



1. 一种装置, 包括:

存储区, 所述存储区存储包括每一个都具有多个数据元素的分组数据的数据;

5 执行单元, 所述执行单元执行具有用于标识第一源数据和第二源数据的格式的分组数据指令集所指定的操作, 其中所述分组数据指令集指定至少如下操作, 包括:

一个或多个分解类型操作, 其中每个导致在所述存储器区生成和存储结果分组数据, 所述结果分组数据包括少于所述第一源数据的数据元素和所述第二源数据的对应数据元素交错的数据元素的全部;

10 一个或多个组装类型的操作, 其中每个导致在所述存储区生成和存储结果分组数据, 所述结果分组数据包括从所述第一源数据和所述第二源数据中的每个数据元素中求出的多个位部分, 所述第一和第二源数据中的所述多个位部分在所述结果分组数据中是交错的;

15 一个或多个分组乘类型操作, 其中每个导致在所述存储器区生成和存储结果分组数据, 所述结果分组数据仅包括作为独立结果元素的高阶或者低阶位, 所述高阶或者低阶位来自与所述第二源数据的对应数据元素相乘后的所述第一源数据的每个数据元素的结果的高阶或者低阶位;

20 一个或多个分组加类型操作, 其中每个导致在所述存储器区生成和存储结果分组数据, 所述结果分组数据包括作为独立的结果元素的与所述第二源数据的对应数据元素进行了相加的所述第一源数据的每个数据元素;

25 一个或多个分组减类型操作, 其中每个导致在所述存储器区生成和存储结果分组数据, 所述结果分组数据包括作为独立的结果元素的被从所述第一源数据的对应数据元素中减去的所述第二源数据的每个数据元素;

30 一个或多个分组比较类型操作, 其中每个导致在所述存储器区生成和存储结果分组数据, 所述结果分组数据包括作为独立的结果元素的掩码, 所述掩码表示将所述第一分组数据中的每个数据元素之一和所述第二分组数据中的对应数据元素进行对应的比较, 所述掩码中的那些对应比较为真的掩码的每一个包括多个都具有第一预定值的位, 并且所述掩码中的那些对应比较为假的掩码的每一个包括多个都具有第二预定值的位; 以及

一个或多个乘加类型操作，其中每个导致在所述存储器区生成和存储结果分组数据，所述结果分组数据包括第一结果元素和第二结果元素，所述第一和第二结果元素被存储而没有对所述第一和第二结果元素进行求和，其中所述第一结果元素表示将所述第一源数据和所述第二源数据的两对对应数据元素相乘的结果的第一个和，所述第二结果元素表示将所述第一源数据和所述第二源数据的不同两对的对应数据元素相乘的结果的第二个和，所述第一和第二结果元素具有比用于生成所述第一和第二结果元素的所述第一和第二源数据中的每个数据元素更高的精度。

2. 权利要求 1 的装置，其中对于所述一个或多个分解类型的操作中的至少一个，结果分组数据包括所述第一源数据和所述第二源数据的高位一半或者低位一半。

3. 权利要求 1 的装置，其中对于所述一个或多个组装类型的操作中的至少一个，所述第一和第二源数据的每个数据元素是  $N$  位数据元素，所述多个位部分的每个是  $N/2$  位结果元素。

4. 权利要求 1-3 中任意一项的装置，其中对于所述一个或多个乘加类型操作中的至少一个，所述第一和第二源数据中的每个数据元素包括  $N$  位数据元素，并且所述第一结果元素和所述第二结果元素中的每一个是  $2N$  位结果元素。

5. 权利要求 1-4 中任意一项的装置，其中所述第一存储区是一个或多个寄存器的组。

6. 权利要求 1-5 中任意一项的装置，其中所述格式包括把第一存储位置指定为源-目的操作数并把第二存储位置指定为源操作数。

7. 权利要求 6 的装置，其中所述源操作数和所述源-目的操作数中的每个被指定为寄存器数，所述源操作数由操作码字节的 0-2 位来指定，所述源-目的操作数由所述操作码字节的 3-5 位指定。

8. 权利要求 1-8 中任意一项的装置，其中所述分组数据指令集还指定至少如下操作，包括：

一个或多个分组移位类型的操作，其中每个导致在所述存储区生成和存储结果分组数据，所述结果分组数据包括作为独立的结果元素的所述第一源数据，所述第一源数据中的每个数据元素已经进行了由所述第二源数据指定的量的移位并且对每个数据元素的所述量的位进行了填充，如果需要，则使用填充值来填充。

9. 权利要求 8 的装置，其中所述一个或多个分组移位类型操作包括：一个或多个分组向右移位算术类型操作，其中每个使所述结果分组

数据包含作为独立的结果元素的所述第一源数据，所述第一源数据的每个数据元素向右移位由所述第二源数据指定的所述量，并且对高阶位进行填充，如果需要，则使用带符号的值来加以填充；

5 一个或多个分组向左移位类型操作，其中每个使所述结果分组数据包含作为独立的结果元素的所述第一源数据，所述第一源数据的每个数据元素向左移位由所述第二源数据指定的所述量，并且对低阶位进行填充，如果需要，则使用 0 来加以填充；以及

10 一个或多个分组向右移位逻辑类型操作，其中每个使所述结果分组数据包含作为独立的结果元素的所述第一源数据，所述第一源数据的每个数据元素向右移位由所述第二源数据指定的所述量，并且对高阶位进行填充，如果需要，则使用 0 加以填充。

10. 权利要求 1-9 中任意一项的装置，其中所述执行单元包括多个电路，其中所述多个电路中的至少一个电路执行多个所述操作。

15 11. 权利要求 10 的装置，其中所述多个电路包括：  
加法器/减法器电路；以及  
乘加电路。

12. 权利要求 1-9 中的任意一项的装置，其中所述执行单元包括：  
第一电路，所述第一电路执行所述组装和分解类型操作；以及  
第二电路，所述第二电路执行所述组装乘和乘加类型操作。

20 13. 权利要求 12 的装置，其中所述执行单元还包括：  
第三电路，所述第三电路执行所述分组加、分组减、分组比较以及  
逻辑类型操作。

14. 权利要求 10-13 中任意一项的装置，还包括：  
25 存储设备，所述存储设备存储多个指令序列，所述存储设备与所述  
执行单元耦合；  
与所述执行单元耦合的显示设备；以及  
与所述执行单元耦合的声音播放设备。

15. 权利要求 14 的装置，其中所述存储设备存储复数乘法例程，所述复数乘法例程包括用于指定所述一个或多个乘加类型操作的一个来使  
30 至少两个复数彼此相乘的指令。

16. 权利要求 14 或 15 的装置，其中所述存储设备存储乘累加例程，所述乘累加例程包括用于指定所述分解和乘加类型操作中的每个的至少一种类型的指令。

17. 权利要求 14-16 中任意一项的装置，其中所述存储设备存储点积

例程，所述点积例程包括用于指定所述乘加、分组移位以及分组加类型操作中每个的至少一种类型的指令。

- 5 18. 权利要求 14-17 中任意一项的装置，其中所述存储设备存储环路滤波器例程，所述环路滤波器例程包括用于指定所述分解、分组移位以及分组加类型操作中的每个的至少一种类型的指令。

19. 权利要求 14-18 中任意一项的装置，其中所述存储设备存储运动估计例程，所述运动估计例程包括用于指定所述分解、带饱和分组减、分组加类型操作和分组逻辑类型操作中的每个的至少一种类型的指令，所述运动估计例程处理可在所述显示设备上显示的数据。

- 10 20. 权利要求 14-19 中任意一项的装置，其中所述存储设备存储离散余弦变换例程，所述离散余弦变换例程包括用于指定所述分组移位和乘加类型操作中的每个的至少一种类型的指令，所述离散余弦变换例程处理可在所述显示设备上显示的数据。

## 在分组数据上操作的指令集

## 技术领域

- 5 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

- 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。  
10 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  
15 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。

- 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  
20 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  
25 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。

- 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，  
30 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。

5 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  
10 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.i.j、6.i.j 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  
15 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。

另一种支持分组数据的通用处理器便是 Motorola 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  
20 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。

MC88110 处理器分组命令通过连接第一寄存器对中的各元素的  $(t \cdot r) / 64$ （其中  $t$  为该分组数据的元素中的位数）个最高有效位进行  
25 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。

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

X=未定义的操作

表1

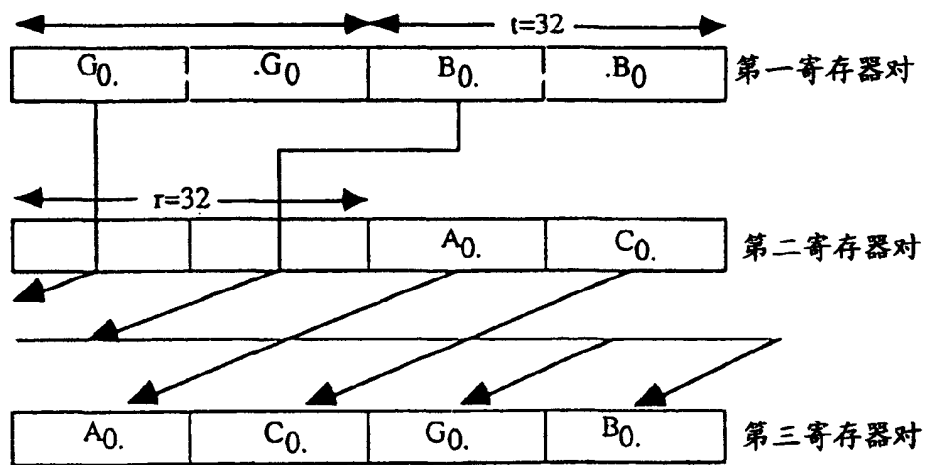


表2

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，  
5 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)





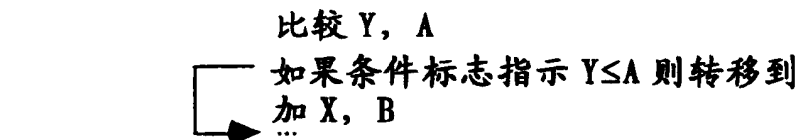
这一乘法指令具有两个缺点。首先这一乘法指令并不断开进位链，从而程序员必须保证在分组数据上执行的运算并不导致溢出。结果，程序员有时必须加入附加的指令来防止这一溢出。第二，这一乘法指令将分组数据中的各元素乘以单一的值（即该 32 位值）。结果，

5 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，

10 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。

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

15 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：if Y>A then X=X+B else X=X。这一函数的伪码编译表示将是：



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

25 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。

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）。

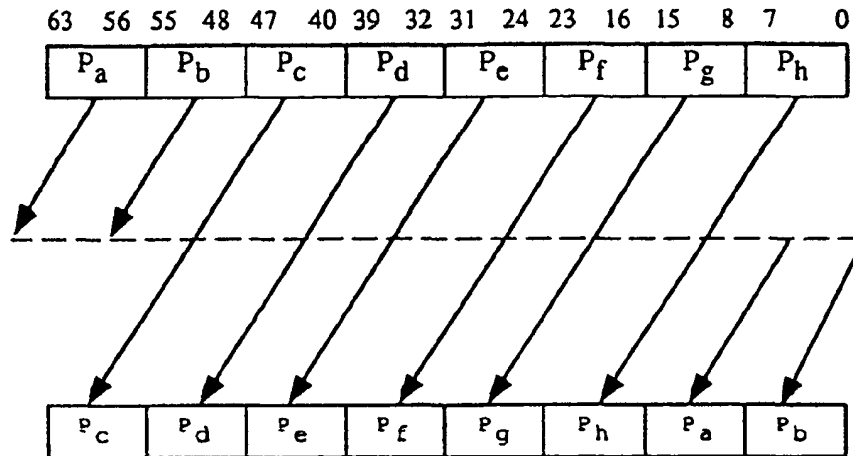


表 6

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

5 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

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

10 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。

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

15 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。

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

20 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。

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。

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。

- 5 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

- 10 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。

图 1 说明按照本发明的一个实施例的示例性计算机系统。

图 2 说明按照本发明的一个实施例的处理器寄存器文件。

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。

- 15 图 4 说明按照本发明的一个实施例的分组数据类型。

图 5a 表示按照本发明的一个实施例的寄存器中分组数据。

图 5b 表示按照本发明的一个实施例的寄存器中分组数据。

图 5c 表示按照本发明的一个实施例的寄存器中分组数据。

- 20 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。

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。

#### 分组加法/减法

图 7a 说明按照本发明的一个实施例执行分组加法的方法。

- 25 图 7b 说明按照本发明的一个实施例执行分组减法的方法。

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 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。

- 30 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。

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。

#### 分组乘法

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。

5 图 13 说明按照本发明的一个实施例执行分组乘法的电路。

#### 乘-加/减

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。

10 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。

#### 分组移位

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。

15 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。

#### 组装

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。

20 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。

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。

#### 分解

25 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。

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。

#### 个数计数

30 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。

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。

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。

5 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。

**分组逻辑运算。**

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。

10 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。

**分组比较**

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。

15 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**

20 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。

**定义**

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。

位 X 至位 Y;

25 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  $15_{10}$ 。

$R_x$ : 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。

30 SRC1, SRC2 与 DEST:

标识存储区 (诸如存储器地址、寄存器等)

Source1-i 与 Result1-i: 表示数据

## 计算机系统

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  
5 CISC（复杂指令集计算）或 RISC（精简指令集计算）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  
10 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。

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

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操作、分解操作、分组加法运算、分  
20 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。

25 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  
30 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

千条这些指令的方式改进了许多多媒体应用的性能。

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。

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。

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。

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  
5 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。

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  
10 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  
15 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。

图 2 说明按照本发明的一个实施例的寄存器的寄存器文件。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  
20 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。

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。  
25 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  
30 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。

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，  
5 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。

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  
10 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<sub>0</sub> 212a 至 R<sub>7</sub> 212h。R<sub>1</sub> 212a、R<sub>2</sub> 212b 及 R<sub>3</sub> 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。  
15 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。

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  
20 107 的数据加载到寄存器文件 150 中的寄存器。

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。

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  
25 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  
30 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。

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  
5 201 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。

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。

10

### 数据与存储格式

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  
15 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。

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。

20 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。

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 R<sub>0</sub> 212a 至 R<sub>h</sub> 212h 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、  
25 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。

30

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。

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。

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 R<sub>0</sub> 212a 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。

#### 控制信号格式

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。

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。SRC1 602 (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，SRC2 603 (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。

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<sub>2</sub>，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。

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。

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。

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. Box 7641, 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 SRC1 602。在一个实施例中，当存在一个 SRC1 602 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 SRC2 603 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 SRC2 603。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、寄存器到寄存器的寻址。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。

25

#### 饱和/不饱和的说明

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。

30

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	$- 2^{63}$	$2^{63} - 1$

表 7

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。

### 分组加法

#### 分组加法运算

10 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。

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 SRC1 602、SRC2 603 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 SRC1 602 与 SRC2 603 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。

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。

假定数据元素长度为 8 位，则执行步骤 705a。执行单元 130 将  
 5 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  
 10 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  
 15 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。

20 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  
 25 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。

30 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。

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<sub>2</sub>。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

5 饱和的加法, 则执行单元 130 产生所示的 Result 分组数据。

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<sub>2</sub> + 10001000<sub>2</sub> = 00001000<sub>2</sub>。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

10 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。

00101010	01010101	01010101	11111111	10000000	01110000	10001111	10001000
<u>+</u> <sup>7</sup>	<u>+</u> <sup>6</sup>	<u>+</u> <sup>5</sup>	<u>+</u> <sup>4</sup>	<u>+</u> <sup>3</sup>	<u>+</u> <sup>2</sup>	<u>+</u> <sup>1</sup>	<u>+</u> <sup>0</sup>
10101010	01010101	10101010	10000001	10000000	11110000	11001111	10001000
<u>=</u> <sup>7</sup>	<u>=</u> <sup>6</sup>	<u>=</u> <sup>5</sup>	<u>=</u> <sup>4</sup>	<u>=</u> <sup>3</sup>	<u>=</u> <sup>2</sup>	<u>=</u> <sup>1</sup>	<u>=</u> <sup>0</sup>
11010100	10101010	11111111	上溢	上溢	上溢	上溢	上溢
7	6	5	4	3	2	1	0

表 8a

15

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

20 下面强调的“上溢”)。



00101010 01010101	01010101 11111111	10000000 01110000	10001111 10001000
+	+	+	+
10101010 01010101	10101010 10000001	10000000 11110000	11001111 10001000
=	=	=	=
11010100 10101010	上溢	上溢	上溢
3	2	1	0

表8b

表 8 c 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 5 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。

00101010 01010101 01010101 11111111	10000000 01110000 10001111 10001000
+	+
10101010 01010101 10101010 10000001	10000000 11110000 11001111 10001000
=	=
11010100 10101011 00000000 10000000	上溢
1	0

表8c

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 10 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。

00101010 01010101 01010101 11111111 10000000 01110000 10001111 10001000
±
10101010 01010101 10101010 10000001 10000000 11110000 11001111 10001000
≡
11010100 10101010/ 00000000/ 10000000/ 00000000/ 01100000/ 01011111/ 00010000

表9

**带符号/不饱和分组加法**

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。

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

表10

**带符号/饱和的分组加法**

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。

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

表11

**分组减法**

**分组减法运算**

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。

- 5 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。

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。

- 10 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。

- 15 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。

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  
 5 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。

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。

10 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。

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。

15

00101010	01010101	01010101	01111111	00000000	11110000	00001111	10001000
<u>-</u> <sup>2</sup>	<u>-</u> <sup>2</sup>	<u>-</u> <sup>2</sup>	<u>-</u> <sup>2</sup>	<u>-</u> <sup>2</sup>	<u>-</u> <sup>2</sup>	<u>-</u> <sup>2</sup>	<u>-</u> <sup>2</sup>
10101010	01010101	10101010	00000001	00000000	11110000	00001111	10001000
<u>=</u> <sup>2</sup>	<u>=</u> <sup>2</sup>	<u>=</u> <sup>2</sup>	<u>=</u> <sup>2</sup>	<u>=</u> <sup>2</sup>	<u>=</u> <sup>2</sup>	<u>=</u> <sup>2</sup>	<u>=</u> <sup>2</sup>
下溢	00000000	下溢	01111110	00000000	00000000	00000000	00000000
7	6	5	4	3	2	1	0

表12

### 分组数据加法/减法电路

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  
 20 800。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<sub>i</sub> 804a 上所接收  
 的位 i 上加上或减去 Source2<sub>i</sub> 805a 上接收的位 i，产生在 Result<sub>i</sub> 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。

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

...	...	...	...	...	...	...	00001111	10001000
$\overline{+}$	$\overline{+}$	$\overline{+}$	$\overline{+}$	$\overline{+}$	$\overline{+}$	$\overline{+}$	$\overline{+}$	$\overline{+}$
...	...	...	...	...	...	...	00001111	10001000
$\overline{=}$	$\overline{=}$	$\overline{=}$	$\overline{=}$	$\overline{=}$	$\overline{=}$	$\overline{=}$	$\overline{=}$	$\overline{=}$
...	...	...	...	...	...	...	00011110	上溢
7	6	5	4	3	2	1	0	

表13a

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。

...	...	...	...	00001111	10001000
$\overline{+}$	$\overline{+}$	$\overline{+}$	$\overline{+}$	$\overline{+}$	$\overline{+}$
...	...	...	...	00001111	10001000
$\overline{=}$	$\overline{=}$	$\overline{=}$	$\overline{=}$	$\overline{=}$	$\overline{=}$
...	...	...	...	00011111	00010000
3	2	1	0		

表13b

加法器/减法器 801a 通过首先反相 Source2<sub>i</sub> 805a 及加 1 形成 Source2<sub>i</sub> 805a 的 2 的补码，从 Source1<sub>i</sub> 804a 减去位 Source2<sub>i</sub> 805a。然后，加法器/减法器 801a 将这一结果加在 Source1<sub>i</sub> 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。

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<sub>i</sub> 906a-h 与 Source2<sub>i</sub> 905 a-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 904 a-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 CTRL 904a-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。

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。

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。

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。

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）。

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。

## 15 分组乘法

### 分组乘法运算

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。

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。

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。

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。

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。

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 SRC1 602、SRC2 603 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。

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 SRC1 602 与 SRC2 603 地址的寄存器 209。寄存器 209 向执行单元 130 提供存储在 SRC1 603 寄存器中的分组数据 (Source1) 及存储在 SRC2 603 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。

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。

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。

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 5 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.

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 10 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.

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. 15

在步骤 1220 将 Result 存储在 DEST 寄存器中.

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 20 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.

11111111 11111111	11111111 00000000	11111111 00000000	00001110 00001000
乘 <sup>3</sup>	乘 <sup>2</sup>	乘 <sup>1</sup>	乘 <sup>0</sup>
00000000 00000000	00000000 00000001	10000000 00000000	00001110 10000001
=	=	=	=
00000000 00000000	00000000 00000000	01111111 10000000	00000000 11001011
3	2	1	0

表14

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。

11111111 11111111	11111111 00000000	11111111 00000000	00001110 00001000
乘 <sup>3</sup>	乘 <sup>2</sup>	乘 <sup>1</sup>	乘 <sup>0</sup>
00000000 00000000	00000000 00000001	10000000 00000000	00001110 10000001
=	=	=	=
00000000 00000000	11111111 11111111	00000000 10000000	00000000 11001011
<sup>3</sup>	<sup>2</sup>	<sup>1</sup>	<sup>0</sup>

表15

表 16 示出分组字数据上的分组乘法低运算的寄存器表示。

5

11111111 11111111	11111111 00000000	11111111 00000000	00001110 00001000
乘 <sup>3</sup>	乘 <sup>2</sup>	乘 <sup>1</sup>	乘 <sup>0</sup>
00000000 00000000	00000000 00000001	10000000 00000000	00001110 10000001
=	=	=	=
00000000 00000000	11111111 00000000	00000000 00000000	10000010 00001000
<sup>3</sup>	<sup>2</sup>	<sup>1</sup>	<sup>0</sup>

表16

分组数据乘法电路

10 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。

15 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。

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。

5 分组字乘法器 1301 包含 4 个  $16 \times 16$  乘法器电路： $16 \times 16$  乘法器 A 1310、 $16 \times 16$  乘法器 B 1311、 $16 \times 16$  乘法器 C 1312 及  $16 \times 16$  乘法器 D 1313。 $16 \times 16$  乘法器 A 1310 具有输入 Source1[15:0] 与 Source2[15:0]， $16 \times 16$  乘法器 B 1311 具有输入 Source1[31:16] 与 Source2[31:16]， $16 \times 16$  乘法器 C 1312 具有输入 Source1[47:32] 与 Source2[47:32]， $16 \times 16$  乘法器 D 1313 具有输入 Source1[63:48] 与 Source2[63:48]。各  $16 \times 16$  乘法器耦合在符号使能 1382 上。各  $16 \times 16$  乘法器产生 32 位积。对于各乘法器，多路复用器（分别为 Mx0 1350、Mx1 1351、Mx2 1352 及 Mx3 1353）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  
15 存储在结果寄存器 1 1371 中。

分组字节乘法器 1302 包含 8 个  $8 \times 8$  乘法器电路： $8 \times 8$  乘法器 A 1320 至  $8 \times 8$  乘法器 H 1337。各  $8 \times 8$  乘法器具有来自各 Source1[63:0] 1331 及 Source2[63:0] 1333 的 8 位输入。例如  $8 \times 8$  乘法器 A 1320 具有输入 Source1[7:0] 与 Source2[7:0]，而  $8 \times 8$  乘法器 H 1327 具有输入 Source1[63:56] 与 Source2[63:56]。各  $8 \times 8$  乘法器耦合在符号使能 1382 上。各  $8 \times 8$  乘法器产生一个 16 位积。对于各乘法器，多路复用器（诸如 Mx4 1360 与 Mx11 1367）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  
20 储在结果寄存器 2 1372 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。

在一个实施例中，通过制造能乘两个  $8 \times 8$  数或一个  $16 \times 16$  数两者的电路，减少用于实现乘法的面积。即将两个  $8 \times 8$  乘法器及一个  $16 \times 16$  乘法器组合成一个  $8 \times 8$  与  $16 \times 16$  乘法器。运算控制 1300 将允  
30 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  $8 \times 8$  乘法、两个  $16$

×16 乘法或一个 32 × 32 乘法。

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 2 1372。

#### 在指令集中包含上述分组乘法运算的优点

5 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；借此解除程序员的责任，不再需要准备数据来防止上溢的指令，并得出更健壮  
10 的代码。

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。

15 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。

#### 乘 - 加/减

20

#### 乘 - 加/减运算

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。

Multiply-Add Source1, Source2				
A1	A2	A3	A4	(源1)
B1	B2	B3	B4	(源2)
=				
A1B1+A2B2		A3B3+A4B4		(结果1)

表17a

25

11111111 11111111	11111111 00000000	01110001 11000111	01110001 11000111
乘 <sup>3</sup>	乘 <sup>2</sup>	乘 <sup>1</sup>	乘 <sup>0</sup>
00000000 00000000	00000000 00000001	10000000 00000000	00000100 00000000
↓	↓	↓	↓
32位中间 结果4	32位中间 结果3	32位中间 结果2	32位中间 结果1
加		加	
11111111 11111111 11111111 00000000		11001000 11100011 10011100 00000000	
		1	0

表17b

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。

Multiply-Subtract Source1, Source2				
A <sub>1</sub>	A <sub>2</sub>	A <sub>3</sub>	A <sub>4</sub>	(源1)
B <sub>1</sub>	B <sub>2</sub>	B <sub>3</sub>	B <sub>4</sub>	(源2)
=				
A <sub>1</sub> B <sub>1</sub> -A <sub>2</sub> B <sub>2</sub>		A <sub>3</sub> B <sub>3</sub> -A <sub>4</sub> B <sub>4</sub>		(结果1)

表12

5

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。

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。

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。

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。

在步骤 1402，解码器 202 通过内部总线 170 存取给出 SRC1 602 与 SRC2 603 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 SRC1 602 寄存器中的分组数据（Source1）及存储在 SRC2 603 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。

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。

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。

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。

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。

在步骤 1420，将 Result 存储在 DEST 寄存器中。

### 分组数据乘-加/减电路

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。

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。

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  $16 \times 16$  乘法器电路： $16 \times 16$  乘法器 A 1510、 $16 \times 16$  乘法器 B 1511、 $16 \times 16$  乘法器 C 1512 及  $16 \times 16$  乘法器 D 1513。 $16 \times 16$  乘法器 A 1510 具有输入 Source1[15:0] 及 Source2[15:0]。 $16 \times 16$  乘法器 B 1511 具有输入 Source1[31:16] 及 Source2[31:16]。 $16 \times 16$  乘法器 C 1512 具有输入 Source1[47:32] 及 Source2[47:32]。 $16 \times 16$  乘法器 D 1513 具有输入 Source1[63:48] 及 Source2[63:48]。 $16 \times 16$  乘法器 A 1510 及  $16 \times 16$  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  $16 \times 16$  乘法器 C 1512 及  $16 \times 16$  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。

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。

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。

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。

5 因此，程序员将必须截断各结果。

#### 在指令集中包含上述乘-加运算的优点

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。

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-  
10 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。

#### 分组移位

#### 分组移位操作

在本发明的一个实施例中，SCR1 寄存器中包含要移位的数据  
15 (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。

20 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  
25 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。

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。

30 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。

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 SRC1 602、SRC2 603 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。

5       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 SRC1 602 及 SRC2 603 地址的寄存器 209。寄存器 209 向执行单元 130 提供 SRC1 602 寄存器中所存储的分组数据 (Source1) 及存储在 SRC2 603 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。

10       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）。

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。

15       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。

假定数据元素的长度为 8 位，便执行步骤 1612。在步骤 1612 中，  
20 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  
25 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。

假定数据元素的长度为 16 位，便执行步骤 1614。在步骤 1614 中  
30 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。

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。

5 在步骤 1620，将 Result 存储在 DEST 寄存器中。

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。

10

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

表19

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

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

表20

15 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。

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

表21

### 分组数据移位电路

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。

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  $i$  1799 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。

移位单元  $i$  1711 及移位单元  $i+1$  1771 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  $i$  1711 具有 Source1:1731 输入、Source2[63:0] 1733 输入、控制  $i$  1701 输入、下一级  $i$  1713 信号、上一级  $i$  1712 输入及存储在结果寄存器  $i$  1751 中的结果。因此，移位单元  $i+1$  1771 具有 Source1  $i+1$  1732 输入、Source2[63:0] 1733 输入、控制  $i+1$  1702 输入、下一级  $i+1$  1773 信号、上一级  $i+1$  1772 输入及存储在结果寄存器  $i+1$  1752 中的结果。

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  
5 Source1 的部分上的移位操作的结果。

位控制 i 1720 是从操作控制 1700 通过分组数据启动 i 1706 启动的。位控制 i 1720 控制下一级 i 1713 及上一级 i+1 1772。例如，假定移位单元 i 1711 负责 Source1 的 8 个最低位，而移位单元 i+1 1771 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 i  
10 1720 将不允许来自移位单元 i+1 1771 的最低位与移位单元 i 1711 的最高位连通。然而，执行分组字上的移位时，则位控制 i 1720 将允许来自移位单元 i+1 1771 的最低位与移位单元 i 1711 的最高位连通。

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+1 1771 在数据元素 1 上操作，而移位单元 i 1711 在数据元素 0 上操作。  
15 移位单元 i+1 1771 将其最低位移出。然而操作控制 1700 将导致位控制 i 1720 停止从上一级 i+1 1721 接收的该位传播到下一级 i 1713。反之，移位单元 i 1711 以符号位填充最高阶位 Source1[7]。

...	...	...	...	...	...	...	...	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		

表22

20 然而，如果执行分组字算术移位，则将移位单元 i+1 1771 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。

25

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

表23

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。

- 5 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。

#### 在指令集中包含上述移位操作的优点

上述分组移位指令导致 Source1 的各元素移位指定的移位计数。

- 10 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。

#### 传送操作

- 15 传送操作向或从寄存器 209 传送数据。在一个实施例中，SRC2 603 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 SRC1 602。在另一实施例中，SRC1 602 等于 DEST 605。

- 20 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。

- 25 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，如果分组数据存储于寄存器 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。

5 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  
10 存储到指定的整数寄存器中。

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  
15 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。

#### 组装操作

在本发明的一个实施例中，SRC1 602 寄存器包含数据 (Source1)，  
20 SRC2 603 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。

25 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。

30 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。

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 SRC1 602、SRC2 603 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，SRC1 602（或 SRC2 603）可用作 DEST 605。

5 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 SRC1 602 与 SRC2 603 地址的寄存器 209。寄存器 209 向执行单元 130 提供存储在 SRC1 602 寄存器中的分组数据 (Source1) 及存储在 SRC2 603 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。

10 在步骤 1803，解码器 202 启动执行单元 130 去执行适当的组操作。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。

20 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。

25 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。

30 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。

- 5 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。

在步骤 1820，将 Result 存储在 DEST 605 寄存器中。

- 10 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 A<sub>L</sub> 表示 Source1 中的数据元素 A 的低阶 8 位。

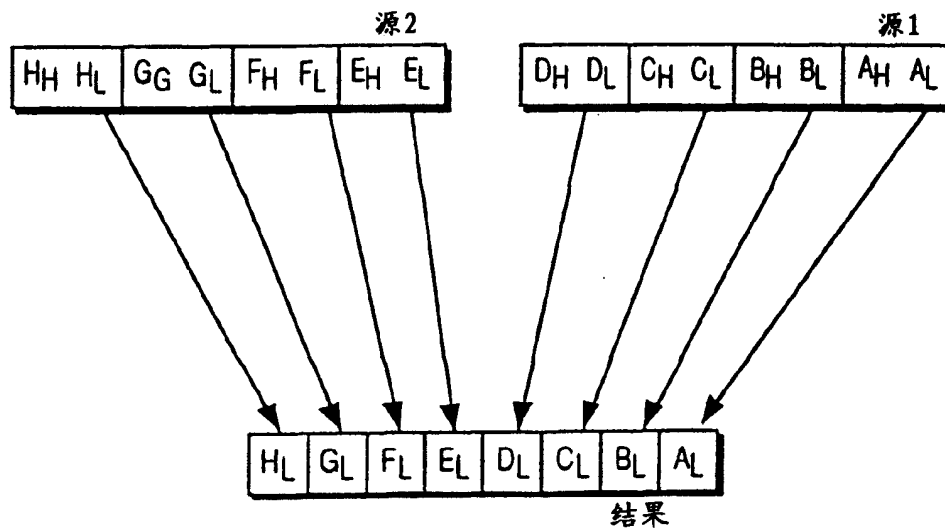


表24

- 15 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。



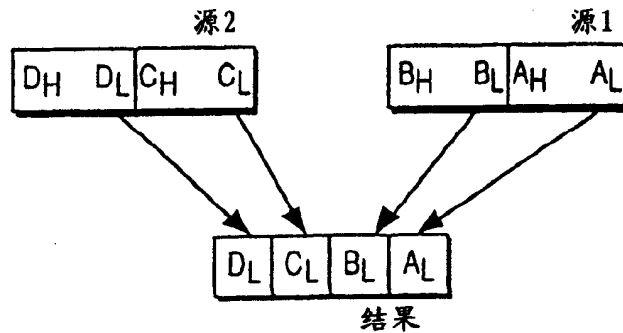


表25

### 组装电路

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  
5 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。

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。

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  
10 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。

15 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。

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  
20 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。

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，或可选地为一个籍位的值 (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。

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。

15

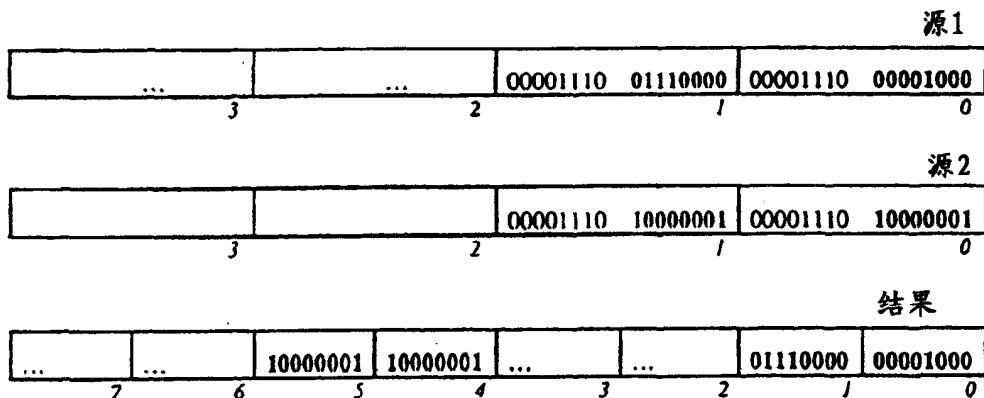


表26

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。

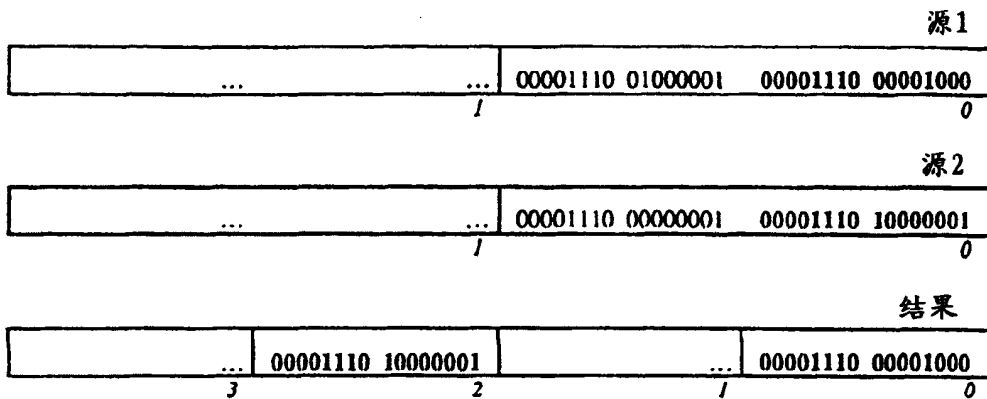


表27

**在指令集中包含上述组装操作的优点**

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：

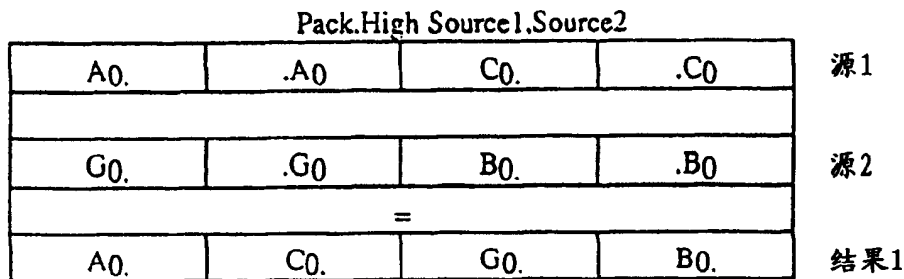


表28

10 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。

**分解操作**

**分解操作**

15 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）。

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。

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。

- 5        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
- 10       步骤 2016。

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

15       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。

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。

20       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。

- 25       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。

在步骤 2020，将 Result 存储在 DEST 605 寄存器中。

- 30       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。

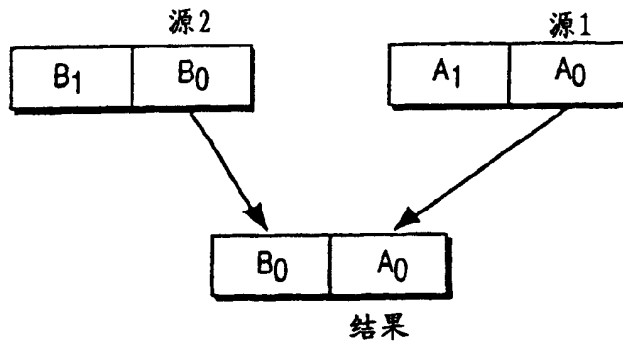


表29

表 30 说明分解字操作 (各数据元素 A<sub>0-3</sub> 及 B<sub>0-3</sub> 包含 16 位) 的寄存器表示。

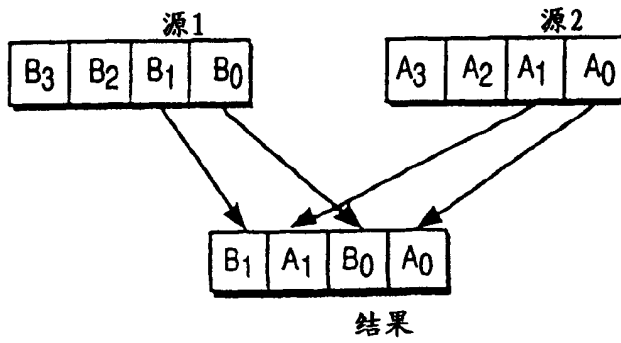


表 30

表 31 说明分解字节操作 (各数据元素 A<sub>0-7</sub> 及 B<sub>0-7</sub> 包含 8 位) 的寄存器表示。

5

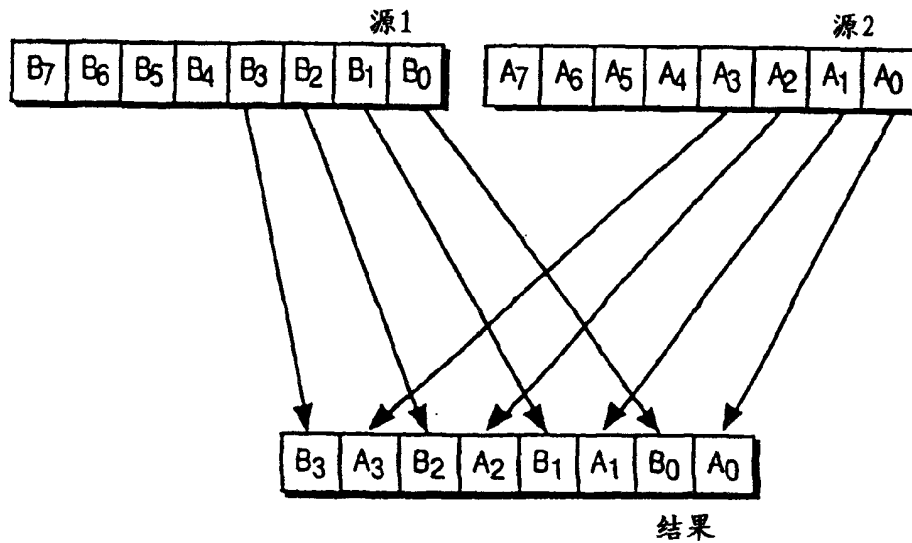


表31

### 分解电路

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。

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。

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。

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.

5 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.

例如, 在表 32 中, 执行了解析字操作. 操作控制 2100 将启动结果寄存器 2153 输出结果 [63: 0] 2160.

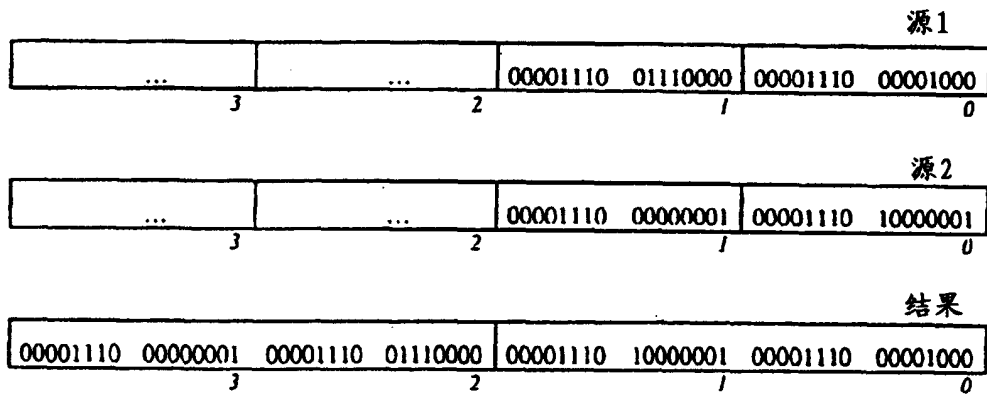


表32

10 然而, 如果执行解析双字, 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.

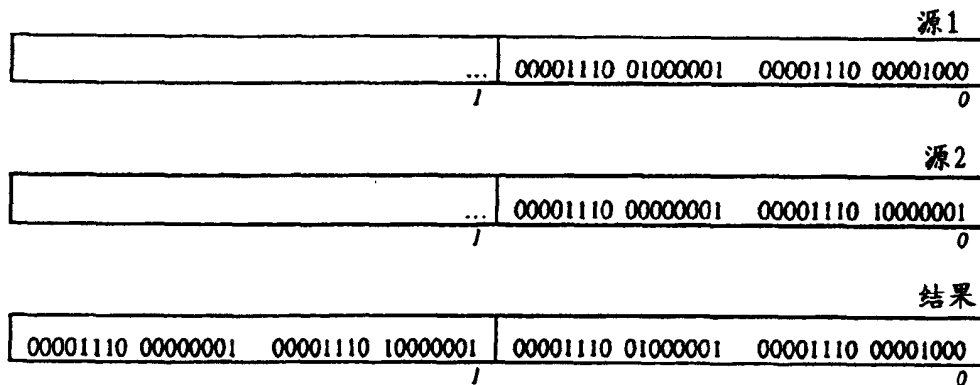


表33

在指令集中包含上述分解指令的优点

15 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。

								源1
00101010	01010101	01010101	11111111	10000000	01110000	10001111	10001000	
7	6	5	4	3	2	1	0	
								源2
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	
7	6	5	4	3	2	1	0	
								结果
00000000	10000000	00000000	01110000	00000000	10001111	00000000	10001000	
		3	2		1		0	

表 34a

- 5 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。

								源1
00101010	01010101	01010101	11111111	10000000	01110000	10001111	10001000	
7	6	5	4	3	2	1	0	
								源2
00000000	00000000	11000000	00000000	11110011	00000000	10001110	10001000	
7	6	5	4	3	2	1	0	
								结果
11110011	10000000	00000000	01110000	10001110	10001111	10001000	10001000	
7	6	5	4	3	2	1	0	

表 34b

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。

**个数计算**

个数计算

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。

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<sub>2</sub>。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。

01110010 00000101	11111111 11111111	01111111 11111111	10001111 10001000
= <sub>3</sub>	= <sub>2</sub>	= <sub>1</sub>	= <sub>0</sub>
00000000 00000110	00000000 00010000	00000000 00001111	00000000 00000111
<sub>3</sub>	<sub>2</sub>	<sub>1</sub>	<sub>0</sub>

表35a

10

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。

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

表35b

15

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。

20

11111111 11111111 11111111 11111111	10000000 11110000 11001111 10001000
=	=
00000000 00000000 00000000 00100000	00000000 00000000 00000000 00001101

表35c

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。

5

11111111 11111111 11111111 11111111 10000000 11110000 11001111 10001000
=
00000000 00000000 00000000 00100000 00000000 00000000 00000000 00101101

表35d

执行个数计数的方法

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 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140 145 150 155 160 165 170 175 180 185 190 195 200 205 210 215 220 225 230 235 240 245 250 255 260 265 270 275 280 285 290 295 300 305 310 315 320 325 330 335 340 345 350 355 360 365 370 375 380 385 390 395 400 405 410 415 420 425 430 435 440 445 450 455 460 465 470 475 480 485 490 495 500 505 510 515 520 525 530 535 540 545 550 555 560 565 570 575 580 585 590 595 600 605 610 615 620 625 630 635 640 645 650 655 660 665 670 675 680 685 690 695 700 705 710 715 720 725 730 735 740 745 750 755 760 765 770 775 780 785 790 795 800 805 810 815 820 825 830 835 840 845 850 855 860 865 870 875 880 885 890 895 900 905 910 915 920 925 930 935 940 945 950 955 960 965 970 975 980 985 990 995 1000 1005 1010 1015 1020 1025 1030 1035 1040 1045 1050 1055 1060 1065 1070 1075 1080 1085 1090 1095 1100 1105 1110 1115 1120 1125 1130 1135 1140 1145 1150 1155 1160 1165 1170 1175 1180 1185 1190 1195 1200 1205 1210 1215 1220 1225 1230 1235 1240 1245 1250 1255 1260 1265 1270 1275 1280 1285 1290 1295 1300 1305 1310 1315 1320 1325 1330 1335 1340 1345 1350 1355 1360 1365 1370 1375 1380 1385 1390 1395 1400 1405 1410 1415 1420 1425 1430 1435 1440 1445 1450 1455 1460 1465 1470 1475 1480 1485 1490 1495 1500 1505 1510 1515 1520 1525 1530 1535 1540 1545 1550 1555 1560 1565 1570 1575 1580 1585 1590 1595 1600 1605 1610 1615 1620 1625 1630 1635 1640 1645 1650 1655 1660 1665 1670 1675 1680 1685 1690 1695 1700 1705 1710 1715 1720 1725 1730 1735 1740 1745 1750 1755 1760 1765 1770 1775 1780 1785 1790 1795 1800 1805 1810 1815 1820 1825 1830 1835 1840 1845 1850 1855 1860 1865 1870 1875 1880 1885 1890 1895 1900 1905 1910 1915 1920 1925 1930 1935 1940 1945 1950 1955 1960 1965 1970 1975 1980 1985 1990 1995 2000 2005 2010 2015 2020 2025 2030 2035 2040 2045 2050 2055 2060 2065 2070 2075 2080 2085 2090 2095 2100 2105 2110 2115 2120 2125 2130 2135 2140 2145 2150 2155 2160 2165 2170 2175 2180 2185 2190 2195 2200 2205 2210 2215 2220 2225 2230 2235 2240 2245 2250 2255 2260 2265 2270 2275 2280 2285 2290 2295 2300 2305 2310 2315 2320 2325 2330 2335 2340 2345 2350 2355 2360 2365 2370 2375 2380 2385 2390 2395 2400 2405 2410 2415 2420 2425 2430 2435 2440 2445 2450 2455 2460 2465 2470 2475 2480 2485 2490 2495 2500 2505 2510 2515 2520 2525 2530 2535 2540 2545 2550 2555 2560 2565 2570 2575 2580 2585 2590 2595 2600 2605 2610 2615 2620 2625 2630 2635 2640 2645 2650 2655 2660 2665 2670 2675 2680 2685 2690 2695 2700 2705 2710 2715 2720 2725 2730 2735 2740 2745 2750 2755 2760 2765 2770 2775 2780 2785 2790 2795 2800 2805 2810 2815 2820 2825 2830 2835 2840 2845 2850 2855 2860 2865 2870 2875 2880 2885 2890 2895 2900 2905 2910 2915 2920 2925 2930 2935 2940 2945 2950 2955 2960 2965 2970 2975 2980 2985 2990 2995 3000 3005 3010 3015 3020 3025 3030 3035 3040 3045 3050 3055 3060 3065 3070 3075 3080 3085 3090 3095 3100 3105 3110 3115 3120 3125 3130 3135 3140 3145 3150 3155 3160 3165 3170 3175 3180 3185 3190 3195 3200 3205 3210 3215 3220 3225 3230 3235 3240 3245 3250 3255 3260 3265 3270 3275 3280 3285 3290 3295 3300 3305 3310 3315 3320 3325 3330 3335 3340 3345 3350 3355 3360 3365 3370 3375 3380 3385 3390 3395 3400 3405 3410 3415 3420 3425 3430 3435 3440 3445 3450 3455 3460 3465 3470 3475 3480 3485 3490 3495 3500 3505 3510 3515 3520 3525 3530 3535 3540 3545 3550 3555 3560 3565 3570 3575 3580 3585 3590 3595 3600 3605 3610 3615 3620 3625 3630 3635 3640 3645 3650 3655 3660 3665 3670 3675 3680 3685 3690 3695 3700 3705 3710 3715 3720 3725 3730 3735 3740 3745 3750 3755 3760 3765 3770 3775 3780 3785 3790 3795 3800 3805 3810 3815 3820 3825 3830 3835 3840 3845 3850 3855 3860 3865 3870 3875 3880 3885 3890 3895 3900 3905 3910 3915 3920 3925 3930 3935 3940 3945 3950 3955 3960 3965 3970 3975 3980 3985 3990 3995 4000 4005 4010 4015 4020 4025 4030 4035 4040 4045 4050 4055 4060 4065 4070 4075 4080 4085 4090 4095 4100 4105 4110 4115 4120 4125 4130 4135 4140 4145 4150 4155 4160 4165 4170 4175 4180 4185 4190 4195 4200 4205 4210 4215 4220 4225 4230 4235 4240 4245 4250 4255 4260 4265 4270 4275 4280 4285 4290 4295 4300 4305 4310 4315 4320 4325 4330 4335 4340 4345 4350 4355 4360 4365 4370 4375 4380 4385 4390 4395 4400 4405 4410 4415 4420 4425 4430 4435 4440 4445 4450 4455 4460 4465 4470 4475 4480 4485 4490 4495 4500 4505 4510 4515 4520 4525 4530 4535 4540 4545 4550 4555 4560 4565 4570 4575 4580 4585 4590 4595 4600 4605 4610 4615 4620 4625 4630 4635 4640 4645 4650 4655 4660 4665 4670 4675 4680 4685 4690 4695 4700 4705 4710 4715 4720 4725 4730 4735 4740 4745 4750 4755 4760 4765 4770 4775 4780 4785 4790 4795 4800 4805 4810 4815 4820 4825 4830 4835 4840 4845 4850 4855 4860 4865 4870 4875 4880 4885 4890 4895 4900 4905 4910 4915 4920 4925 4930 4935 4940 4945 4950 4955 4960 4965 4970 4975 4980 4985 4990 4995 5000 5005 5010 5015 5020 5025 5030 5035 5040 5045 5050 5055 5060 5065 5070 5075 5080 5085 5090 5095 5100 5105 5110 5115 5120 5125 5130 5135 5140 5145 5150 5155 5160 5165 5170 5175 5180 5185 5190 5195 5200 5205 5210 5215 5220 5225 5230 5235 5240 5245 5250 5255 5260 5265 5270 5275 5280 5285 5290 5295 5300 5305 5310 5315 5320 5325 5330 5335 5340 5345 5350 5355 5360 5365 5370 5375 5380 5385 5390 5395 5400 5405 5410 5415 5420 5425 5430 5435 5440 5445 5450 5455 5460 5465 5470 5475 5480 5485 5490 5495 5500 5505 5510 5515 5520 5525 5530 5535 5540 5545 5550 5555 5560 5565 5570 5575 5580 5585 5590 5595 5600 5605 5610 5615 5620 5625 5630 5635 5640 5645 5650 5655 5660 5665 5670 5675 5680 5685 5690 5695 5700 5705 5710 5715 5720 5725 5730 5735 5740 5745 5750 5755 5760 5765 5770 5775 5780 5785 5790 5795 5800 5805 5810 5815 5820 5825 5830 5835 5840 5845 5850 5855 5860 5865 5870 5875 5880 5885 5890 5895 5900 5905 5910 5915 5920 5925 5930 5935 5940 5945 5950 5955 5960 5965 5970 5975 5980 5985 5990 5995 6000 6005 6010 6015 6020 6025 6030 6035 6040 6045 6050 6055 6060 6065 6070 6075 6080 6085 6090 6095 6100 6105 6110 6115 6120 6125 6130 6135 6140 6145 6150 6155 6160 6165 6170 6175 6180 6185 6190 6195 6200 6205 6210 6215 6220 6225 6230 6235 6240 6245 6250 6255 6260 6265 6270 6275 6280 6285 6290 6295 6300 6305 6310 6315 6320 6325 6330 6335 6340 6345 6350 6355 6360 6365 6370 6375 6380 6385 6390 6395 6400 6405 6410 6415 6420 6425 6430 6435 6440 6445 6450 6455 6460 6465 6470 6475 6480 6485 6490 6495 6500 6505 6510 6515 6520 6525 6530 6535 6540 6545 6550 6555 6560 6565 6570 6575 6580 6585 6590 6595 6600 6605 6610 6615 6620 6625 6630 6635 6640 6645 6650 6655 6660 6665 6670 6675 6680 6685 6690 6695 6700 6705 6710 6715 6720 6725 6730 6735 6740 6745 6750 6755 6760 6765 6770 6775 6780 6785 6790 6795 6800 6805 6810 6815 6820 6825 6830 6835 6840 6845 6850 6855 6860 6865 6870 6875 6880 6885 6890 6895 6900 6905 6910 6915 6920 6925 6930 6935 6940 6945 6950 6955 6960 6965 6970 6975 6980 6985 6990 6995 7000 7005 7010 7015 7020 7025 7030 7035 7040 7045 7050 7055 7060 7065 7070 7075 7080 7085 7090 7095 7100 7105 7110 7115 7120 7125 7130 7135 7140 7145 7150 7155 7160 7165 7170 7175 7180 7185 7190 7195 7200 7205 7210 7215 7220 7225 7230 7235 7240 7245 7250 7255 7260 7265 7270 7275 7280 7285 7290 7295 7300 7305 7310 7315 7320 7325 7330 7335 7340 7345 7350 7355 7360 7365 7370 7375 7380 7385 7390 7395 7400 7405 7410 7415 7420 7425 7430 7435 7440 7445 7450 7455 7460 7465 7470 7475 7480 7485 7490 7495 7500 7505 7510 7515 7520 7525 7530 7535 7540 7545 7550 7555 7560 7565 7570 7575 7580 7585 7590 7595 7600 7605 7610 7615 7620 7625 7630 7635 7640 7645 7650 7655 7660 7665 7670 7675 7680 7685 7690 7695 7700 7705 7710 7715 7720 7725 7730 7735 7740 7745 7750 7755 7760 7765 7770 7775 7780 7785 7790 7795 7800 7805 7810 7815 7820 7825 7830 7835 7840 7845 7850 7855 7860 7865 7870 7875 7880 7885 7890 7895 7900 7905 7910 7915 7920 7925 7930 7935 7940 7945 7950 7955 7960 7965 7970 7975 7980 7985 7990 7995 8000 8005 8010 8015 8020 8025 8030 8035 8040 8045 8050 8055 8060 8065 8070 8075 8080 8085 8090 8095 8100 8105 8110 8115 8120 8125 8130 8135 8140 8145 8150 8155 8160 8165 8170 8175 8180 8185 8190 8195 8200 8205 8210 8215 8220 8225 8230 8235 8240 8245 8250 8255 8260 8265 8270 8275 8280 8285 8290 8295 8300 8305 8310 8315 8320 8325 8330 8335 8340 8345 8350 8355 8360 8365 8370 8375 8380 8385 8390 8395 8400 8405 8410 8415 8420 8425 8430 8435 8440 8445 8450 8455 8460 8465 8470 8475 8480 8485 8490 8495 8500 8505 8510 8515 8520 8525 8530 8535 8540 8545 8550 8555 8560 8565 8570 8575 8580 8585 8590 8595 8600 8605 8610 8615 8620 8625 8630 8635 8640 8645 8650 8655 8660 8665 8670 8675 8680 8685 8690 8695 8700 8705 8710 8715 8720 8725 8730 8735 8740 8745 8750 8755 8760 8765 8770 8775 8780 8785 8790 8795 8800 8805 8810 8815 8820 8825 8830 8835 8840 8845 8850 8855 8860 8865 8870 8875 8880 8885 8890 8895 8900 8905 8910 8915 8920 8925 8930 8935 8940 8945 8950 8955 8960 8965 8970 8975 8980 8985 8990 8995 9000 9005 9010 9015 9020 9025 9030 9035 9040 9045 9050 9055 9060 9065 9070 9075 9080 9085 9090 9095 9100 9105 9110 9115 9120 9125 9130 9135 9140 9145 9150 9155 9160 9165 9170 9175 9180 9185 9190 9195 9200 9205 9210 9215 9220 9225 9230 9235 9240 9245 9250 9255 9260 9265 9270 9275 9280 9285 9290 9295 9300 9305 9310 9315 9320 9325 9330 9335 9340 9345 9350 9355 9360 9365 9370 9375 9380 9385 9390 9395 9400 9405 9410 9415 9420 9425 9430 9435 9440 9445 9450 9455 9460 9465 9470 9475 9480 9485 9490 9495 9500 9505 9510 9515 9520 9525 9530 9535 9540 9545 9550 9555 9560 9565 9570 9575 9580 9585 9590 9595 9600 9605 9610 9615 9620 9625 9630 9635 9640 9645 9650 9655 9660 9665 9670 9675 9680 9685 9690 9695 9700 9705 9710 9715 9720 9725 9730 9735 9740 9745 9750 9755 9760 9765 9770 9775 9780 9785 9790 9795 9800 9805 9810 9815 9820 9825 9830 9835 9840 9845 9850 9855 9860 9865 9870 9875 9880 9885 9890 9895 9900 9905 9910 9915 9920 9925 9930 9935 9940 9945 9950 9955 9960 9965 9970 9975 9980 9985 9990 9995 10000

在步骤 2202，解码器 202 通过内部总线 170 存取寄存器文件 150 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140 145 150 155 160 165 170 175 180 185 190 195 200 205 210 215 220 225 230 235 240 245 250 255 260 265 270 275 280 285 290 295 300 305 310 315 320 325 330 335 340 345 350 355 360 365 370 375 380 385 390 395 400 405 410 415 420 425 430 435 440 445 450 455 460 465 470 475 480 485 490 495 500 505 510 515 520 525 530 535 540 545 550 555 560 565 570 575 580 585 590 595 600 605 610 615 620 625 630 635 640 645 650 655 660 665 670 675 680 685 690 695 700 705 710 715 720 725 730 735 740 745 750 755 760 765 770 775 780 785 790 795 800 805 810 815 820 825 830 835 840 845 850 855 860 865 870 875 880 885 890 895 900 905 910 915 920 925 930 935 940 945 950 955 960 965 970 975 980 985 990 995 1000

过内部总线 170 将分组数据传递给执行单元 130。

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。

- 5        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。

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。

#### 15                    在一个数据元素上执行个数计数的方法

- 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、CCarry1c、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 CSum2a、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。
- 30        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<sub>2</sub>时。Result 将为 16 并用 0000000000010000<sub>2</sub>表示。

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。

#### 执行个数计数的电路

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。

10 图 24 示出一个电路，其中 Source1 总线 2401 通过 Source1<sub>m</sub> 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  
15 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  
20 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。

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。

#### 25 在一个数据元素上执行个数计数的电路

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  
30 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。

popcnt 电路 2408a 采用 4- $\rightarrow$ 2 进位保留加法器（除非另有指定，CSA 将指 4- $\rightarrow$ 2 进位保留加法器），popcnt 电路 2408a-d 中可能采用的 4- $\rightarrow$ 2 进位保留加法器是本技术中众所周知的。4- $\rightarrow$ 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- $\rightarrow$ 2 进位保留加法器。这四个 4- $\rightarrow$ 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。

虽然采用了 4- $\rightarrow$ 2 进位保留加法器，替代实施例中可采用 3- $\rightarrow$ 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。

Source1<sub>15:0</sub> 2406a 携带 Source1 的位 15 至位 0。第一个四位耦合在 4- $\rightarrow$ 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 2510 a-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 CSA 2530 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。

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。

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。

#### 在指令集中加入上述个数计数操作的优点

- 5 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。
- 10 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。

#### 逻辑运算

#### 逻辑运算

- 15 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”。

- 20 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。

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。

- 25 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 SRC1 602、SRC2 603 及 DEST 604 地址。

- 30 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 SRC1 602 及 SRC2 603 地址的寄存器 209。寄存器 209 向执行单元 130 提供存储在 SRC1 602 寄存器中的分组数据 (Source1) 及存储在 SRC2 603 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。

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。

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  
5 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。

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

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。

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

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  
20 Result 位 63 至 0。

在步骤 2620, 将 Result 存储在 DEST 寄存器中。

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。  
25

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

表36

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。

5

#### 分组数据逻辑电路

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。

10 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。

15 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。

#### 在指令集中加入上述逻辑运算的优点

20 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。

#### 分组比较

##### 分组比较操作

25 在本发明的一个实施例中，SRC1 602 寄存器中包含要比较的数据 (Source1)，SRC2 603 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。

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  
5 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。

10 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。

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 SRC1 602、SRC2 603 及 DEST 605 地址; 饱和/不饱和 (对比较操作没有  
15 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, SRC1 602 (或 SRC2 603) 可用作 DEST 605。

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 SRC1 602 与 SRC2 603 地址的寄存器 209。寄存器 209 向执行单元 130 提供存储在 SRC1 602 寄存器中的分组数据 (Source1) 及存  
20 储在 SRC2 603 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。

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。

25 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  
30 32 位数据元素长度的分组比较 (分组双字 403)。

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.

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.

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.

在步骤 2820, 将 Result 存储在 DEST 605 寄存器中.

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<sub>2</sub>.

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

25

表 37

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。

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

5

表 38

分组数据比较电路

在一个实施例中，在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。

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。

15 比较单元 i 2911 及比较单元 i+1 2971 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。

20 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 i 2901 输入、下一级 i 2913 信号、上一级 i 2912 输入及存储在结

果寄存器  $i+1$  2951 中的结果。因此，比较单元  $i+1$  2971 具有 Source $1_{i+1}$  2932 输入、Source $2_{i+1}$  2934 输入、控制  $i+1$  2902 输入、下一级  $i+1$  2973 信号、上一级  $i+1$  2972 输入及存储在结果寄存器  $i+1$  2952 中的结果。

Source $1_n$  输入通常是 Source $1$  的一个 8 位部分。8 位表示最小类型的

5 数据元素，一个分组字节 401 的数据元素。Source $2$  输入是 Source $2$  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

10 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。

15 取决于所要求的关系及 Source $1$  与 Source $2$  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

20 示该比较单元在其上操作的 Source $1$  与 Source $2$  的部分上的比较操作的结果。

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  $i+1$  2972。例如，假定比较单元  $i$  2911 负责 Source $1$  与 Source $2$  的 8 个最低位，而比较

25 单元  $i+1$  2971 负责 Source $1$  与 Source $2$  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  $i+1$  2971 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

30  $i+1$  2971 的结果（在一个实施例中为溢出）信息传递给比较单元  $i$  2911。

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

$i+1$  2971 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  $i+1$  2971 比较一个字的最高 8 位并通过上一级  $i+1$  2972 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  $i+1$  2972 及下一级  $i$  2913 接收的结果信息。

...	...	...	...	...	...	00001110	00001000
> 7	> 6	> 5	> 4	> 3	> 2	> 1	> 0
...	...	...	...	...	...	00001110	10001000
↓	↓	↓	↓	↓	↓	↓	↓
...	...	...	...	...	...	00000000	11111111
7	6	5	4	3	2	1	0

表 39

10 然而，如果执行分组字带符号大于比较，则比较单元  $i+1$  2971 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。

...	...	...	00001110	00001000
> 3	> 2	> 1	>	0
...	...	...	00001110	10000001
↓	↓	↓	↓	↓
...	...	...	00000000	00000000
3	2	1		0

表 40

15

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。

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。

在指令集中加入上述分组

比较操作的优点

- 5 上述分组比较指令将 Source1 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 (if Y>A then X=X+B; else X=X)，如下面
- 10 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=Y0-1
>	>	
00000000	00000001	Source2=A0-1
=		
FFFFFFFF	00000000	掩码

Packed AND Source3,Mask		
00000005	0000000A	Source3=B0-1
>	>	
FFFFFFFF	00000000	掩码
=		
00000005	00000000	结果

Packed Add Source4, Result		
00000010	00000020	Source4=X0-1
>	>	
00000005	00000000	结果
=		
00000015	00000020	新 X0-1 值

表41

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。

5 多媒体算法示例

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。

10 1) 复数乘法

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， $r_1 i_1$  与  $r_2 i_2$ ）的乘法是按照下列等式执行的：

实部 =  $r_1 \cdot r_2 - i_1 \cdot i_2$

15 虚部 =  $r_1 \cdot i_2 + r_2 \cdot i_1$

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。

Multiply-Add Source1, Source2				
r1	i2	r1	i1	源1
r2	-i2	i2	r2	源2
=				
实部 $r_1 r_2 - i_1 i_2$		虚部 $r_1 i_2 + r_2 i_1$		结果1

表42a

20 作为另一示例，表 42b 示出用来将三个复数一起乘的指令。

Multiply-Add Source1, Source2				
r1	i1	r1	i1	源1
r2	-i2	i2	r2	源2
=				
实部1 $r1r2-i1i2$		虚部1 $r1i2+r2i1$		结果1

Packed Shift Right Source1, Source2				
实部1		虚部1		结果1
16				
=				
	实部1		虚部1	结果2

Pack Result2, Result2				
	实部1		虚部1	结果2
	实部1		虚部1	结果2
=				
实部1	虚部1	实部1	虚部1	结果3

Multiply-Add Result3, Source3				
实部1 $r1r2-i1i2$	虚部1 $r1i2+r2i1$	实部1 $r1r2-i1i2$	虚部1 $r1i2+r2i1$	结果3
r3	-i3	i3	r3	源3
=				
实部2		虚部2		结果4

表42b



## 2) 乘累加运算

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。

5

Multiply-Add Source1, Source2				
0	0	A <sub>1</sub>	A <sub>2</sub>	源1
0	0	B <sub>1</sub>	B <sub>2</sub>	源2
=				
0		A <sub>1</sub> B <sub>1</sub> +A <sub>2</sub> B <sub>2</sub>		结果1

Multiply-Add Source3, Source4				
0	0	A <sub>3</sub>	A <sub>4</sub>	源3
0	0	B <sub>3</sub>	B <sub>4</sub>	源4
=				
0		A <sub>3</sub> A <sub>4</sub> +B <sub>3</sub> B <sub>4</sub>		结果2

Unpacked Add Result1, Result2		
0	A <sub>1</sub> B <sub>1</sub> +A <sub>2</sub> B <sub>2</sub>	结果1
0	A <sub>3</sub> A <sub>4</sub> +B <sub>3</sub> B <sub>4</sub>	结果2
=		
0	A <sub>1</sub> B <sub>1</sub> +A <sub>2</sub> B <sub>2</sub> +A <sub>3</sub> A <sub>4</sub> +B <sub>3</sub> B <sub>4</sub>	结果3

表43

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

A1	A2	A3	A4	源1
B1	B2	B3	B4	源2
=				
A1B1+A2B2		A3B3+A4B4		结果1

## Multiply-Add Source3, Source4

A5	A6	A7	A8	源3
B5	B6	B7	B8	源4
=				
A5B5+A6B6		A7B7+A8B8		结果2

## Packed Add Result1, Result2

A1B1+A2B2		A3B3+A4B4		结果1
A5B5+A6B6		A7B7+A8B8		结果2
=				
A1B1+A2B2+A5B5+A6B6		A3B3+A4B4+A7B7+A8B8		结果3

## Unpack High Result3, Source5

A1B1+A2B2+A5B5+A6B6		A3B3+A4B4+A7B7+A8B8		结果3
0		0		源5
=				
0		A1B1+A2B2+A5B5+A6B6		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

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

表44

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。

Multiply-Add Source1, Source2				
A <sub>1</sub>	A <sub>2</sub>	C <sub>1</sub>	C <sub>2</sub>	源1
B <sub>1</sub>	B <sub>2</sub>	D <sub>1</sub>	D <sub>2</sub>	源2
=				
$A_1B_1+A_2B_2$		$C_1D_1+C_2D_2$		结果1

表45

5

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				
A1	A2	C1	C2	源1
B1	B2	D1	D2	源2
=				
A1B1+A2B2		C1D1+C2D2		结果1

Multiply-Add Source3, Source4				
A3	A4	C3	C4	源3
B3	B4	D3	D4	源4
=				
A3B3+A4B4		C3D3+C4D4		结果2

Packed Add Result1, Result2		
A1B1+A2B2	C1D1+C2D2	结果1
A3B3+A4B4	C3D3+C4D4	结果2
=		
A1B1+A2B2+A3B3+A4B4	C1D1+C2D2+C3D3+C4D4	结果6

表46

### 3) 点积算法

- 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。

两个长度 N 的序列 A 与 B 的点积定义为：

$$\text{Result} = \sum_{i=0}^{N-1} A_i \cdot B_i$$

- 10 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。

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：

- 5           1) 使用传送指令从A序列取4个16位值来生成Source1;  
               2) 使用传送指令从B序列取4个16位值来生成Source2; 以及  
               3) 使用乘-加、分组加法及移位指令如上所述相乘与累加。

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  
 10 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。

#### 4) 二维环路滤波器

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。  
 15 波。

1	2	1
2	4	2
1	2	1

表47

为了计算位置(x, y)上的象素的新值，使用下述等式：

20 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)

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  
 25 二滤波器则在垂直方向上。

表48示出象素数据的一个8×8块的表示。

←8→							
A <sub>0</sub>	A <sub>1</sub>	A <sub>2</sub>	A <sub>3</sub>	A <sub>4</sub>	A <sub>5</sub>	A <sub>6</sub>	A <sub>7</sub>
B <sub>0</sub>	B <sub>1</sub>	B <sub>2</sub>	B <sub>3</sub>	B <sub>4</sub>	B <sub>5</sub>	B <sub>6</sub>	B <sub>7</sub>
C <sub>0</sub>	C <sub>1</sub>	C <sub>2</sub>	C <sub>3</sub>	C <sub>4</sub>	C <sub>5</sub>	C <sub>6</sub>	C <sub>7</sub>
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
I <sub>0</sub>	I <sub>1</sub>	I <sub>2</sub>	I <sub>3</sub>	I <sub>4</sub>	I <sub>5</sub>	I <sub>6</sub>	I <sub>7</sub>

表48

执行下述步骤来实现象素数据的这一  $8 \times 8$  块上的滤波器的水平通过:

- 1) 使用传送指令存取 8 个 8 位象素值作为分组数据;
  - 5     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;
  - 3) 复制 Source1 两次生成 Source2 与 Source3;
  - 4) 在 Source1 上执行非分组向右移位 16 位;
  - 5) 在 Source3 上执行非分组向左移位 16 位;
  - 10    6) 通过执行下述分组加法生成 (Source1 + 2\* Source2 + Source3);
    - a) Source1 = Source1 + Source2;
    - b) Source1 = Source1 + Source2;
    - c) Source1 = Source1 + Source3;
  - 15    7) 作为一个  $8 \times 8$  中间结果数组的一部分存储得出的分组字数据;
- 以及
- 8) 重复这些步骤直到生成如下面表 49 中所示的整个  $8 \times 8$  中间结果数组 (如 IA<sub>0</sub> 表示来自表 49 的 A<sub>0</sub> 的中间结果)。

←16→

IA <sub>0</sub>	IA <sub>1</sub>	IA <sub>2</sub>	IA <sub>3</sub>	IA <sub>4</sub>	IA <sub>5</sub>	IA <sub>6</sub>	IA <sub>7</sub>
IB <sub>0</sub>	IB <sub>1</sub>	IB <sub>2</sub>	IB <sub>3</sub>	IB <sub>4</sub>	IB <sub>5</sub>	IB <sub>6</sub>	IB <sub>7</sub>
IC <sub>0</sub>	IC <sub>1</sub>	IC <sub>2</sub>	IC <sub>3</sub>	IC <sub>4</sub>	IC <sub>5</sub>	IC <sub>6</sub>	IC <sub>7</sub>
·	·	·	·	·	·	·	·
·	·	·	·	·	·	·	·
·	·	·	·	·	·	·	·
II <sub>0</sub>	II <sub>1</sub>	II <sub>2</sub>	II <sub>3</sub>	II <sub>4</sub>	II <sub>5</sub>	II <sub>6</sub>	II <sub>7</sub>

表 49

执行下述步骤来实现  $8 \times 8$  中间结果数组上的滤波器的垂直通过:

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

←16→

IA <sub>0</sub>	IA <sub>1</sub>	IA <sub>2</sub>	IA <sub>3</sub>	源1
IB <sub>0</sub>	IB <sub>1</sub>	IB <sub>2</sub>	IB <sub>3</sub>	源2
IC <sub>0</sub>	IC <sub>1</sub>	IC <sub>2</sub>	IC <sub>3</sub>	源3

表50

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

- 10 a) Source1 = Source1 + Source2;
- b) Source1 = Source1 + Source2;
- c) Source1 = Source1 + Source3;

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;

- 15 4) 组装带有饱和的结果 Source1, 将 16 位值转换回 8 位象素值;
- 5) 将得出的分组字节数据作为一个  $8 \times 8$  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

6) 重复这些步骤直到生成整个  $8 \times 8$  结果数组为止。

值得指出的是,  $8 \times 8$  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。

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。

### 5) 运动估计 (Motion Estimation)

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)。

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  $16 \times 16$  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:

- 1) 利用传送指令从块 A 中取 8 个 8 位值生成 Source1;
- 2) 利用传送指令从块 B 中取 8 个 8 位值生成 Source2;
- 3) 执行带饱和的分组减法从 Source2 中减去 Source1 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);
- 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);
- 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；

6) 重复这些步骤直到处理完  $16 \times 16$  块。

5 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。

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。

### 10 6) 离散余弦变换

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。

15 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。

20 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：

$$X = a*x + b*y$$

$$Y = c*x - d*y$$

25 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。

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：

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)；

30 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

3) 利用 Source1 与 Source2 执行乘 - 加指令生成 Result (见下面

表 51)。

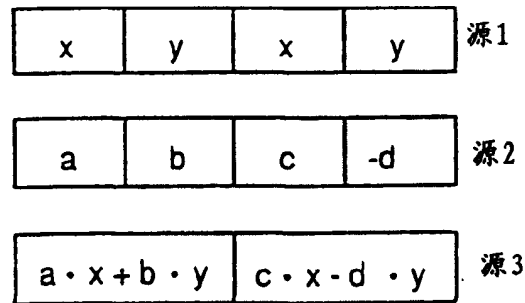


表51

5 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  
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。

IEEE 文件规定了电视会议必须执行的逆 DCT 所用的精度。(见 IEEE  
10 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. 345 East 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  
因为它使用 16 位输入来生成 32 位输出。

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。

#### 替代实施例

15 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；  
20 以及 4) 一个电路单元来执行个数计数操作。

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  
25 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）。

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。

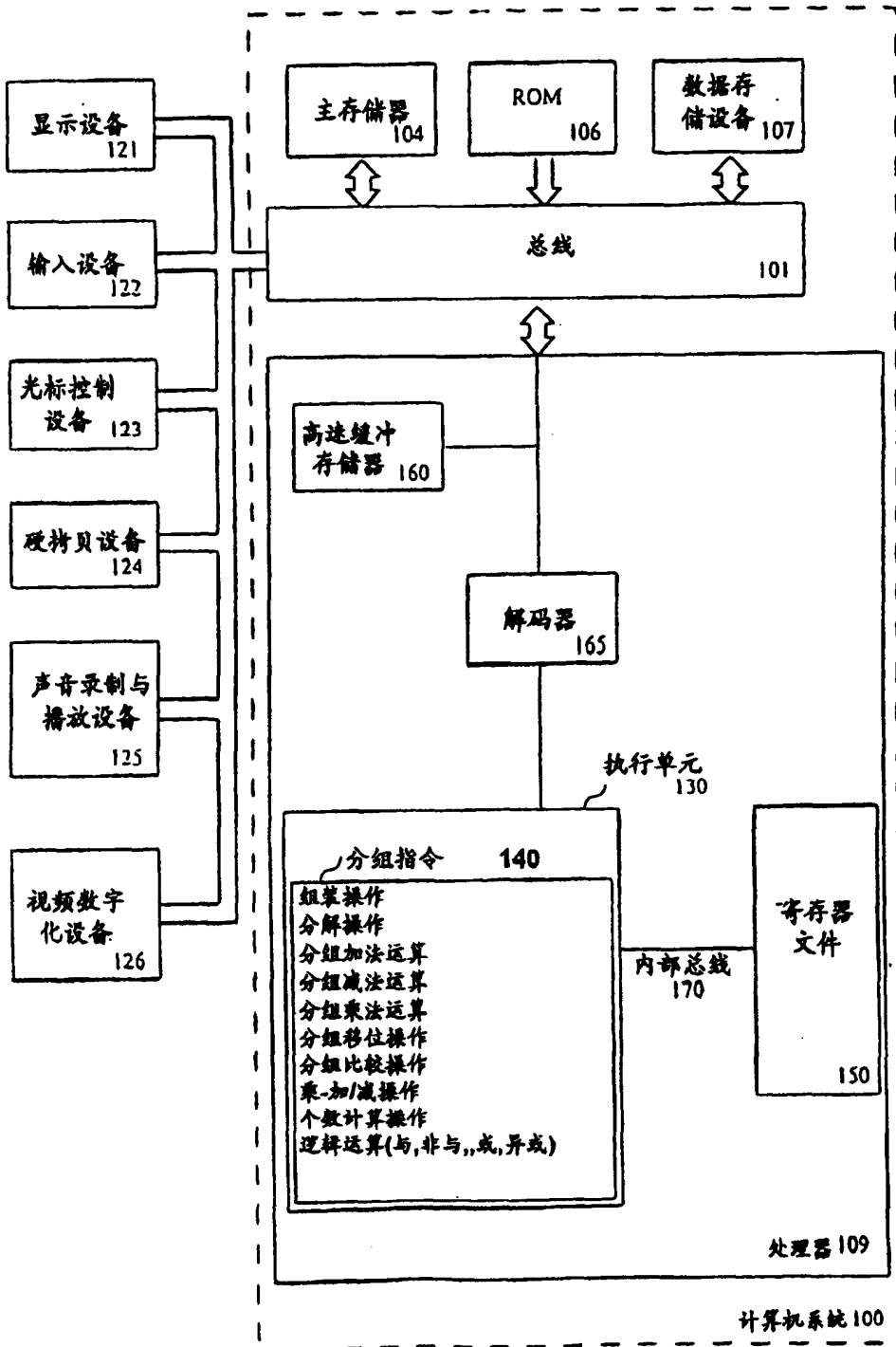


图 1

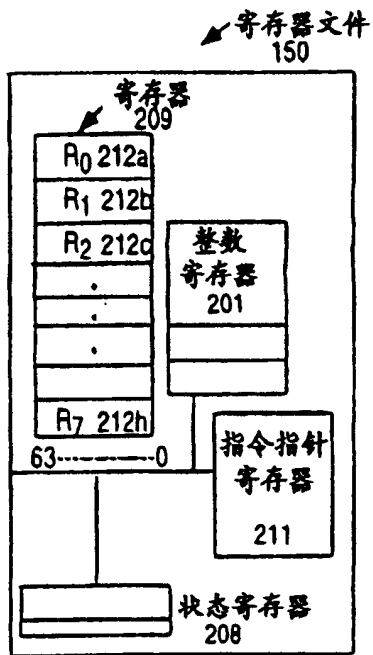


图 2

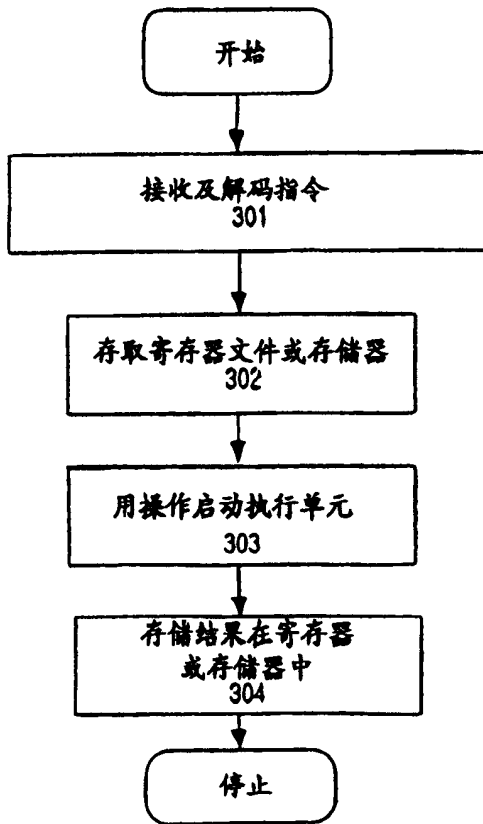


图 3

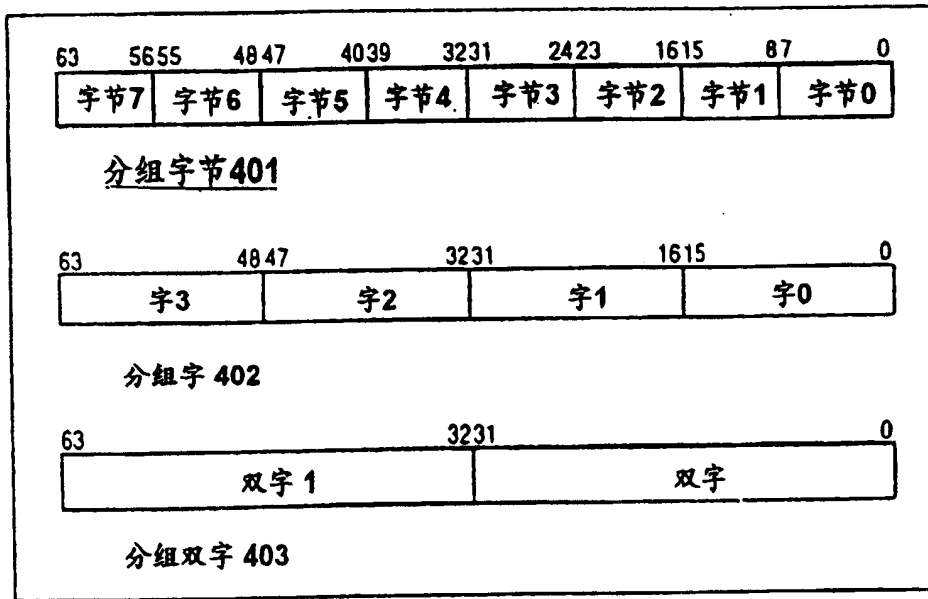


图 4

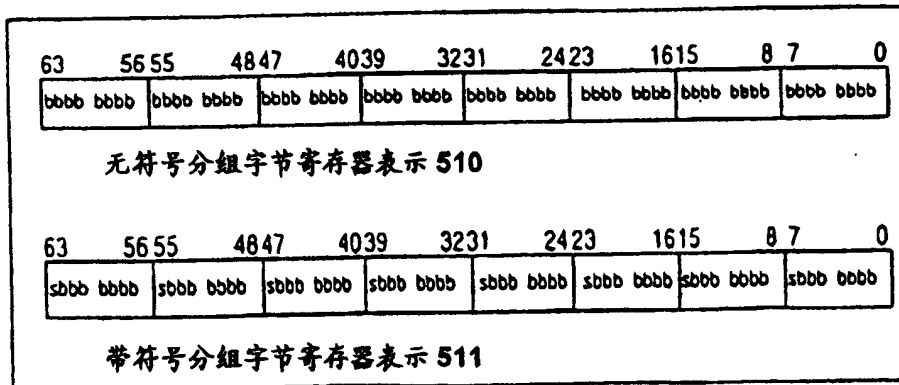


图 5 a

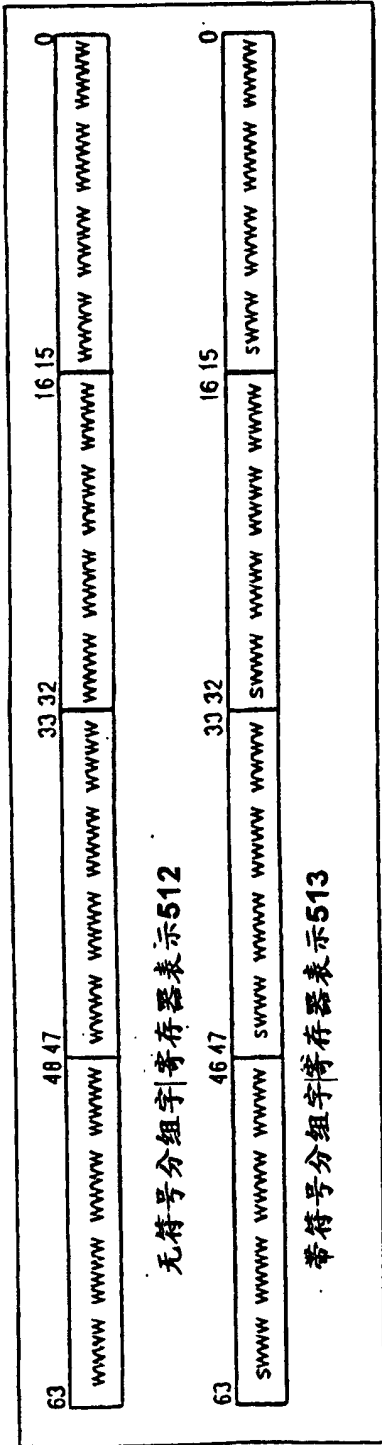


图 5 b

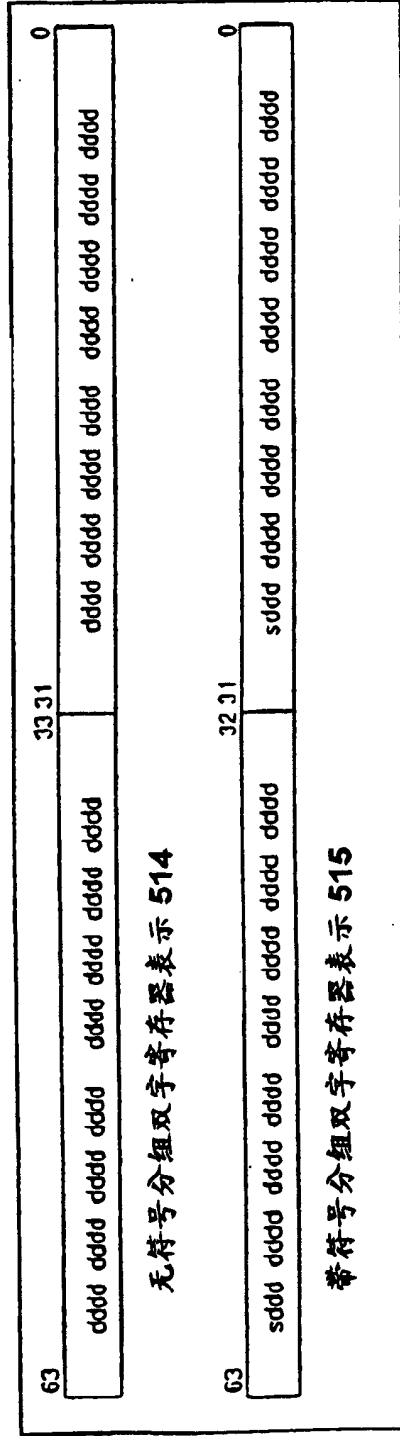


图 5 c

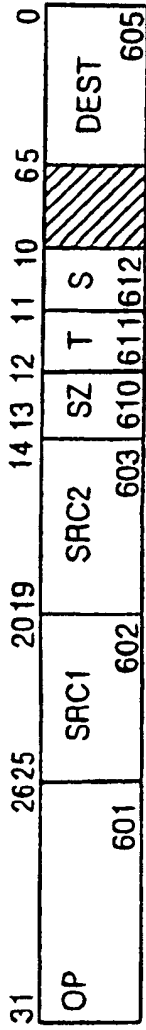


图 6 a

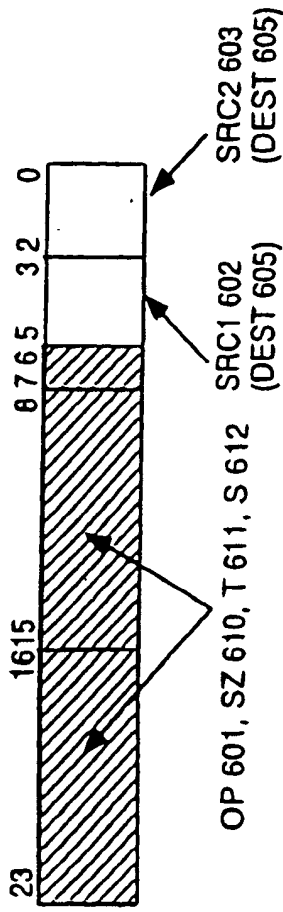


图 6 b



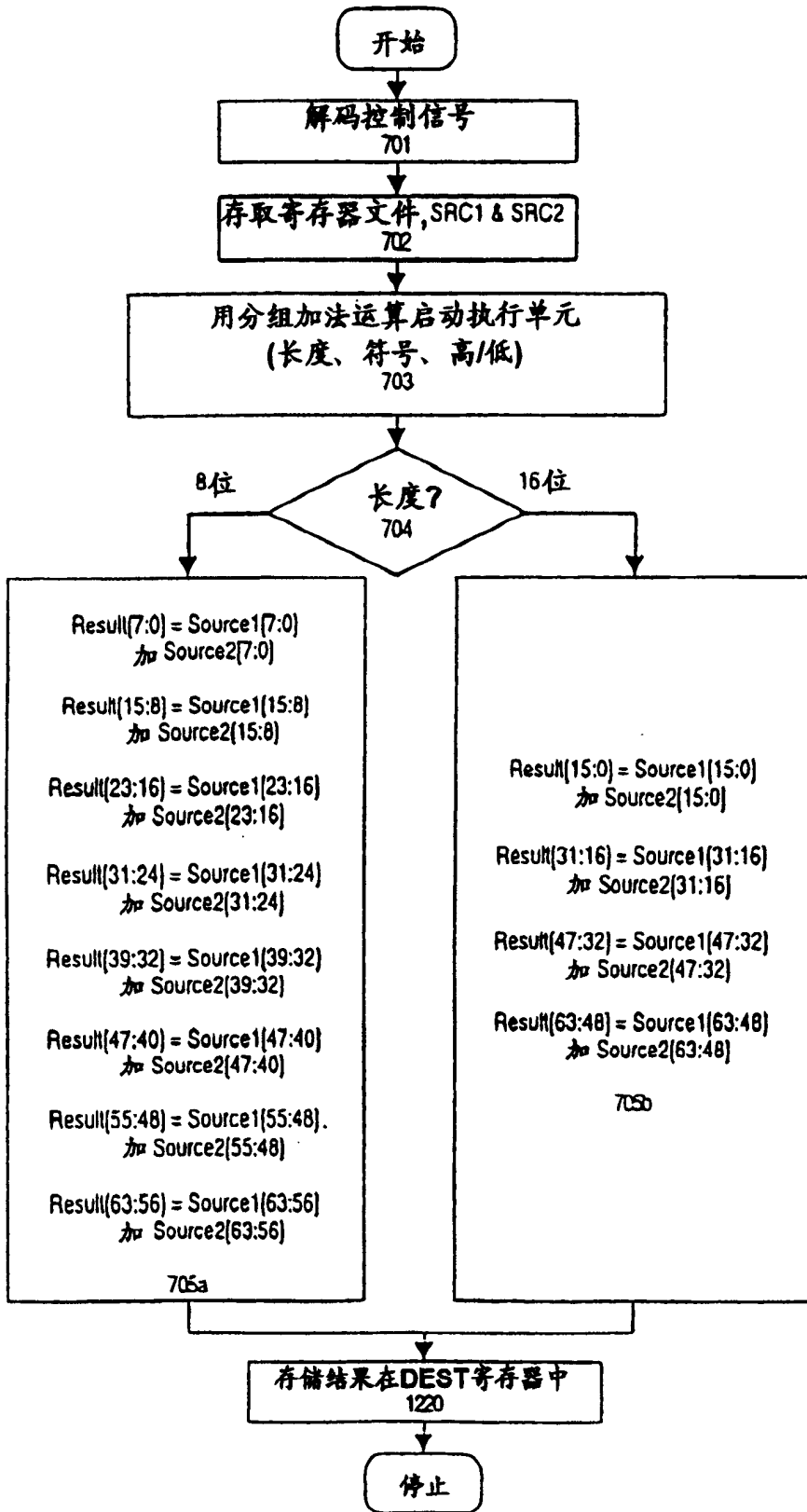


图 7 a

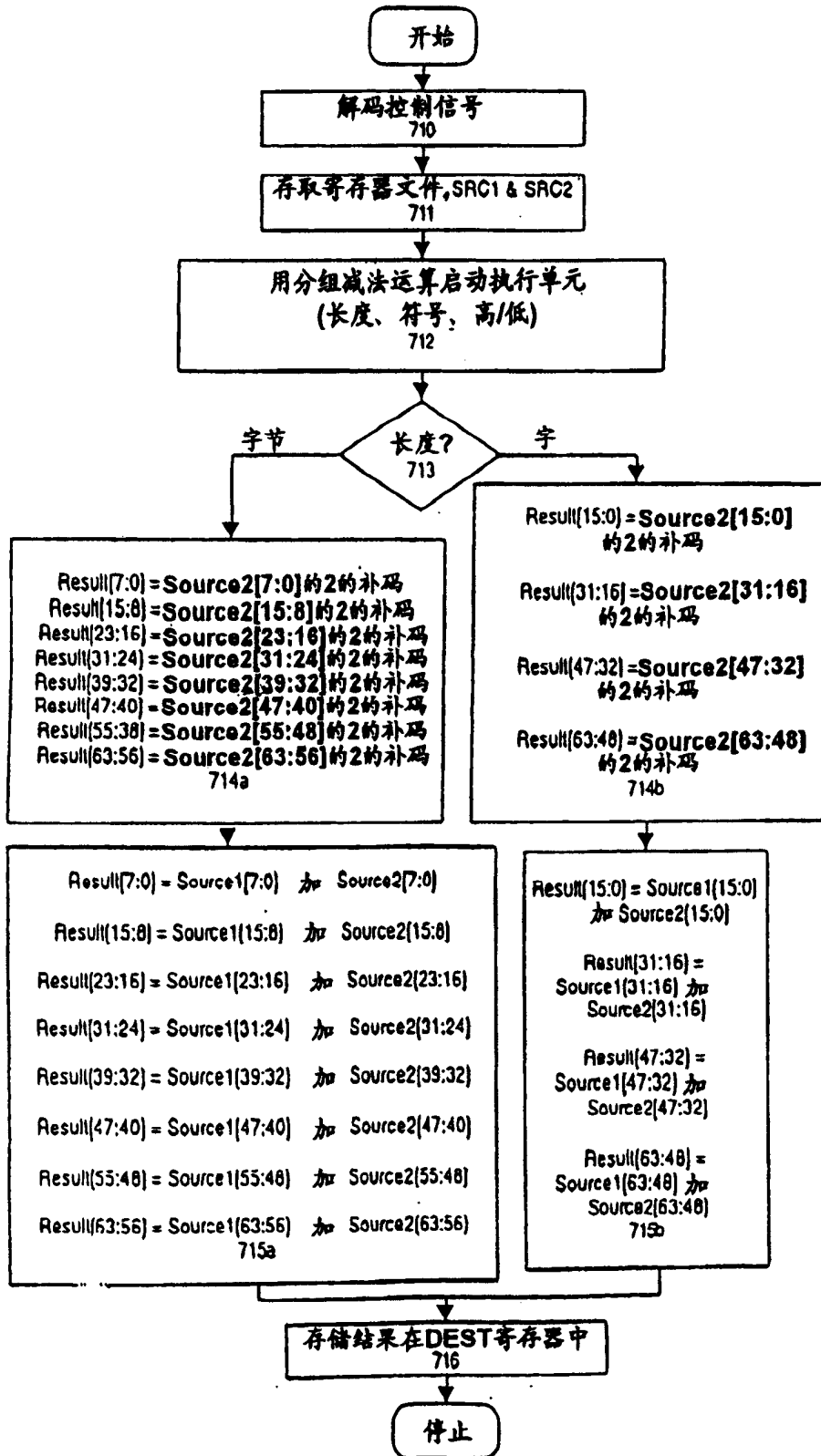


图 7 b

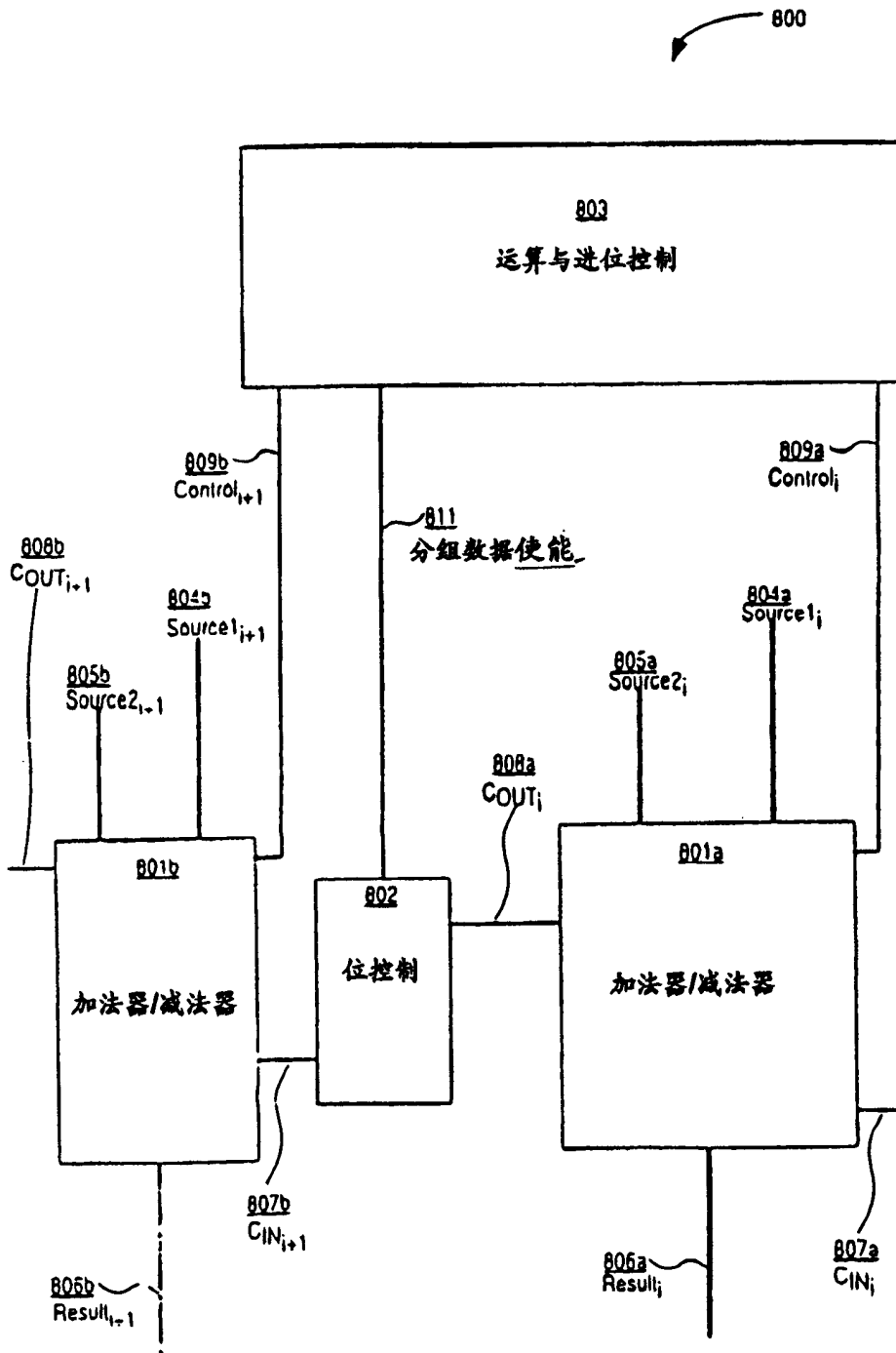


图 8

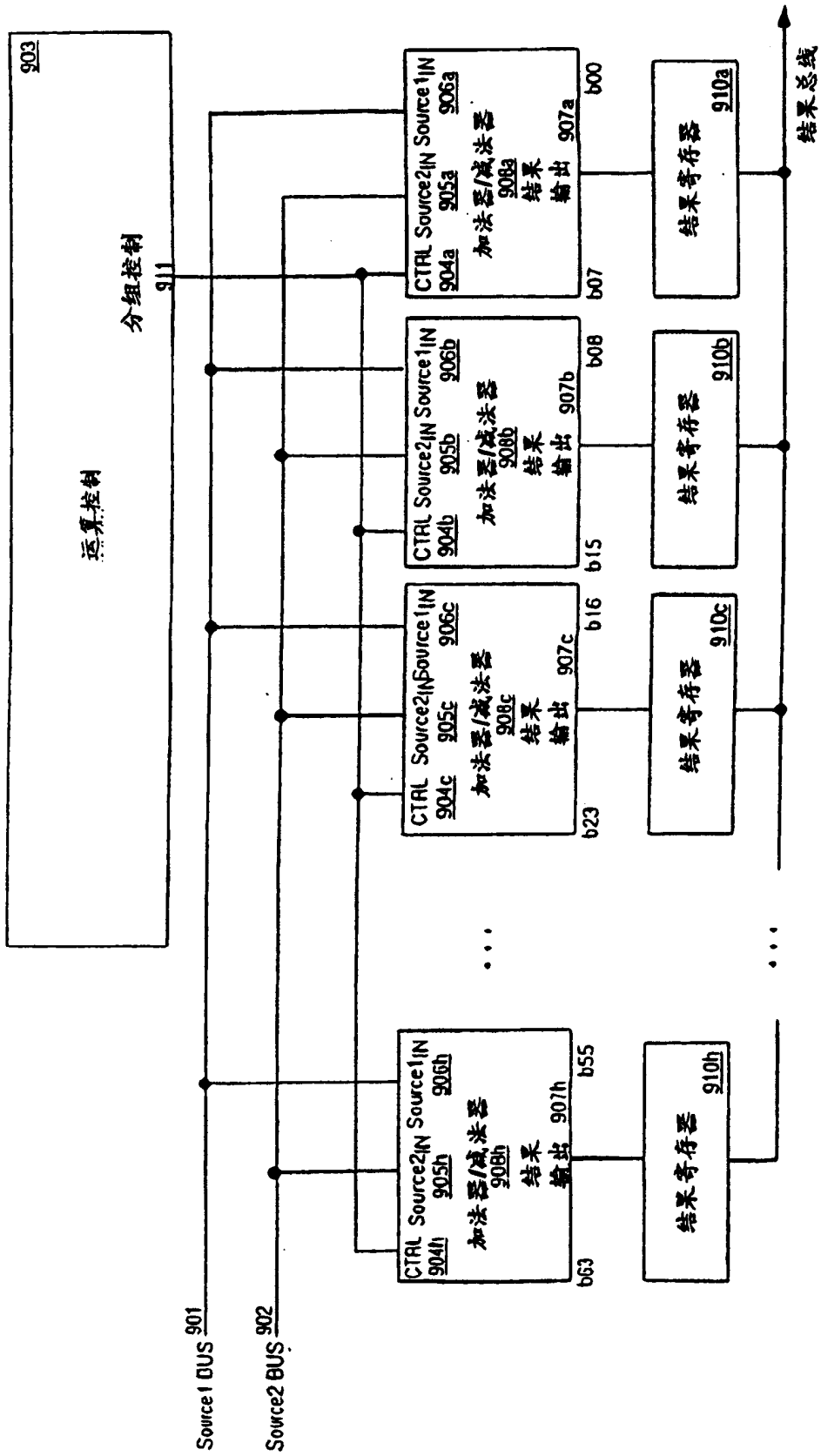


图 9

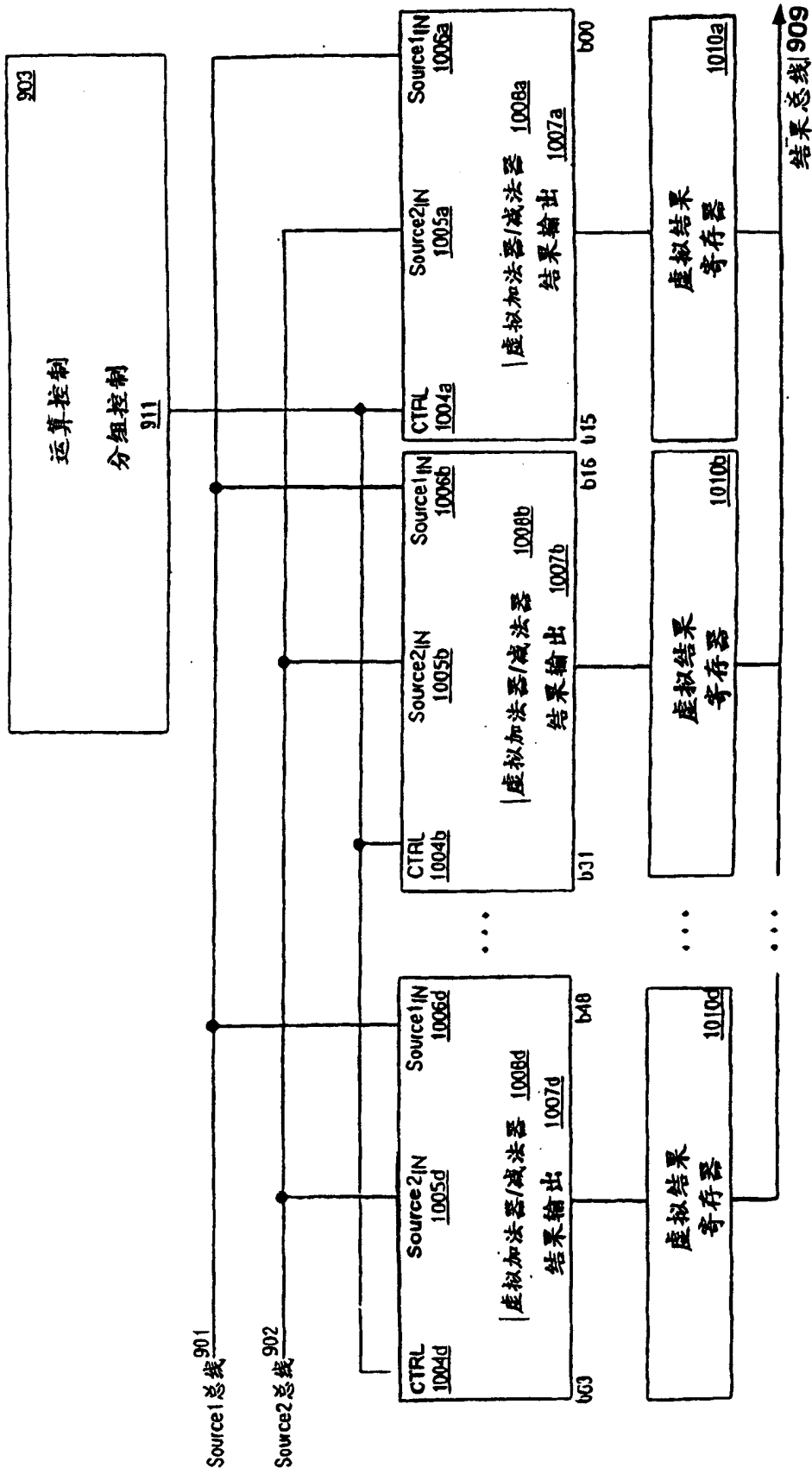


图 10

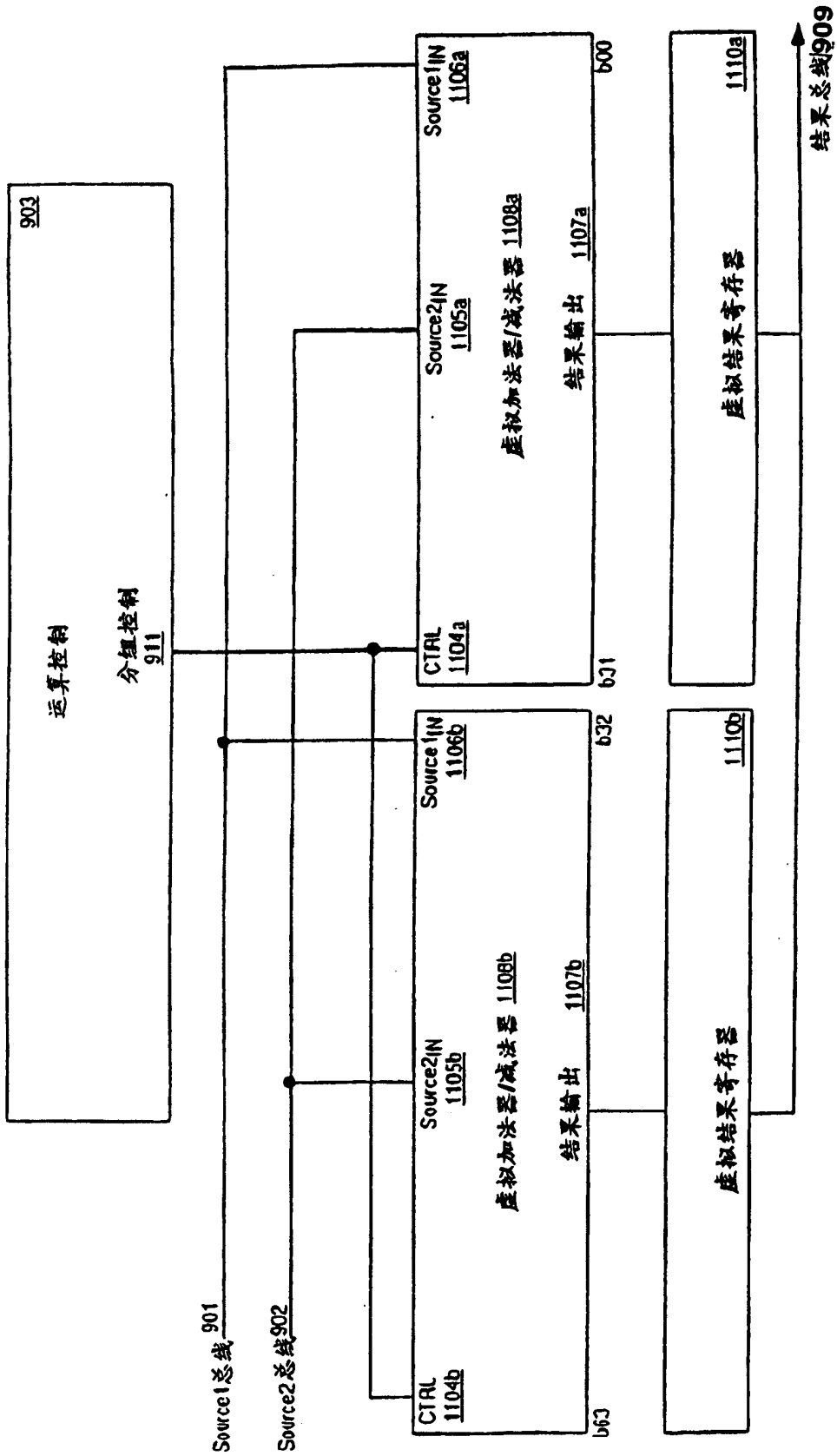


图 11

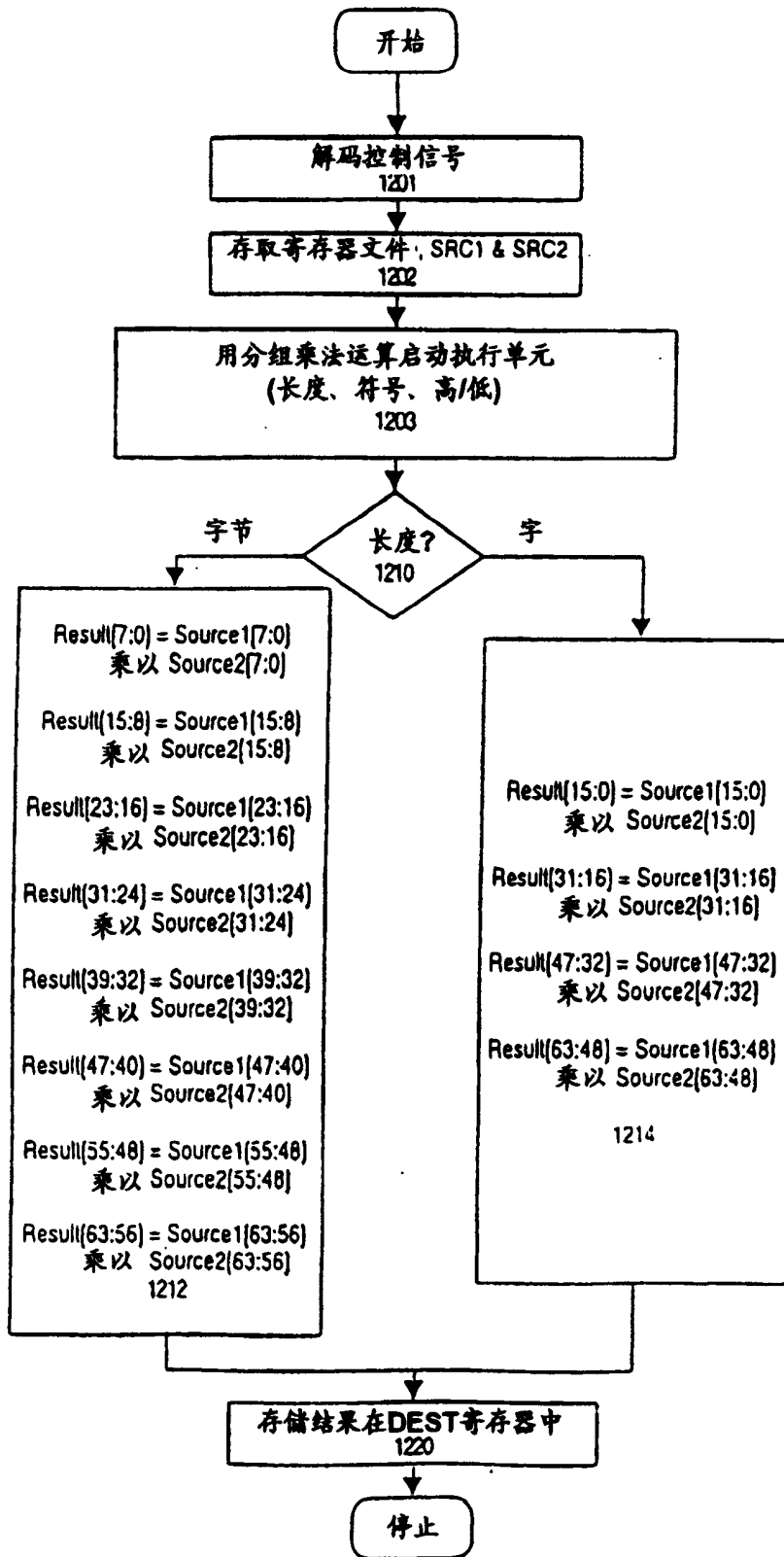


图 12

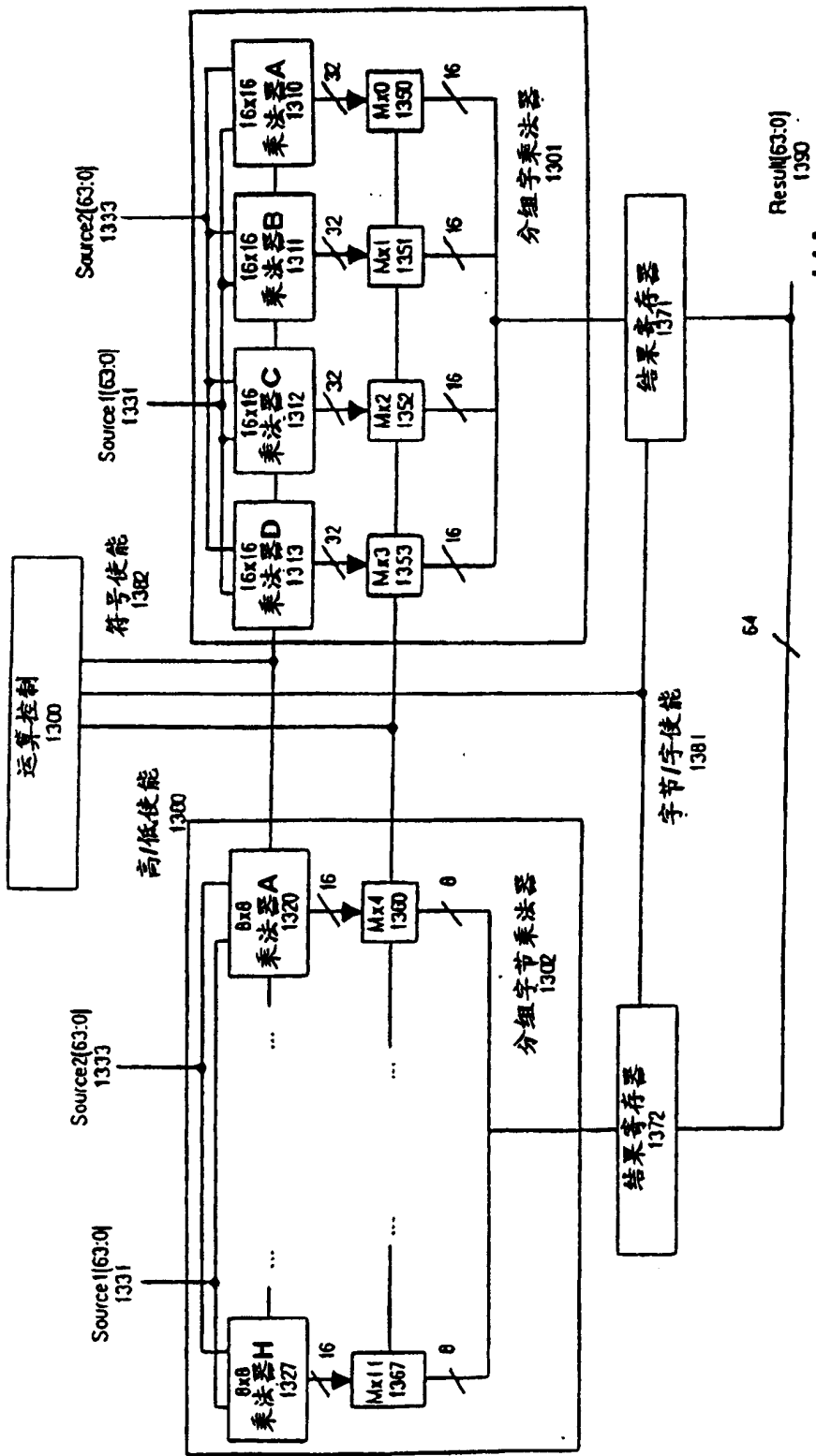


图 13



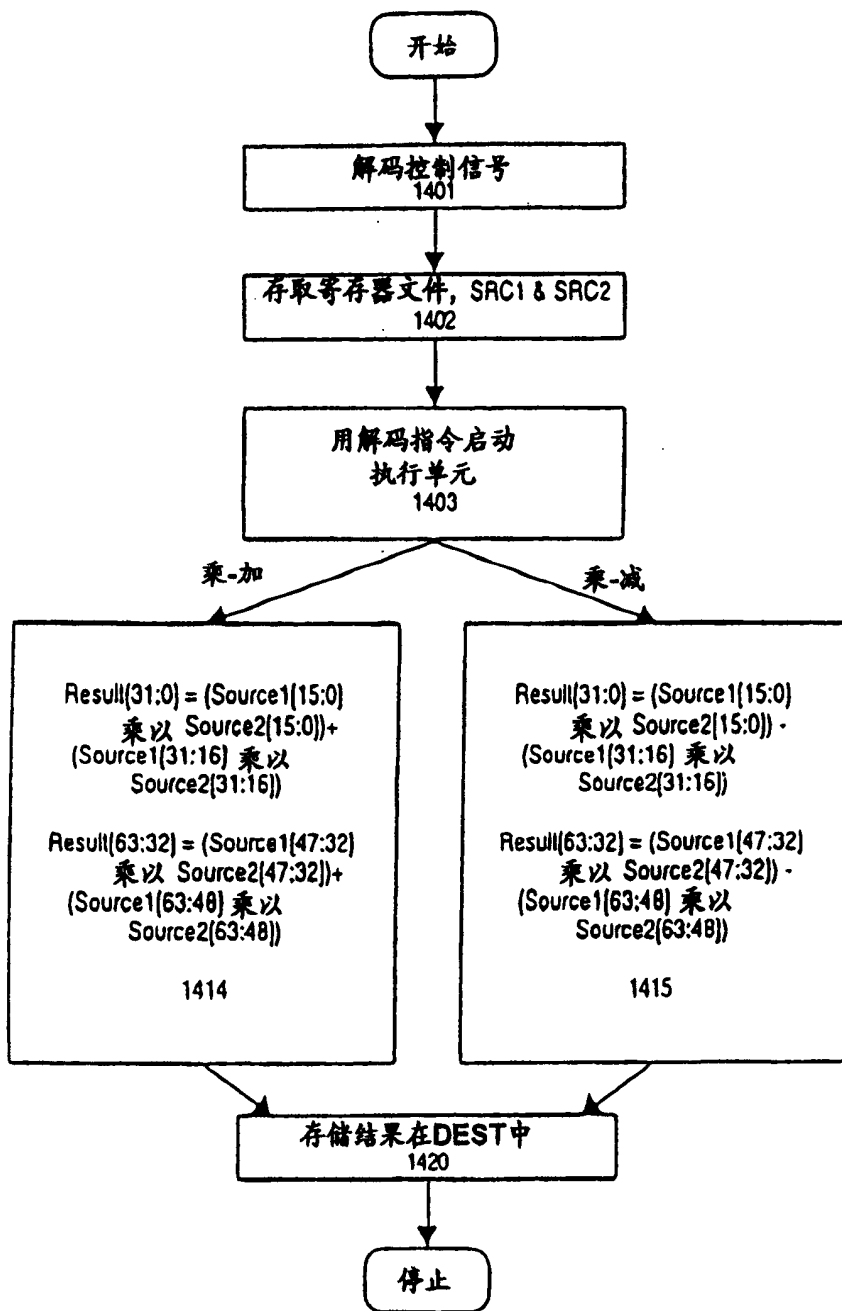


图 14

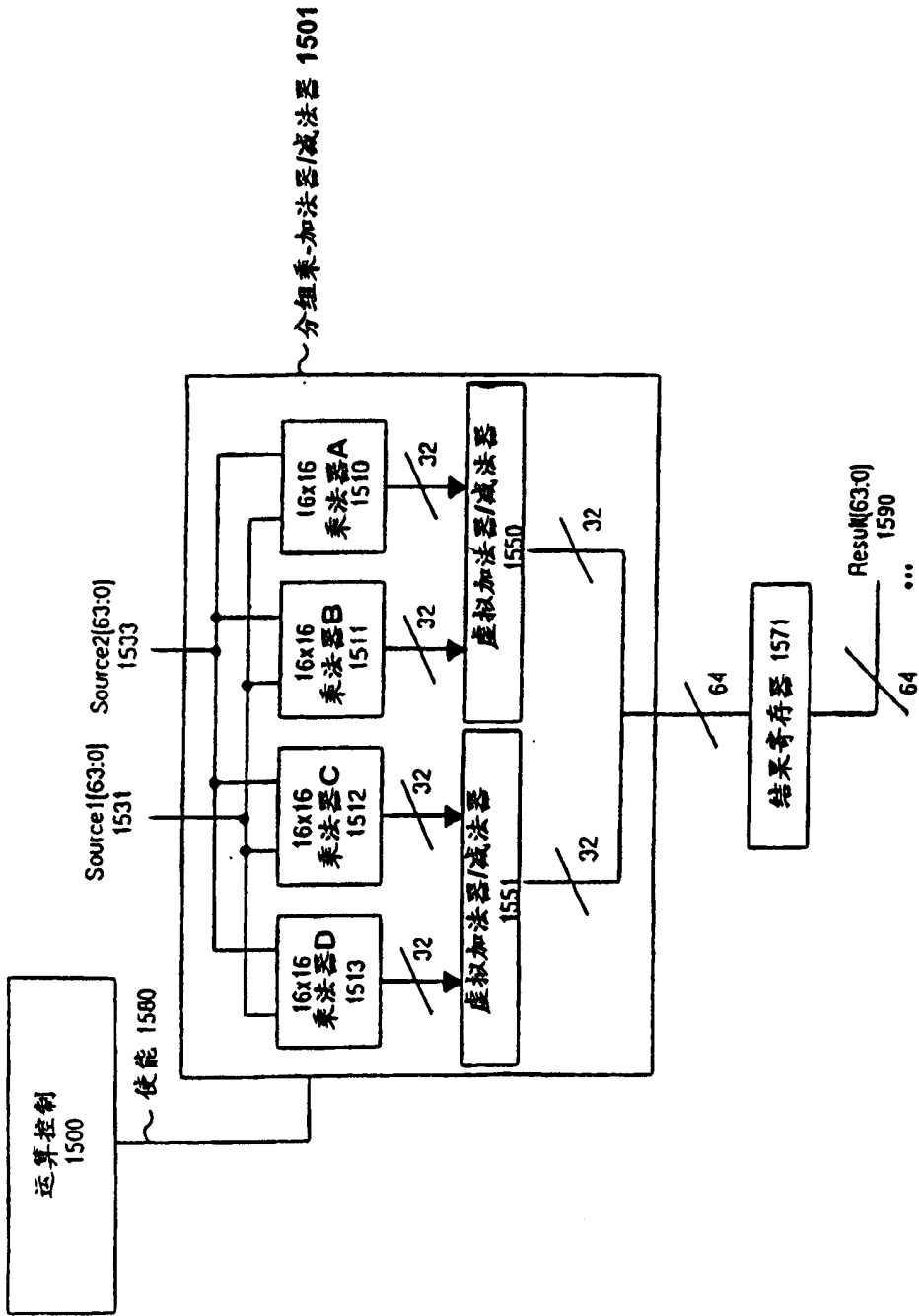


图 15

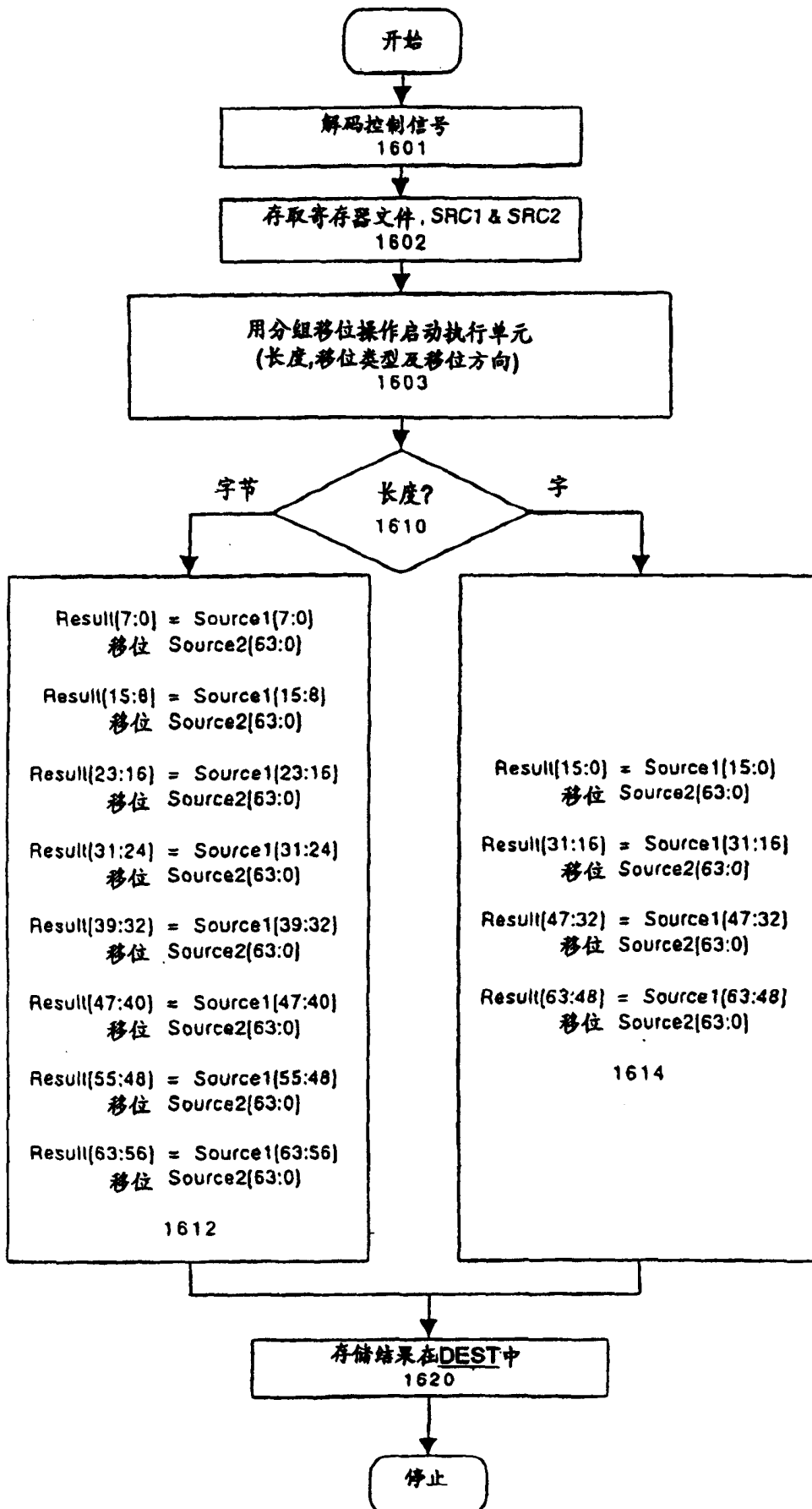


图 16



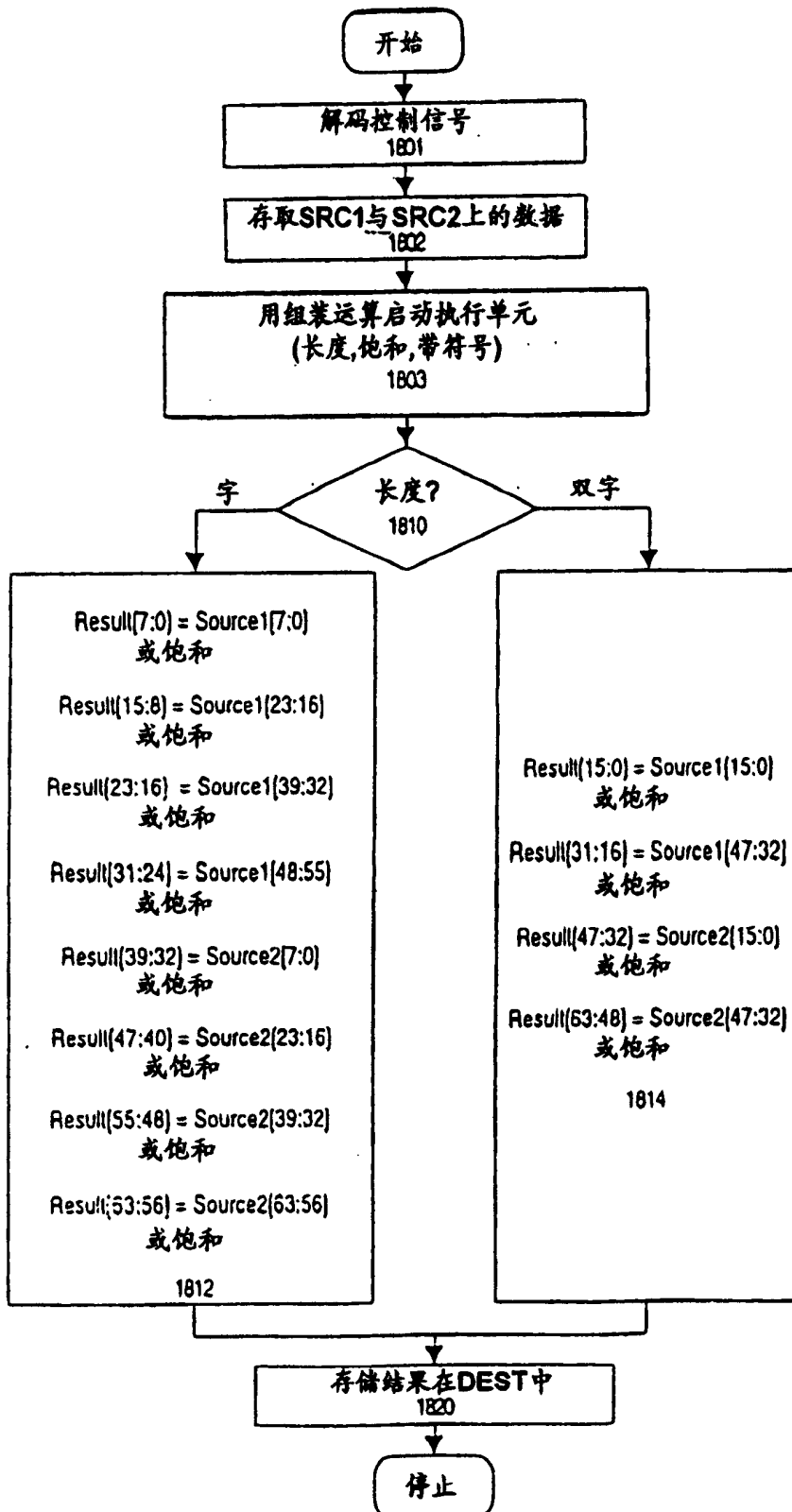


图 18

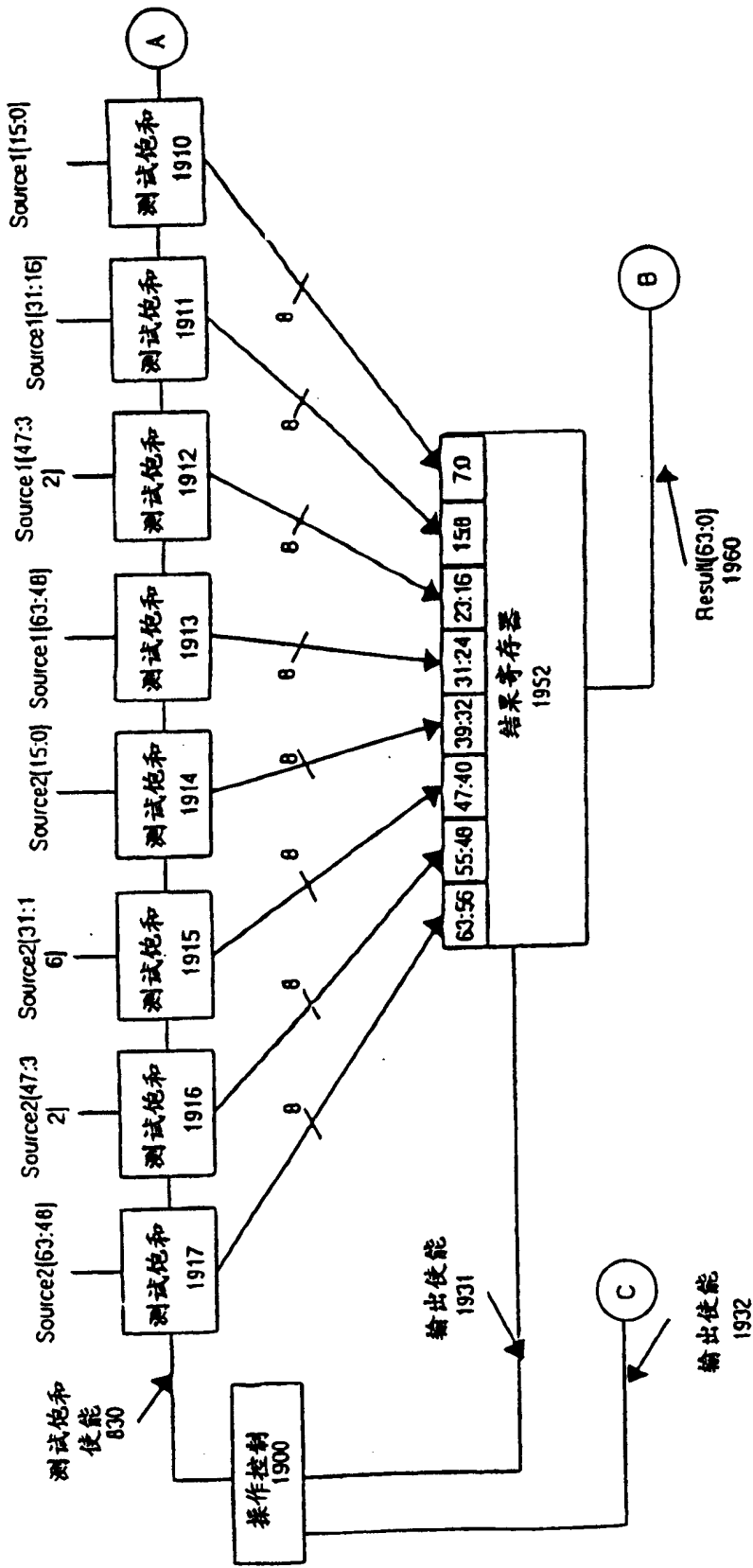


图19 a

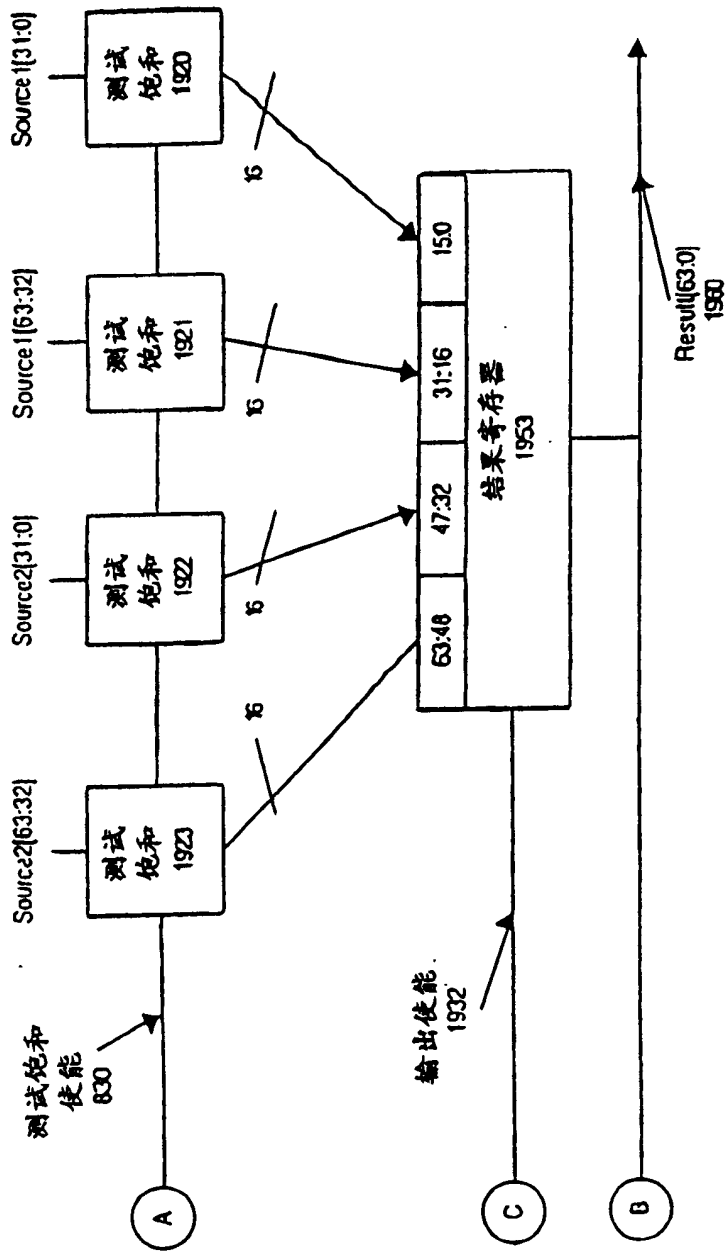


图 19 b

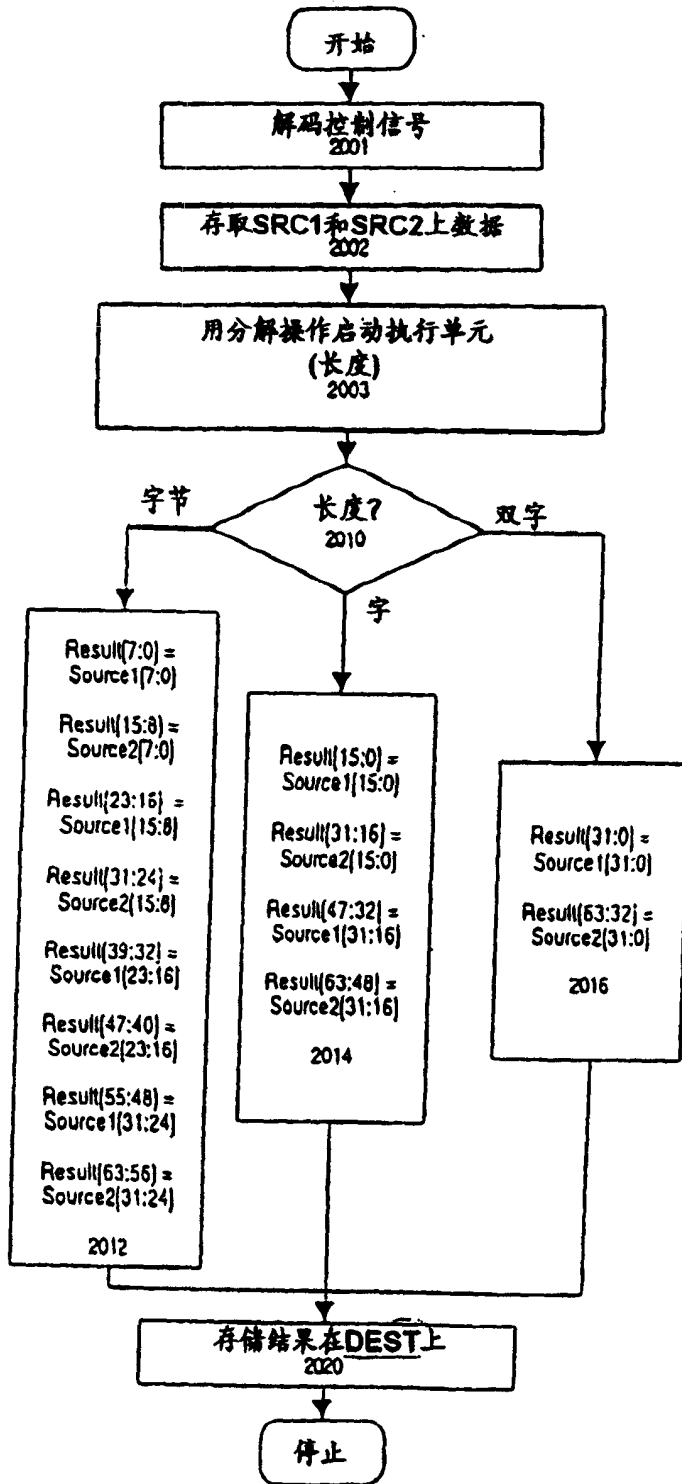


图 20



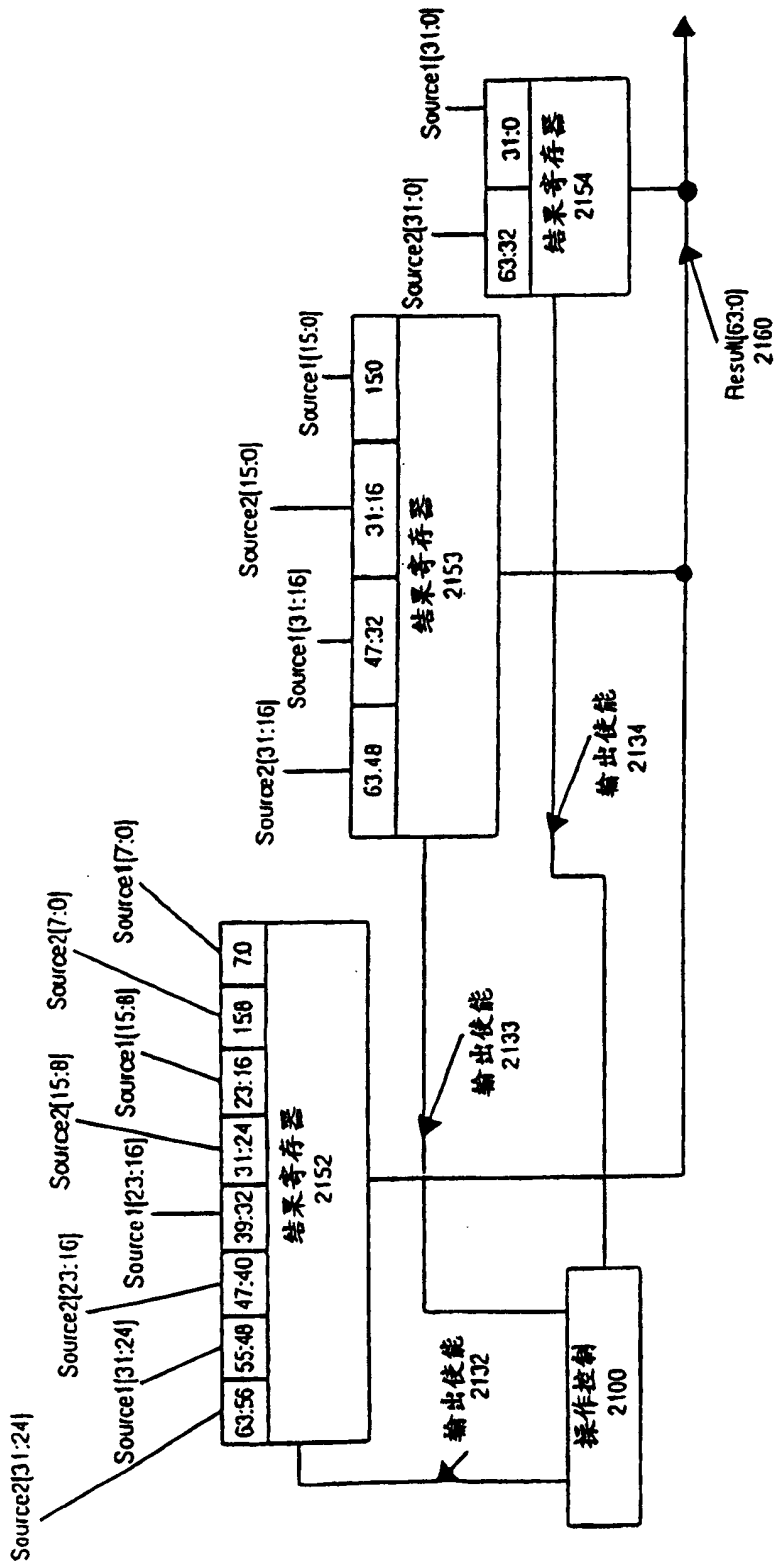


图 21

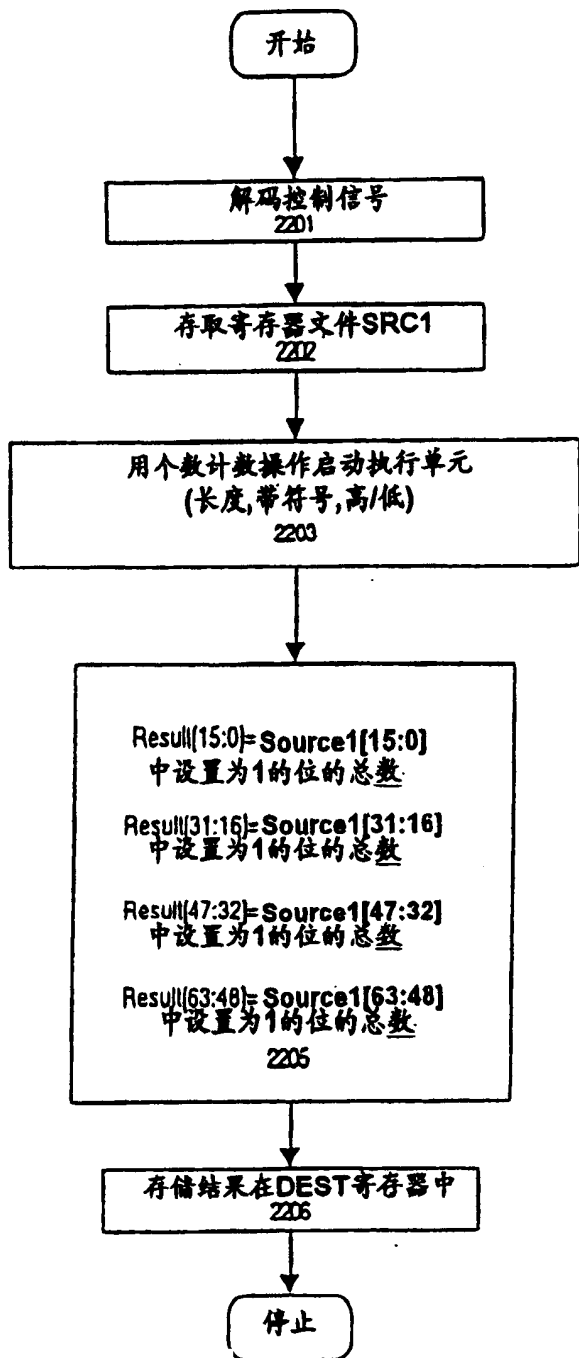


图 22

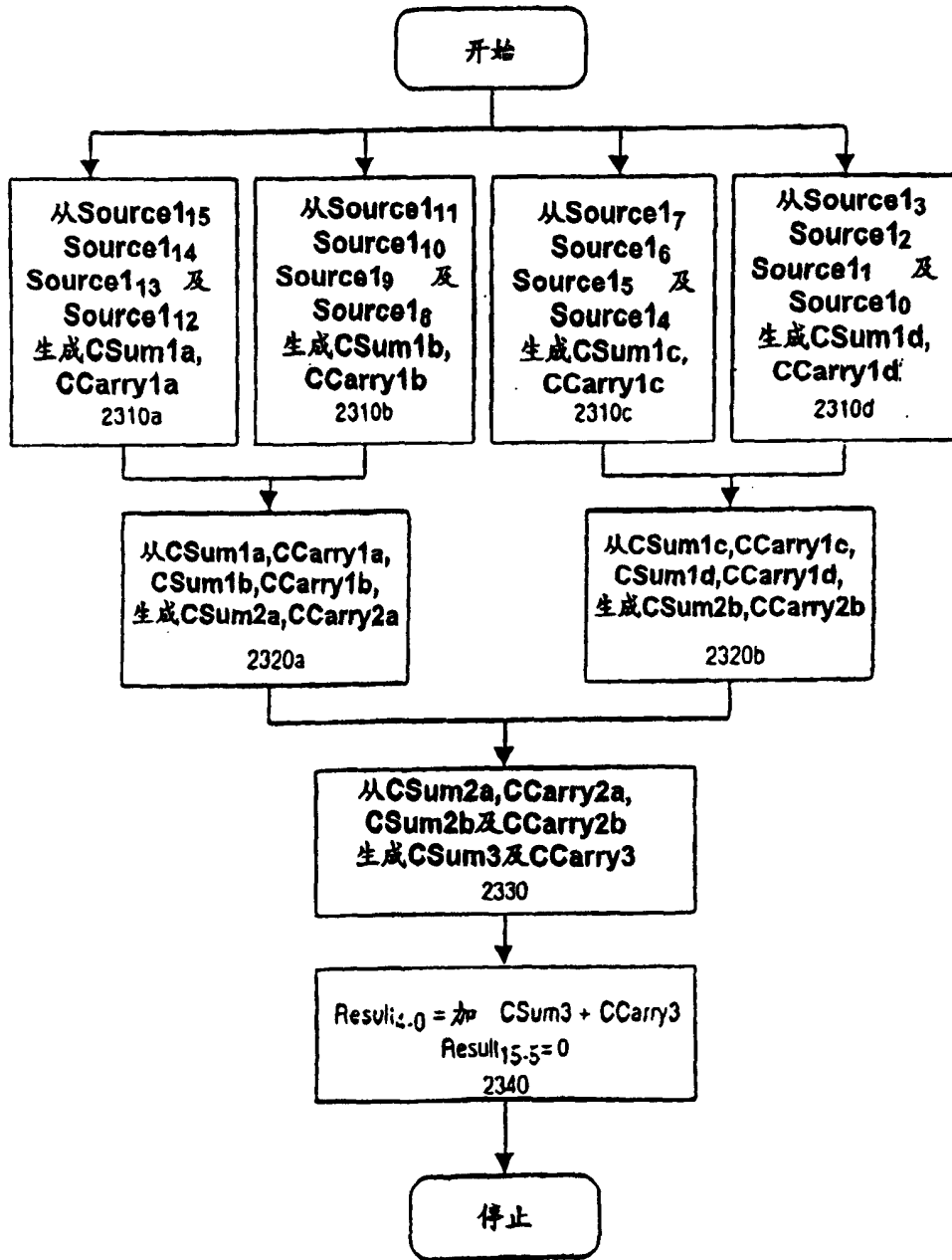


图 23

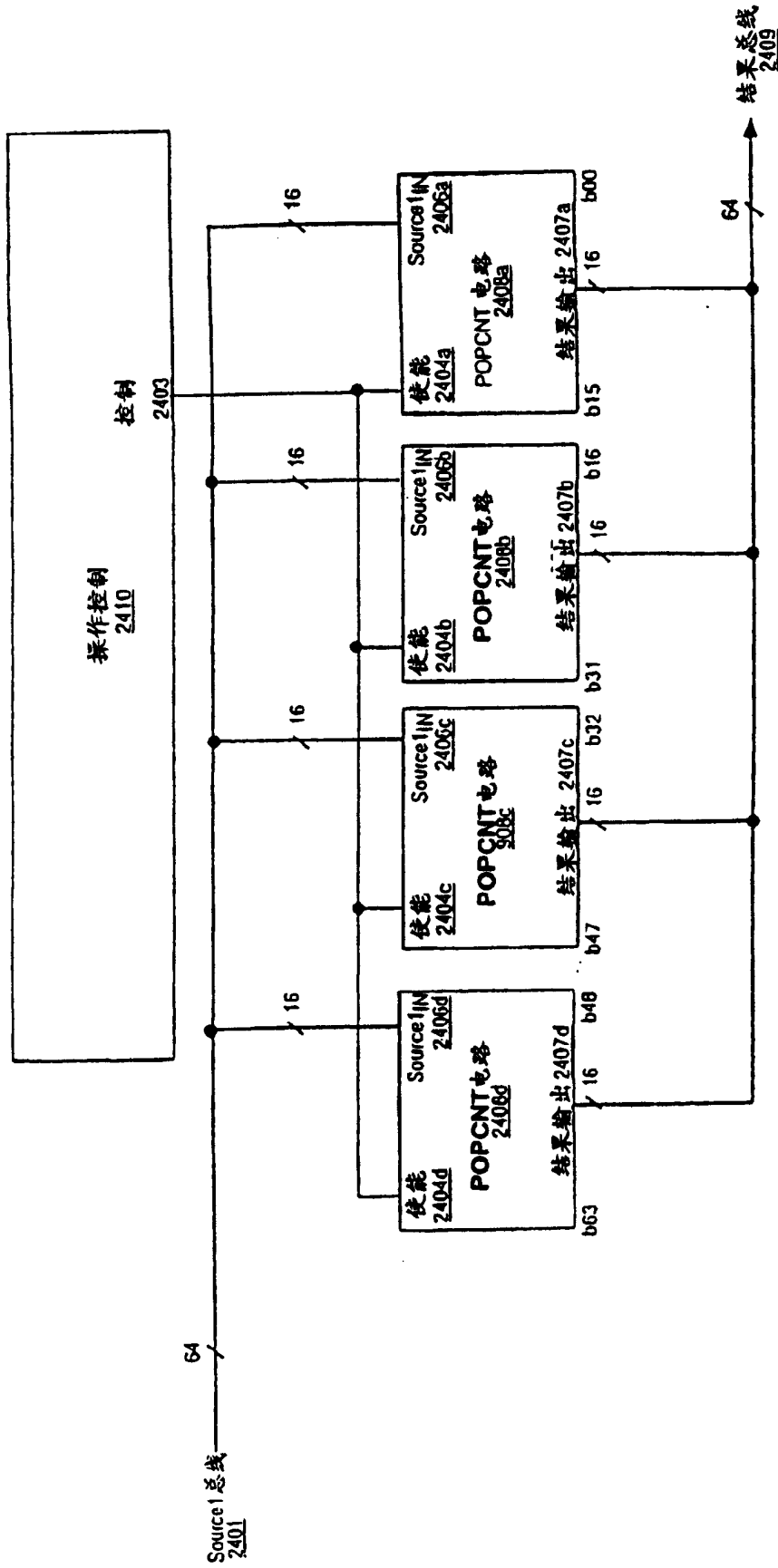


图 24

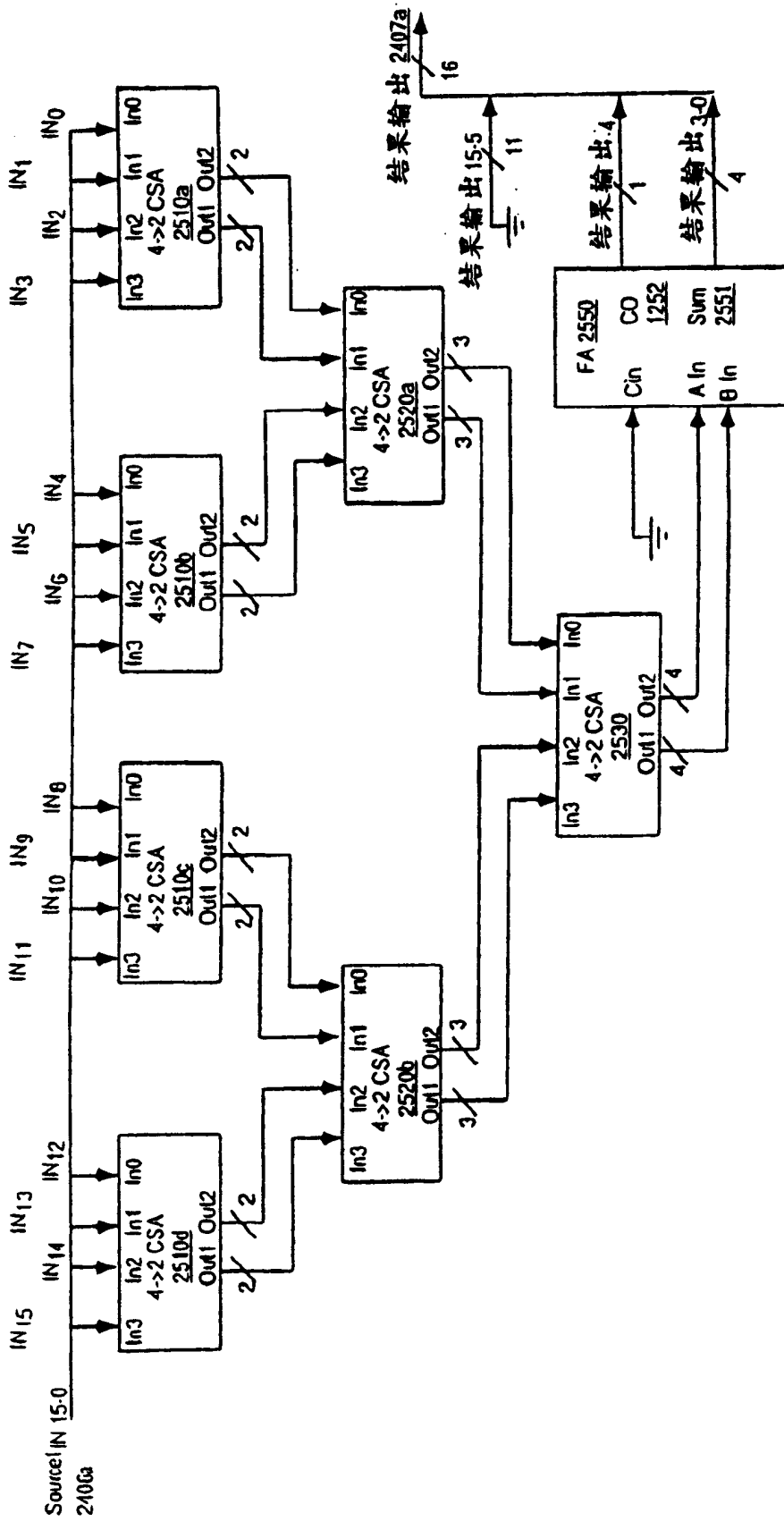


图 25

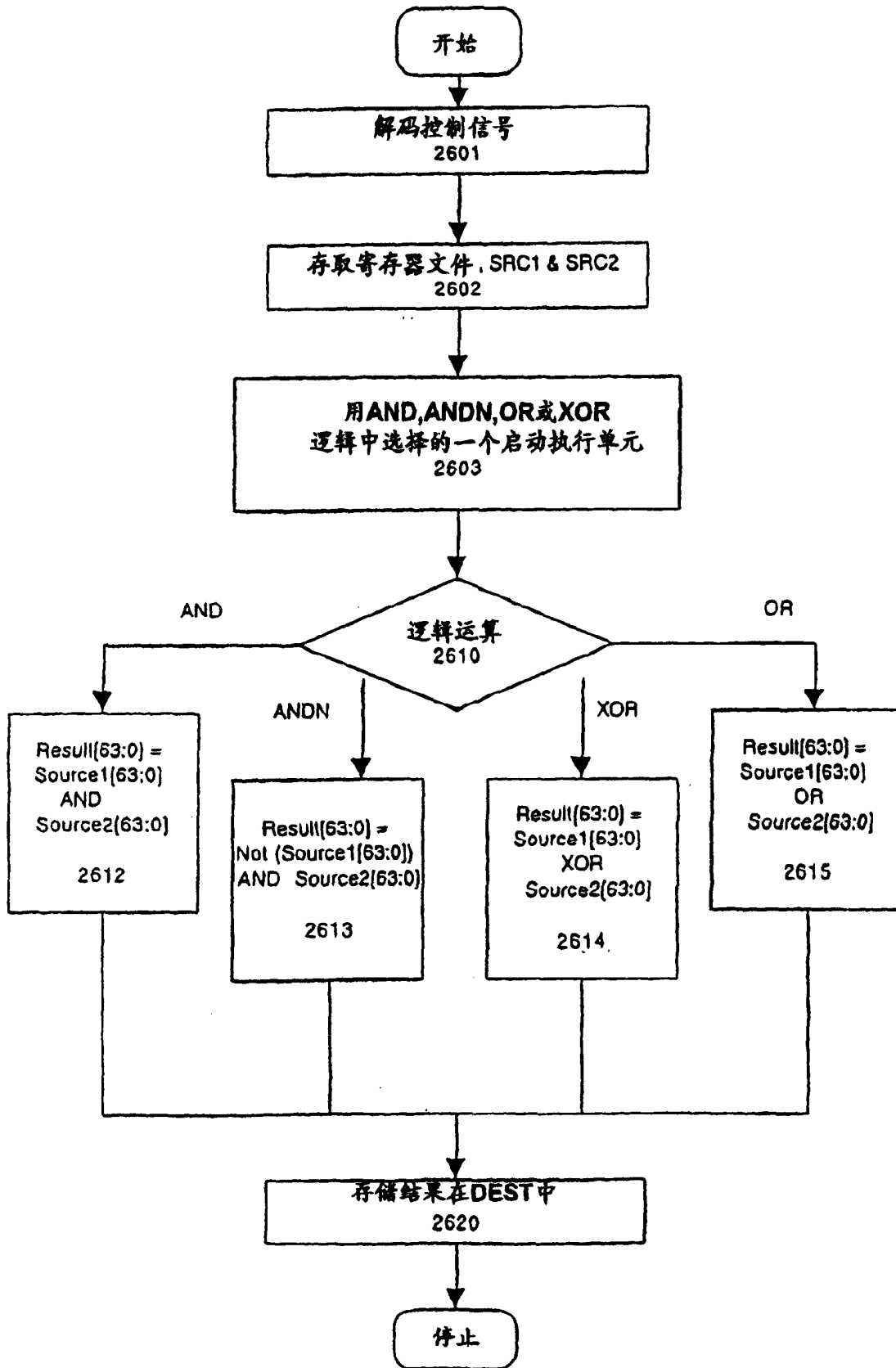


图 26

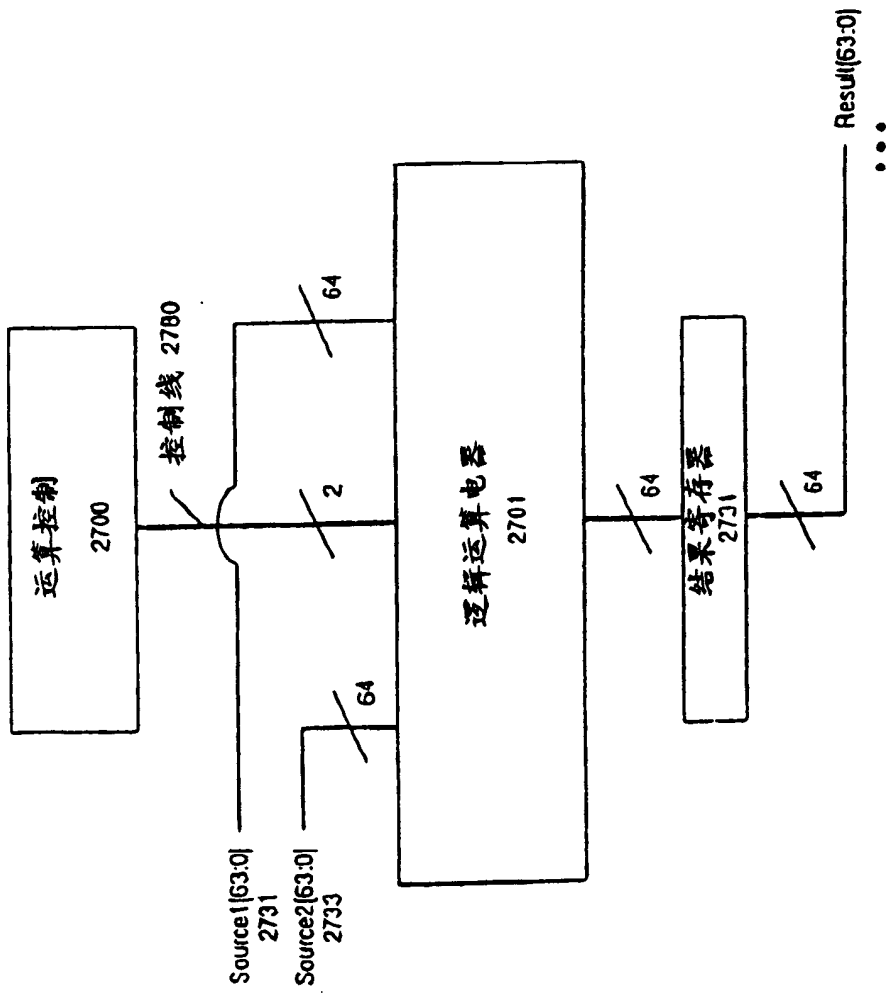


图 27

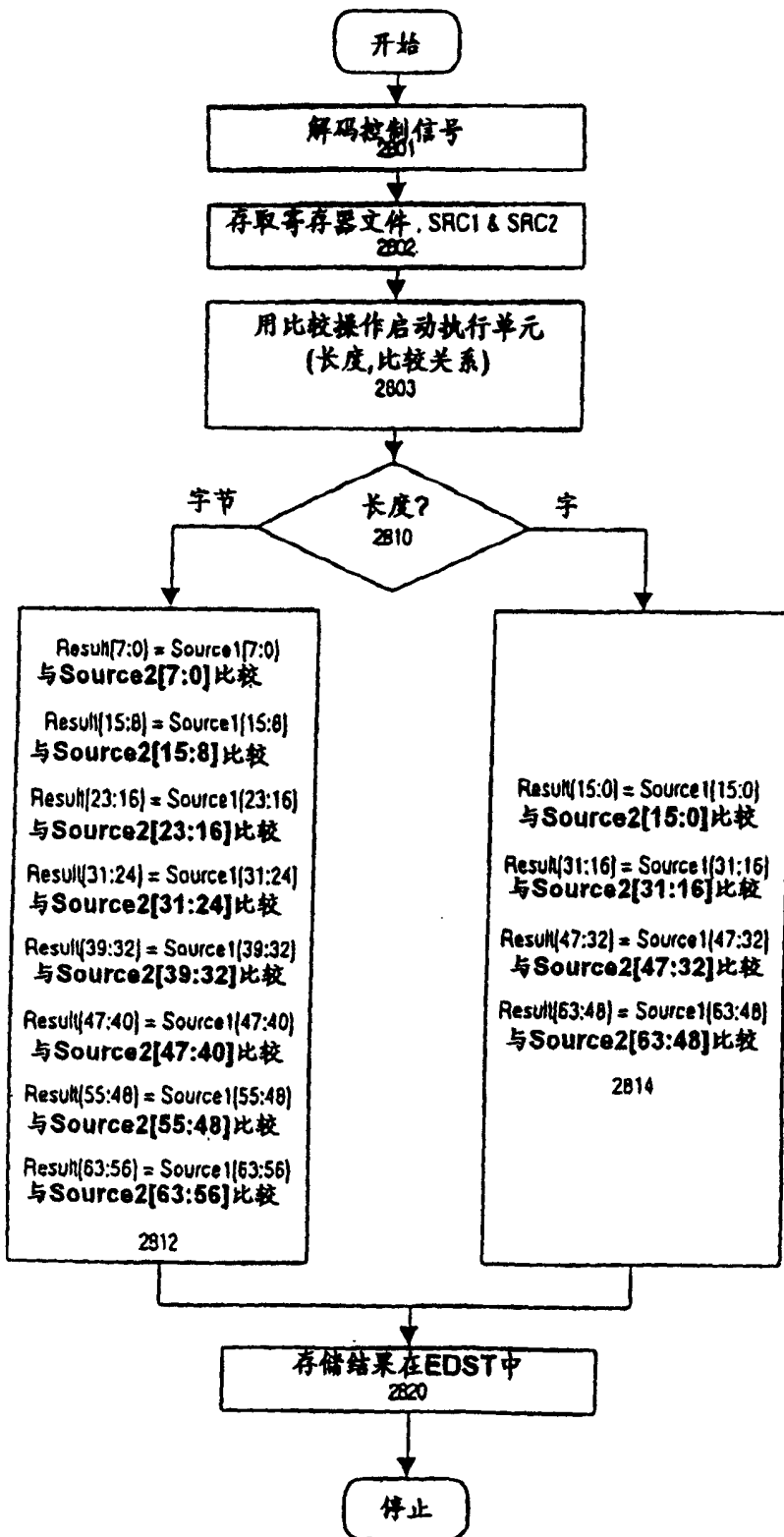


图 28



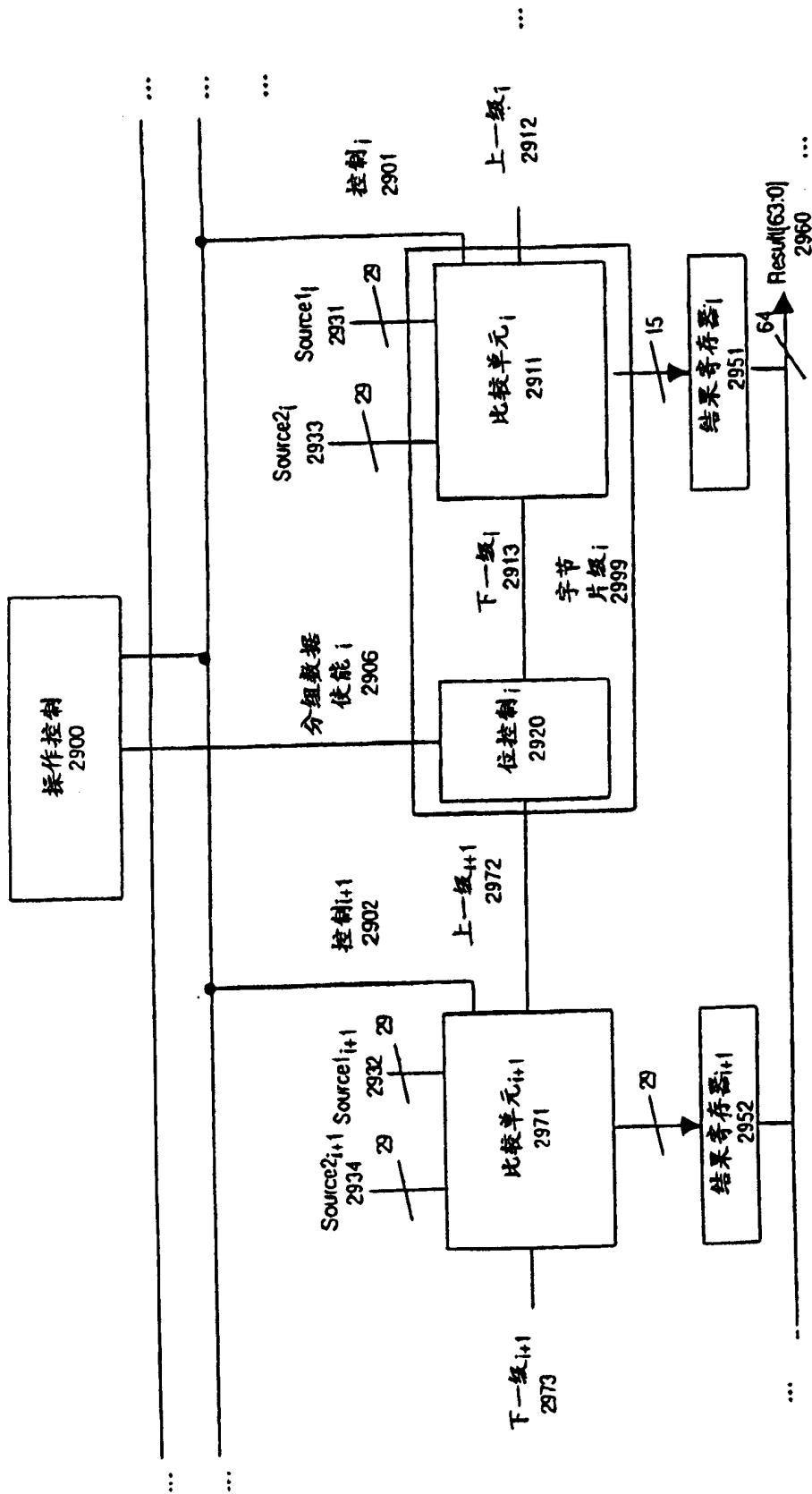


图 29