位，并指定了一个整数寄存器，则将该整数寄存器中的 8 个字节数据加载到寄存器 209 中的寄存器或从该寄存器存储到指定的整数寄存器中。

[0295] 在一个实施例中，整数是表示为 32 位的。在执行从寄存器 209 到寄存器 201 的传送操作时，则只将分组数据的低 32 位传送到指定的整数寄存器。在一个实施例中，将高阶 32 位变成 0。类似地，当执行从整数寄存器 201 到寄存器 209 的传送时，只加载寄存器 209 中的一个寄存器的低 32 位。在一个实施例中，处理器 109 支持寄存器 209 中的寄存器与存储器之间的 32 位传送操作。在另一实施例中，只在分组数据的高阶 32 位上执行只有 32 位的传送。

[0296] 组装操作

[0297] 在本发明的一个实施例中，SRC1602 寄存器包含数据 (Source1)，SRC2603 寄存器包含数据 (Source2)，而 DEST 605 寄存器将包含操作的结果数据 (Result)。这便是，将 Source1 的部分与 Source2 的部分组装在一起生成 Result。

[0298] 在一个实施例中，组装操作通过将源分组字（或双字）的低阶字节（或字）组装进 Result 的字节（或字）中而将分组字（或双字）转换成分组字节（或字）。在一个实施例中，组装操作将 4 个分组字转换成分组双字。这一操作可选择地以带符号数据执行。此外，这一操作可以选择地以饱和执行。在一个替代实施例中，加入了在各数据元素的高阶部分上操作的附加的组装操作。

[0299] 图 18 为说明按照本发明的一个实施例在分组数据上执行组装操作的方法的流程图。

[0300] 在步骤 1801，解码器 202 解码处理器 109 接收的控制信号 207。从而，解码器 202 解

码出 :适当的组装操作的操作码 ;寄存器 209 中的 SRC1602、SRC2603 及 DEST 605 地址 ;饱和 / 不饱和,带符号 / 无符号及分组数据中的数据元素长度。如上所述,SRC1602(或 SRC2603)可用作 DEST 605。

[0301] 在步骤 1802,解码器 202 通过内部总线 170 存取寄存器文件 150 中给定 SRC1602 与 SRC2603 地址的寄存器 209。寄存器 209 向执行单元 130 提供存储在 SRC1602 寄存器中的分组数据 (Source1) 及存储在 SRC2603 寄存器中的分组数据 (Source2)。即寄存器 209 通过内部总线 170 将分组数据传递给执行单元 130。

[0302] 在步骤 1803,解码器 202 启动执行单元 130 去执行适当的组装操作。解码器 202 还通过内部总线 170 传递饱和及 Source1 和 Source2 中的数据元素的长度。可选用饱和来使结果数据元素中的数据的值成为最大。如果 Source1 或 Source2 中的数据元素的值大于或小于 Result 的数据元素所能表示的值的范围,则将对应的结果数据元素设定在其最高或最低值上。例如,如果 Source1 与 Source2 的字数据元素中的带符号值小于 0x80(或对于双字 0x8000),则将结果字节 (或字) 数据元素箝位到 0x80(或对于双字 0x8000) 上。如果 Source1 与 Source2 的字数据元素中的带符号值大于 0x7F(或对于双字 0x7FFF),则将结果字节 (或字) 数据元素箝位到 0x7F(或 0x7FFF) 上。

[0303] 在步骤 1810,数据元素的长度确定下面要执行哪一步骤。如果数据元素的长度为 16 位 (分组字 402 数据),则执行单元 130 执行步骤 1812。然而,如果分组数据中的数据元素的长度为 32 位 (分组双字 403 数据),则执行单元 130 执行步骤 1814。

[0304] 假定源数据元素的长度为 16 位,便执行步骤 1812。在步骤 1812 中,执行以下操作。Source1 位 7 至 0 为 Result 位 7 至 0。Source1 位 23 至 16 为 Result 位 15 至 8。Source1 位 39 至 32 为 Result 位 23 至 16。Source1 位 63 至 56 为 Result 位 31 至 24。Source2 位 7 至 0 为 Result 位 39 至 32。Source2 位 23 至 16 为 Result 位 47 至 40。Source2 位 39 至 32 为 Result 位 55 至 48。Source2 位 63 至 56 为 Result 位 31 至 24。如果设定饱和,则测试各字的高阶位来确定是否应箝位 Result 数据元素。

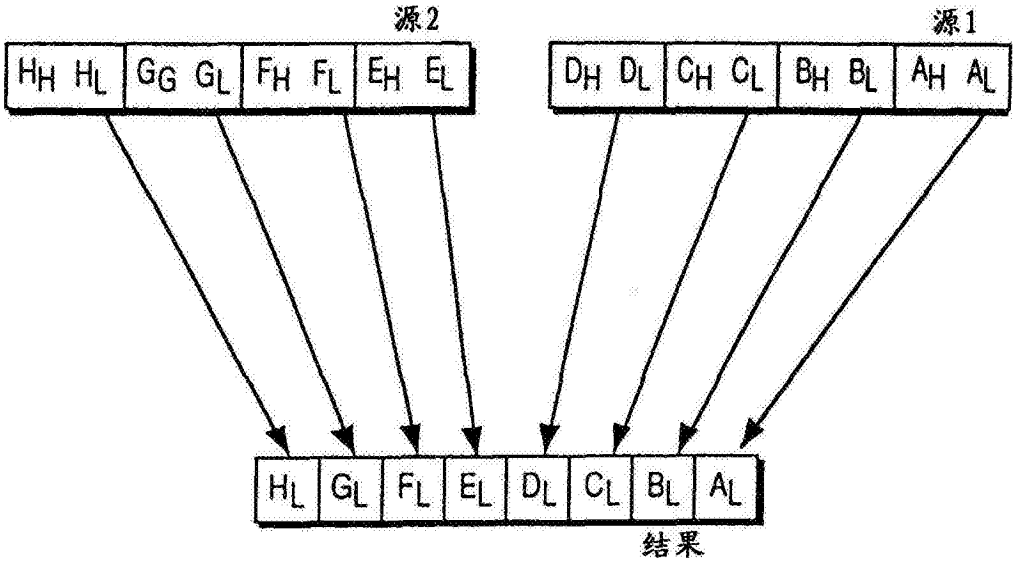
[0305] 假定源数据元素的长度为 32 位,便执行步骤 1814。在步骤 1814 中,执行下述操作。Source1 位 15 至 0 为 Result 位 15 至 0。Source1 位 47 至 32 为 Result 位 31 至 16。Source2 位 15 至 0 为 Result 位 47 至 32。Source2 位 47 至 32 为 Result 位 63 至 48。如果设定了饱和,则测试各双字的高阶位来确定是否应将 Result 数据元素箝位。

[0306] 在一个实施例中,同时执行步骤 1812 的组装。然而在另一实施例中,串行执行这一组装。在另一实施例中,一些组装是同时执行的而一些是串行执行的。这一讨论也适用于步骤 1814 的组装。

[0307] 在步骤 1820,将 Result 存储在 DEST 605 寄存器中。

[0308] 表 24 说明组装字操作的寄存器表示。加下标的 Hs 与 Ls 分别表示 Source1 与 Source2 中的各 16 位数据元素的高与低阶位。例如 AL 表示 Source1 中的数据元素 A 的低阶 8 位。

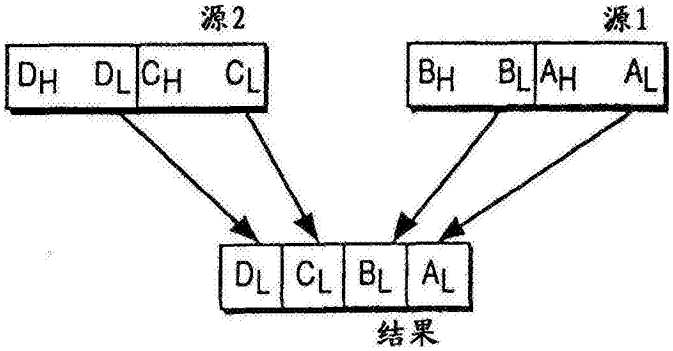
[0309]



[0310] 表 24

[0311] 表 25 说明组装双字操作的寄存器表示，其中加下标的 Hs 与 Ls 分别表示 Source1 与 Source2 中的各 32 位数据元素的高低阶位。

[0312]



[0313] 表 25

[0314] 组装电路

[0315] 在本发明的一个实施例中，为了达到组装操作的高效执行，采用了并行性。图 19a 与 19b 示出按照本发明的一个实施例在分组数据上执行组装操作的电路。该电路能有选择地执行带饱和的组装操作。

[0316] 图 19a 与 19b 的电路包括操作控制 1900、结果寄存器 1952、结果寄存器 1953、8 个 16 位到 8 位测试饱和电路及 4 个 32 位到 16 位测试饱和电路。

[0317] 操作控制 1900 接收来自解码器 202 的信息来启动组装操作。操作控制 1900 使用饱和值为各测试饱和电路启动饱和测试。如果源分组数据的长度为字分组数据 503，则操作控制 1900 设定输出使能 1931。这便启动结果寄存器 1952 的输出。如果源分组数据的长度为双字分组数据 504，则操作控制 1900 设定输出使能 1932。这便启动输出寄存器 1953 的输出。

[0318] 各测试饱和电路能有选择地测试饱和。如果禁止饱和测试，则各测试饱和电路只将低阶位传递到结果寄存器中对应的位置上。如果允许测试饱和，则各测试饱和电路测试高阶位来确定是否应箝位结果。

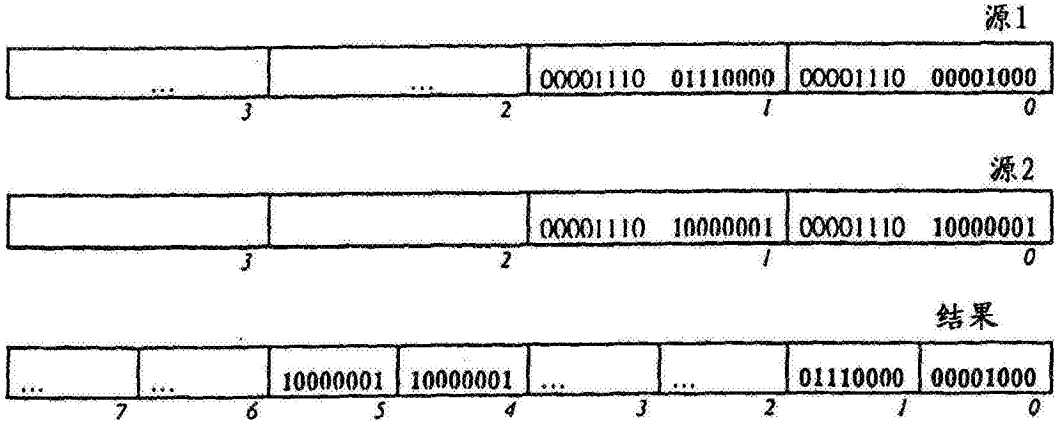
[0319] 测试饱和 1910 至测试饱和 1917 具有 16 位输入与 8 位输出。8 位输出为输入的

低 8 位，或可选地是一个箝位的值 (0x80、0x7F、或 0xFF)。测试饱和 1910 接收 Source1 位 15 至 0 并向结果寄存器 1952 输出位 7 至 0。测试饱和 1911 接收 Source1 位 31 至 16 并向结果寄存器 1952 输出位 15 至 8。测试饱和 1912 接收 Source1 位 47 至 32 并向结果寄存器 1952 输出位 23 至 16。测试饱和 1913 接收 Source1 位 63 至 48 并向结果寄存器 1952 输出位 31 至 24。测试饱和 1914 接收 Source2 位 15 至 0 并向结果寄存器 1952 输出位 39 至 32。测试结果 1915 接收 Source2 位 31 至 16 并向结果寄存器 1952 输出位 47 至 40。测试饱和 1916 接收 Source2 位 47 至 32 并向结果寄存器 1952 输出位 55 至 48。测试饱和 1917 接收 Source2 位 63 至 48 并向结果寄存器 1952 输出位 63 至 56。

[0320] 测试饱和 1920 至测试饱和 1923 具有 32 位输入与 16 位输出。16 位输出为输入的低 16 位，或可选地为一个箝位的值 (0x8000、0x7FFF 或 0xFFFF)。测试饱和 1920 接收 Source1 位 31 至 0 并为结果寄存器 1953 输出位 15 至 0。测试饱和 1921 接收 Source1 位 63 至 32 并为结果寄存器 1953 输出位 31 至 16。测试饱和 1922 接收 Source2 位 31 至 0 并为结果寄存器 1953 输出位 47 至 32。测试饱和 1923 接收 Source2 位 63 至 32 并为结果寄存器 1953 输出位 63 至 48。

[0321] 例如，在表 26 中，执行不带饱和的无符号字组装。操作控制 1900 将启动结果寄存器 1952 输出结果 [63:0]1960。

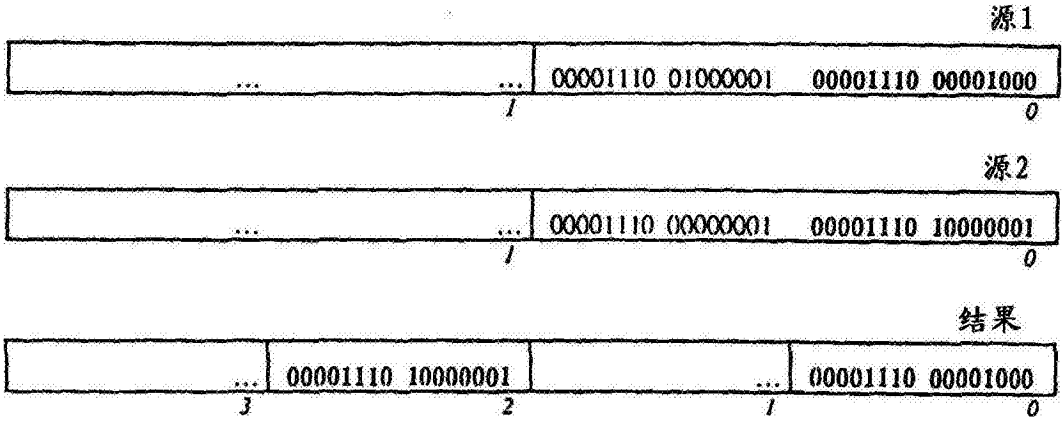
[0322]



[0323] 表 26

[0324] 然而，如果执行不带饱和的无符号双字组装，操作控制 1900 将启动结果寄存器 1953 输出结果 [63:0]1960。表 27 示出这一结果。

[0325]



[0326] 表 27

[0327] 在指令集中包含上述组装操作的优点

[0328] 上述组装指令组装来自 Source1 与 Source2 中各数据元素的预定数目的位来生成 Result。以这一方式，处理器 109 能以少到先有技术通用处理器中所需的指令一半的指令中组装数据。例如，从四个 32 位数据元素生成包含 4 个 16 位数据元素的结果只需一条指令（与两条指令相对照），如下面所示：

[0329]

Pack.High Source1,Source2

A_0	A_0	C_0	C_0	源1
G_0	G_0	B_0	B_0	源2
		=		
A_0	C_0	G_0	B_0	结果1

[0330] 表 28

[0331] 典型的多媒体应用组装大量数据。从而，通过将组装这些数据所需的指令数目减少到一半，便提高了这些多媒体应用的性能。

[0332] 分解操作

[0333] 分解操作

[0334] 在一个实施例中，分解操作交错两个源分组数据的低位分组字节、字或双字以生成结果分组字节、字或双字。这里将这一操作称作分解低操作。在另一实施例中，分解操作也可能交错高阶元素（称作分解高操作）。

[0335] 图 20 为说明按照本发明的一个实施例在分组数据上执行分解操作的方法的流程图。

[0336] 首先执行步骤 2001 与 2002。在步骤 2003，解码器 202 启动执行单元 130 去执行分解操作。解码器 202 通过内部总线 170 传递 Source1 与 Source2 中的数据元素的长度。

[0337] 在步骤 2010，数据元素的长度确定下面要执行哪一步骤。如果数据元素的长度为 8 位（分组字节 401 数据），则执行单元 130 执行步骤 2012。然而，如果分组数据中的数据元素的长度为 16 位（分组字 402 数据）则执行单元 130 执行步骤 2014。然而，如果分组数据中的数据元素的长度为 32 位（分组双字 503 数据），则执行单元 130 执行步骤 2016。

[0338] 假定源数据元素长度为 8 位，便执行步骤 2012。在步骤 2012 中，执行下述操作。Source1 位 7 至 0 为 Result 位 7 至 0。Source2 位 7 至 0 为 Result 位 15 至 8。Source1 位 15 至 8 为 Result 位 23 至 16。Source2 位 15 至 8 为 Result 位 31 至 24。Source1 位 23 至 16 为 Result 位 39 至 32。Source2 位 23 至 16 为 Result 位 47 至 40。Source1 位 31 至 24 为 Result 位 55 至 48。Source2 位 31 至 24 为 Result 位 63 至 56。

[0339] 假定源数据元素的长度为 16 位，则执行步骤 2014。在步骤 2014 中，执行下述操作。Source1 位 15 至 0 为 Result 位 15 至 0。Source2 位 15 至 0 为 Result 位 31 至 16。Source1 位 31 至 16 为 Result 位 47 至 32。Source2 位 31 至 16 为 Result 位 63 至 48。

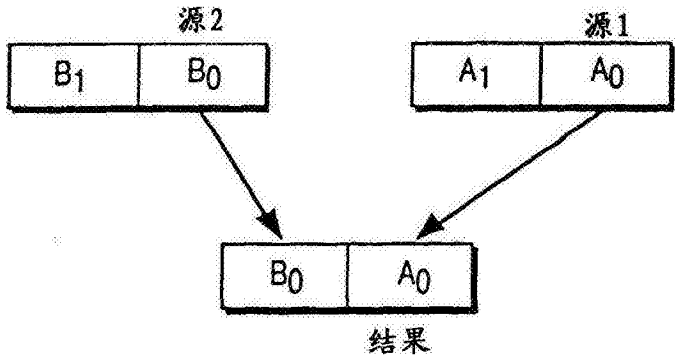
[0340] 假定源数据元素长度为 32 位，则执行步骤 2016。在步骤 2016 中，执行下述操作。Source1 位 31 至 0 为 Result 位 31 至 0。Source2 位 31 至 0 为 Result 位 63 至 32。

[0341] 在一个实施例中，同时执行步骤 2012 的分解。然而，在另一实施例中，串行执行这一分解。在另一实施例中，一些分解是同时执行的而一些则是串行执行的。这一讨论也适用于步骤 2014 与步骤 2016 的分解。

[0342] 在步骤2020,将Result存储在DEST 605寄存器中。

[0343] 表29说明分解双字操作（各数据元素$A_{0\ 1}$及$B_{0\ 1}$包含32位）的寄存器表示。

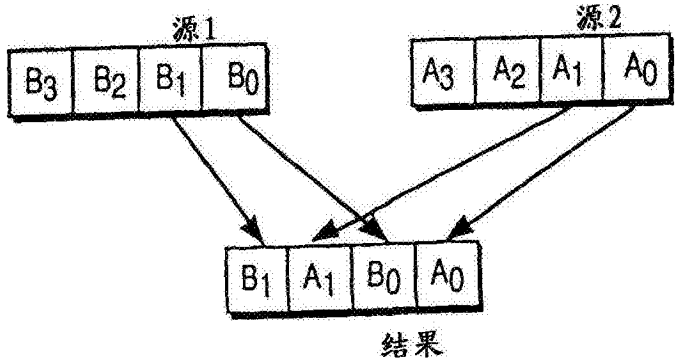
[0344]



[0345] 表29

[0346] 表30说明分解字操作（各数据元素$A_{0\ 3}$及$B_{0\ 3}$包含16位）的寄存器表示。

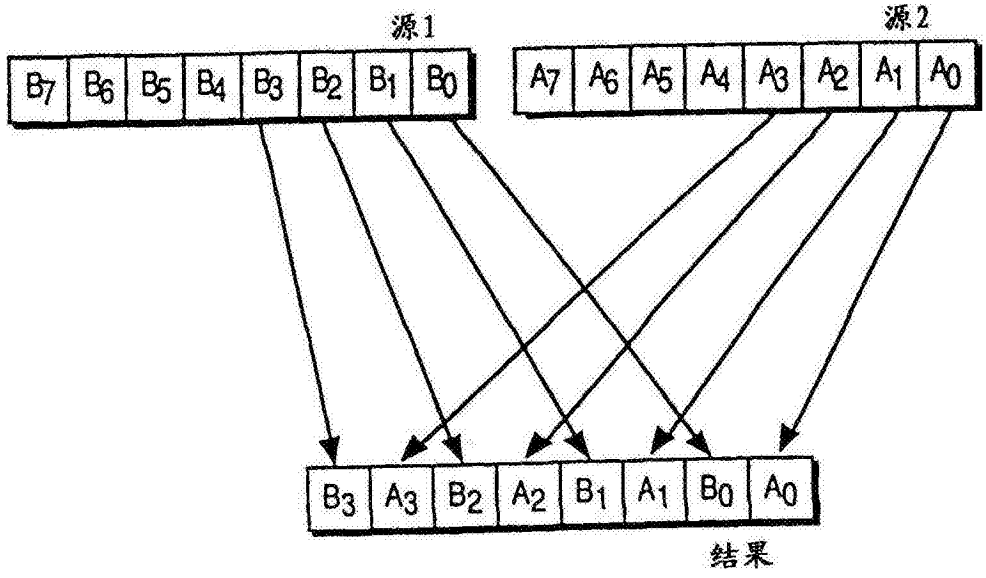
[0347]



[0348] 表30

[0349] 表31说明分解字节操作（各数据元素$A_{0\ 7}$及$B_{0\ 7}$包含8位）的寄存器表示。

[0350]



[0351] 表31

[0352] 分解电路

[0353] 图21示出按照本发明的一个实施例在分组数据上执行分解操作的电路。图21的

电路包含操作控制电路 2100、结果寄存器 2152、结果寄存器 2153 及结果寄存器 2154。

[0354] 操作控制 2100 接收来自解码器 202 的信息以启动分解操作。如果源分组数据的长度为字节分组数据 502，则操作控制 2100 设定输出使能 2132。这便启动结果寄存器 2152 的输出。如果源分组数据的长度为字分组数据 503，则操作控制 2100 设置输出使能 2133。这便启动输出寄存器 2153 的输出。如果源分组数据的长度为双字分组数据 504，则操作控制 2100 设置输出使能 2134。这便启动输出结果寄存器 2154 的输出。

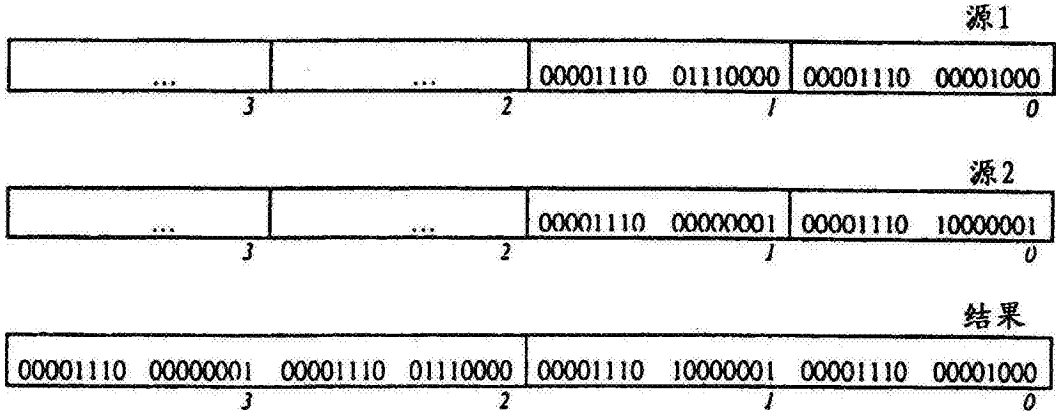
[0355] 结果寄存器 2152 具有下述输入。Source1 位 7 至 0 为结果寄存器 2152 的位 7 至 0。Source2 位 7 至 0 为结果寄存器 2152 的位 15 至 8。Source1 位 15 至 8 为结果寄存器 2152 的位 23 至 16。Source2 位 15 至 8 为结果寄存器 2152 的位 31 至 24。Source1 位 23 至 16 为结果寄存器 2152 的位 39 至 32。Source2 位 23 至 16 为结果寄存器 2152 的位 47 至 40。Source1 位 31 至 24 为结果寄存器 2152 的位 55 至 48。Source2 位 31 至 24 为结果寄存器 2152 的位 63 至 56。

[0356] 结果寄存器 2153 具有下述输入。Source1 位 15 至 0 为结果寄存器 2153 的位 15 至 0。Source2 位 15 至 0 为结果寄存器 2153 的位 31 至 16。Source1 位 31 至 16 为结果寄存器 2153 的位 47 至 32。Source2 位 31 至 16 为结果寄存器 2153 的位 63 至 48。

[0357] 结果寄存器 2154 具有下述输入。Source1 位 31 至 0 为结果寄存器 2154 的位 31 至 0。Source2 位 31 至 0 为结果寄存器 2154 的位 63 至 32。

[0358] 例如，在表 32 中，执行了分解字操作。操作控制 2100 将启动结果寄存器 2153 输出结果 [63:0]2160。

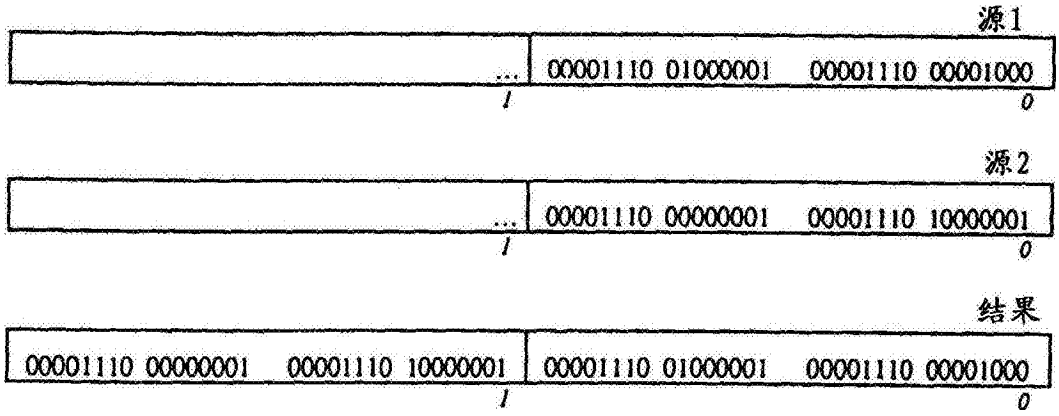
[0359]



[0360] 表 32

[0361] 然而，如果执行分解双字，操作控制 2100 将启动结果寄存器 2154 输出 Result[63:0]2160。表 33 示出这一结果。

[0362]

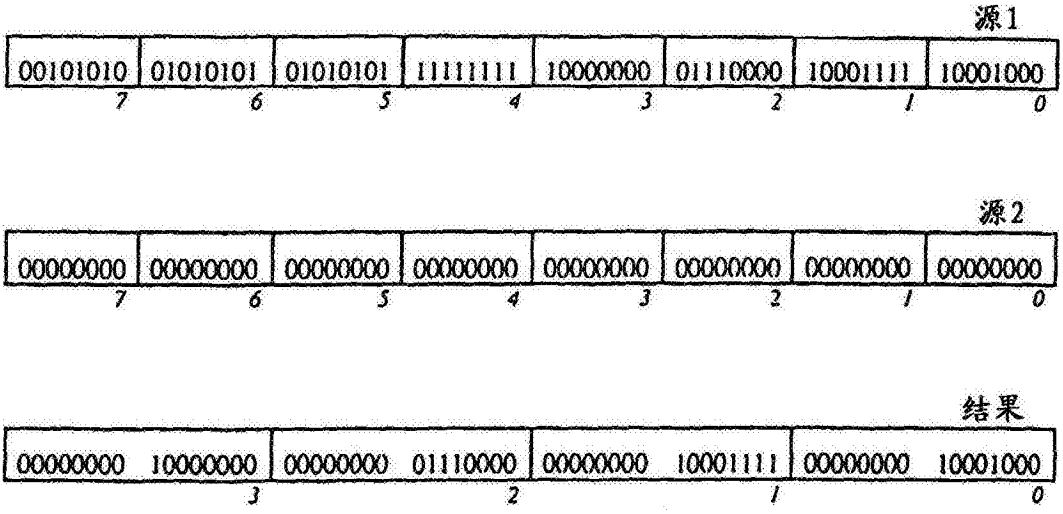


[0363] 表 33

[0364] 在指令集中包含上述分解指令的优点

[0365] 通过将上述分解指令加入指令集中，可以交错或者分解分组数据。这一分解指令通过使 Source2 中的数据元素全为 0，便能用来分解分组数据。分解字节的实例示出在表 34a 中。

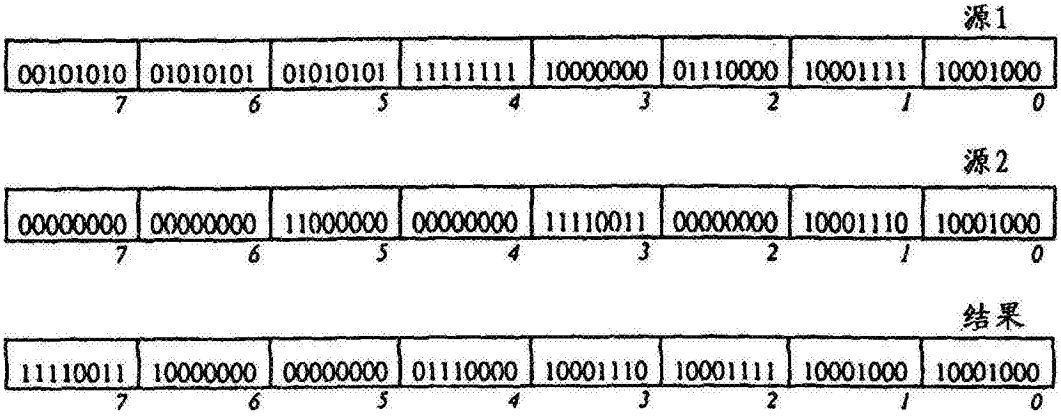
[0366]



[0367] 表 34a

[0368] 同一分解指令可用来交错数据，如表 34b 中所示。在多种多媒体算法中交错是有用的。例如，交错可用于转置矩阵及插值象素。

[0369]



[0370] 表 34b

[0371] 从而，通过在处理器 109 支持的指令集中加入这一分解指令，处理器 109 更为通用并能在更高的性能级上执行需要这一功能的算法。

[0372] 个数计算

[0373] 个数计算

[0374] 本发明的一个实施例允许要在分组数据上执行的个数计数操作。即，本发明为第一分组数据的各数据元素生成一个结果数据元素。各结果数据元素表示在第一分组数据的各对应数据元素中置位的位数。在一个实施例中，计数置位成 1 的总位数。

[0375] 表 35a 说明在分组数据上的个数计数操作的寄存器表示。第一行的位是 Source1 分组数据的分组数据表示。第二行的位是 Result 分组数据的分组数据表示。各数据元素位下面的数据字为数据元素号。例如，Source1 数据元素 0 为 1000111110001000_2。因此，如果数据元素长度为 16 位（字数据），并且执行个数计数操作，执行单元 130 生成所示的 Result 分组数据。

[0376]

01110010 00000101	11111111 11111111	01111111 11111111	10001111 10001000
3	2	1	0
00000000 00000110	00000000 00010000	00000000 00001111	00000000 00000111
3	2	1	0

[0377] 表 35a

[0378] 在另一实施例中，个数计数是在 8 位数据元素上执行的。表 35b 说明在具有 8 个 8 位分组数据元素的分组数据上的个数计数的寄存器表示。

[0379]

01111111	01010101	10101010	10000001	10000000	11111111	11001111	00000000
2	6	5	4	3	2	1	0
00000111	00000100	00000100	00000010	00000001	00001000	00000110	00000000
7	6	5	4	3	2	1	0

[0380] 表 35b

[0381] 在另一实施例中，个数计数是在 32 位数据元素上执行的。表 35c 说明在具有两个 32 位分组数据元素的分组数据上的个数计数的寄存器表示。

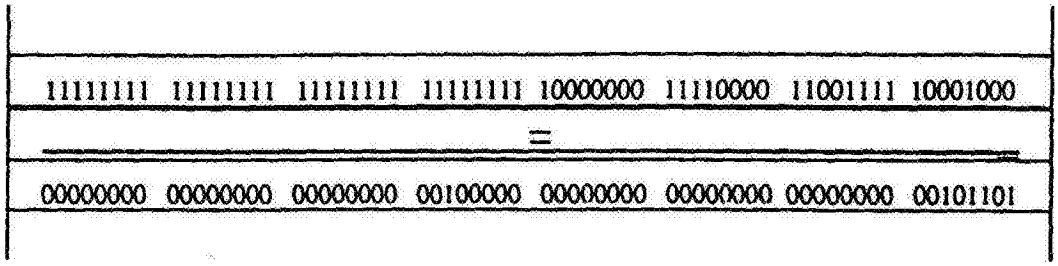
[0382]

11111111 11111111 11111111 11111111	10000000 11110000 11001111 10001000
1	0
00000000 00000000 00000000 00100000	00000000 00000000 00000000 00001101
1	0

[0383] 表 35c

[0384] 个数计数也能在 64 位整数数据上执行。即，求出 64 位数据中置位成 1 的总位数。表 35d 说明在 64 位整数数据上的个数计数的寄存器表示。

[0385]



[0386] 表 35d

[0387] 执行个数计数的方法

[0388] 图 22 为说明按照本发明的一个实施例在分组数据上执行个数计数操作的方法的流程图。在步骤 2201，解码器 202 响应一个控制信号 207 的接收，解码该控制信号 207。在一个实施例中，控制信号 207 是通过总线 101 供给的。在另一实施例中，控制信号 207 是高速缓冲存储器 160 供给的。从而，解码器 202 解码出：个数计数的操作码、以及寄存器 209 中的 SRC1602 及 DEST 605 地址。注意在当前本发明的实施例中不使用 SRC2603。在这一实施例中也不使用饱和 / 不饱和、带符号 / 无符号及分组数据中的数据元素长度。在本发明的当前实施例中，只支持 16 位数据元素长度的分组加法。然而，熟悉本技术的人员会理解能在具有 8 个分组字节数据元素或两个分组双字数据元素的分组数据上执行个数计数。

[0389] 在步骤 2202，解码器 202 通过内部总线 170 存取寄存器文件 150 中给出 SRC1602 地址的寄存器 209。寄存器 209 向执行单元 130 提供存储在这一地址上的寄存器中的分组数据 Source1，即寄存器 209 通过内部总线 170 将分组数据传递给执行单元 130。

[0390] 在步骤 2130，解码器 202 启动执行单元 130 去执行个数计数操作。在一个替代实施例中，解码器 202 还通过内部总线 170 传递分组数据元素长度。

[0391] 在步骤 2205，假定数据元素长度为 16 位，则执行单元 130 求出 Source1 位 15 至 0 中置位的位的总数，产生 Result 分组数据的位 15 至 0。与这一求总数并行，执行单元 130 求 Source1 位 31 至 16 的总数，产生 Result 分组数据的位 31 至位 16。与这些总数的生成并行，执行单元 130 总计 Source1 的位 47 至位 32，产生 Result 分组数据的位 47 至位 32。与这些总计的生成并行，执行单元 130 总计 Source1 的位 63 至位 48，产生 Result 分组数据的位 63 至位 48。

[0392] 在步骤 2206，解码器 202 启动寄存器 209 中带有目的地寄存器的 DEST 605 地址的寄存器。从而，将 Result 分组数据存储在由 DEST 605 寻址的寄存器中。

[0393] 在一个数据元素上执行个数计数的方法

[0394] 图 23 为说明按照本发明的一个实施例在一个分组数据的一个数据元素上执行个数计数操作及生成一个结果分组数据的单个结果数据元素的方法。在步骤 2310a，从 Source1 位 15、14、13 与 12 生成一个列和 CSum 1a 及列进位 CCarry 1a。在步骤 2310b，从 Source1 位 11、10、9 与 8 生成列和 CSum1b 及列进位 CCarry1b。在步骤 2310C 从 Source1 位 7、6、5 与 4 生成列和 CSum1c 及列进位 CCarry1c。在步骤 2310d，从 Source1 位 3、2、1 与 0 生成列和 CSum1d 及列进位 CCarry1d。在本发明的一个实施例中，步骤 2310a-d 是并行执行的。在步骤 2320a，从 CSum1a、CCarry 1a、CSum1b 与 CCarry 1b 生成列和 CSum2a 及列进位 CCarry 2b。在步骤 2320b，从 CSum1c、CCarry1、CSum1d 与 CCarry1d 生成列和 CSum2b 及列进位 CCarry2b。在本发明的一个实施例中，步骤 2320a-b 是并行执行的。在步骤 2330，

从 CSumm2a、CCarry 2a、CSum2b 及 CCarry 2b 生成列和 CSum3 及列进位 CCarry3。在步骤 2340，从 CSum3 与 CCarry3 生成 Result 结果。在一个实施例中，Result 是以 16 位表示的。在这一实施例中，由于只需要位 4 至位 0 来表示 Source1 中置位的位的最大数目，将位 15 至 5 设定为 0。Source1 的最大位数为 16。这出现在 Source1 等于 1111111111111111_2时。Result 将为 16 并用 0000000000010000_2表示。

[0395] 从而，为了为 64 位分组数据上的个数计数操作计算 4 个结果数据元素，要为分组数据中的每一个数据元素执行图 23 的步骤。在一个实施例中，4 个 16 位结果数据元素是并行计算的。

[0396] 执行个数计数的电路

[0397] 图 24 说明按照本发明的一个实施例在具有 4 个字数据元素的分组数据上执行个数计数操作的电路。图 25 说明按照本发明的一个实施例在分组数据的一个字数据元素上执行个数计数操作的详细电路。

[0398] 图 24 示出一个电路，其中 Source1 总线 2401 通过 $Source1_{IN}$2406a-d 将信息信号带到 popcnt 电路 2408a-d。从而 popcnt 电路 2408a 求出 Source1 的位 16 至位 0 中置位的位的总数，生成 Result 的位 15 至位 0。popcnt 电路 2408b 求出 Source1 的位 31 至位 16 中置位的位的总数，生成 Result 的位 31 至位 16。popcnt 电路 2408c 求出 Source1 的位 47 至位 32 中置位的位的总数，生成 Result 的位 47 至位 32。popcnt 电路 2408d 求出 Source1 的位 63 至位 48 中的置位的位的总数，生成 Result 的位 63 至位 48。启动 2404a-d 通过控制 2403 从操作控制 2410 接收启动 popcnt 电路 2408a-d 执行个数计数操作的控制信号，及将 Result 放置在 Result 总线 2409 上。给予了上面描述及图 1-6b 及 22-25 中的描述与说明，熟悉本技术的人员将能建立这一电路。

[0399] popcnt 电路 2408a-d 通过结果输出 2407a-d 将分组个数计数操作的结果信息传递到 Result 总线 2409 上。然后将这一结果信息存储在 DEST 605 寄存器地址所指定的整数寄存器中。

[0400] 在一个数据元素上执行个数

[0401] 计数的电路

[0402] 图 25 示出在分组数据的一个字数据元素上执行个数计数操作的详细电路。具体地，图 25 示出 popcnt 电路 2408a 的一部分。为了达到采用个数计数操作的应用的最大性能，应在一个时钟周期内完成这个操作。因此，假定存取寄存器及存储结果需要时钟周期的一定百分比，图 24 的电路在一个时钟周期大约 80％的时间内完成其操作。这一电路具有允许处理器 109 在一个时钟周期中在四个 16 位数据元素上执行个数计数操作的优点。

[0403] popcnt 电路 2408a 采用 4- ＞ 2 进位保留加法器（除非另有指定，CSA 将指 4- ＞ 2 进位保留加法器），popcnt 电路 2408a-d 中可能采用的 4- ＞ 2 进位保留加法器是本技术中众所周知的。4- ＞ 2 进位保留加法器为将 4 个操作数相加得出两个和的加法器。由于 popcnt 电路 2408a 中的个数计数操作包含 16 位，第一级包含 4 个 4- ＞ 2 进位保留加法器。这四个 4- ＞ 2 进位保留加法器将 16 个一位操作数变换成 8 个 2 位和。第二级将 8 个 2 位和变换成 4 个 3 位和，而第三级将 4 个 3 位和变换成两个 4 位和。然后一个 4 位全加法器将两个四位和相加生成最终结果。

[0404] 虽然采用了 4- ＞ 2 进位保留加法器，替代实施例中可采用 3- ＞ 2 进位保留加法

器。另外，也可使用若干个全加法器；然而，这种配置不能象图 25 中所示的实施例那样快地提供结果。

[0405] $Source1_{IN15\ 0}$2406a 携带 Source1 的位 15 至位 0。第一个四位耦合在 4- ＞ 2 进位保留加法器 (CSA 2510a) 的输入上。下面的四位耦合在 CSA 2510b 的输入上。再下面的四位耦合在 CSA 2510c 的输入上。最后四位耦合在 CSA 2510d 的输入上。各 CSA 2510a-d 生成两个 2 位输出。将 CSA 2510a 的两个 2 位输出耦合到 CSA 2520a 的两个输入上。将 CSA 2510b 的两个 2 位输出耦合到 CSA 2520a 的其它两个输入上。将 CSA 2510c 的两个 2 位输出耦合到 CSA 2520b 的两个输入上。将 CSA 2510d 的两个 2 位输出耦合到 CSA 2520b 其余两个输入上。各 CSA 2520a-b 生成两个 3 位输出。将 2520a 的两个 3 位输出耦合到 CSA2530 的两个输入上。将 2520b 的两个 3 位输出耦合到 CSA 2530 的其余两个输入上。CSA 2530 生成两个 4 位输出。

[0406] 将这些两个 4 位输出耦合到全加法器 (FA 2550) 的两个输入上。FA 2550 将两个 4 位输入相加并传递 Result 输出 2407a 的位 3 至位 0 作为该两个 4 位输入相加的总和。FA 2550 通过进位输出 (CO 2552) 生成 Result 输出 2407a 的位 4。在一个替代实施例中，采用 5 位全加法器来生成 Result 输出 2407a 的位 4 至位 0。在任一情况中，都将 Result 输出 2407a 的位 15 至位 5 固定在 0 上。同样，将对全加法器的任何进位输入固定在 0 上。

[0407] 虽然在图 25 中未示出，熟悉本技术的人员会理解可将 Result 输出 2407a 多路复用或缓冲存储到 Result 总线 2409 上。多路复用器受到使能 2404a 的控制。这将允许其它执行单元电路将数据写到 Result 总线 2409 上。

[0408] 在指令集中加入上述个数计数操作的优点

[0409] 上述个数计数指令计数诸如 Source1 等分组数据的各数据元素中置位的位的数目。从而，通过在指令集中加入这一指令，便可在一条单一指令中在分组数据上执行个数计数操作。相反，先有技术通用处理器必须执行许多指令来分解 Source1，在各分解的数据元素上单个地执行该功能，然后组装结果供进一步分组处理。

[0410] 从而，通过在处理器 109 支持的指令集中加入这一个数计数指令，便提高了需要这一功能的算法的性能。

[0411] 逻辑运算

[0412] 逻辑运算

[0413] 在本发明的一个实施例中，SRC1 寄存器包含分组数据 (Source1)，SRC2 寄存器包含分组数据 (Source2)，而 DEST 寄存器将包含在 Source1 与 Source2 上执行所选择的逻辑运算的结果 (Result)。例如，如果选择了逻辑“与”运算，则将 Source1 与 Source2 逻辑“与”。

[0414] 在本发明的一个实施例中，支持下述逻辑运算：逻辑“与”、逻辑“非与”(ANDN)、逻辑“或”及逻辑“异或”(XOR)。逻辑“与”、“或”及“异或”运算是本技术中众所周知的。逻辑“非与”(ANDN) 运算使 Source2 与 Source1 的逻辑“非”进行“与”运算。虽然本发明是关于这些逻辑运算描述的，其它实施例可实现其它逻辑运算。

[0415] 图 26 为说明按照本发明的一个实施例在分组数据上执行若干种逻辑运算的方法的流程图。

[0416] 在步骤 2601，解码器 202 解码处理器 109 所接收的控制信号 207。从而，解码器

202 解码出 :适当的逻辑运算 (即“与”、“非与”、“或”或“异或”) 的操作码 ;寄存器 209 中的 SRC1602、SRC2603 及 DEST604 地址。

[0417] 在步骤 2602,解码器 202 通过内部总线 170 存取寄存器文件 150 中给出 SRC1602 及 SRC2603 地址的寄存器 209。寄存器 209 向执行单元 130 提供存储在 SRC1602 寄存器中的分组数据 (Source1) 及存储在 SRC2603 寄存器中的分组数据 (Source2)。即,寄存器 209 通过内部总线 170 将分组数据传递给执行单元 130。

[0418] 在步骤 2603,解码器 202 启动执行单元 130 去执行分组逻辑运算中所选择的一种。

[0419] 在步骤 2610,分组逻辑运算中所选择的一种确定下面执行哪一步骤。如果选择了逻辑“与”运算,执行单元 130 执行步骤 2612 ;如果选择了逻辑“非与”运算,执行单元 130 执行步骤 2613 ;如果选择了逻辑“或”运算,执行单元 130 执行步骤 2614 ;而如果选择了逻辑“异或”运算,执行单元 130 执行步骤 2615。

[0420] 假定选择了逻辑“与”运算,便执行步骤 2612。在步骤 2612 中, Source1 位 63 至 0 和 Source2 位 63 至 0 进行“与”运算生成 Result 位 63 至 0。

[0421] 假定选择了逻辑“非与”运算,便执行步骤 2613。在步骤 2613 中, Source1 位 63 至 0 和 Source2 位 63 至 0 进行“非与”运算生成 Result 位 63 至 0。

[0422] 假定选择了逻辑“或”运算,便执行步骤 2614。在步骤 2614 中, Source1 位 63 至 0 和 Source2 位 63 至 0 进行“或”运算生成 Result 位 63 至 0。

[0423] 假定选择了逻辑“异或”运算,便执行步骤 2615。在步骤 2615 中, Source1 位 63 至 0 和 Source2 位 63 至 0 进行“异或”运算生成 Result 位 63 至 0。

[0424] 在步骤 2620,将 Result 存储在 DEST 寄存器中。

[0425] 表 36 说明分组数据上的逻辑“非与”运算的寄存器表示。第一行的位是 Source1 的分组数据表示。第二行的位是 Source2 的分组数据表示。第三行的位是 Result 的分组数据表示。各数据元素位下方的数字为数据元素号。例如, Source1 数据元素 2 为 1111111100000000_2。

[0426]

11111111 11111111	11111111 00000000	11111111 00000000	00001110 00001000
逻辑“非与” 3	逻辑“非与” 2	逻辑“非与” 1	逻辑“非与” 0
00000000 00000000	00000000 00000001	10000000 00000000	00001110 10000001
=	=	=	=
00000000 00000000	00000000 00000001	00000000 00000000	00000000 10000001
3	2	1	0

[0427] 表 36

[0428] 虽然本发明是相对于在 Source1 与 Source2 中的对应数据元素上执行同一逻辑运算描述的,替代实施例可支持允许在逐个元素的基础上选择在对应的数据元素上要执行的逻辑运算的指令。

[0429] 分组数据逻辑电路

[0430] 在一个实施例中,能够在和非分组数据上的单一逻辑运算相同数目的时钟周期中在多个数据元素上出现上述逻辑运算。为了达到在相同数目的时钟周期中的执行,采用了

并行性。

[0431] 图 27 说明按照本发明的一个实施例在分组数据上执行逻辑运算的电路。操作控制 2700 控制执行逻辑运算的电路。操作控制 2700 处理控制信号并在控制线 2780 上输出选择信号。这些选择信号向逻辑运算电路 2701 传递“与”、“非与”、“或”及“异或”运算中选择的一种。

[0432] 逻辑运算电路 2701 接收 Source1[63:0] 及 Source2[63:0] 并执行选择信号指定的逻辑运算以生成 Result。逻辑运算电路 2701 将 Result[63:0] 传递给结果寄存器 2731。

[0433] 在指令集中加入上述逻辑运算的优点

[0434] 上述逻辑指令执行逻辑“与”、逻辑“非与”、逻辑“或”及逻辑“异或”。这些指令在需要数据的逻辑操作的任何应用中是有用的。通过在处理器 109 支持的指令集中加入这些指令，便可以在一条指令中在分组数据上执行这些逻辑运算。

[0435] 分组比较

[0436] 分组比较操作

[0437] 在本发明的一个实施例中，SRC1602 寄存器中包含要比较的数据 (Source1)，SRC2603 寄存器中包含要相对于它进行比较的数据 (Source2)，而 DEST 605 寄存器中将包含比较的结果 (Result)。即，用 Source2 的各数据元素按指定的关系独立地与 Source1 中各数据元素比较。

[0438] 在本发明的一个实施例中，支持下述比较关系 :等于 ;带符号的大于 ;带符号的大于或等于 ;无符号大于 ;或无符号大于或等于。在每对对应的数据元素中测试这种关系。例如，Source1[7:0] 大于 Source2[7:0]，结果为 Result[7:0]。如果比较结果满足该关系，则在一个实施例中将 Result 中的对应数据元素设置成全 1。如果比较的结果不满足该关系，则将 Result 中的对应数据元素设置成全 0。

[0439] 图 28 为说明按照本发明的一个实施例在分组数据上执行分组比较操作的方法的流程图。

[0440] 在步骤 2801，解码器 202 解码处理器 109 接收的控制信号 207。从而，解码器 202 解码出 :适当比较操作的操作码 ;寄存器 209 中的 SRC1602、SRC2603 及 DEST 605 地址 ;饱和 / 不饱和 (对比较操作没有必要)，带符号 / 无符号及分组数据中的数据元素的长度。如上所述，SRC1602(或 SRC2603) 可用作 DEST 605。

[0441] 在步骤 2802，解码器 202 通过内部总线 170 存取寄存器文件 150 中给定 SRC1602 与 SRC2603 地址的寄存器 209。寄存器 209 向执行单元 130 提供存储在 SRC1602 寄存器中的分组数据 (Source1) 及存储在 SRC2603 寄存器中的分组数据 (Source2)。即，寄存器 209 通过内部总线 170 将分组数据传递给执行单元 130。

[0442] 在步骤 2803，解码器 202 启动执行单元 130 去执行适当的分组比较操作。解码器 202 还通过内部总线 170 传递数据元素的长度及比较操作的关系。

[0443] 在步骤 2810，数据元素的长度确定下面要执行的步骤。如果数据元素的长度为 8 位 (分组字节 401 数据)，则执行单元 130 执行步骤 2812。然而，如果分组数据中的数据元素的长度为 16 位 (分组字 402 数据)，则执行单元 130 执行步骤 2814。在一个实施例中，只支持 8 位与 16 位数据元素长度的分组比较。然而，在另一实施例中，也支持 32 位数据元素长度的分组比较 (分组双字 403)。

[0444] 假定数据元素的长度为 8 位，则执行步骤 2812。在步骤 2812 中，执行下述操作。将 Source1 位 7 至 0 对 Source2 位 7 至 0 比较生成 Result 位 7 至 0。将 Source1 位 15 至 8 对 Source2 位 15 至 8 比较生成 Result 位 15 至 8。将 Source1 位 23 至 16 对 Source2 位 23 至 16 比较生成 Result 位 23 至 16。将 Source1 位 31 至 24 对 Source2 位 31 至 24 比较生成 Result 位 31 至 24。将 Source1 位 39 至 32 对 Source2 位 39 至 32 比较生成 Result 位 39 至 32。将 Source1 位 47 至 40 对 Source2 位 47 至 40 比较生成 Result 位 47 至 40。将 Source1 位 55 至 48 对 Source2 位 55 至 48 比较生成 Result 位 55 至 48。将 Source1 位 63 至 56 对 Source2 位 63 至 56 比较生成 Result 位 63 至 56。

[0445] 假定数据元素的长度为 16 位，便执行步骤 2814。在步骤 2814 中，执行下述操作。将 Source1 位 15 至 0 对 Source2 位 15 至 0 比较生成 Result 位 15 至 0。将 Source1 位 31 至 16 对 Source2 位 31 至 16 比较生成 Result 位 31 至 16。将 Source1 位 47 至 32 对 Source2 位 47 至 32 比较生成 Result 位 47 至 32。将 Source1 位 63 至 48 对 Source2 位 63 至 48 比较生成 Result 位 63 至 48。

[0446] 在一个实施例中，步骤 2812 的比较是同时执行的。然而，在另一实施例中，这些比较是串行执行的。在另一实施例中，一些比较是同时执行的而一些则是串行执行的。这一讨论也适用于步骤 2814 中的比较。

[0447] 在步骤 2820，将 Result 存储在 DEST 605 寄存器中。

[0448] 表 37 说明分组比较无符号大于操作的寄存器表示。第一行的位是 Source1 的分组数据表示。第二行的位是 Source2 的数据表示。第三行的位是 Result 的分组数据表示。各数据元素位下面的数字是数据元素号。例如，Source1 数据元素 3 为 10000000_2。

[0449]

00101010	01010101	01010101	11111111	10000000	01110000	10001111	10001000
7 >	6 >	5 >	4 >	3 >	2 >	1 >	0 >
00000000	00000000	10000000	00000000	11110011	00000000	10001110	10001000
↓	↓	↓	↓	↓	↓	↓	↓
11111111	11111111	00000000	11111111	00000000	11111111	11111111	00000000
7	6	5	4	3	2	1	0

[0450] 表 37

[0451] 表 38 说明分组字节数据上的分组比较带符号大于或等于操作的寄存器表示。

[0452]

00101010	01010101	01010101	11111111	10000000	01110000	10001111	10001000
7 >=	6 >=	5 >=	4 >=	3 >=	2 >=	1 >=	0 >=
00000000	00000000	10000000	00000000	11110011	00000000	10001110	10001000
↓	↓	↓	↓	↓	↓	↓	↓
11111111	11111111	11111111	00000000	00000000	11111111	00000000	11111111
7	6	5	4	3	2	1	0

[0453] 表 38

[0454] 分组数据比较电路

[0455] 在一个实施例中，在和非分组数据上的单个比较操作相同数目的时钟周期中能在多个数据元素上产生比较操作。为了达到相同数目的时钟周期中的执行，采用了并行性。即，同时指示寄存器在数据元素上执行比较操作。下面更详细地讨论这一点。

[0456] 图 29 示出按照本发明的一个实施例在分组数据的各个字节上执行分组比较操作的电路。图 29 示出经过修改的字节片比较电路、字节片级 i 2999 的使用。除了最高位数据元素字节片以外的各字节片都包含一个比较单元及位控制。最高位数据元素字节片只需一个比较单元。

[0457] 比较单元 i2911 及比较单元 i+12971 各允许来自 Source1 的 8 位与来自 Source2 的对应的 8 位进行比较。在一个实施例中，各比较单元象已知的 8 位比较单元一样操作。这一已知的 8 位比较电路包含允许从 Source1 减去 Source2 的字节片电路。处理减法的结果来确定比较操作的结果。在一个实施例中，减法结果包含一个溢出信息。测试这一溢出信息来判定比较操作的结果是否为真。

[0458] 各比较单元具有一个 Source1 输入、一个 Source2 输入、一个控制输入、一个下一级信号、一个上一级信号及一个结果输出。因此，比较单元 i 2911 具有 $Source1_i$ 2931 输入、$Source2_i$ 2933 输入、控制 i2901 输入、下一级 i 2913 信号、上一级 i2912 输入及存储在结果寄存器 i 2951 中的结果。因此，比较单元 i+12971 具有 $Source1_{i+1}$ 2932 输入、$Source2_{i+1}$ 2934 输入、控制 i+12902 输入、下一级 i+12973 信号、上一级 i+12972 输入及存储在结果寄存器 i+12952 中的结果。

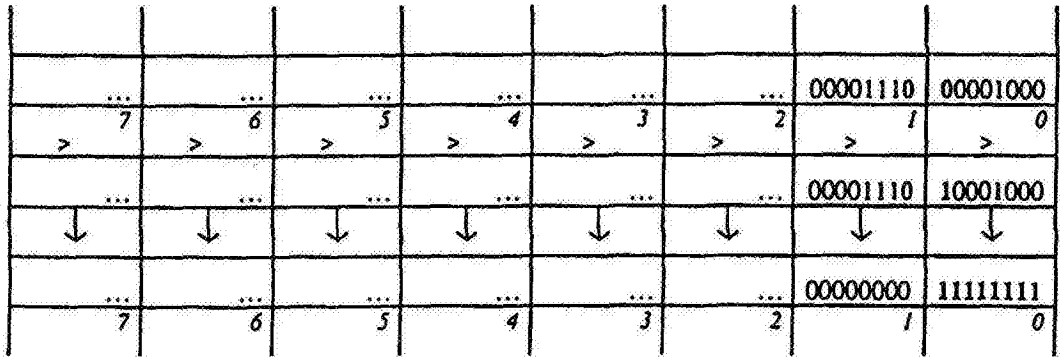
[0459] Source1n 输入通常是 Source1 的一个 8 位部分。8 位表示最小类型的数据元素，一个分组字节 401 的数据元素。Source2 输入是 Source2 的对应 8 位部分。操作控制 2900 传输控制信号来启动各比较单元执行所要求的比较。控制信号是从比较的关系（诸如带符号大于）及数据元素的长度（诸如字节或字）确定的。下一级信号是从该比较单元的位控制接收的。当使用大于字节长度的数据元素时，位控制单元有效地组合比较单元。例如，当比较字分组数据时，第一比较单元与第二比较单元之间的位控制单元将使得这两个比较单元作为一个 16 位比较单元工作。类似地，第三与第四比较单元之间的控制单元将使得这两个比较单元作为一个比较单元工作。这可以继续到四个分组字数据元素。

[0460] 取决于所要求的关系及 Source1 与 Source2 的值，比较单元通过允许较高阶比较单元的结果向下传播到较低阶比较单元或反过来以执行比较。这便是，各比较单元将利用位控制 i 2920 传递的信息来提供比较结果。如果使用双字分组数据，则四个比较单元一起工作以形成用于各数据元素的一个 32 位长的比较单元。各比较单元的结果输出表示该比较单元在其上操作的 Source1 与 Source2 的部分上的比较操作的结果。

[0461] 位控制 i 2920 是从操作控制 2900 通过分组数据使能 i 2906 启动的。位控制 i 2920 控制下一级 i 2913 与上一级 i+12972。例如，假定比较单元 i 2911 负责 Source1 与 Source2 的 8 个最低位，而比较单元 i+12971 负责 Source1 与 Source2 的下一 8 位。如果在分组字节数据上执行比较，位控制 i 2920 将不允许来自比较单元 i+12971 的结果信息传递到比较单元 i 2911，反之亦然。然而，如果在分组字上执行比较，则位控制 i 2920 将允许来自比较单元 i 2911 的结果（在一个实施例中为溢出）信息传递到比较单元 i+1，以及来自比较单元 i+12971 的结果（在一个实施例中为溢出）信息传递给比较单元 i 2911。

[0462] 例如，在表 39 中，执行分组字节带符号大于比较。假定比较单元 i+12971 在数据元素 1 上操作，而比较单元 i 2911 在数据元素 0 上操作。比较单元 i+12971 比较一个字的最高 8 位并通过上一级 i+12972 传递该结果信息。比较单元 i 2911 比较该字的最低 8 位并通过下一级 i 2913 传递该结果信息。然而操作控制 2900 将使得位控制 i 2920 停止在比较单元之间传播从上一级 i+12972 及下一级 i 2913 接收的结果信息。

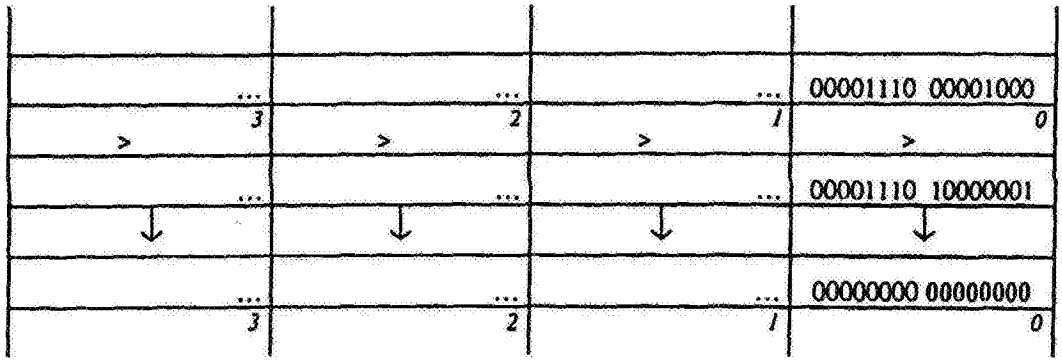
[0463]



[0464] 表 39

[0465] 然而，如果执行分组字带符号大于比较，则比较单元 i+12971 的结果将传递到比较单元 i 2911，反之亦然。表 40 示出这一结果。这在类型的传递对分组双字比较同样允许。

[0466]



[0467] 表 40

[0468] 各比较单元可选地耦合在结果寄存器上。结果寄存器临时存储比较操作的结果直到能将完整的结果 Result[63:0]2960 传输到 DEST 605 寄存器为止。

[0469] 对于一个完整的 64 位分组比较电路，采用 8 个比较单元及 7 个位控制单元。这一电路也能用来在 64 位非分组数据上执行比较，借此利用同一电路来执行非分组比较操作与分组比较操作。

[0470] 在指令集中加入上述分组

[0471] 比较操作的优点

[0472] 上述分组比较指令将 Source 1 与 Source2 的比较结果作为分组掩码存储。如上所述，数据上的条件转移是不可预测的，并因为它们破坏转移预测算法，因此浪费了处理器性能。然而，通过生成分组掩码，这一比较指令减少了需要的基于数据的条件转移的数目。例如，可以在分组数据上执行函数 (ifY ＞ A then X ＝ X+B ;else X ＝ X)，如下面表 41 中所示（表 41 中所示的值为以 16 进制符号示出的）。

Compare.Greater_Than Source1,Source2

00000001	00000000	Source1=Y_{0-1}
>	>	
00000000	00000001	Source2=A_{0-1}
=		
FFFFFFFF	00000000	掩码

[0473]

Packed AND Source3,Mask

00000005	0000000A	Source3=B_{0-1}
>	>	
FFFFFFFF	00000000	掩码
=		
00000005	00000000	结果

Packed Add Source4, Result

00000010	00000020	Source4=X_{0-1}
>	>	
00000005	00000000	结果
=		
00000015	00000020	新 X_{0-1} 值

[0474] 表 41

[0475] 从上述示例中可见，不再需要条件转移。由于不需要转移指令，当使用这一比较指令来执行这一与其它类似操作时，推测性地预测转移的处理器不会有性能降低。从而，通过在处理器 109 支持的指令集中提供这一比较指令，处理器 109 便能在较高的性能级上执行需要这一功能的算法。

[0476] 多媒体算法示例

[0477] 为说明所公开的指令集的通用性，下面描述若干多媒体算法示例。在一些情况中，可以用类似的分组数据指令来执行这些算法中的某些步骤。在下面的示例中，已经省略了需要使用通用处理器指令来管理数据传送、循环及条件转移的若干步骤。

[0478] 1)复数乘法

[0479] 所公开的乘-加指令能用来在单一的指令中将两个复数相乘，如表 42a 中所示。两个复数（诸如，r1i1 与 r2i2）的乘法是按照下列等式执行的：

[0480] 实部＝ r1 • r2-i1 • i2

[0481] 虚部＝ r1 • i2+r2 • i1

[0482] 如果将这一指令实现成在一时钟周期中完成，本发明便能在一个时钟周期中将两个复数相乘。

[0483]

Multiply-Add Source1, Source2

r_1	i_2	r_1	i_1	源1
r_2	$-i_2$	i_2	r_2	源2
=				
实部 $r_1r_2-i_1i_2$		虚部 $r_1i_2+r_2i_1$		结果1

[0484] 表 42a

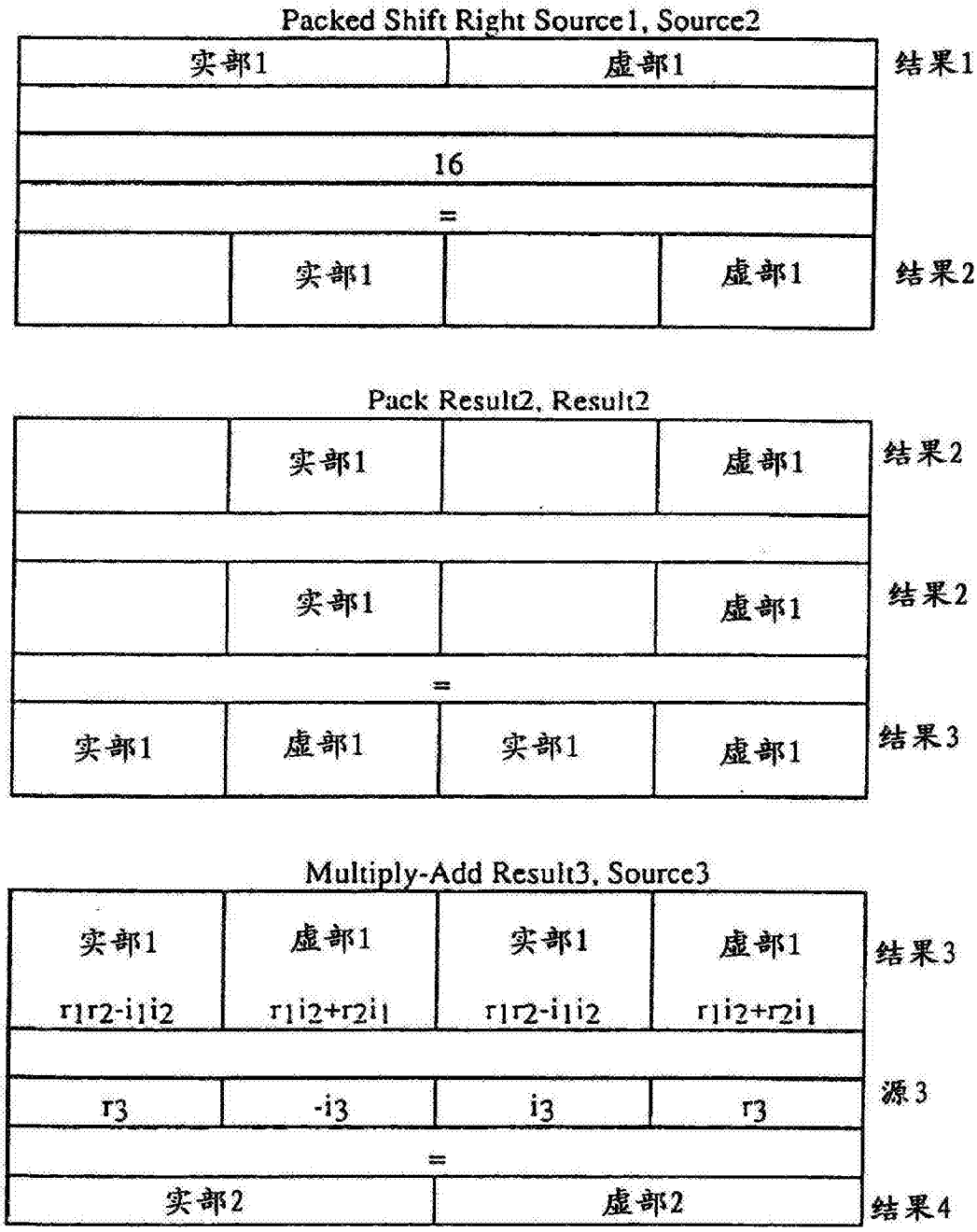
[0485] 作为另一示例，表 42b 示出用来将三个复数一起乘的指令。

[0486]

Multiply-Add Source1, Source2

r_1	i_1	r_1	i_1	源1
r_2	$-i_2$	i_2	r_2	源2
=				
实部1 $r_1r_2-i_1i_2$		虚部1 $r_1i_2+r_2i_1$		结果1

[0487]



[0488] 表 42b

[0489] 2)乘累加运算

[0490] 所公开的指令也能用来乘与累加值。例如,可将两组 4 个数据元素 ($A_{1\ 4}$与$B_{1\ 4}$) 相乘与累加,如下面表 43 中所示。在一个实施例中,表 43 中所示的各指令是实现成在每个时钟周期中完成的。

Multiply-Add Source1, Source2

0	0	A_1	A_2	源1
0	0	B_1	B_2	源2
=				
0		$A_1B_1+A_2B_2$		结果1

Multiply-Add Source3, Source4

0	0	A_3	A_4	源3
0	0	B_3	B_4	源4
=				
0		$A_3A_4+B_3B_4$		结果2

[0491]

Unpacked Add Result1, Result2

0	$A_1B_1+A_2B_2$	结果1
0	$A_3A_4+B_3B_4$	结果2
=		
0	$A_1B_1+A_2B_2+A_3A_4+B_3B_4$	结果3

[0492] 表 43

[0493] 如果各组中的数据元素的数目超过 8 个且为 4 的倍数，如果如下面表 44 中所示执行，这些组的乘法与累加需要更少的指令。

Multiply-Add Source1, Source2

A_1	A_2	A_3	A_4	源1
B_1	B_2	B_3	B_4	源2
=				
$A_1B_1+A_2B_2$		$A_3B_3+A_4B_4$		结果1

Multiply-Add Source3, Source4

A_5	A_6	A_7	A_8	源3
B_5	B_6	B_7	B_8	源4
=				
$A_5B_5+A_6B_6$		$A_7B_7+A_8B_8$		结果2

[0494]

Packed Add Result1, Result2

$A_1B_1+A_2B_2$	$A_3B_3+A_4B_4$	结果1
$A_5B_5+A_6B_6$	$A_7B_7+A_8B_8$	结果2
=		
$A_1B_1+A_2B_2+A_5B_5+A_6B_6$	$A_3B_3+A_4B_4+A_7B_7+A_8B_8$	结果3

Unpack High Result3, Source5

$A_1B_1+A_2B_2+A_5B_5+A_6B_6$	$A_3B_3+A_4B_4+A_7B_7+A_8B_8$	结果3
0	0	源5
=		
0	$A_1B_1+A_2B_2+A_5B_5+A_6B_6$	结果4

Unpack Low Result3, Source5

$A_1B_1+A_2B_2+A_5B_5+A_6B_6$	$A_3B_3+A_4B_4+A_7B_7+A_8B_8$	结果3
0	0	源5
=		
0	$A_3B_3+A_4B_4+A_7B_7+A_8B_8$	结果5

[0495]

Packed Add Result4, Result5

0	$A_1B_1+A_2B_2+A_5B_5+A_6B_6$	结果4
0	$A_3B_3+A_4B_4+A_7B_7+A_8B_8$	结果5
=		
0	总计	结果6

[0496] 表 44

[0497] 作为另一示例，表 45 示出组 A 与 B 以及组 C 与 D 的分开乘法与累加，其中这些组中各组包含两个数据元素。

[0498]

Multiply-Add Source1, Source2

A_1	A_2	C_1	C_2	源1
B_1	B_2	D_1	D_2	源2
=				
$A_1B_1+A_2B_2$		$C_1D_1+C_2D_2$		结果1

[0499] 表 45

[0500] 作为另一示例，表 46 示出组 A 与 B 以及组 C 与 D 的分开乘法与累加，其中这些组中各组包含 4 个数据元素。

Multiply-Add Source1, Source2

A_1	A_2	C_1	C_2	源1
B_1	B_2	D_1	D_2	源2
=				
$A_1B_1+A_2B_2$		$C_1D_1+C_2D_2$		结果1

[0501]

Multiply-Add Source3, Source4

A_3	A_4	C_3	C_4	源3
B_3	B_4	D_3	D_4	源4
=				
$A_3B_3+A_4B_4$		$C_3D_3+C_4D_4$		结果2

Packed Add Result1, Result2

$A_1B_1+A_2B_2$	$C_1D_1+C_2D_2$	结果1
$A_3B_3+A_4B_4$	$C_3D_3+C_4D_4$	结果2
=		
$A_1B_1+A_2B_2+A_3B_3+A_4B_4$	$C_1D_1+C_2D_2+C_3D_3+C_4D_4$	结果6

[0502] 表 46

[0503] 3)点积算法

[0504] 点积（亦称内积）用在信号处理与矩阵运算中。例如，在计算矩阵的积、数字滤波操作（诸如 FIR 与 IIR 滤波）及计算相关序列时使用点积。由于许多语音压缩算法（如 GSM、G.728、CELP 及 VSELP）及高保真压缩算法（诸如 MPEG 及分频段编码）广泛地利用数字滤波及相关计算，提高点积的性能等于提高这些算法的性能。

[0505] 两个长度 N 的序列 A 与 B 的点积定义为：

[0506] $$Result = \sum_{i=0}^{N-1} Ai \bullet Bi$$

[0507] 执行点积计算广泛利用乘累加运算，其中将各序列的对应元素相乘，并累加这些结果以构成点积结果。

[0508] 通过包含传送、分组加法、乘－加及分组移位操作，本发明允许使用分组数据执行点积计算。例如，如果使用包含 4 个 16 位元素的分组数据类型，便可用下述操作在各包含 4 个值的两个序列上执行点积计算：

[0509] 1) 使用传送指令从 A 序列取 4 个 16 位值来生成 Source1；

[0510] 2) 使用传送指令从 B 序列取 4 个 16 位值来生成 Source2 ;以及

[0511] 3) 使用乘 - 加、分组加法及移位指令如上所述相乘与累加。

[0512] 对于带有稍多元素的矢量,使用表 46 中所示的方法并在最后将最终结果加在一起。其它支持指令包含用于初始化累加器寄存器的分组 OR 与 XOR 指令,用于在计算的最后级上移出不需要的值的分组移位指令。利用已存在处理器 109 的指令集中的指令完成循环控制操作。

[0513] 4)二维环路滤波器

[0514] 二维环路滤波器用在一些多媒体算法中。例如,下面表 47 中所示的滤波器系数可用在视频会议算法中以便在象素数据上执行低通滤波。

[0515]
$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

[0516] 表 47

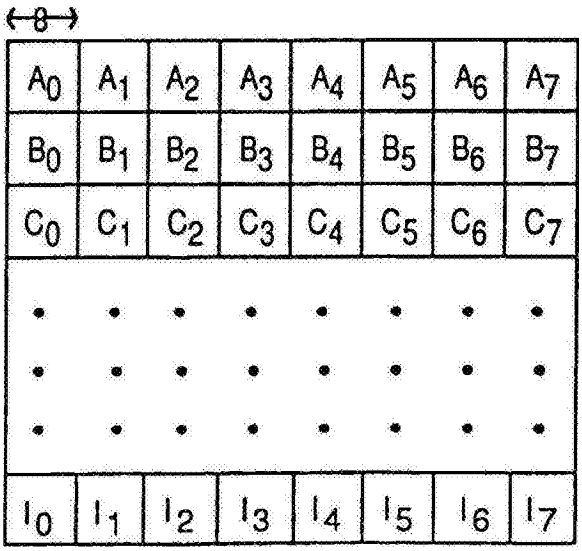
[0517] 为了计算位置 (x, y) 上的象素的新值,使用下述等式 :

[0518] 结果象素 = (x-1,y-1)+2(x,y-1)+(x+1,y-1)+2(x-1,y)+4(x,y)+2(x+1,y)+(x-1,y+1)+2(x,y-1)+(x+1,y+1)

[0519] 通过包含组装、分组、传送、分组移位及分组加法,本发明允许使用分组数据执行二维环路滤波器。按照上述环路滤波器的一种实现,这一环路滤波器是作为两个简单的一维滤波器应用的 - 即,上述二维滤波器能用作两个 121 滤波器。第一滤波器在水平方向上,而第二滤波器则在垂直方向上。

[0520] 表 48 示出象素数据的一个 8×8 块的表示。

[0521]



[0522] 表 48

[0523] 执行下述步骤来实现象素数据的这一 8×8 块上的滤波器的水平通过 :

[0524] 1) 使用传送指令存取 8 个 8 位象素值作为分组数据 ;

[0525] 2) 将这 8 个 8 位象素分解成包含 4 个 8 位象素的 16 位分组数据 (Source1) 以保持累加中的精度 ;

[0526] 3) 复制 Source 1 两次生成 Source2 与 Source3；

[0527] 4) 在 Source1 上执行非分组向右移位 16 位；

[0528] 5) 在 Source3 上执行非分组向左移位 16 位；

[0529] 6) 通过执行下述分组加法生成 (Source1+2*Source2+Source3)；

[0530] a) Source1 = Source1+Source2；

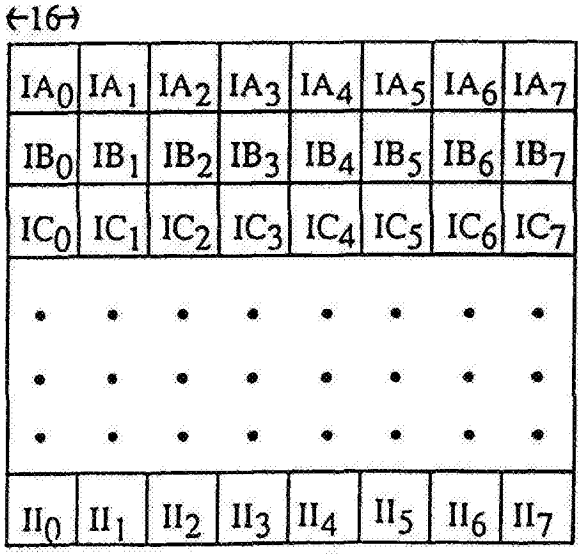
[0531] b) Source1 = Source1+Source2；

[0532] c) Source1 = Source1+Source3；

[0533] 7) 作为一个 8×8 中间结果数组的一部分存储得出的分组字数据；以及

[0534] 8) 重复这些步骤直到生成如下面表 49 中所示的整个 8×8 中间结果数组（如 IA_0 表示来自表 49 的 A_0 的中间结果）。

[0535]

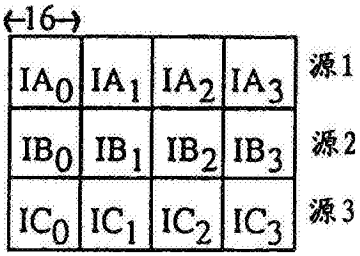


[0536] 表 49

[0537] 执行下述步骤来实现 8×8 中间结果数组上的滤波器的垂直通过：

[0538] 1) 使用传送指令存取来自该中间结果数组的 4×4 的数据块作为分组数据以生成 Source1、Source2 及 Source3（如见作为示例的表 50）；

[0539]



[0540] 表 50

[0541] 2) 通过执行下述分组加法生成 (Source1+2*Source2+Source3)：

[0542] a) Source1 = Source1+Source2；

[0543] b) Source1 = Source1+Source2；

[0544] c) Source1 = Source1+Source3；

[0545] 3) 在得出的 Source1 上执行分组向右移位 4 位生成加权值之和 - 实际上除以 16；

[0546] 4) 组装带有饱和的结果 Source1，将 16 位值转换回 8 位象素值；

[0547] 5) 将得出的分组字节数据作为一个 8×8 结果数组的一部分存储（对于表 50 中所

示的例子，这四个字节表示 B0、B1、B2 与 B3 的新象素值）；以及

[0548] 6) 重复这些步骤直到生成整个 8×8 结果数组为止。

[0549] 值得指出的是，8×8 结果数组的顶与底行是用不同的算法确定的，为了不冲淡本发明在这里未描述该算法。

[0550] 从而通过在处理器 109 上提供组装、分解、传送、分组移位及分组加法指令，本发明的性能明显高于先有技术通用处理器，后者必须一次一个数据元素地执行这些滤波器所要求的操作。

[0551] 5)运动估计 (Motion Estimation)

[0552] 运动估计用在若干种多媒体应用中（诸如，电视会议及 MPEG（高质量电视播放））。对于电视会议，运动估计用来减少必须在终端之间传输的数据量。运动估计通过将视频帧分成固定大小的视频块进行。对于帧 1 中的各块，确定在帧 2 中是否有包含相似图象的块。如果帧 2 中包含这样的块，便能用对帧 1 中的运动矢量引用来描述该块。这样，不是传输表示该块的所有数据，只需要将一个运动矢量传输到接收终端。例如，如果帧 1 中的一块相似于帧 2 中的一块且在相同的屏幕坐标上，只需要为该块发送一个运动矢量 0。然而，如果帧 1 中的一块相似于帧 2 中的一块但在不同的屏面坐标上，只需要发送指示该块的新位置的一个运动矢量即可。按照一种实现，为了确定帧 1 中的块 A 是否相似于帧 2 中的块 B，确定象素值之间的绝对差值之和。和越低，块 A 与块 B 越相似（即如果和为 0，块 A 等于块 B）。

[0553] 通过包含传送、分解、分组加法、带饱和的分组减法及逻辑运算，本发明允许用分组数据执行运动估计。例如，如果两个 16×16 的视频块是用作为分组数据存储的两个 8 位象素值的数组表示的，可用下述步骤计算出这两块中的象素值的绝对差值：

[0554] 1) 利用传送指令从块 A 中取 8 个 8 位值生成 Source1；

[0555] 2) 利用传送指令从块 B 中取 8 个 8 位值生成 Source2；

[0556] 3) 执行带饱和的分组减法从 Source2 中减去 Source 1 生成 Source3- 通过带饱和的减法，Source3 中将只包含这一减法的正结果（即负结果成为 0）；

[0557] 4) 执行带饱和的分组减法从 Source1 中减去 Source2 生成 Source4- 通过带饱和的减法，Source4 中将只包含这一减法的正结果（即负结果成为 0）；

[0558] 5) 在 Source3 与 Source4 上执行分组或运算 (OR) 产生 Source5- 通过执行这一或运算，Source5 中包含 Source1 与 Source2 的绝对值；

[0559] 6) 重复这些步骤直到处理完 16×16 块。

[0560] 将得出的 8 位绝对值分解成 16 位数据元素以便允许 16 位精度，然后使用分组加法求和。

[0561] 从而，通过在处理器 109 上提供传送、分解、分组加法、带饱和的分组减法及逻辑运算，本发明比先有技术通用处理器有了明显的性能提高，后者必须一次一个数据元素地执行运动估计计算的加法与绝对差值。

[0562] 6)离散余弦变换

[0563] 离散余弦变换 (DCT) 是用在许多信号处理算法中的著名函数。尤其是视频与图象压缩算法广泛地利用这一变换。

[0564] 在图象与视频压缩算法中，使用 DCT 将一块象素从空间表示变换到频率表示。在频率表示中，将画面信息分成频率分量，某些分量比其它分量更重要。压缩算法有选择地量

化或丢弃对重构画面内容并无不利影响的频率分量。以这一方式达到压缩。

[0565] DCT 有许多实现，其中最流行的是基于快速傅里叶变换 (FFT) 计算流程建模的某种快速变换方法。在该快速变换中，将 N 阶变换分解成 N/2 阶变换的组合并重组结果。可将这一分解一直进行到到达最小的二阶变换为止。通常将这一初等二阶变换核称作蝶形运算。蝶形运算表示如下：

[0566] X = a*x+b*y

[0567] Y = c*x-d*y

[0568] 其中 a、b、c 与 d 称作系数，x 与 y 为输入数据，而 X 与 Y 则为变换输出。

[0569] 通过包含传送、乘 - 加及分组移位指令，本发明允许以下述方式使用分组数据执行 DCT 计算：

[0570] 1) 利用传送及分解指令取表示 x 与 y 的两个 16 位值生成 Source1（见下面表 51）；

[0571] 2) 如下面表 51 中所示生成 Source2- 注意 Source2 在若干次蝶形运算上可重复使用；以及

[0572] 3) 利用 Source1 与 Source2 执行乘 - 加指令生成 Result（见下面表 51）。

[0573]

x	y	x	y	源1
a	b	c	-d	源2
a·x+b·y		c·x-d·y		源3

[0574] 表 51

[0575] 在一些情况中，蝶形运算的系数为 1。对于这些情况，蝶形运算退化成只有加与减，这可以利用分组加法与分组减法指令来执行。

[0576] IEEE 文件规定了电视会议必须执行的逆 DCT 所用的精度。（见 IEEE 电路与系统协会，“8×8 逆离散余弦变换的实现的 IEEE 标准规范”，IEEE Std. 1180-1990，IEEE Inc. 345East 47th st.，NY，NY 10017，USA，1991 年 3 月 18 日）。公开的乘 - 加指令满足这一要求的精度因为它使用 16 位输入来生成 32 位输出。

[0577] 从而通过在处理器 109 上提供传送、乘 - 加及分组移位操作，本发明相比于每次必须一个数据元素地执行 DCT 计算的加法与乘法的先有技术通用处理器，本发明有了明显的性能提高。

[0578] 替代实施例

[0579] 虽然已将本发明描述成其中各个不同的运算具有独立的电路，但也能实现替代的实施例使不同运算共同某些电路。例如，在一个实施例中使用下列电路：1) 单个算术逻辑单元 (ALU) 来执行分组加法、分组减法、分组比较及分组逻辑运算；2) 一个电路单元来执行组装、分解及分组移位操作；3) 一个电路单元来执行分组乘法及乘 - 加运算；以及 4) 一个电路单元来执行个数计数操作。

[0580] 这里使用了对应与相应的名词来指称存储在两个或更多分组数据中的数据元素之间的预定关系。在一个实施例中，这一关系是基于分组数据中的数据元素的位位置的。

例如，第一分组数据的数据元素 0（例如以分组字节格式存储在位位置 0-7 中）对应于第二分组数据的数据元素 0（例如以分组字节格式存储在位位置 0-7 中）。然而，在不同的实施例中这一关系可以不同。例如，第一与第二分组数据中的对应数据元素可能具有不同长度。作为另一示例，不是第一分组数据的最低位数据元素对应于第二分组数据的最低位数据元素（及以此类推），而且第一与第二分组数据中的数据元素可以以某种其它次序互相对应。作为另一示例，第一与第二分组数据中的数据元素不是具有一一对应，而数据元素可在不同的比率上对应（例如，第一分组数据的一个或多个数据元素可对应于第二分组数据中的两个或更多不同数据元素）。

[0581] 虽已用若干实施例描述了本发明，熟悉本技术者将理解本发明不限于所述实施例。可在所附权利要求书的精神与范围内修改或改变来实现本发明的方法与装置。因此，该说明书应认为是示例性的而不是对本发明的限制。

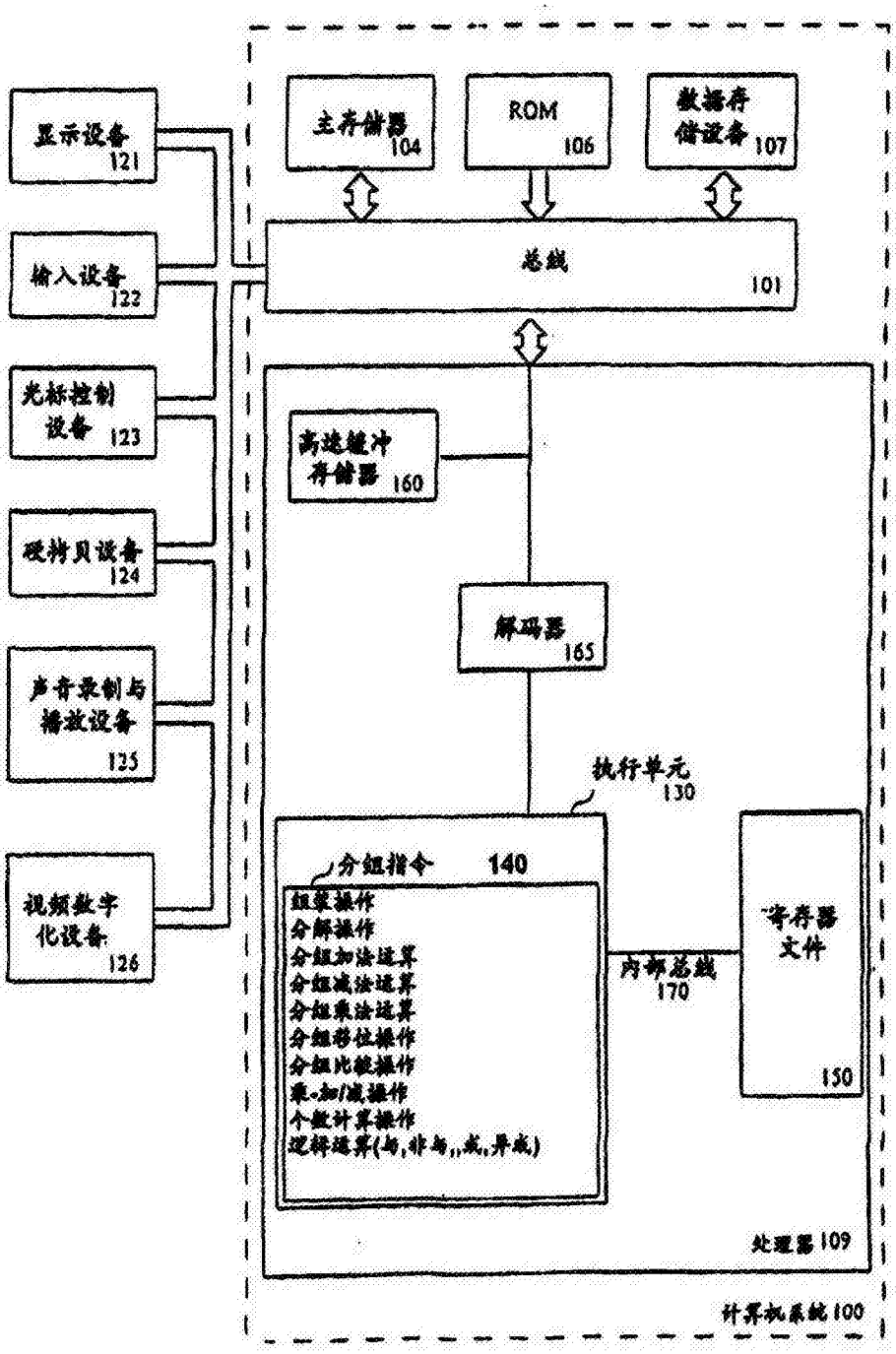

显示设备 121
输入设备 122
光标控制设备 123
硬拷贝设备 124
声音录制与播放设备 125
视频数字化设备 126
主存储器 104
ROM 106
数据存储设备 107
总线 101
高速缓冲存储器 160
解码器 165
执行单元 130
分组指令 140
组聚操作
分解操作
分组加法运算
分组减法运算
分组乘法运算
分组移位操作
分组比较操作
乘-加/减操作
个数计算操作
逻辑运算(与,非与,,或,,异或)
内部总线 170
寄存器文件 150
处理器 109
计算机系统 100

图 1

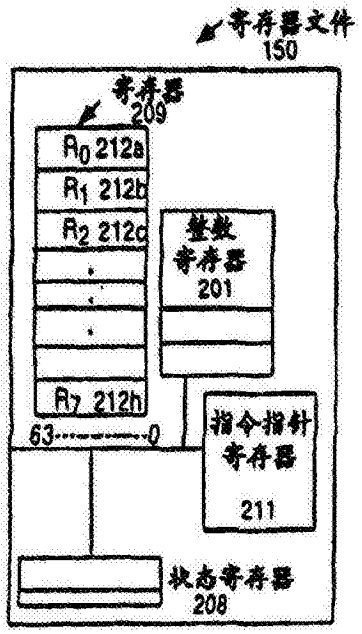

寄存器文件
150
寄存器
209
R0 212a
R1 212b
R2 212c
R7 212h
整数
寄存器
201
63.....0
指令指针
寄存器
211
状态寄存器
208

图 2

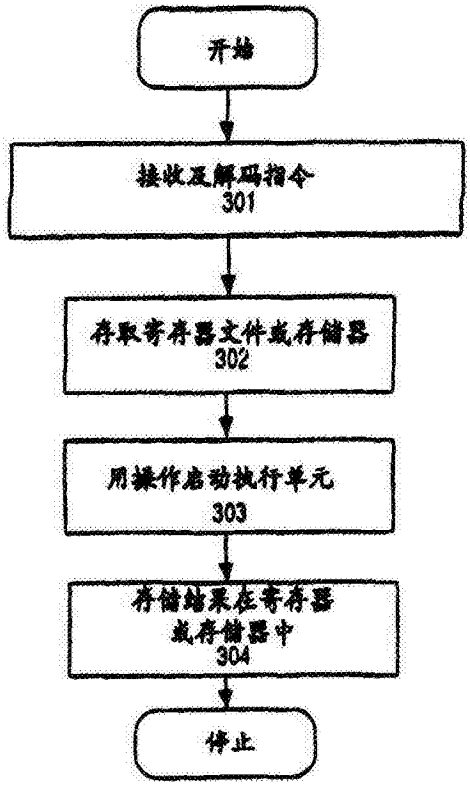

开始
接收及解码指令
301
存取寄存器文件或存储器
302
用操作启动执行单元
303
存储结果在寄存器
或存储器中
304
停止

图 3

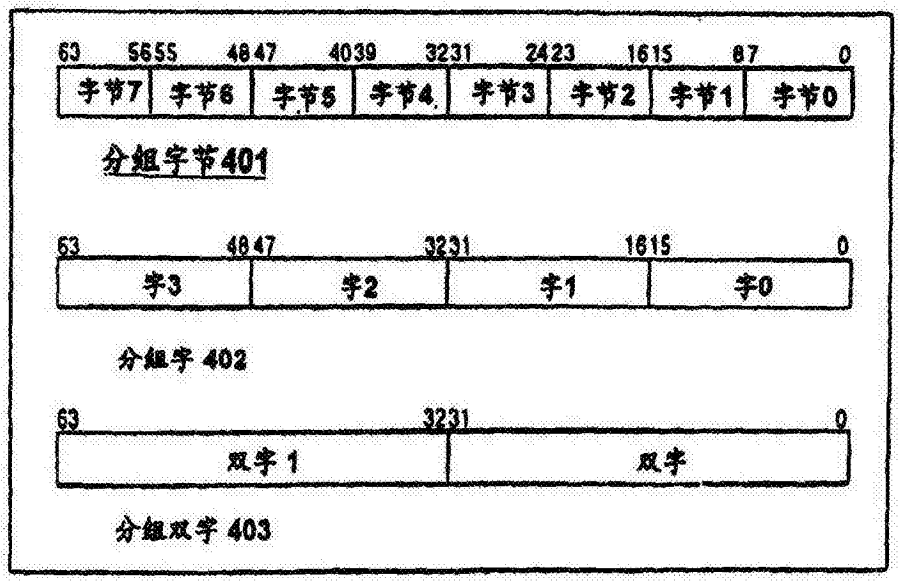

63 5655 4847 4039 3231 2423 1615 87 0
字节7 字节6 字节5 字节4 字节3 字节2 字节1 字节0
分组字节401
63 4847 3231 1615 0
字3 字2 字1 字0
分组字 402
63 3231 0
双字 1 双字
分组双字 403

图 4

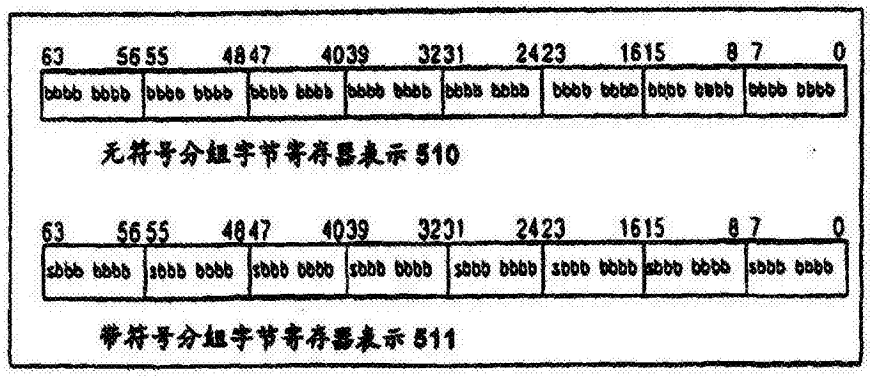

63 56 55 48 47 40 39 32 31 24 23 16 15 8 7 0
bbbb bbbb bbbb bbbb bbbb bbbb bbbb bbbb bbbb bbbb bbbb bbbb bbbb bbbb bbbb bbbb
无符号分组字节寄存器表示 510
63 56 55 48 47 40 39 32 31 24 23 16 15 8 7 0
sbbb bbbb sbbb bbbb sbbb bbbb sbbb bbbb sbbb bbbb sbbb bbbb sbbb bbbb sbbb bbbb
带符号分组字节寄存器表示 511

图 5a

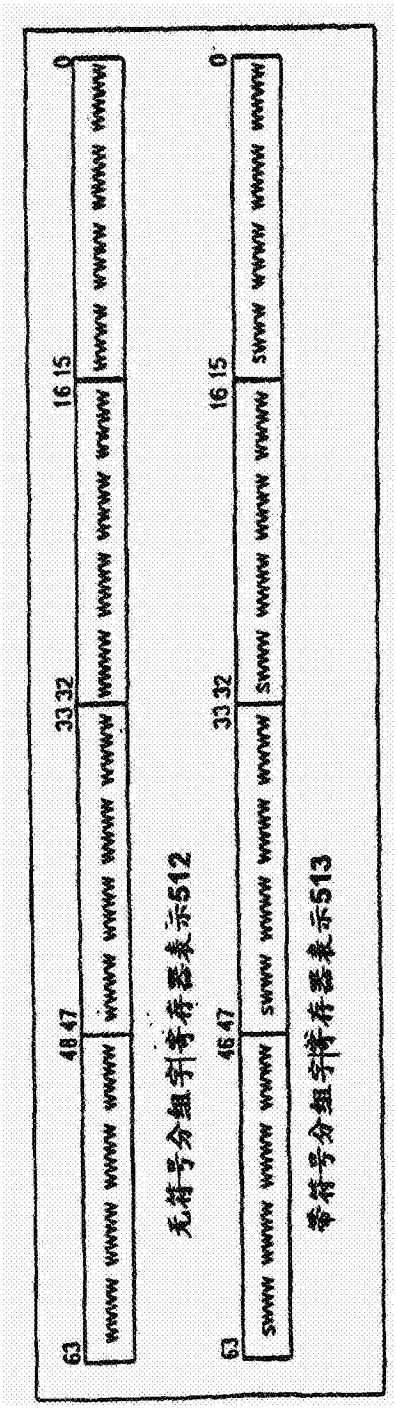

63
48 47
33 32
16 15
0
无符号分组字寄存器表示512
63
46 47
33 32
16 15
0
带符号分组字寄存器表示513

图 5b

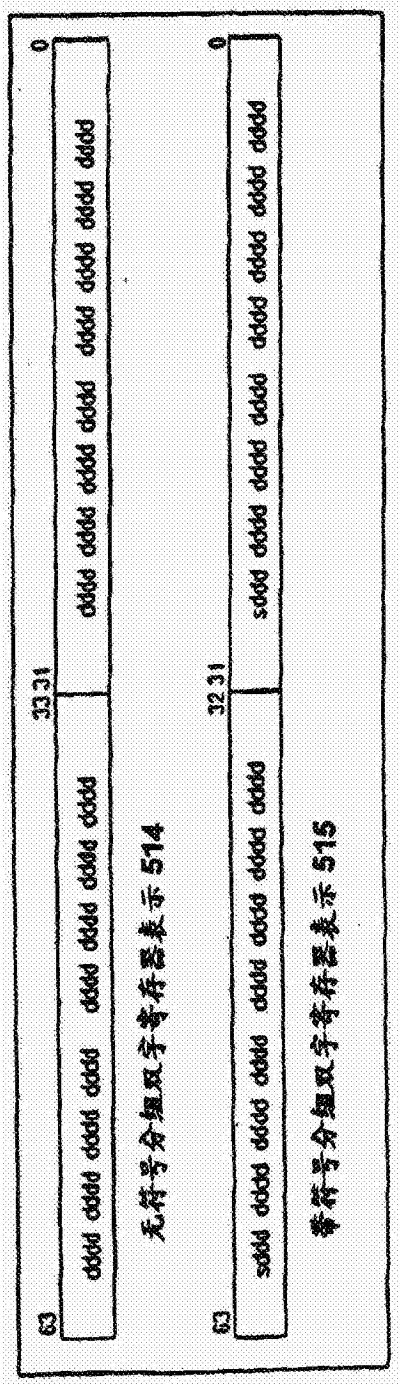

63
33 31
0
无符号分组双字寄存器表示 514
63
32 31
0
带符号分组双字寄存器表示 515

图 5c

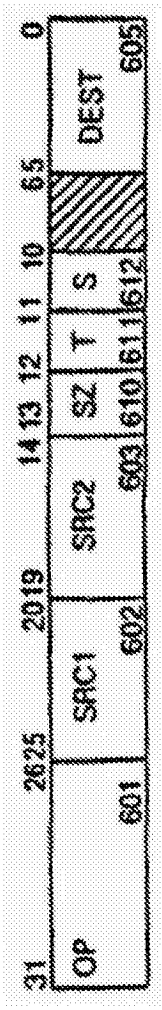

31
26 25
OP
601
20 19
SRC1
602
14 13
SRC2
603
12
S2
610
11
T
611
10
S
612
6 5
0
DEST
605

图 6a

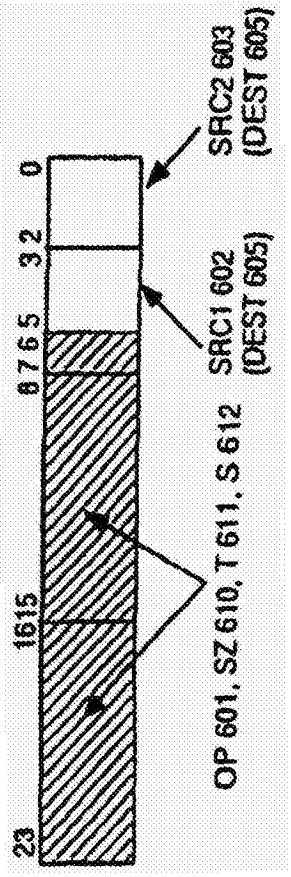

SRC2 603
(DEST 605)
SRC1 602
(DEST 605)
OP 601, SZ 610, T 611, S 612
0
32
8765
1615
23

图 6b

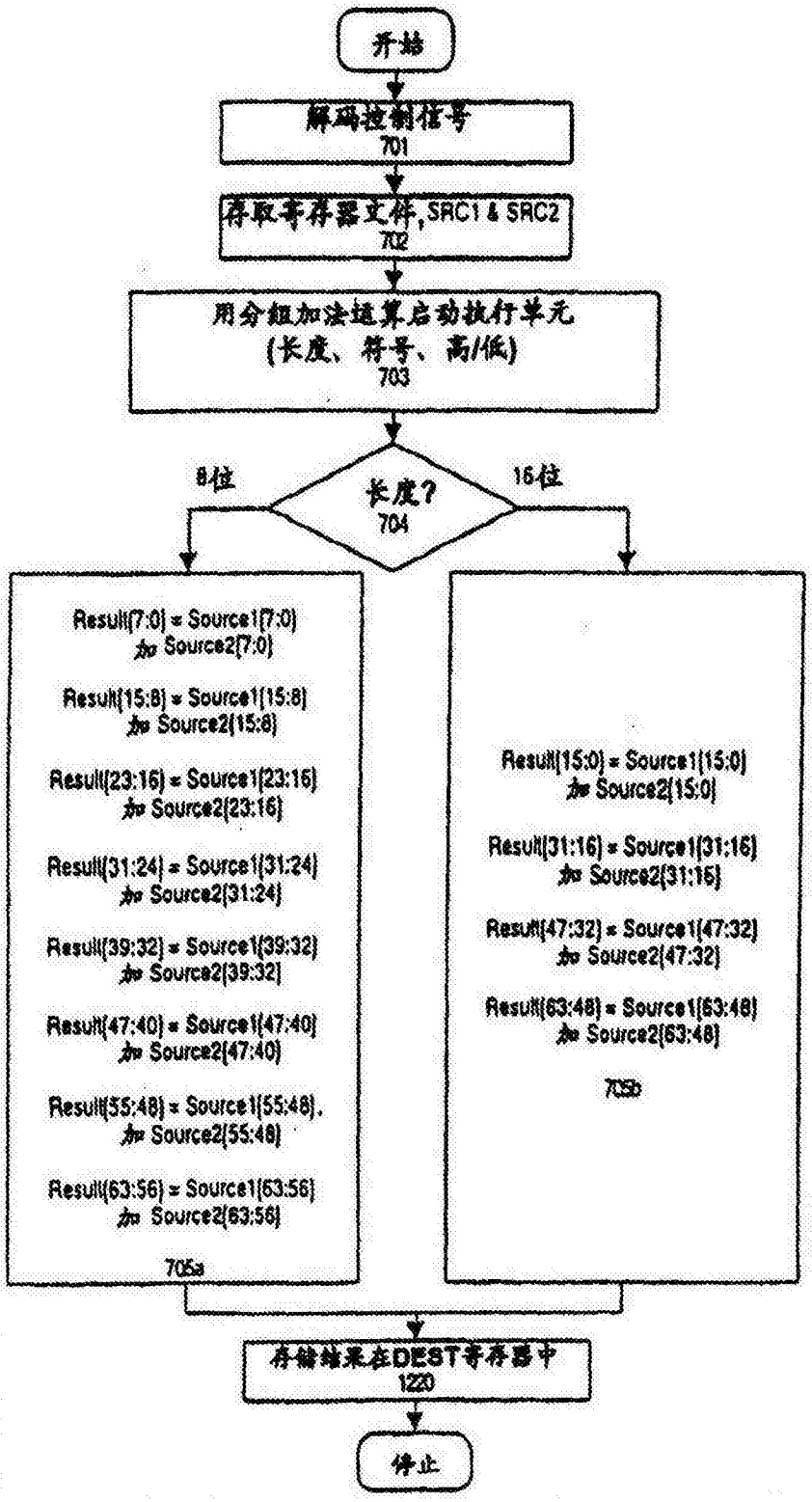

开始
解码控制信号
701
存取寄存器文件, SRC1 & SRC2
702
用分组加法运算启动执行单元
(长度、符号、高/低)
703
长度?
704
8位
16位
Result[7:0] = Source1[7:0]
加 Source2[7:0]
Result[15:8] = Source1[15:8]
加 Source2[15:8]
Result[23:16] = Source1[23:16]
加 Source2[23:16]
Result[31:24] = Source1[31:24]
加 Source2[31:24]
Result[39:32] = Source1[39:32]
加 Source2[39:32]
Result[47:40] = Source1[47:40]
加 Source2[47:40]
Result[55:48] = Source1[55:48].
加 Source2[55:48]
Result[63:56] = Source1[63:56]
加 Source2[63:56]
705a
Result[15:0] = Source1[15:0]
加 Source2[15:0]
Result[31:16] = Source1[31:16]
加 Source2[31:16]
Result[47:32] = Source1[47:32]
加 Source2[47:32]
Result[63:48] = Source1[63:48]
加 Source2[63:48]
705b
存储结果在DEST寄存器中
1220
停止

图 7a

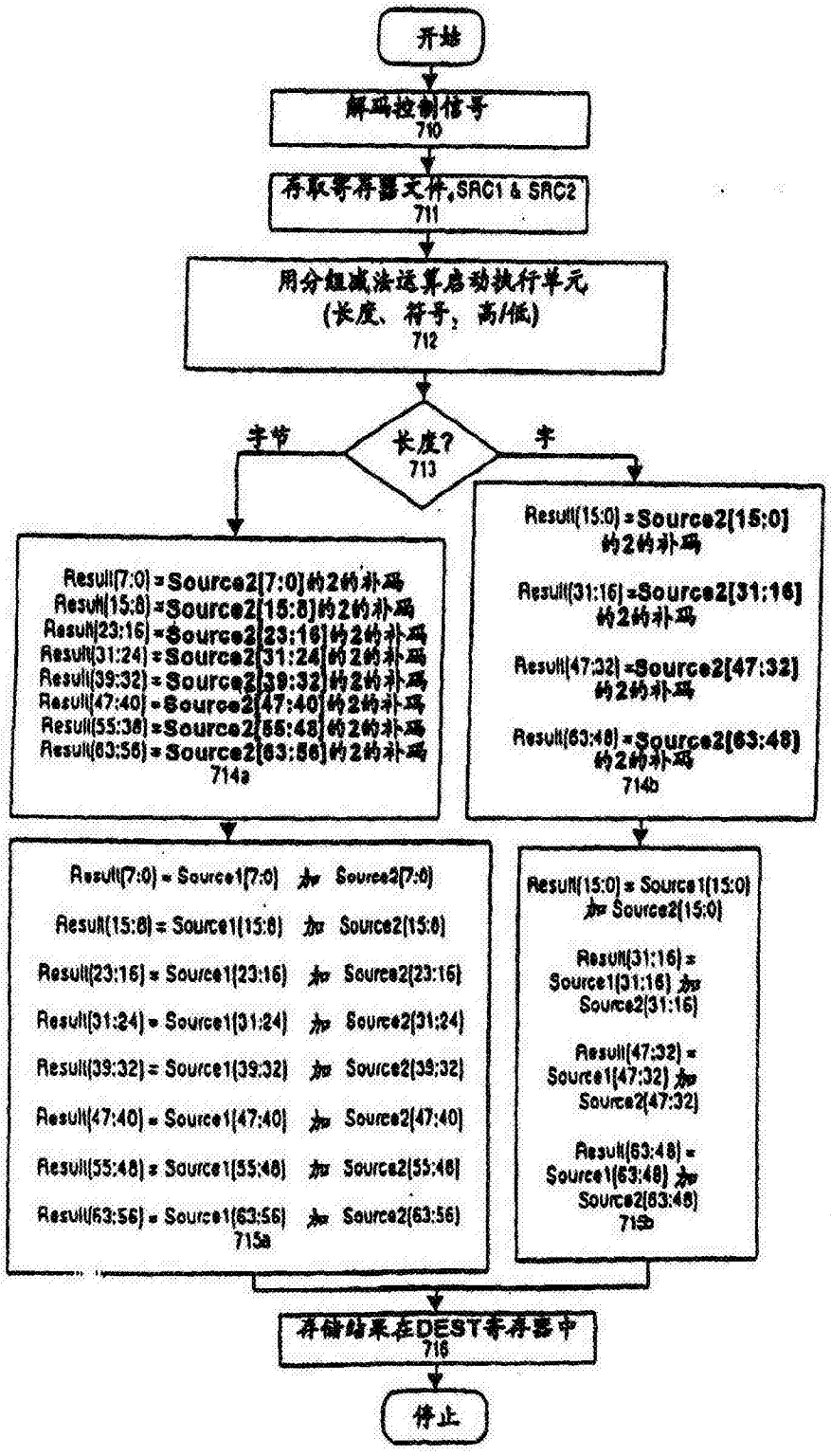

开始
解码控制信号
710
存取寄存器文件,SRC1 & SRC2
711
用分组减法运算启动执行单元
(长度、符号，高/低)
712
长度?
713
字节
字
Result[7:0] = Source2[7:0]的2的补码
Result[15:8] = Source2[15:8]的2的补码
Result[23:16] = Source2[23:16]的2的补码
Result[31:24] = Source2[31:24]的2的补码
Result[39:32] = Source2[39:32]的2的补码
Result[47:40] = Source2[47:40]的2的补码
Result[55:48] = Source2[55:48]的2的补码
Result[63:56] = Source2[63:56]的2的补码
714a
Result[15:0] = Source2[15:0]的2的补码
Result[31:16] = Source2[31:16]的2的补码
Result[47:32] = Source2[47:32]的2的补码
Result[63:48] = Source2[63:48]的2的补码
714b
Result[7:0] = Source1[7:0] 加 Source2[7:0]
Result[15:8] = Source1[15:8] 加 Source2[15:8]
Result[23:16] = Source1[23:16] 加 Source2[23:16]
Result[31:24] = Source1[31:24] 加 Source2[31:24]
Result[39:32] = Source1[39:32] 加 Source2[39:32]
Result[47:40] = Source1[47:40] 加 Source2[47:40]
Result[55:48] = Source1[55:48] 加 Source2[55:48]
Result[63:56] = Source1[63:56] 加 Source2[63:56]
715a
Result[15:0] = Source1[15:0] 加 Source2[15:0]
Result[31:16] = Source1[31:16] 加 Source2[31:16]
Result[47:32] = Source1[47:32] 加 Source2[47:32]
Result[63:48] = Source1[63:48] 加 Source2[63:48]
715b
存储结果在DEST寄存器中
716
停止

图 7b

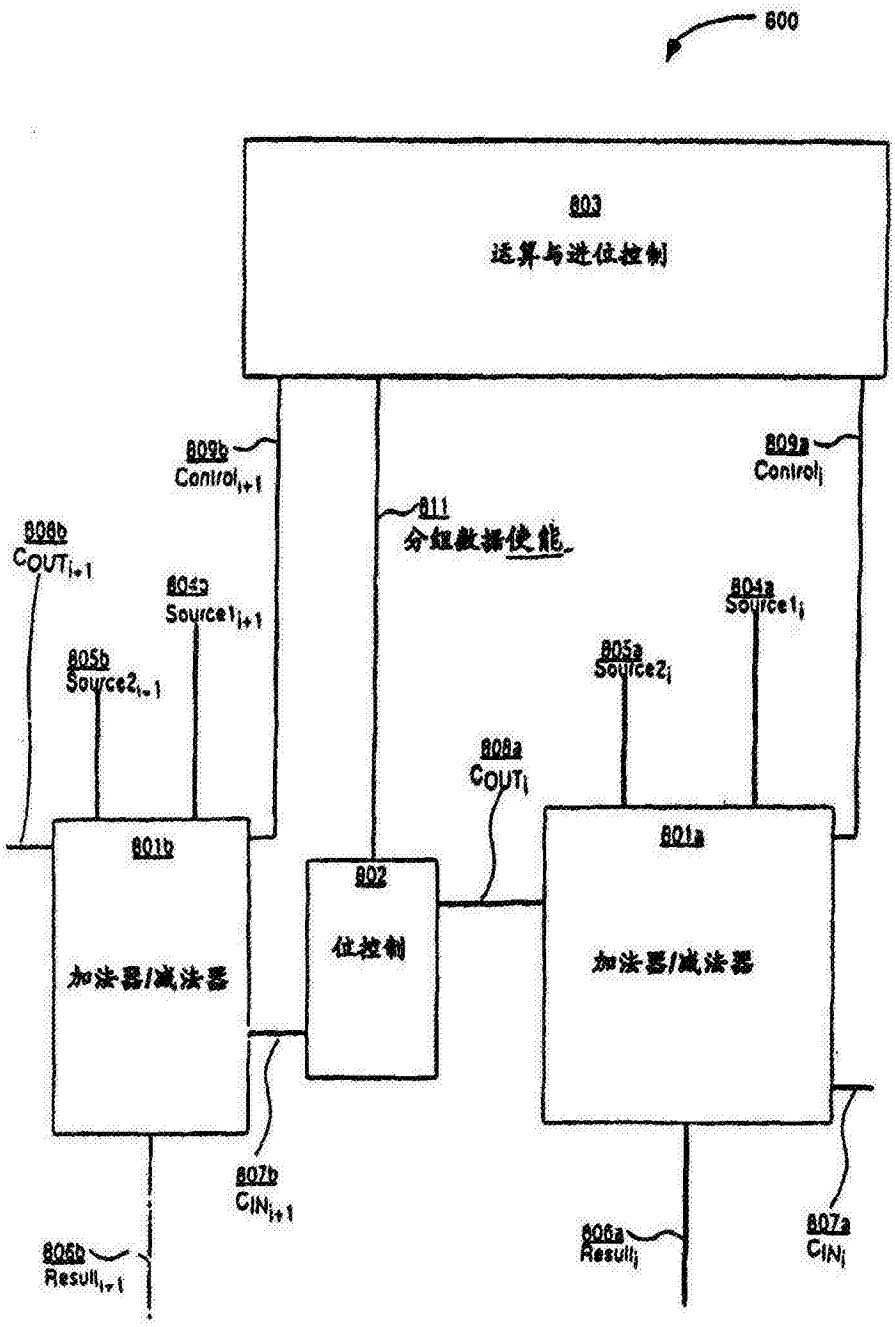

800
803
运算与进位控制
809b
Control$_{i+1}$
809a
Control$_i$
811
分组数据使能
808b
COUT$_{i+1}$
804b
Source1$_{i+1}$
804a
Source1$_i$
805b
Source2$_{i+1}$
805a
Source2$_i$
808a
COUT$_i$
801b
加法器/减法器
802
位控制
801a
加法器/减法器
807b
CIN$_{i+1}$
806b
Result$_{i+1}$
806a
Result$_i$
807a
CIN$_i$

图 8

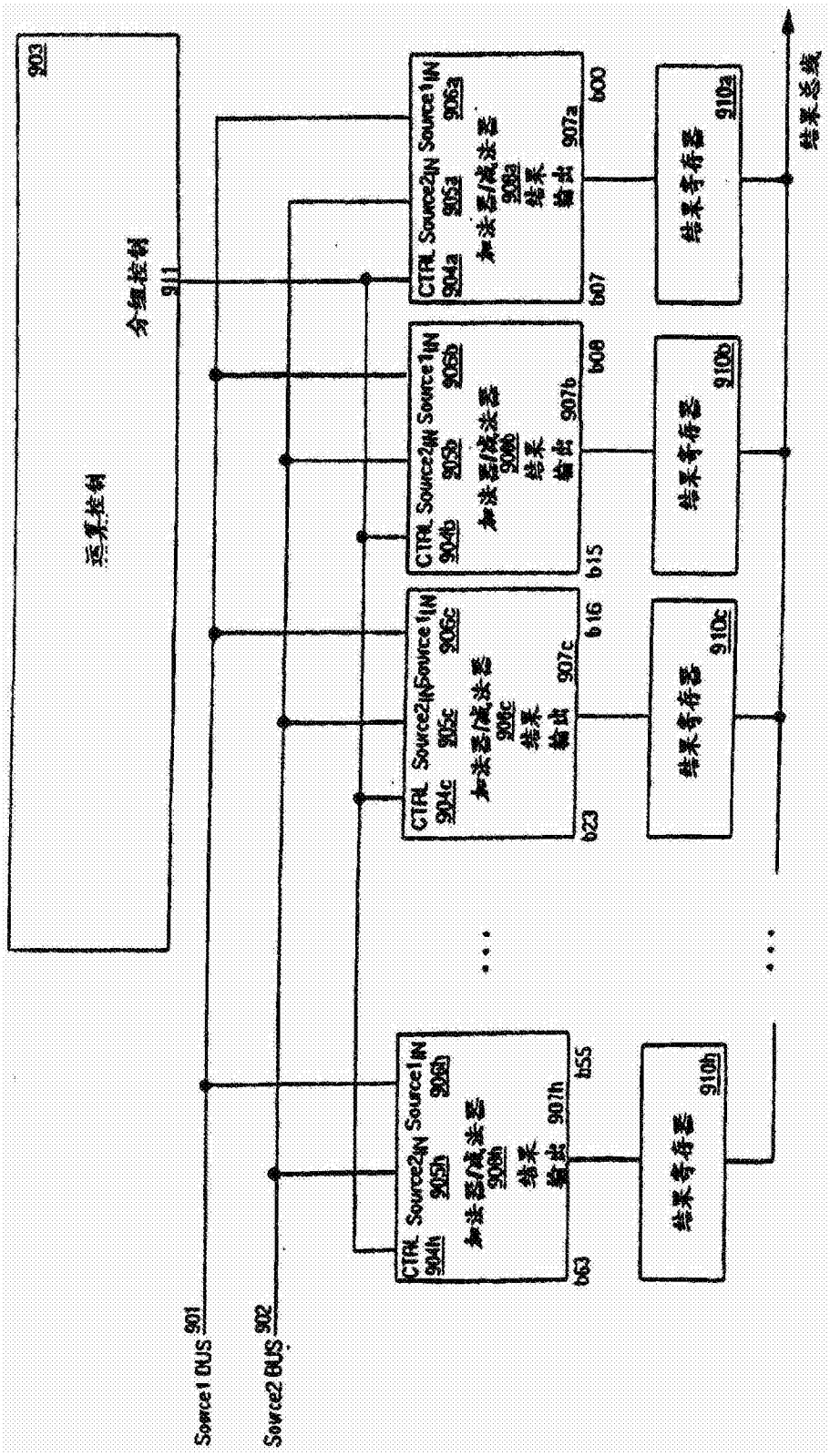

903
运算控制
分组控制
911
CTRL Source2/N Source1/N
904a 905a
加法器/减法器
906a
结果输出
907a
b00
b07
结果寄存器
910a
结果总线
CTRL Source2/N Source1/N
904b 905b
加法器/减法器
906b
结果输出
907b
b08
b15
结果寄存器
910b
CTRL Source2/N Source1/N
904c 905c
加法器/减法器
906c
结果输出
907c
b16
b23
结果寄存器
910c
CTRL Source2/N Source1/N
904h 905h
加法器/减法器
906h
结果输出
907h
b55
b63
结果寄存器
910h
Source1 BUS 901
Source2 BUS 902

图 9

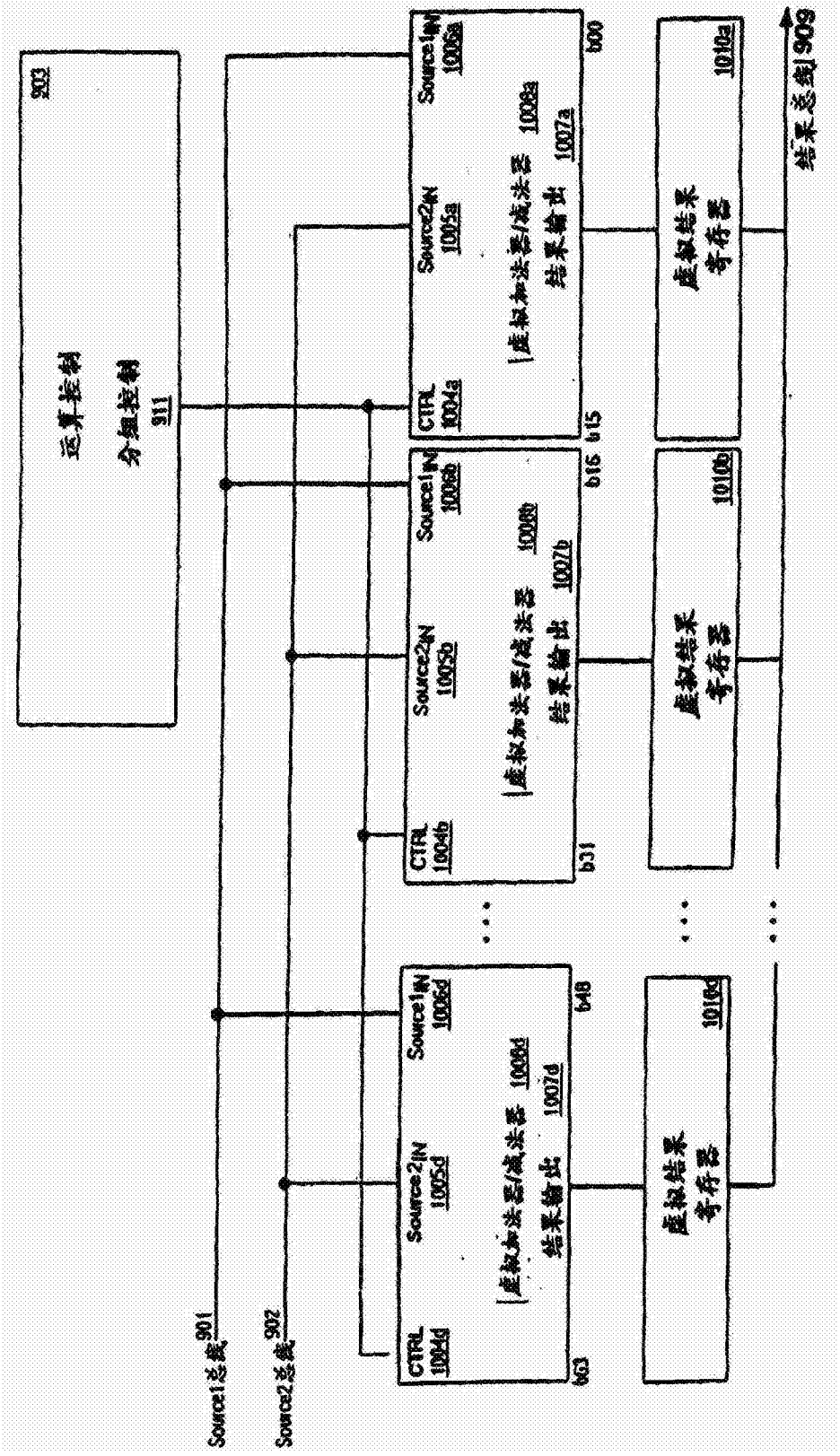

903
运算控制
分组控制
911
Source1总线 901
Source2总线 902
结果总线 909
CTRL 1004a
Source1[N] 1006a
Source2[N] 1005a
虚拟加法器/减法器 1008a
结果输出 1007a
虚拟结果寄存器 1010a
b00
b15
CTRL 1004b
Source1[N] 1006b
Source2[N] 1005b
虚拟加法器/减法器 1008b
结果输出 1007b
虚拟结果寄存器 1010b
b16
b31
CTRL 1004d
Source1[N] 1006d
Source2[N] 1005d
虚拟加法器/减法器 1008d
结果输出 1007d
虚拟结果寄存器 1010d
b48
b63

图 10

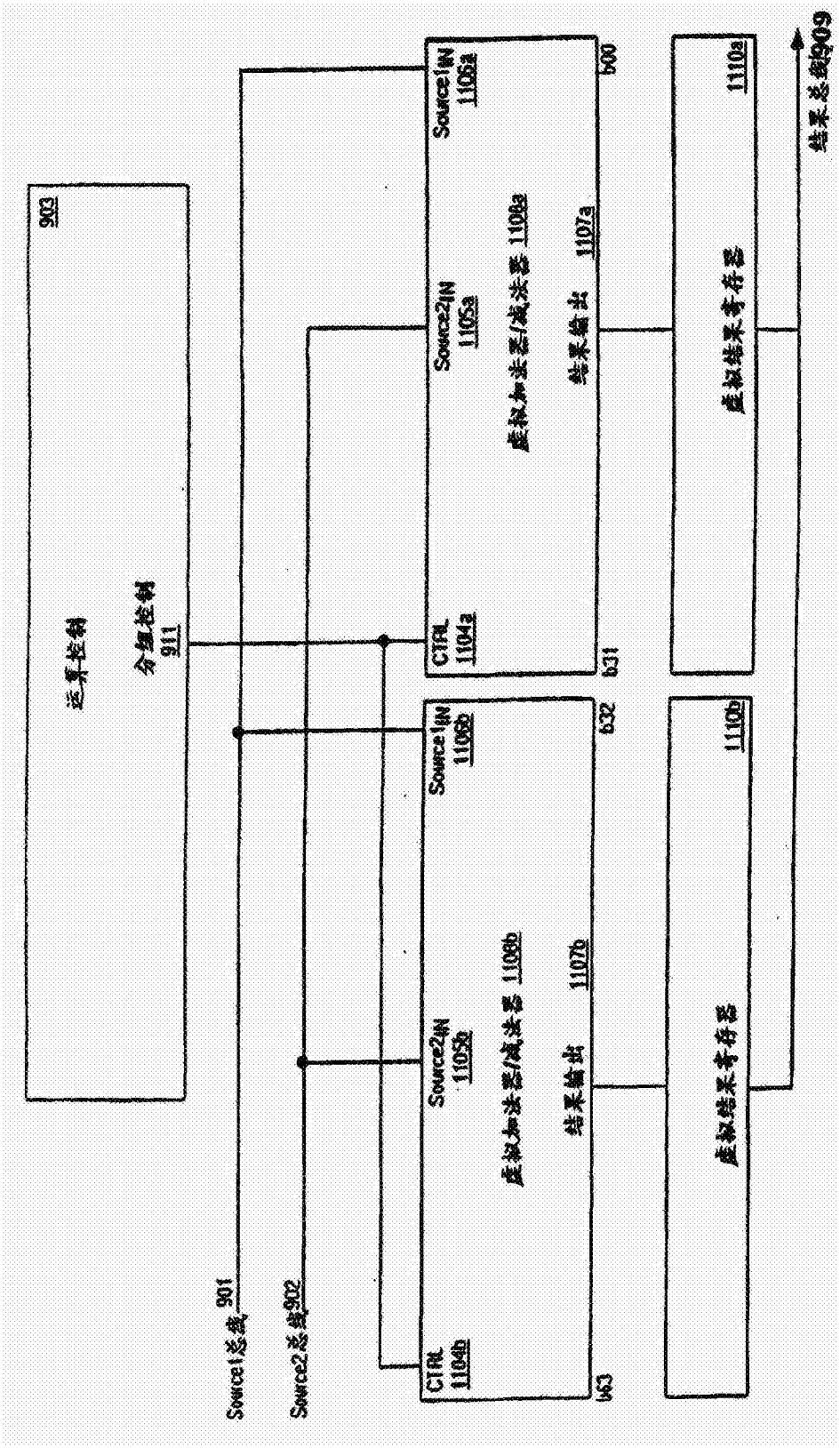

903
运算控制
分组控制
911
Source1总线901
Source2总线902
CTRL
1104a
Source1IN
1106a
Source2IN
1105a
虚拟加法器/减法器 1108a
结果输出 1107a
600
631
虚拟结果寄存器
1110a
结果总线909
CTRL
1104b
Source1IN
1106b
Source2IN
1105b
虚拟加法器/减法器 1108b
结果输出 1107b
632
633
虚拟结果寄存器
1110b

图 11

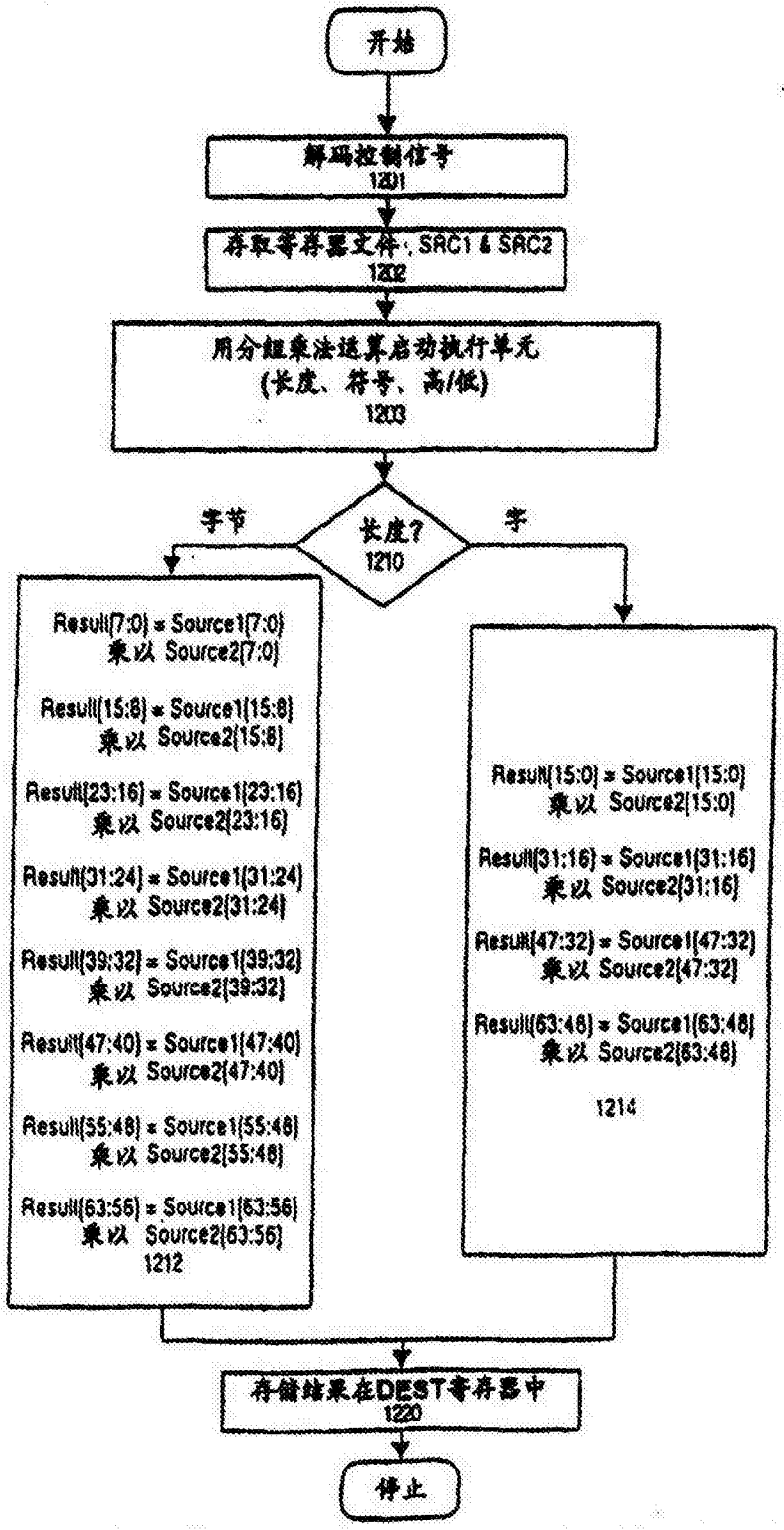

开始
解码控制信号
1201
存取寄存器文件：SRC1 & SRC2
1202
用分组乘法运算启动执行单元
(长度、符号、高/低)
1203
长度?
1210
字节
字
Result(7:0) = Source1(7:0)
乘以 Source2(7:0)
Result(15:8) = Source1(15:8)
乘以 Source2(15:8)
Result(23:16) = Source1(23:16)
乘以 Source2(23:16)
Result(31:24) = Source1(31:24)
乘以 Source2(31:24)
Result(39:32) = Source1(39:32)
乘以 Source2(39:32)
Result(47:40) = Source1(47:40)
乘以 Source2(47:40)
Result(55:48) = Source1(55:48)
乘以 Source2(55:48)
Result(63:56) = Source1(63:56)
乘以 Source2(63:56)
1212
Result(15:0) = Source1(15:0)
乘以 Source2(15:0)
Result(31:16) = Source1(31:16)
乘以 Source2(31:16)
Result(47:32) = Source1(47:32)
乘以 Source2(47:32)
Result(63:48) = Source1(63:48)
乘以 Source2(63:48)
1214
存储结果在DEST寄存器中
1220
停止

图 12

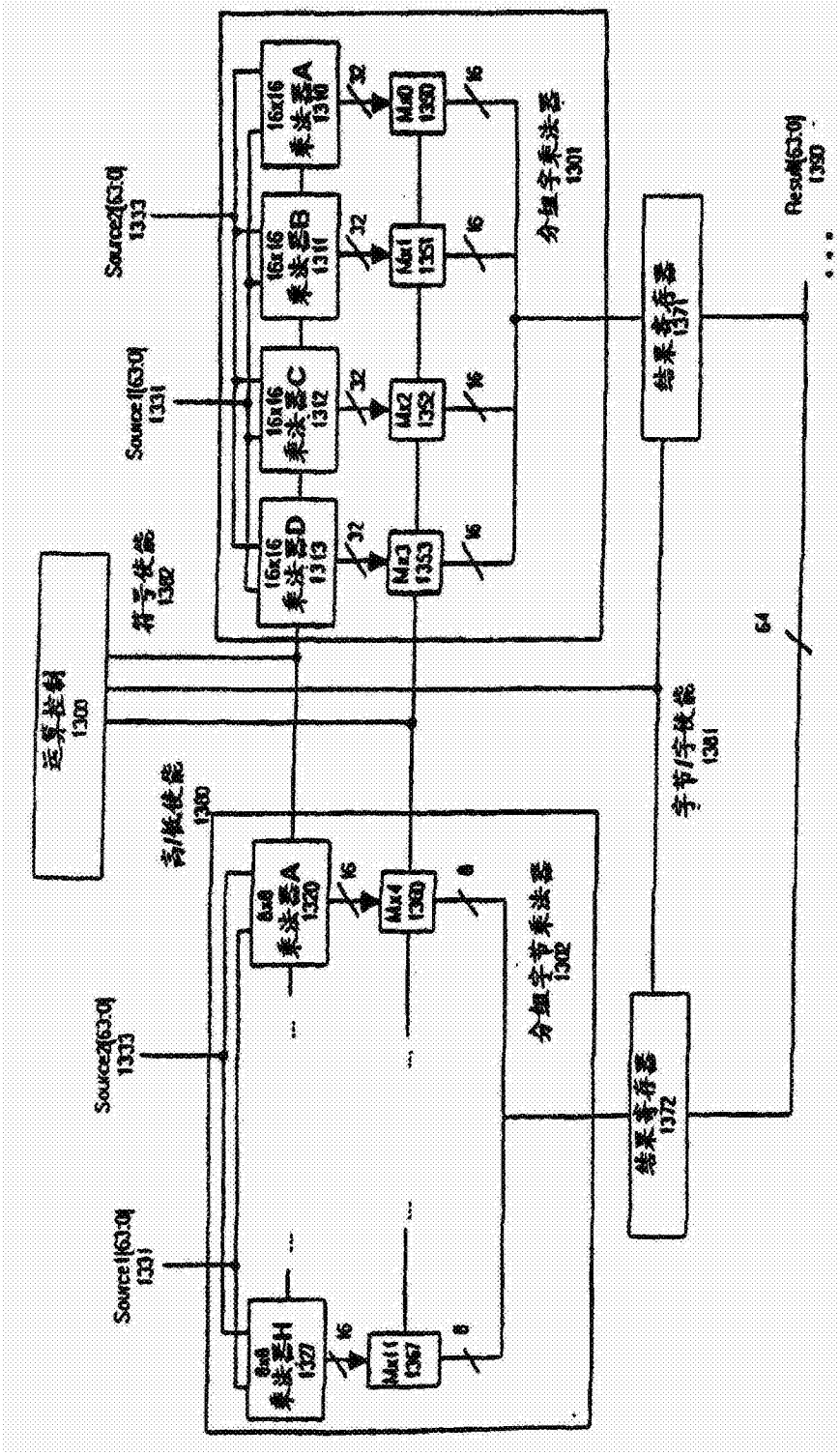

运算控制
1300
符号使能
1302
字节/字使能
1301
16x16
乘法器A
1310
16x16
乘法器B
1311
16x16
乘法器C
1312
16x16
乘法器D
1313
Mux0
1350
Mux1
1351
Mux2
1352
Mux3
1353
分组字乘法器
1301
结果寄存器
1370
Result[63:0]
1350
Source2[63:0]
1333
Source1[63:0]
1331
字节/字使能
1300
8x8
乘法器A
1320
8x8
乘法器H
1327
Mux4
1360
Mux11
1367
分组字节乘法器
1302
结果寄存器
1372
Source2[63:0]
1333
Source1[63:0]
1331
64

图 13

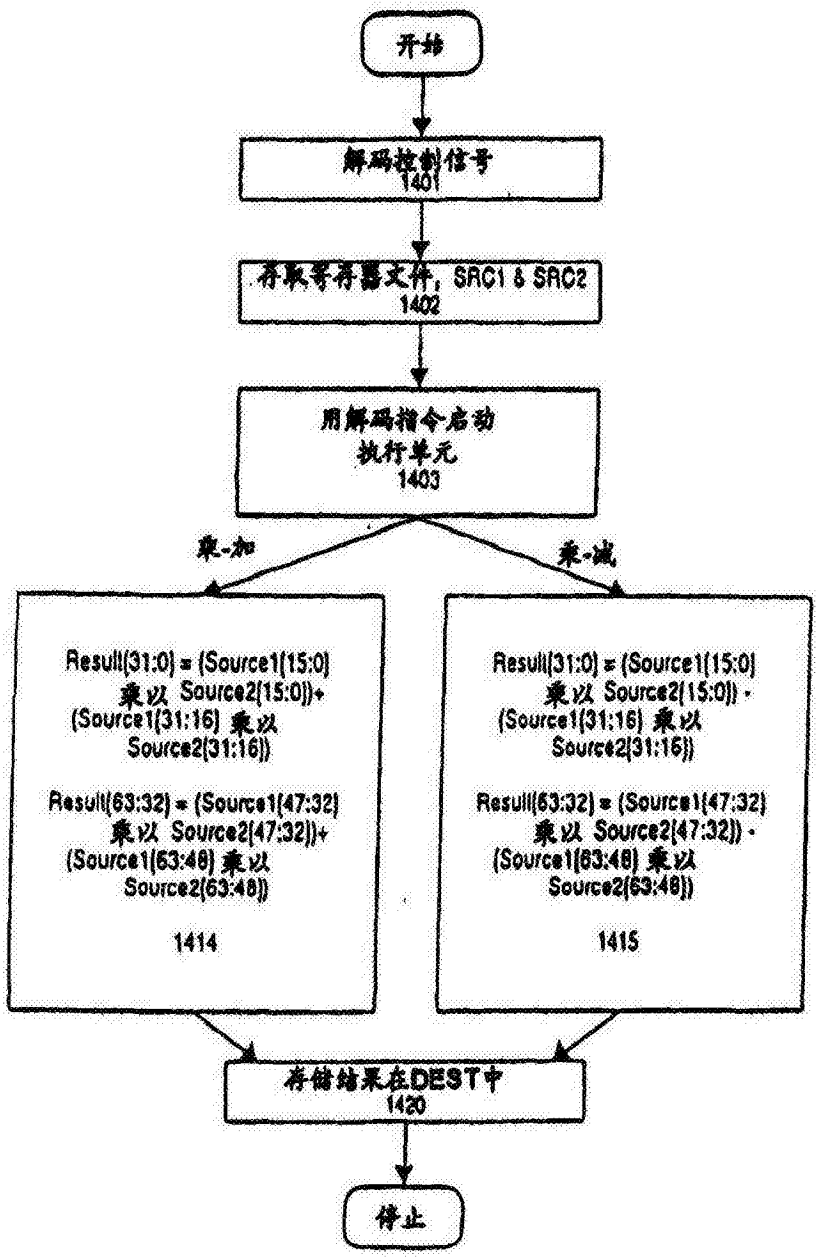

开始
解码控制信号
1401
存取寄存器文件，SRC1 & SRC2
1402
用解码指令启动
执行单元
1403
乘-加
乘-减
Result[31:0] = (Source1[15:0]
乘以 Source2[15:0])+
(Source1[31:16] 乘以
Source2[31:16])
Result[63:32] = (Source1[47:32]
乘以 Source2[47:32])+
(Source1[63:48] 乘以
Source2[63:48])
1414
Result[31:0] = (Source1[15:0]
乘以 Source2[15:0]) -
(Source1[31:16] 乘以
Source2[31:16])
Result[63:32] = (Source1[47:32]
乘以 Source2[47:32]) -
(Source1[63:48] 乘以
Source2[63:48])
1415
存储结果在DEST中
1420
停止

图 14

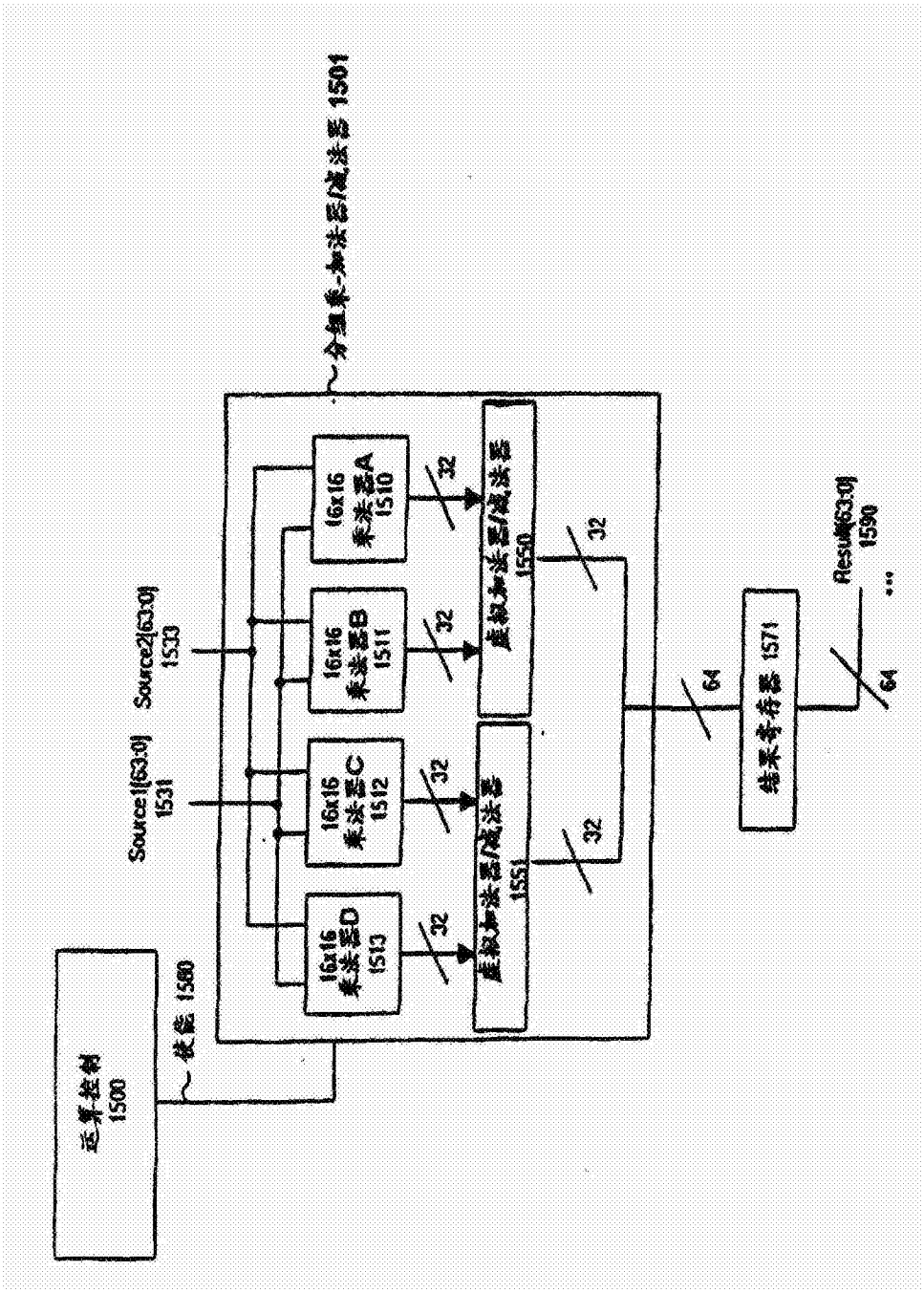

分组乘-加法器/减法器 1501
16x16 乘法器A 1510
16x16 乘法器B 1511
16x16 乘法器C 1512
16x16 乘法器D 1513
32
32
32
32
虚拟加法器/减法器 1550
虚拟加法器/减法器 1551
32
32
64
结果寄存器 1571
64
Result[63:0] 1590
...
Source2[63:0] 1533
Source1[63:0] 1531
使能 1580
运算控制 1500

图 15

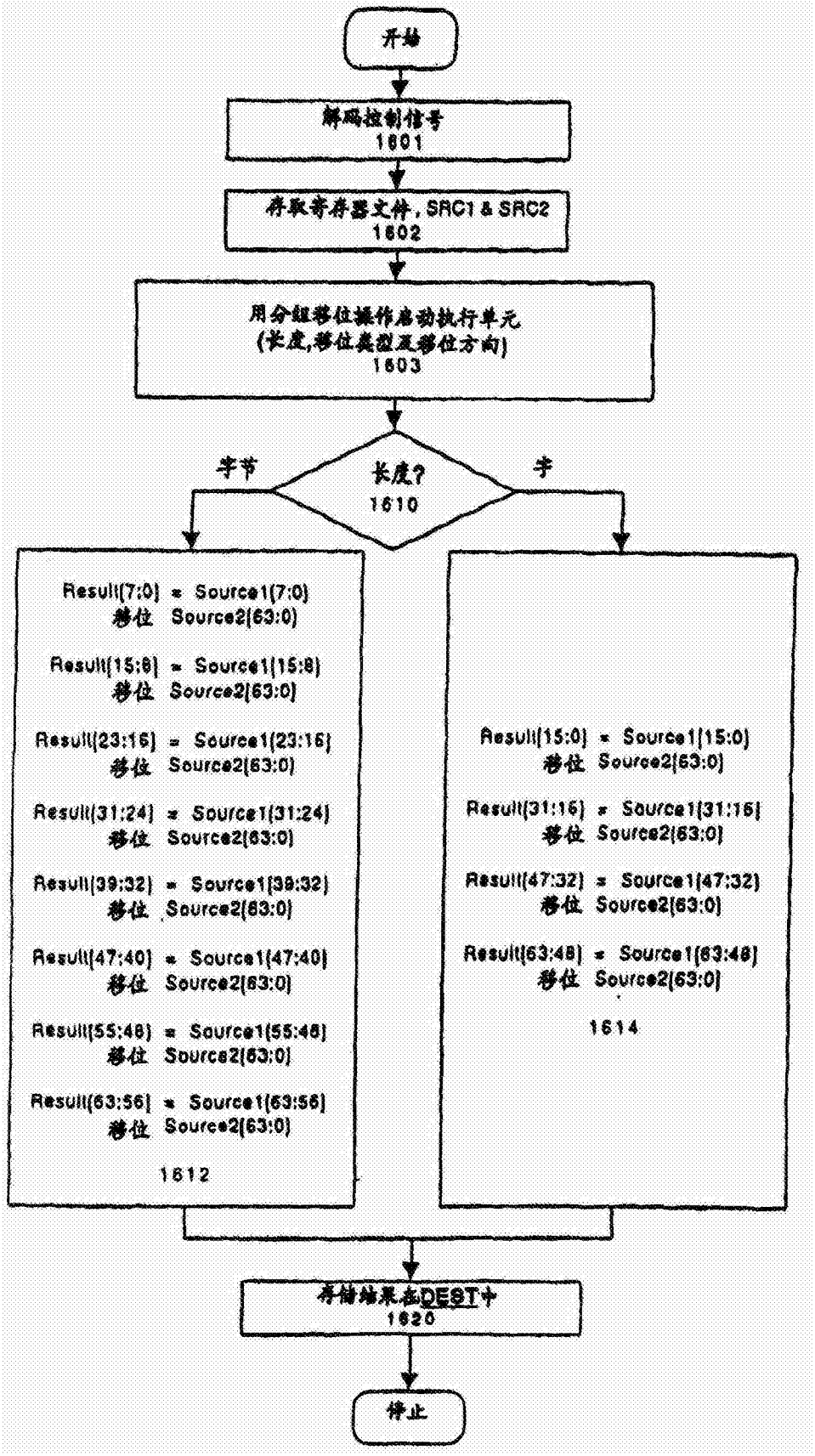

开始
解码控制信号
1601
存取寄存器文件，SRC1 & SRC2
1602
用分组移位操作启动执行单元
(长度,移位类型及移位方向)
1603
字节
长度?
1610
字
Result[7:0] = Source1[7:0]
移位 Source2[63:0]
Result[15:8] = Source1[15:8]
移位 Source2[63:0]
Result[23:16] = Source1[23:16]
移位 Source2[63:0]
Result[31:24] = Source1[31:24]
移位 Source2[63:0]
Result[39:32] = Source1[39:32]
移位 Source2[63:0]
Result[47:40] = Source1[47:40]
移位 Source2[63:0]
Result[55:48] = Source1[55:48]
移位 Source2[63:0]
Result[63:56] = Source1[63:56]
移位 Source2[63:0]
1612
Result[15:0] = Source1[15:0]
移位 Source2[63:0]
Result[31:16] = Source1[31:16]
移位 Source2[63:0]
Result[47:32] = Source1[47:32]
移位 Source2[63:0]
Result[63:48] = Source1[63:48]
移位 Source2[63:0]
1614
存储结果在DEST中
1620
停止

图 16

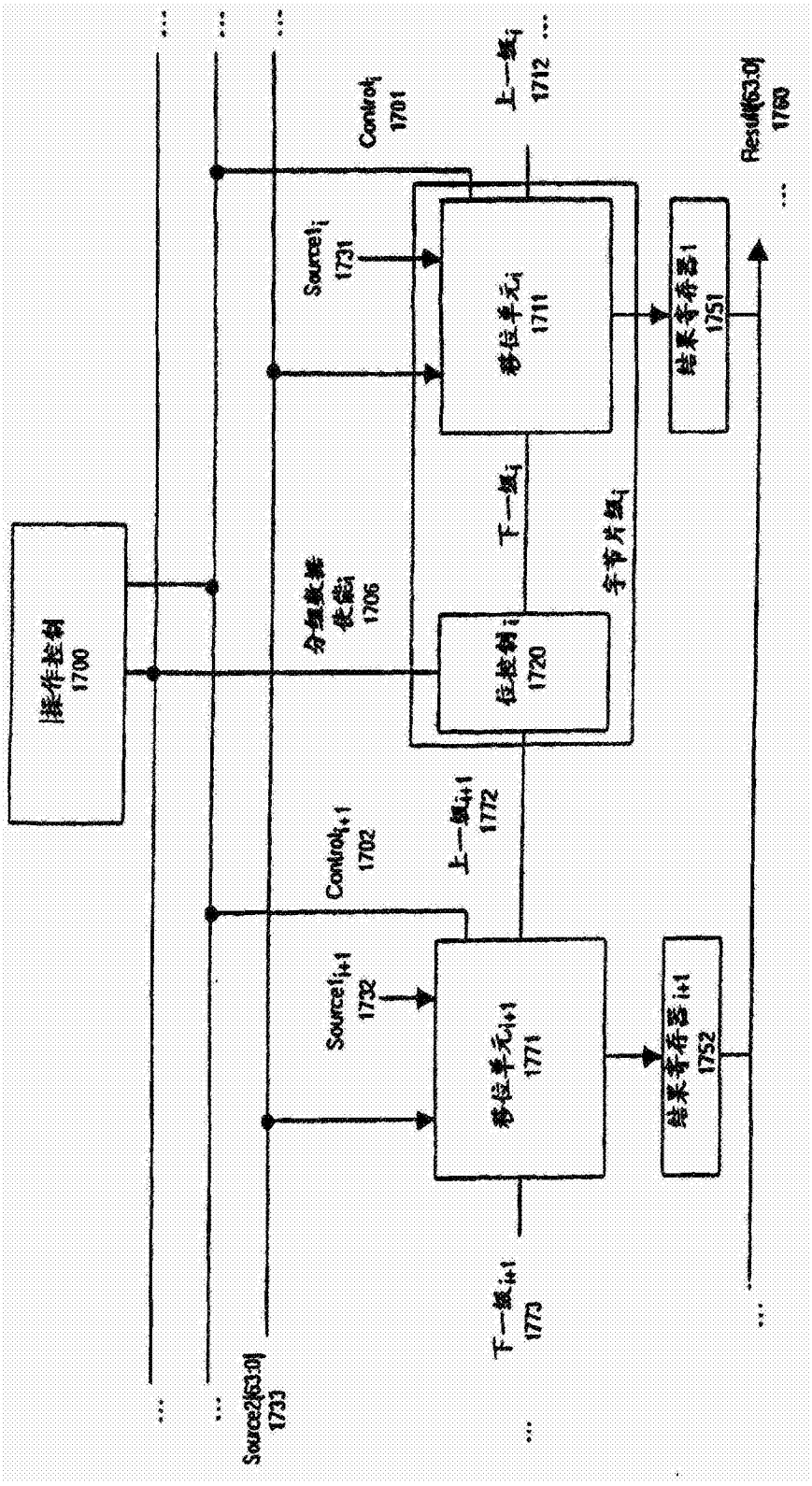

操作控制
1700
Control$_i$
1701
上一组$_i$
1712
Result[63:0]
1760
Source1$_i$
1731
移位单元$_i$
1711
结果寄存器1
1751
下一组$_i$
分组数据
使能$_i$
1706
字节片组$_i$
位控制 i
1720
Control$_{i+1}$
1702
上一组$_{i+1}$
1772
Source1$_{i+1}$
1732
移位单元$_{i+1}$
1771
结果寄存器 i+1
1752
下一组$_{i+1}$
1773
Source2[63:0]
1733

图 17

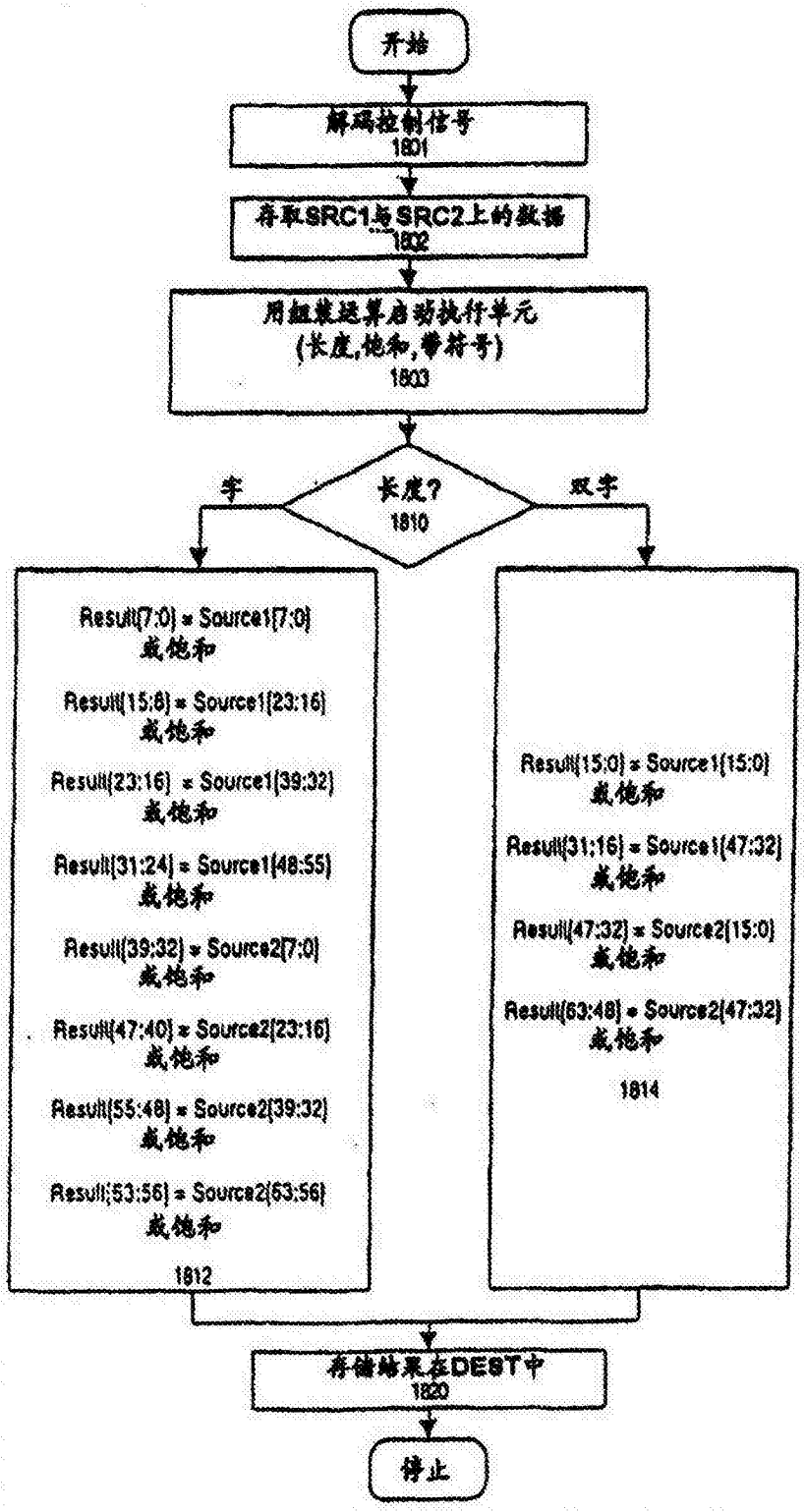

开始
解码控制信号
1801
存取SRC1与SRC2上的数据
1802
用组装运算启动执行单元
(长度,饱和,带符号)
1803
长度?
1810
字
双字
Result[7:0] = Source1[7:0]
或饱和
Result[15:8] = Source1[23:16]
或饱和
Result[23:16] = Source1[39:32]
或饱和
Result[31:24] = Source1[48:55]
或饱和
Result[39:32] = Source2[7:0]
或饱和
Result[47:40] = Source2[23:16]
或饱和
Result[55:48] = Source2[39:32]
或饱和
Result[63:56] = Source2[63:56]
或饱和
1812
Result[15:0] = Source1[15:0]
或饱和
Result[31:16] = Source1[47:32]
或饱和
Result[47:32] = Source2[15:0]
或饱和
Result[63:48] = Source2[47:32]
或饱和
1814
存储结果在DEST中
1820
停止

图 18

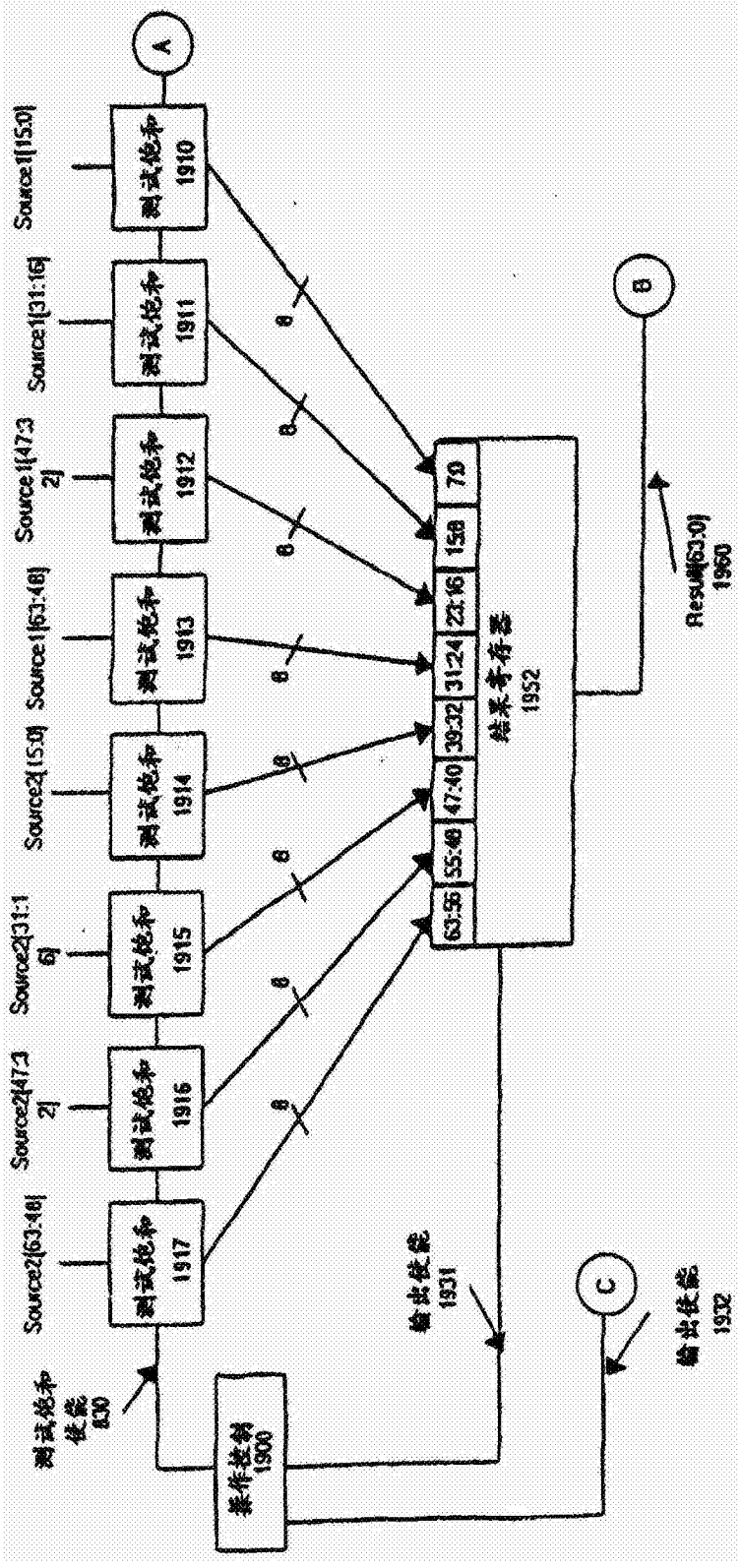

A
B
C
Source1[15:0]
Source1[31:16]
Source1[47:32]
Source1[63:48]
Source2[15:0]
Source2[31:16]
Source2[47:32]
Source2[63:48]
测试使能 830
测试比较 1910
测试比较 1911
测试比较 1912
测试比较 1913
测试比较 1914
测试比较 1915
测试比较 1916
测试比较 1917
0
结果寄存器 1952
79:58
57:40
39:32
31:24
23:16
15:8
7:0
63:56
55:48
47:40
Result[63:0] 1960
操作控制 1900
输出使能 1931
输出使能 1932

图 19a

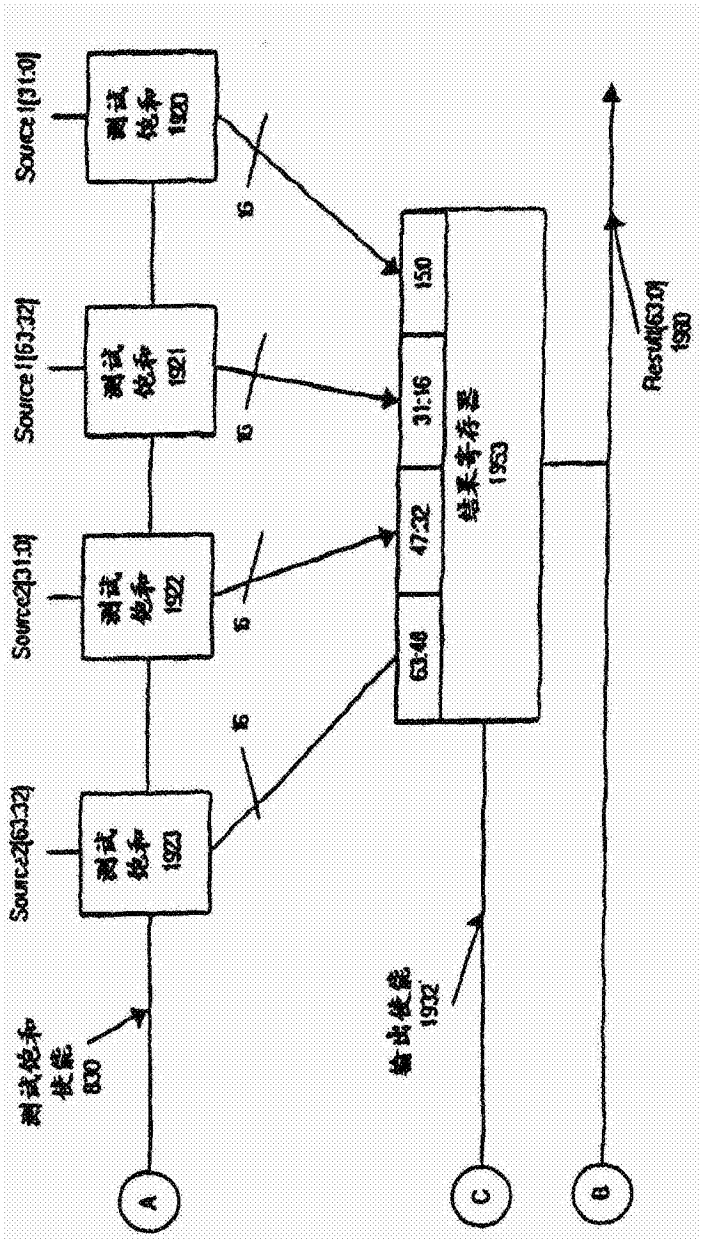

Source1[31:0]
测试使能 1920
Source1[63:32]
测试使能 1921
Source2[31:0]
测试使能 1922
Source2[63:32]
测试使能 1923
测试使能 800
16
16
16
16
150
31:16
47:32
63:48
结果寄存器 1933
输出使能 1932
Result[63:0] 1930
A
C
B

图 19b

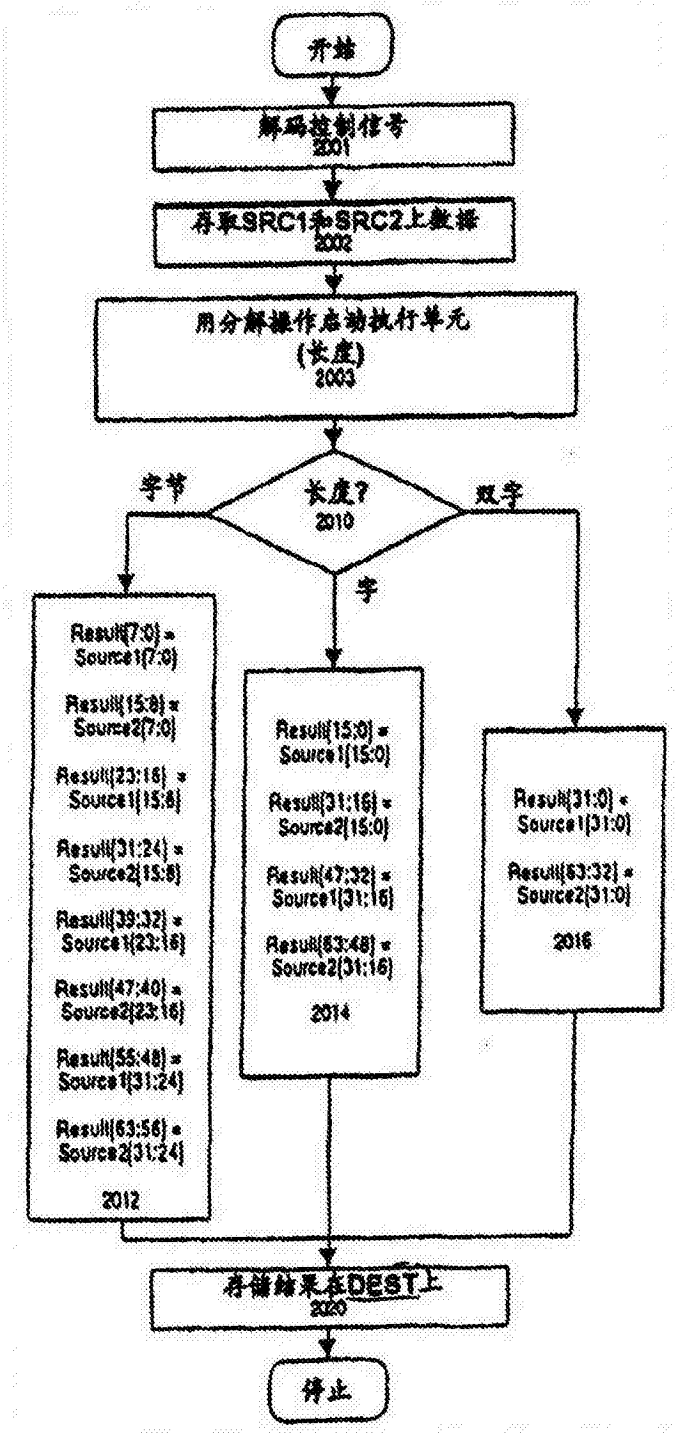

开始
解码控制信号
2001
存取SRC1和SRC2上数据
2002
用分解操作启动执行单元
(长度)
2003
长度?
2010
字节
字
双字
Result[7:0] = Source1[7:0]
Result[15:8] = Source2[7:0]
Result[23:16] = Source1[15:8]
Result[31:24] = Source2[15:8]
Result[39:32] = Source1[23:16]
Result[47:40] = Source2[23:16]
Result[55:48] = Source1[31:24]
Result[63:56] = Source2[31:24]
2012
Result[15:0] = Source1[15:0]
Result[31:16] = Source2[15:0]
Result[47:32] = Source1[31:16]
Result[63:48] = Source2[31:16]
2014
Result[31:0] = Source1[31:0]
Result[63:32] = Source2[31:0]
2016
存储结果在DEST上
2020
停止

图 20

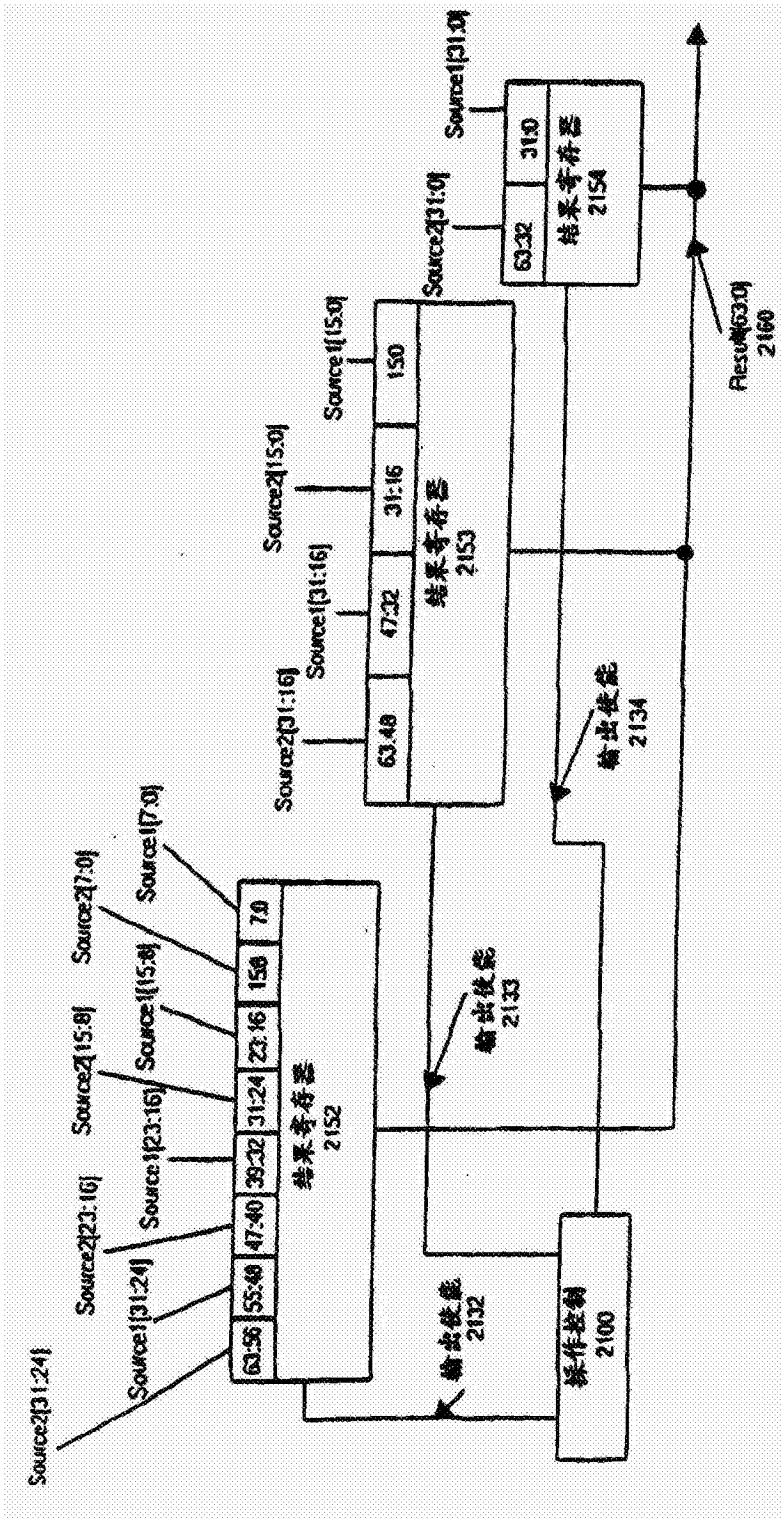

Source2[31:24]
Source1[31:24]
Source2[23:16]
Source1[23:16]
Source2[15:8]
Source1[15:8]
Source2[7:0]
Source1[7:0]
63:56
55:48
47:40
39:32
31:24
23:16
15:8
7:0
结果寄存器
2152
输出使能
2132
操作控制
2100
输出使能
2133
输出使能
2134
Source2[31:16]
Source1[31:16]
Source2[15:0]
Source1[15:0]
63:48
47:32
31:16
15:0
结果寄存器
2153
Source2[31:0]
Source1[31:0]
63:32
31:0
结果寄存器
2154
Result[63:0]
2160

图 21

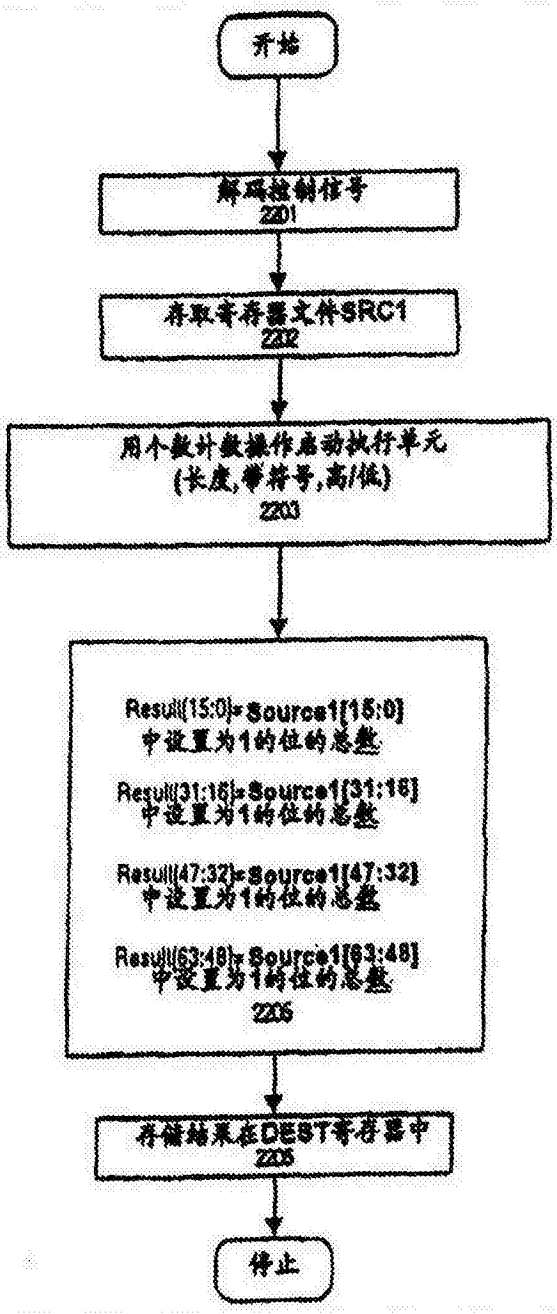

开始
解码控制信号
2201
存取寄存器文件SRC1
2202
用个数计数操作启动执行单元
(长度,带符号,高/低)
2203
Result[15:0]=Source1[15:0]
中设置为1的位的总数
Result[31:16]=Source1[31:16]
中设置为1的位的总数
Result[47:32]=Source1[47:32]
中设置为1的位的总数
Result[63:48]=Source1[63:48]
中设置为1的位的总数
2206
存储结果在DEST寄存器中
2205
停止

图 22

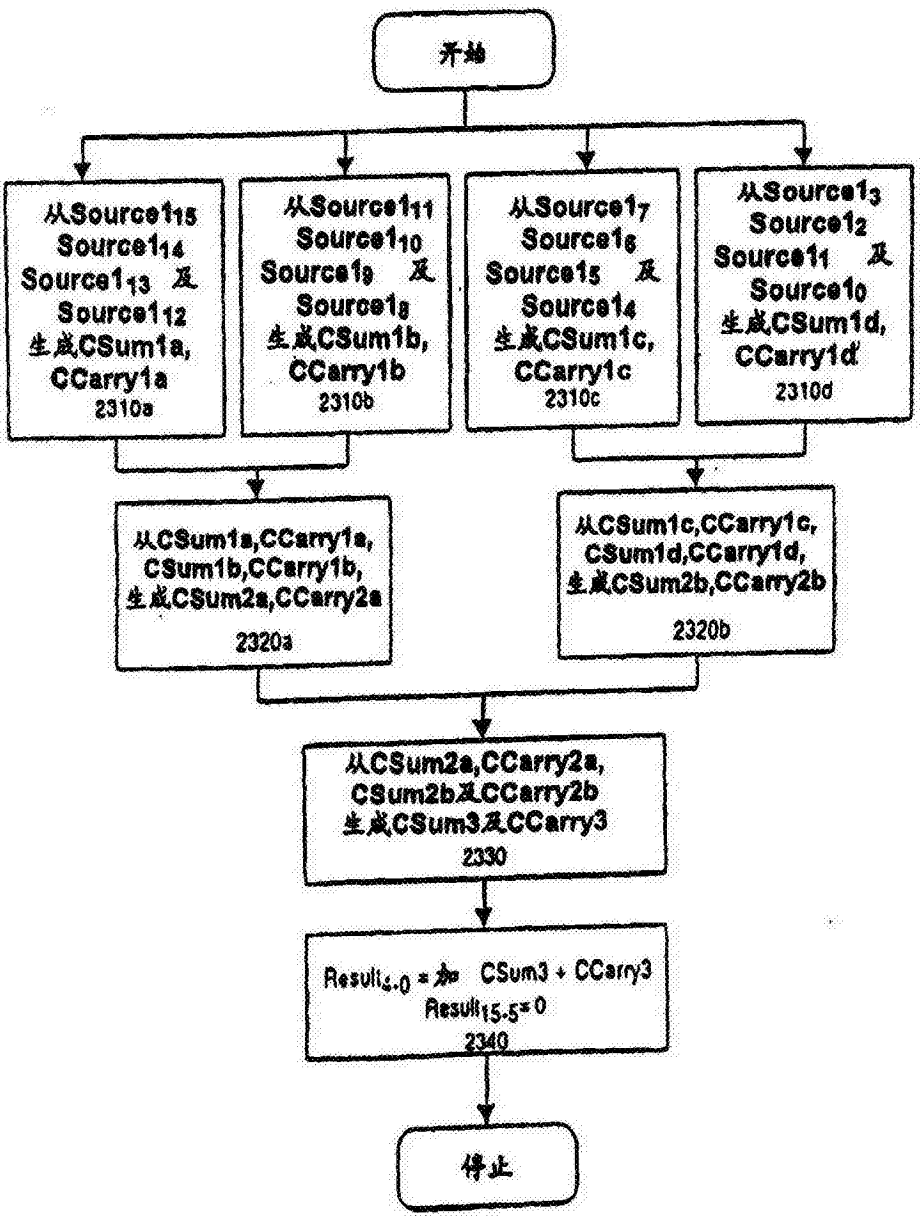

开始
从Source1$_{15}$ Source1$_{14}$ Source1$_{13}$ 及 Source1$_{12}$ 生成CSum1a, CCarry1a
2310a
从Source1$_{11}$ Source1$_{10}$ Source1$_{9}$ 及 Source1$_{8}$ 生成CSum1b, CCarry1b
2310b
从Source1$_{7}$ Source1$_{6}$ Source1$_{5}$ 及 Source1$_{4}$ 生成CSum1c, CCarry1c
2310c
从Source1$_{3}$ Source1$_{2}$ Source1$_{1}$ 及 Source1$_{0}$ 生成CSum1d, CCarry1d
2310d
从CSum1a,CCarry1a, CSum1b,CCarry1b, 生成CSum2a,CCarry2a
2320a
从CSum1c,CCarry1c, CSum1d,CCarry1d, 生成CSum2b,CCarry2b
2320b
从CSum2a,CCarry2a, CSum2b及CCarry2b 生成CSum3及CCarry3
2330
Result$_{4:0}$ = 加 CSum3 + CCarry3
Result$_{15:5}$ = 0
2340
停止

图 23

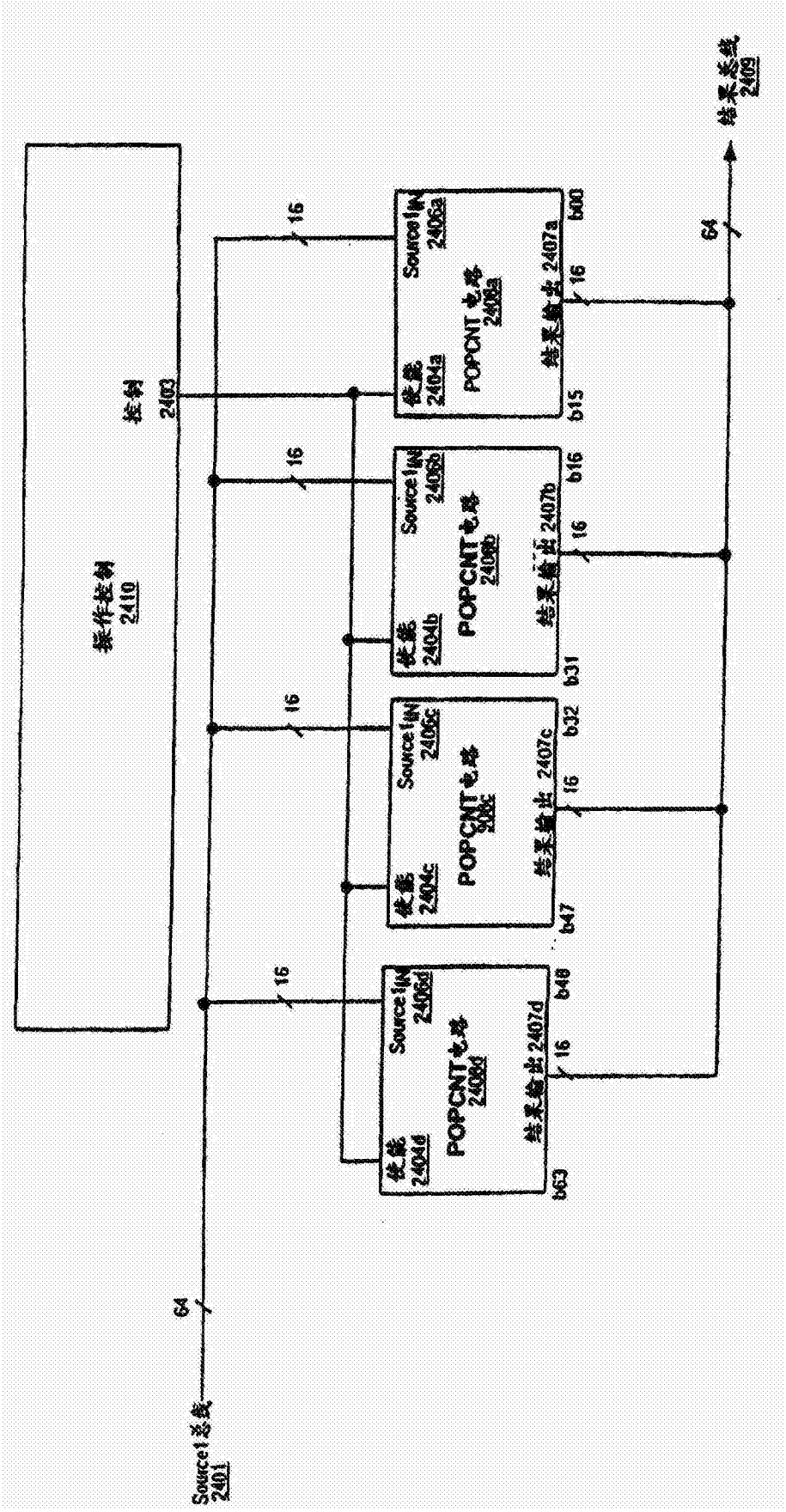

操作控制
2410
控制
2403
Source1总线
2401
64
16
使能
2404a
Source1IN
2406a
POPCNT电路
2408a
结果输出 2407a
b00
b15
使能
2404b
Source1IN
2406b
POPCNT电路
2408b
结果输出 2407b
b16
b31
使能
2404c
Source1IN
2406c
POPCNT电路
2408c
结果输出 2407c
b32
b47
使能
2404d
Source1IN
2406d
POPCNT电路
2408d
结果输出 2407d
b48
b63
结果总线
2409

图 24

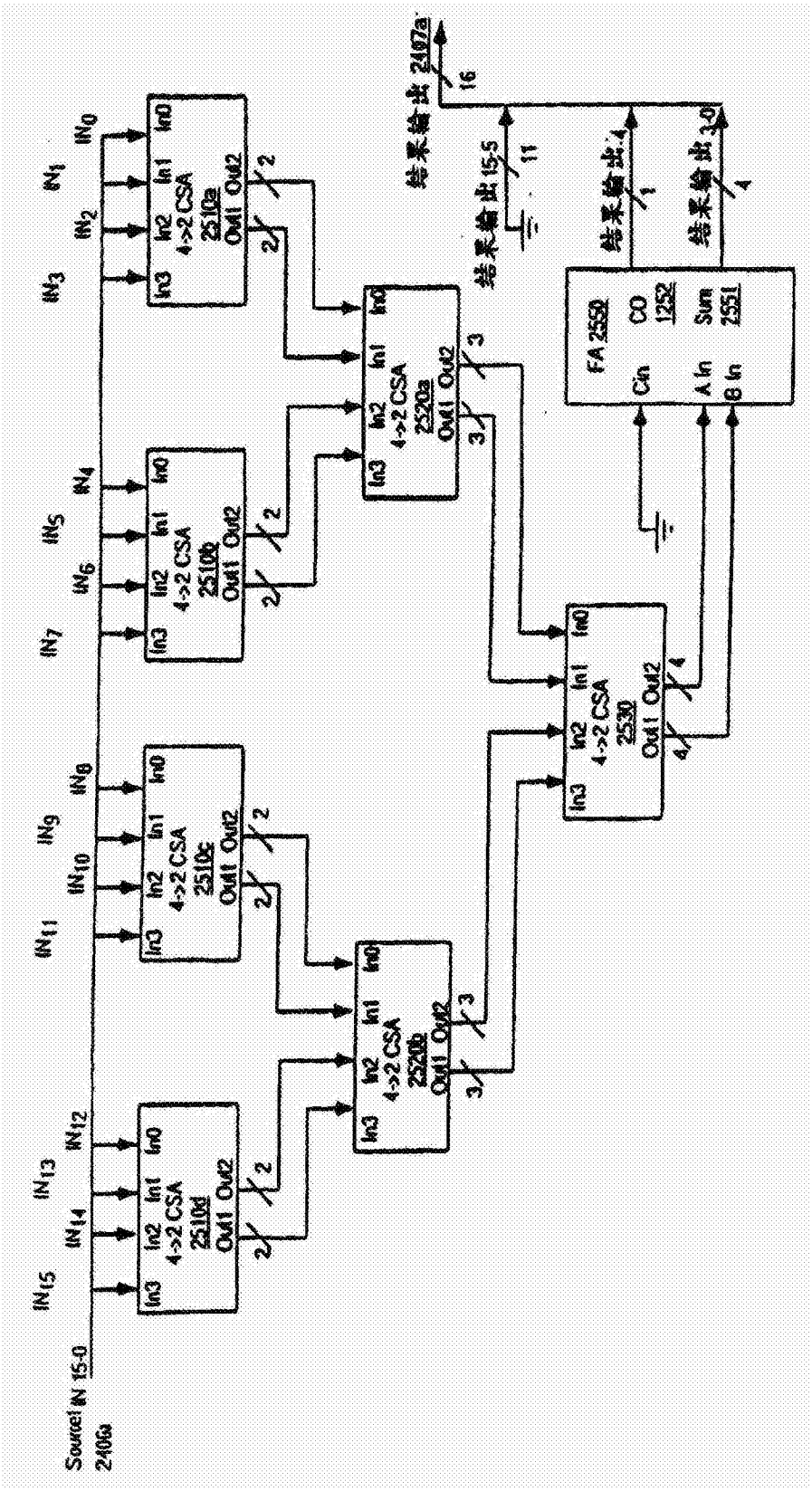

结果输出 2407a
16
结果输出 15:5
11
结果输出 4
结果输出 3:0
4
FA 2550
CO
1252
Sum
2551
Cin
A In
B In
4->2 CSA
2510a
2520a
2530
2510b
2520b
2510c
2510d
In0
In1
In2
In3
Out1
Out2
Source IN 15:0
2406a

图 25

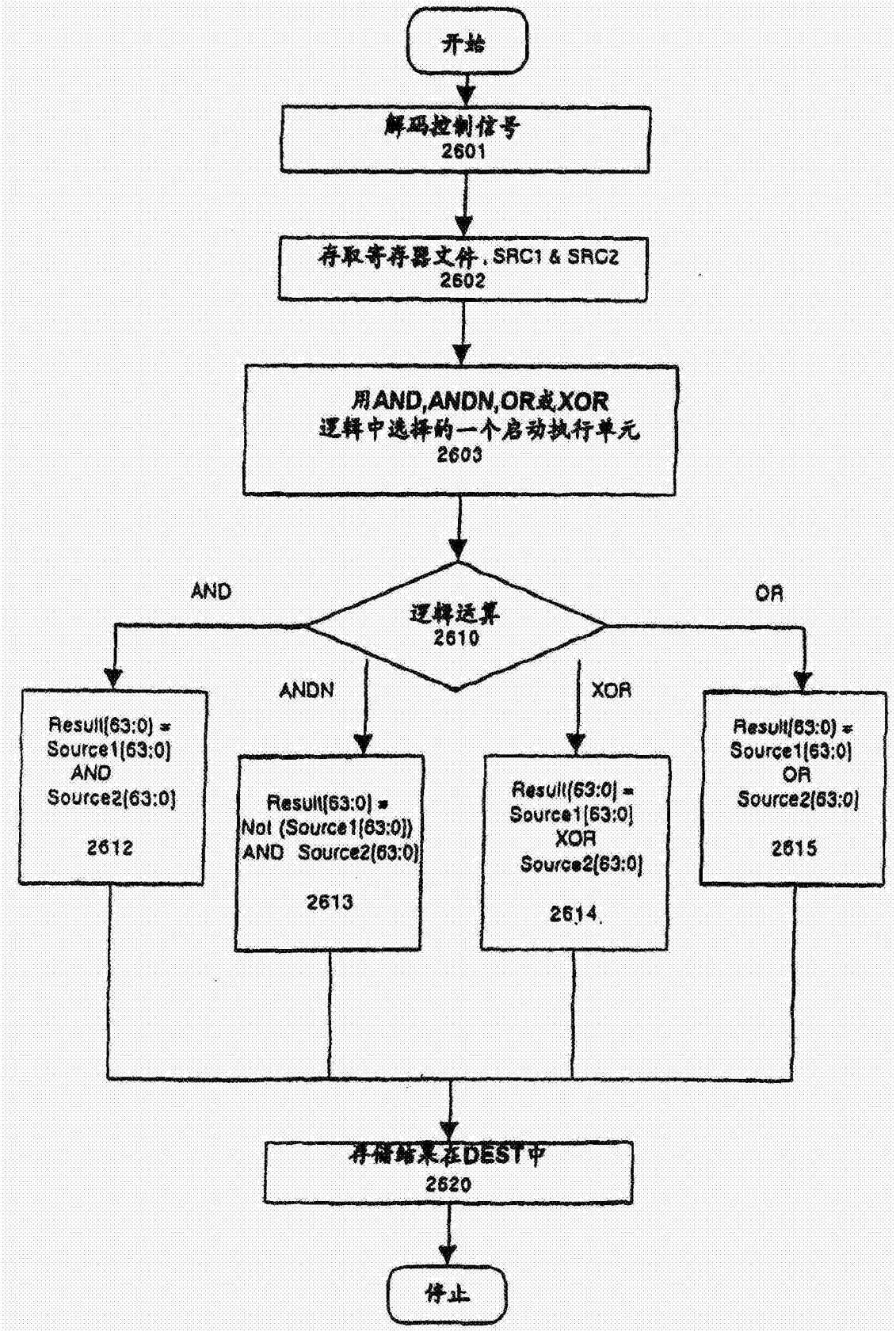

开始
解码控制信号
2601
存取寄存器文件，SRC1 & SRC2
2602
用AND,ANDN,OR或XOR
逻辑中选择的一个启动执行单元
2603
逻辑运算
2610
AND
ANDN
XOR
OR
Result[63:0] =
Source1[63:0]
AND
Source2[63:0]
2612
Result[63:0] =
Not (Source1[63:0])
AND Source2[63:0]
2613
Result[63:0] =
Source1[63:0]
XOR
Source2[63:0]
2614
Result[63:0] =
Source1[63:0]
OR
Source2[63:0]
2615
存储结果在DEST中
2620
停止

图 26

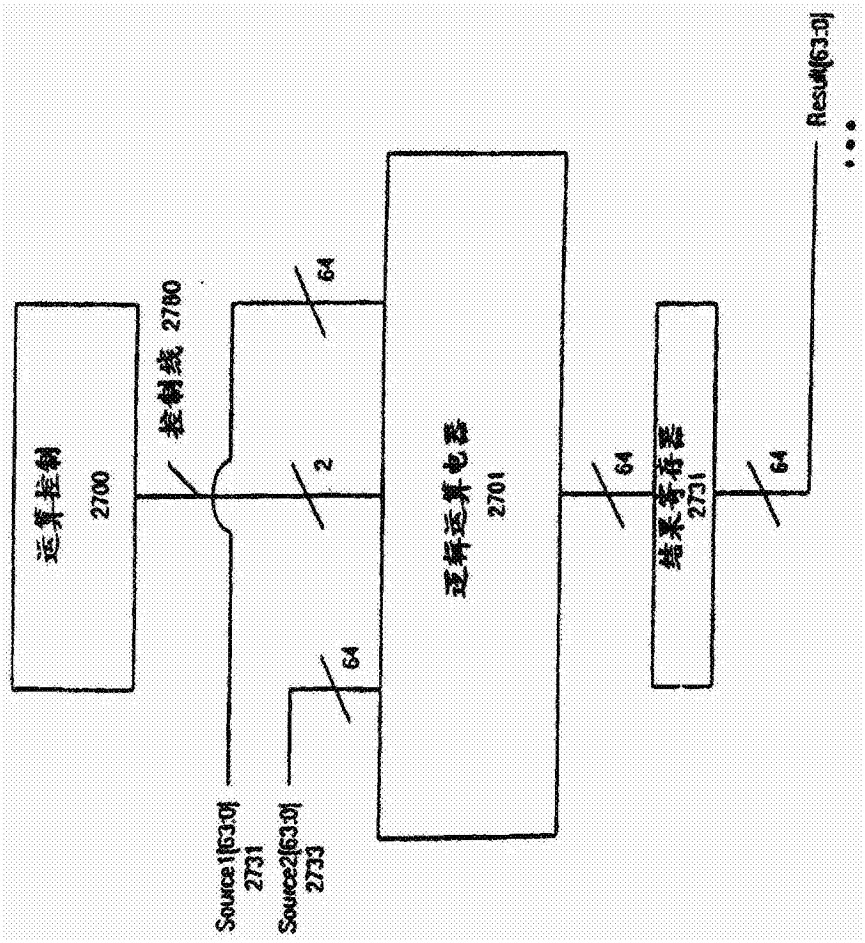

运算控制
2700
控制线 2780
64
2
64
Source1[63:0]
2731
Source2[63:0]
2733
逻辑运算电路
2701
64
结果寄存器
2731
64
Result[63:0]

图 27

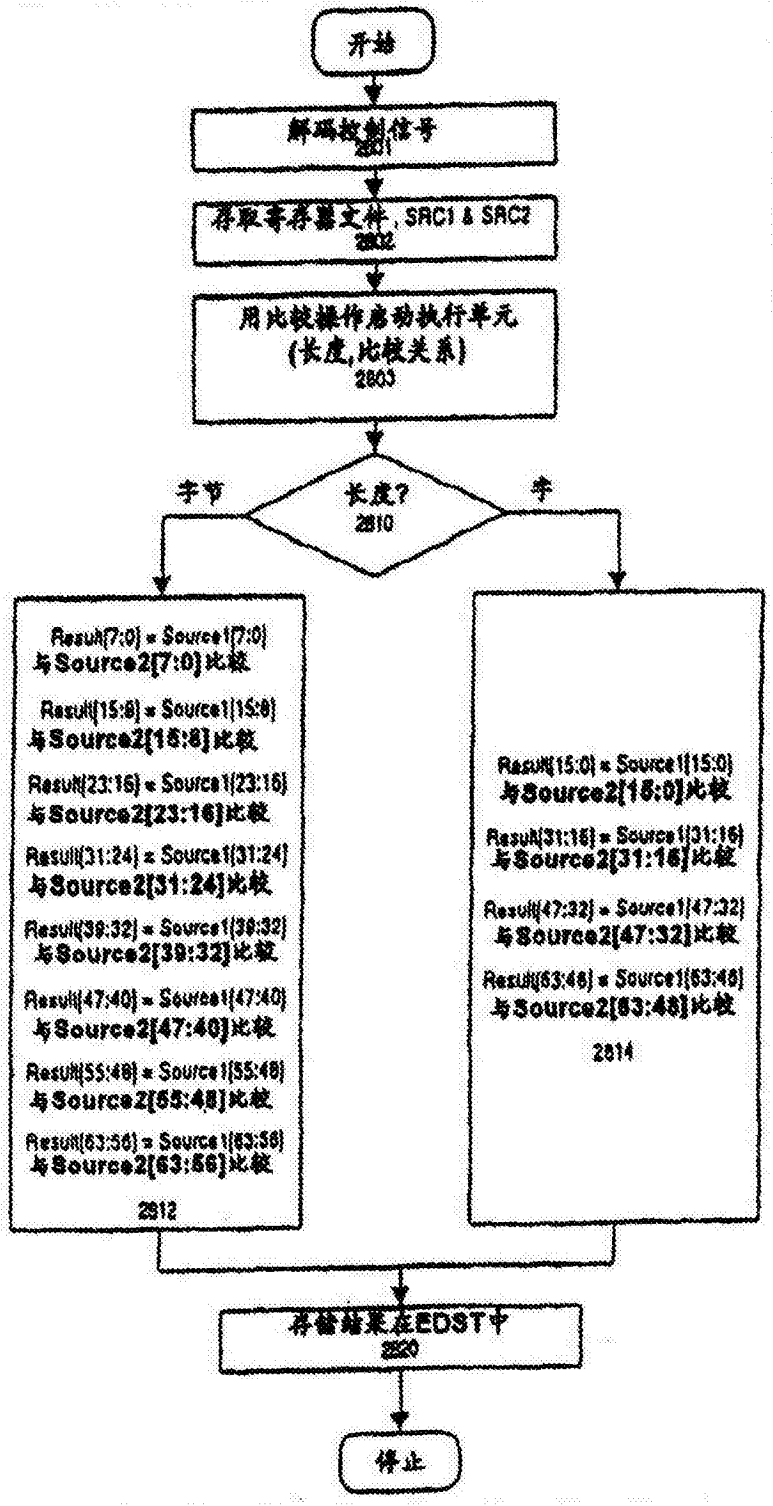

开始
解码控制信号
2801
存取寄存器文件, SRC1 & SRC2
2802
用比较操作启动执行单元
(长度,比较关系)
2803
长度?
2810
字节
半
Result[7:0] = Source1[7:0]
与Source2[7:0]比较
Result[15:8] = Source1[15:8]
与Source2[15:8]比较
Result[23:16] = Source1[23:16]
与Source2[23:16]比较
Result[31:24] = Source1[31:24]
与Source2[31:24]比较
Result[39:32] = Source1[39:32]
与Source2[39:32]比较
Result[47:40] = Source1[47:40]
与Source2[47:40]比较
Result[55:48] = Source1[55:48]
与Source2[55:48]比较
Result[63:56] = Source1[63:56]
与Source2[63:56]比较
2812
Result[15:0] = Source1[15:0]
与Source2[15:0]比较
Result[31:16] = Source1[31:16]
与Source2[31:16]比较
Result[47:32] = Source1[47:32]
与Source2[47:32]比较
Result[63:48] = Source1[63:48]
与Source2[63:48]比较
2814
存储结果在EDST中
2820
停止

图 28

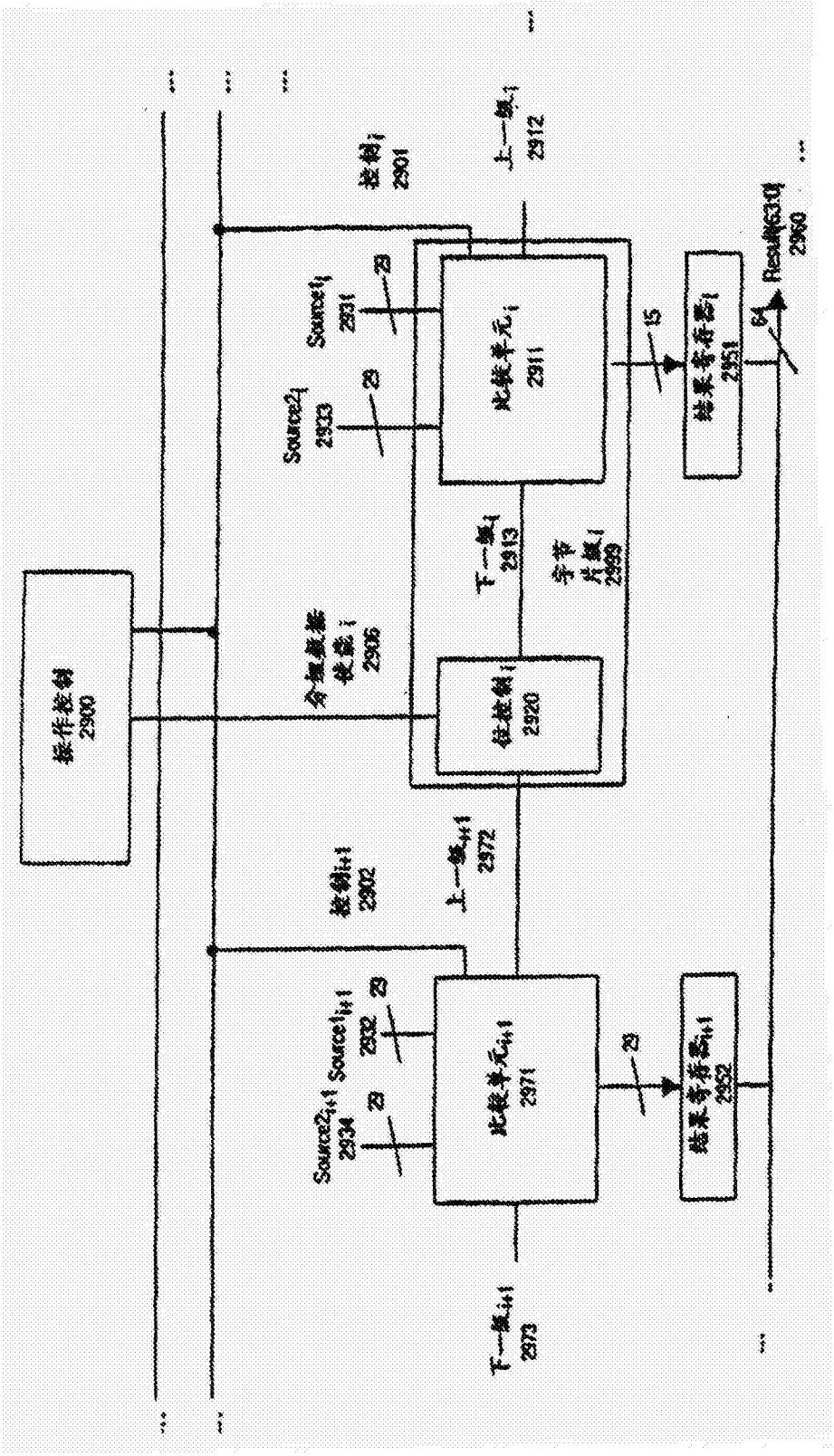

操作控制
2900
控制i
2901
上一级i
2912
Source1i
2931
Source2i
2933
29
比较单元i
2911
15
结果寄存器i
2951
64
Result[63:0]
2960
下一级i
2913
字节
片级i
2999
分组数据
使能i
2906
位控制i
2920
控制i+1
2902
上一级i+1
2972
Source1i+1
2932
Source2i+1
2934
比较单元i+1
2971
结果寄存器i+1
2952
下一级i+1
2973

图 29