(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 101930352 B
(45)授权公告日 2016.05.18

(21)申请号 201010194848.4

(22)申请日 1996.07.17

(30)优先权数据
08/521360 1995.08.31 US

(62)分案原申请数据
96197839.2 1996.07.17

(73)专利权人 英特尔公司
地址 美国加利福尼亚州

(72)发明人 A·D·佩勒格 Y·雅里 M·米塔尔 L·M·门内梅尔 B·艾坦 A·F·格卢 C·杜龙 E·科瓦施 W·维特

(74)专利代理机构 中国专利代理(香港)有限公司 72001
代理人 王洪斌

(51)Int.Cl.
G06F 7/57(2006.01)
G06F 7/60(2006.01)
G06F 7/544(2006.01)
G06F 9/302(2006.01)
G06F 9/315(2006.01)
G06F 15/78(2006.01)

(56)对比文件
JP 昭62-84335 A,1987.04.17,
CN 1040105 A,1990.02.28,
US 4985848 A,1991.01.15,

审查员 邹予婷

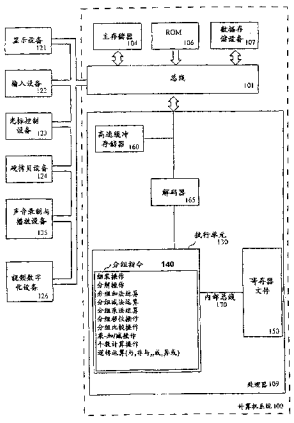
权利要求书6页 说明书55页 附图29页

(54)发明名称

控制移位分组数据的位校正的装置

(57)摘要

本发明涉及控制移位分组数据的位校正的装置，提供一种在处理器中加入支持典型的多媒体应用所要求的分组数据上的操作的指令集的装置。在一个实施例中，本发明包括具有存储区(150)、解码器(165)及多个电路(130)的处理器。该多个电路提供若干指令的执行来操作分组数据。在这一实施例中，这些指令包含组装、分解、分组乘法、分组加法、分组减法、分组比较及分组移位。



CN 101930352 B

1.一种处理器，包括：

高速缓冲存储器，用于存储控制信号；

与所述高速缓冲存储器耦合的解码器，用于从高速缓冲存储器接收所述控制信号，所述控制信号的操作字段提供关于由处理器执行的运算是分组加法运算的信息；

寄存器文件，包括用于存储第一分组操作数的第一寄存器、用于存储第二分组操作数的第二寄存器和用于存储所述分组加法运算的结果的目的寄存器；

执行单元，用于执行所述分组加法运算以便并行地将第一分组操作数与第二分组操作数的对应数据元素单独地相加，并且将所述分组加法运算的结果存储在所述目的寄存器中。

2.如权利要求1所述的处理器，其中，所述数据元素具有选自包含8位、16位、32位的组中的大小。

3.如权利要求1所述的处理器，其中，当所述结果溢出且运算采用不饱和时，忽略所述结果的进位位。

4.如权利要求1所述的处理器，其中，第一分组操作数和第二分组操作数包括带符号或无符号分组操作数。

5.如权利要求1所述的处理器，其中，所述分组加法运算是带符号饱和加法运算、无符号饱和加法运算、带符号不饱和加法运算以及无符号不饱和加法运算中的一种。

6.一种处理器，包括：

高速缓冲存储器，用于存储控制信号；

与所述高速缓冲存储器耦合的解码器，用于从高速缓冲存储器接收所述控制信号，所述控制信号的操作字段提供关于由处理器执行的运算是分组乘法运算的信息；

寄存器文件，包括用于存储第一分组操作数的第一寄存器、用于存储第二分组操作数的第二寄存器和用于存储分组乘法操作的各个乘积的一部分的目的寄存器；

执行单元，用于执行所述分组乘法运算以便并行地将第一分组操作数的各个数据元素与第二分组操作数的各个对应数据元素独立地相乘，并且将对应数据元素的各个乘积的高阶位或低阶位部分存储在所述目的寄存器中。

7.如权利要求6所述的处理器，其中，所述数据元素具有选自包含8位、16位、32位的组中的大小。

8.如权利要求6所述的处理器，其中，所述分组乘法运算包括包含乘法高带符号运算、乘法高无符号运算、乘法低运算的组中的一种运算。

9.一种处理器，包括：

高速缓冲存储器，用于存储控制信号；

与所述高速缓冲存储器耦合的解码器，用于从高速缓冲存储器接收所述控制信号，所述控制信号的操作字段提供关于由处理器执行的运算是分组乘-加运算的信息；

寄存器文件，包括用于存储第一分组操作数的第一寄存器、用于存储第二分组操作数的第二寄存器和目的寄存器；

执行单元，用于执行所述分组乘-加运算以便并行地将第一分组操作数的各个数据元素与第二分组操作数的对应数据元素相乘以产生一组中间结果，并行地将所述中间结果成对相加以产生结果数据元素并且将所述结果数据元素存储在所述目的寄存器中。

10.如权利要求9所述的处理器，其中，所述数据元素具有16位或8位大小。

11.一种处理器，包括：

高速缓冲存储器，用于存储控制信号；

与所述高速缓冲存储器耦合的解码器，用于从高速缓冲存储器接收所述控制信号，所述控制信号的操作字段提供关于由处理器执行的运算是分组比较运算的信息；

寄存器文件，包括用于存储第一分组操作数的第一寄存器、用于存储第二分组操作数的第二寄存器和用于存储比较结果的目的寄存器；

执行单元，用于执行所述分组比较运算以便并行地将第一分组操作数的各个数据元素与第二分组操作数的对应数据元素按指定的关系进行比较，并且将所述比较结果存储在所述目的寄存器中。

12.如权利要求11所述的处理器，其中，所述执行单元进一步用于响应确定被比较的一对数据元素相等，而将所述目的操作数中的对应数据元素设置成全1。

13.如权利要求11所述的处理器，其中，所述执行单元进一步用于响应确定第一操作数的数据元素大于第二操作数的对应数据元素，而将所述目的操作数中的对应数据元素设置成全1。

14.如权利要求11所述的处理器，其中，所述数据元素具有选自包含32位、16位、8位的组中的大小。

15.如权利要求11所述的处理器，其中，第一、第二和目的寄存器具有用于分组数据的64位。

16.一种处理器，包括：

高速缓冲存储器，用于存储控制信号；

与所述高速缓冲存储器耦合的解码器，用于从高速缓冲存储器接收所述控制信号，所述控制信号的操作字段提供关于由处理器执行的运算是分组移位运算的信息；

寄存器文件，包括用于存储第一分组操作数的第一寄存器、用于存储第二分组操作数的第二寄存器和用于存储移位结果的目的寄存器；

执行单元，用于执行所述分组移位运算以便并行地将第一分组操作数的各个数据元素移位第二分组操作数中的移位计数，并且将所述移位结果存储在所述目的寄存器中。

17.如权利要求16所述的处理器，其中，所述执行单元进一步用于执行第一操作数的各个数据元素的逻辑向左移位所述移位计数，并且用0来填充各个数据元素的最低有效位。

18.如权利要求16所述的处理器，其中，所述执行单元进一步用于执行第一操作数的各个数据元素的逻辑向右移位所述移位计数，并且用0来填充各个数据元素的高阶位。

19.如权利要求16所述的处理器，其中，所述执行单元进一步用于执行第一操作数的各个数据元素的向下移位所述移位计数，并且用所述数据元素的符号位的初始值来填充各个数据元素的高阶位。

20.如权利要求16所述的处理器，其中，所述执行单元进一步用于分别对于8、16或32位的分组数据元素，在所述移位计数大于7、15或31时用所述数据元素的符号位的初始值来填充各个目的数据元素。

21.一种处理器，包括：

高速缓冲存储器，用于存储控制信号；

与所述高速缓冲存储器耦合的解码器，用于从高速缓冲存储器接收所述控制信号，所述控制信号的操作字段提供关于由处理器执行的运算是组装运算的信息；

寄存器文件，包括用于存储第一分组操作数的第一寄存器、用于存储第二分组操作数的第二寄存器和用于存储目的操作数的目的寄存器；

执行单元，用于执行所述组装运算以便将来自第一分组操作数的数据元素和来自第二分组操作数的数据元素的一部分位组装成目的操作数，并且将所述目的操作数存储在所述目的寄存器中。

22.如权利要求21所述的处理器，其中，所述执行单元进一步用于将第一分组操作数的各个数据元素的低阶位与第二分组操作数的各个数据元素的低阶位组装成所述目的操作数。

23.一种处理器，包括：

高速缓冲存储器，用于存储控制信号；

与所述高速缓冲存储器耦合的解码器，用于从高速缓冲存储器接收所述控制信号，所述控制信号的操作字段提供关于由处理器执行的运算是分解运算的信息；

寄存器文件，包括用于存储第一分组操作数的第一寄存器、用于存储第二分组操作数的第二寄存器和用于存储结果的目的寄存器；

执行单元，用于执行所述分解运算以便分解或交织第一和第二分组操作数的高阶数据元素或者低阶数据元素，并且将所述结果存储在所述目的寄存器中。

24.一种处理器，包括：

高速缓冲存储器，用于存储控制信号；

与所述高速缓冲存储器耦合的解码器，用于从高速缓冲存储器接收所述控制信号，所述控制信号的操作字段提供关于由处理器执行的运算是分组逻辑运算的信息；

寄存器文件，包括用于存储第一分组操作数的第一寄存器、用于存储第二分组操作数的第二寄存器和用于存储结果的目的寄存器；

执行单元，用于对第一操作数和第二操作数执行逻辑运算，并且将所述结果存储在所述目的寄存器中。

25.如权利要求24所述的处理器，其中，所述分组逻辑运算包括逻辑“与非”运算，以便对第一操作数执行逻辑“非”，之后与第二操作数执行逻辑“与”，以提供所述结果。

26.如权利要求24所述的处理器，其中，所述分组逻辑运算包括逻辑“或”运算、逻辑“与”运算或逻辑“异或”运算。

27.一种计算机系统，包括：

包含第一寄存器的处理器；以及

耦合在所述处理器上的存储区，其中存储有：

在第一分组数据及第二分组数据上操作的组装指令，所述第一分组数据至少包含第一数据元素及第二数据元素，所述第二分组数据至少包含第三数据元素及第四数据元素，所述第一数据元素，所述第二数据元素、所述第三数据元素及所述第四数据元素均包含一组位，所述处理器响应接收所述组装指令，组装所述第一数据元素、所述第二数据元素、所述第三数据元素及所述第四数据元素的每一个的一部分以构成第三分组数据；

在第四分组数据及第五分组数据上操作的分解指令，所述第四分组数据至少包含第五

数据元素及第六数据元素，所述第五分组数据至少包含对应于所述第五数据元素的第七数据元素及对应于所述第六数据元素的第八数据元素，所述第五数据元素、所述第六数据元素、所述第七数据元素及所述第八数据元素的每一个均包含一组位，所述处理器响应接收所述分解指令，生成至少包含来自所述第四分组数据的所述第五数据元素及来自所述第五分组数据的所述第七数据元素的第六分组数据；

分组加法指令，所述处理器响应接收所述分组加法指令，并行将所述第四分组数据及所述第五分组数据的对应数据元素独立相加；

分组减法指令，所述处理器响应接收所述分组减法指令，并行将所述第四分组数据及所述第五分组数据的对应数据元素独立相减；

分组移位指令，所述处理器响应接收所述分组移位指令，并行单独移位至少所述第一数据元素一个指定的计数及所述第二数据元素一个指定的计数；以及

分组比较指令，所述处理器响应接收所述分组比较指令，按照指定的关系并行单独比较来自所述第四分组数据及所述第五分组数据的对应数据元素，并作为结果将一个分组掩码存储在所述第一寄存器中，所述分组掩码至少包含均含有预定位数的第一掩码元素及第二掩码元素，所述第一掩码元素中的每一位指示所述第四分组数据中的所述第五数据元素与所述第五分组数据中的所述第七数据元素比较的所述结果，所述第二掩码元素中的每一位指示所述第四分组数据中的所述第六数据元素与所述第五分组数据中的所述第八数据元素比较的所述结果。

28.权利要求27的计算机系统，所述存储区中还存储有分组乘法指令，所述处理器响应接收所述分组乘法指令，并行将所述第四分组数据及所述第五分组数据的对应数据元素单独相乘。

29.权利要求27的计算机系统，所述存储区中还存储有分组乘－加指令，所述处理器响应接收所述分组乘－加指令，将所述第一数据元素与所述第三数据元素相乘以生成第一中间结果，并将所述第二数据元素与所述第四数据元素相乘以生成第二中间结果，以及将所述第一中间结果与所述第二中间结果相加以生成最终结果中的第九数据元素。

30.权利要求29的计算机系统，其中所述第一数据元素、所述第二数据元素、所述第三数据元素及所述第四数据元素均包含预定数目的位，及其中所述第九数据元素包含两倍于所述预定数目的位。

31.权利要求27的计算机系统，所述存储区中还存储有个数计数指令，所述处理器响应接收所述个数计数指令，并行确定所述第一数据元素中有多少位设置成预定值及所述第二数据元素中有多少位设置成所述预定值。

32.权利要求27的计算机系统，所述存储区中还存储有，

第一分组逻辑指令，所述处理器响应接收所述第一分组逻辑指令，并行对来自所述第四分组数据及所述第五分组数据的对应数据元素进行逻辑“与”运算；

第二分组逻辑指令，所述处理器响应接收所述第二分组逻辑指令，并行对来自所述第四分组数据的数据元素及来自所述第五分组数据的对应数据元素的逻辑“非”进行逻辑“与”运算；

第三分组逻辑指令，所述处理器响应接收所述第三分组逻辑指令，并行对来自所述第四分组数据及所述第五分组数据的对应数据元素进行逻辑“或”运算；以及

第四分组逻辑指令，所述处理器响应接收所述第四分组逻辑指令，并行对来自所述第四分组数据及所述第五分组数据的对应数据元素进行逻辑“异或”运算。

33.权利要求27的计算机系统，其中所述移位是算术移位。

34.权利要求27的计算机系统，其中所述移位是逻辑移位。

35.权利要求27的计算机系统，其中所述移位是向右方向上的。

36.权利要求27的计算机系统，其中所述移位是向左方向上的。

37.权利要求27的计算机系统，其中所述第一数据元素，所述第二数据元素、所述第三数据元素及所述第四数据元素均包含预定数目的位。

38.权利要求37的计算机系统，其中所述部分包含所述预定数目的位的一半。

39.权利要求27的计算机系统，其中所述第五数据元素、所述第六数据元素、所述第七数据元素及所述第八数据元素均包含所述预定数目的位。

40.一种操作第一分组数据及第二分组数据的方法，所述第一分组数据至少包含第一数据元素及第二数据元素，所述第二分组数据至少包含对应于所述第一数据元素的第三数据元素及对应于所述第二数据元素的第四数据元素，所述第一数据元素、所述第二数据元素、所述第三数据元素及所述第四数据元素中的每一个均包含一组位，所述方法包括计算机实现的下述步骤：

接收一条指令；

确定所述指令是否是组装指令、分解指令、分组加法指令、分组减法指令、分组移位指令及分组比较指令之一；

如果所述指令为所述组装指令，则组装所述第一数据元素、所述第二数据元素、所述第三数据元素及所述第四数据元素的每一个的一部分以构成第三分组数据；

如果所述指令为所述分解指令，生成至少包含来自所述第一分组数据的所述第一数据元素及来自所述第二分组数据的所述第三数据元素的第四分组数据；

如果所述指令为所述分组加法指令，并行将所述第一分组数据及所述第二分组数据的对应数据元素单独相加；

如果所述指令为所述分组减法指令，并行将所述第一分组数据及所述第二分组数据的对应数据元素单独相减；

如果所述指令为所述分组移位指令，并行单独移位至少所述第一数据元素及所述第二数据元素指定的计数；以及

如果所述指令为所述分组比较指令，则按照指定的关系并行单独比较来自所述第一分组数据及所述第二分组数据的对应数据元素，并作为结果生成一个分组掩码，所述分组掩码至少包含均含有预定数目的位的第一掩码元素及第二掩码元素，所述第一掩码元素中的每一个位指示比较所述第一分组数据中的所述第一数据元素与所述第二分组数据中的所述第三数据元素的所述结果，所述第二掩码元素中的每一个位指示比较所述第一分组数据中的所述第二数据元素与所述第二分组数据中的所述第四数据元素的所述结果。

41.权利要求40的方法，其中：

所述确定步骤还包括确定所述指令是否是分组乘法指令；以及

所述方法还包括下述步骤：

如果所述指令为所述分组乘法指令，便并行将所述第一分组数据及所述第二分组数据

的对应数据元素单独相乘。

42.权利要求40的方法，其中：

所述确定步骤还包括确定所述指令是否是分组乘－加指令；以及

所述方法还包括下述步骤：

如果所述指令为所述分组乘－加指令，执行下述步骤：

将所述第一数据元素及所述第三数据元素相乘以生成第一中间结果；

将所述第二数据元素与所述第四数据元素相乘以生成第二中间结果；以及

将所述第一中间结果与所述第二中间结果相加以生成最终结果中的第五数据元素。

43.权利要求42的方法，其中所述第一数据元素、所述第二数据元素、所述第三数据元素及所述第四数据元素均包含预定数目的位，及其中所述第四数据元素与所述第五数据元素的每一个均包含两倍于所述预定数目的位。

44.权利要求40的方法，其中：

所述确定步骤还包括确定所述指令是否是个数计数指令；以及

所述方法还包括下述步骤：

如果所述指令是所述个数计数指令，并行确定在所述第一数据元素中多少位设置在预定值上，及在所述第二数据元素中多少位设置在所述预定值上。

45.权利要求40的方法，其中：

所述确定步骤还包括确定所述指令是否是多条分组逻辑指令之一；以及

所述方法还包括下述步骤：

如果所述指令为所述多条分组逻辑指令的第一条，则并行将来自所述第一分组数据及所述第二分组数据的对应数据元素进行逻辑“与”运算，

如果所述指令为所述多条分组逻辑指令的第二条，则并行将来自所述第一分组数据的数据元素与来自所述第二分组数据的对应数据元素的逻辑“非”进行逻辑“与”运算；

如果所述指令为所述多条分组逻辑指令的第三条，则并行将来自所述第一分组数据及所述第二分组数据的对应数据元素进行逻辑“或”运算；以及

如果所述指令为所述多条分组逻辑指令的第四条，则并行将来自所述第一分组数据及所述第二分组数据的对应数据元素进行逻辑“异或”运算。

46.权利要求40的方法，其中所述单独移位的步骤是作为算术移位与逻辑移位之一执行的。

47.权利要求40的方法，其中所述单独移位的步骤是这样执行的，使所述第一数据元素及所述第二数据元素两者都在向右方向及向左方向之一上移位。

48.权利要求42的方法，其中所述第一数据元素、所述第二数据元素、所述第三数据元素及所述第四数据元素均包含预定数目的位。

49.权利要求48的方法，其中所述部分包含所述预定数目的位的一半。

50.权利要求40的方法，其中所述确定步骤是由解码器执行的。

控制移位分组数据的位校正的装置

[0001] 本申请是申请日为1996年7月17日、申请号为03132844.X、发明名称为“能够使用分组数据指令来执行快速变换操作的方法和装置”的专利申请的分案申请。

技术领域

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

背景技术

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

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

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

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

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

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

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

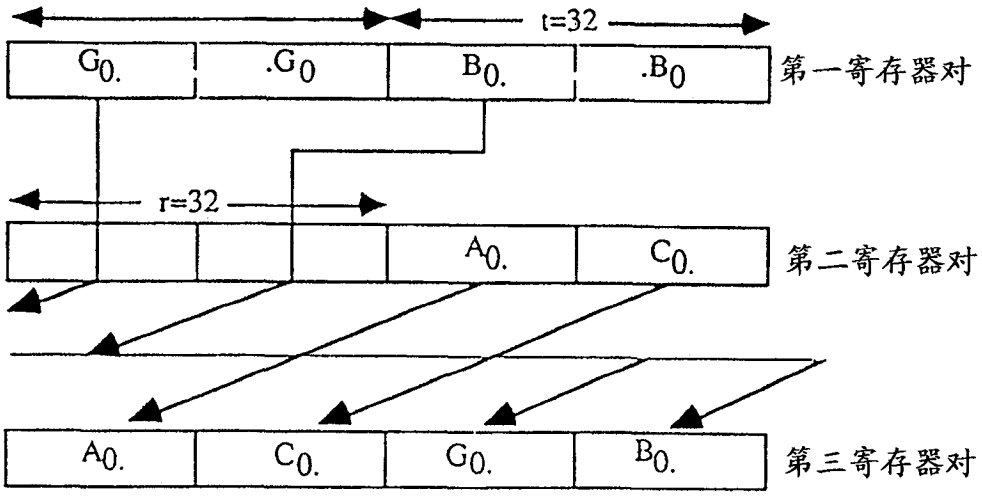
[0009]

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

[0010] X=未定义的操作

[0011] 表1

[0012]



[0013] 表2

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

[0015] ppack Source1,Source2

[0016]

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

[0017] ppack Result1,Source3

[0018]

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

[0019] 表3

[0020] MC88110处理器分解命令通过将来自分组数据的4、8或16位数据元素放入两倍长(8、16或32位)的数据元素的低位一半中,并填充以零,即将得出的数据元素的较高位设定为零进行操作。下面表4中示出了这一分解命令的操作的一个例子。

[0021] 第一寄存器对

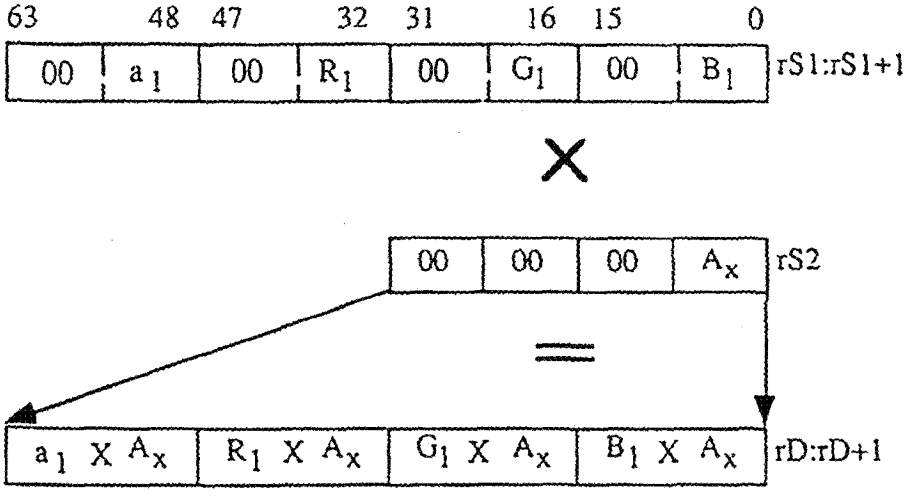
[0022]

00101010	01010101	01010101	11111111	10000000	01110000	10001111	10001000
7	6	5	4	3	2	1	0
解开							
第二寄存器对							
00000000 10000000		00000000 01110000		00000000 10001111		00000000 10001000	
3		2		1		0	

[0023] 表4

[0024] MC88110处理器分组乘法指令将64位分组数据的各元素乘以一个32位值,如同该分组数据表示单一的值一样,如下面表5中所示。

[0025]



[0026] 表5

[0027] 这一乘法指令具有两个缺点。首先这一乘法指令并不断开进位链，从而程序员必须保证在分组数据上执行的运算并不导致溢出。结果，程序员有时必须加入附加的指令来防止这一溢出。第二，这一乘法指令将分组数据中的各元素乘以单一的值(即该32位值)。结果，用户没有选择分组数据中哪些元素乘以该32位值的灵活性。因此，程序员必须制备数据使得分组数据中的每一个元素上都需要相同的乘法或者每当需要对该数据中少于全部元素进行乘法时浪费处理时间来分解数据。因此程序员不能并行利用多个乘数来执行多个乘法。例如，将8个不同的数据片相乘，每一数据片一个字长，需要四次单独的乘法运算。各运算每次乘两个字，实际上浪费了用于位16以上的位的数据线与电路。

[0028] MC88110处理器分组比较指令比较来自第一分组数据与第二分组数据的对应的32位数据元素。两个比较中各个可能返回小于(＜)或大于等于(≥)之一，得出四种可能的组合。该指令返回一个8位结果串；四位表示符合四种可能条件中哪一种，四位表示这些位的补码。根据这一指令的结果的条件转移能以两种方式实现：1)用一序列条件转移；或2)用跳转表。该指令的问题在于它需要根据数据的条件转移来执行函数的事实，诸如：ifY＞A then X＝X+B else X＝X。这一函数的伪码编译表示将是：

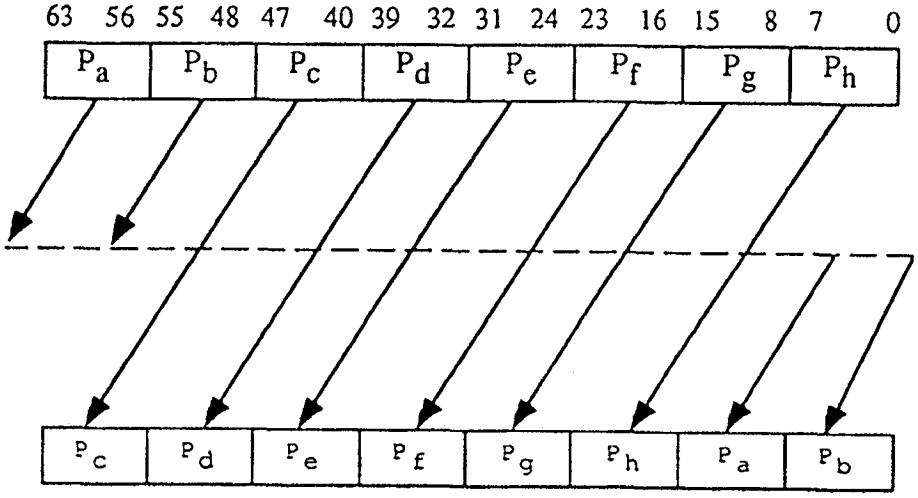
[0029]

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

[0030] 新的微处理器试图通过推测转移到哪里来加快执行。如果预测正确，便不损失性能并且存在着提高性能的潜力。然而如果预测错误，便损失性能。因此，预测得好的鼓励是巨大的。然而根据数据的转移(诸如上面的)呈现为不可预测的方式，这破坏了预测算法并得出更多的错误预测。结果，使用这一比较指令来建立根据数据的条件转移要付出性能上的高昂代价。

[0031] 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)。

[0032]



[0033] 表6

[0034] 由于旋转指令使移出寄存器的高位移入寄存器的低位，MC88110处理器并不支持单个地移位分组数据中的各元素。结果，要求单独移位分组数据类型中各元素的编程算法

需要:1)分解数据,2)单独地在各元素上执行移位,及3)将结果组装成结果分组数据供进一步分组数据处理。

发明内容

[0035] 本发明描述了在处理器中加入支持典型的多媒体应用所要求的操作的分组数据指令集的方法与装置。在一个实施例中,本发明包括一个处理器及一个存储区。存储区中包含若干指令供处理器执行以操作分组数据。在这一实施例中,这些指令包括组装、分解、分组加法、分组减法、分组乘法、分组移位及分组比较。

[0036] 处理器对接收组装指令作出响应,组装来自至少两个分组数据中的数据元素的一部分位以构成第三分组数据。作为比较,处理器对接收该分解指令作出响应,生成包含来自第一分组数据操作数的至少一个数据元素及来自第二分组数据操作数的至少一个对应数据元素的第四分组数据。

[0037] 处理器响应接收该分组加法指令单独地将来自至少两个分组数据的对应数据元素并行加在一起。作为对比,处理器响应接收该分组减法指令单独地将来自至少两个分组数据的对应数据元素并行相减。

[0038] 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。

[0039] 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。

[0040] 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

[0041] 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。

[0042] 图1说明按照本发明的一个实施例的示例性计算机系统。

[0043] 图2说明按照本发明的一个实施例的处理器的寄存器文件。

[0044] 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。

[0045] 图4说明按照本发明的一个实施例的分组数据类型。

[0046] 图5a表示按照本发明的一个实施例的寄存器中分组数据。

[0047] 图5b表示按照本发明的一个实施例的寄存器中分组数据。

[0048] 图5c表示按照本发明的一个实施例的寄存器中分组数据。

[0049] 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。

[0050] 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。

[0051] 分组加法/减法

[0052] 图7a说明按照本发明的一个实施例执行分组加法的方法。

[0053] 图7b说明按照本发明的一个实施例执行分组减法的方法。

[0054] 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。

[0055] 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。

[0056] 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。

[0057] 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。

[0058] 分组乘法

[0059] 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。

[0060] 图13说明按照本发明的一个实施例执行分组乘法的电路。

[0061] 乘-加/减

[0062] 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。

[0063] 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。

[0064] 分组移位

[0065] 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。

[0066] 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。

[0067] 组装

[0068] 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。

[0069] 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。

[0070] 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。

[0071] 分解

[0072] 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。

[0073] 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。

[0074] 个数计数

[0075] 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。

[0076] 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。

[0077] 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。

[0078] 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。

[0079] 分组逻辑运算。

[0080] 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。

[0081] 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。

[0082] 分组比较

[0083] 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。

[0084] 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

[0085] 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。

[0086] 定义

[0087] 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。

[0088] 位X至位Y；

[0089] 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}。

[0090] 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。

[0091] SRC1，SRC2与DEST：

[0092] 标识存储区(诸如存储器地址、寄存器等)

[0093] Source1-i与Result1-i：表示数据

[0094] 计算机系统

[0095] 图1说明按照本发明的一个实施例的示范性计算机系统100。计算机系统100包括用于传递信息的总线101或其它通信硬件与软件，及用于处理信息的与总线101耦合的处理器109。处理器109表示包含CISC(复杂指令集计算)或RISC(精减指令集计算)类型体系结构在内的任何类型体系结构的中央处理单元。计算机系统100还包括耦合在总线101上用于存储信息及要由处理器109执行的指令的随机存取存储器(RAM)或其它动态存储设备(称作主存储器104)。在处理器109执行指令期间，主存储器104也可用来存储临时变量或其它中间信息。计算机系统100还包括耦合在总线101上用于存储静态信息及处理器109的指令的只读存储器(ROM)106与/或其它静态存储设备。数据存储设备107耦合在总线101上用于存储信息与指令。

[0096] 图1还示出处理器109包括执行单元130、寄存器文件150、高速缓冲存储器160、解码器165及内部总线170。当然，处理器109还包含其它电路，为了不冲淡本发明而未示出它们。

[0097] 执行单元130用于执行处理器109所接收的指令。除了识别通常在通用处理器实现的指令，执行单元130还识别在分组数据格式上执行操作的分组指令集140中的指令。在一个实施例中，分组指令集140包含以此后描述的方式支持组装操作、分解操作、分组加法运算、分组减法运算、分组乘法运算、分组移位操作、分组比较操作、乘-加运算、乘-减运算、个数计算操作及一组分组逻辑运算(包含分组“与”、分组“与非”、分组“或”及分组“异或”)的指令。虽然描述了包含这些指令的分组指令集140的一个实施例，其它实施例可包含这些指令的子集或超集。

[0098] 通过包含这些指令，可以用分组数据执行多媒体应用中所使用的许多算法所需要的操作。从而，可以编写这些算法来组装必要的数据及在分组数据上执行必要的操作，而无须分解这些分组数据来一次在一个数据元素上执行一个或多个操作。如上所述，这比不支持某些多媒体算法所要求的分组数据操作的先有技术通用处理器(即，如果多媒体算法要求不能在分组数据上执行的操作，则程序必须分解该数据，单独地在分开的元素上执行操作，然后将结果组装成分组结果供进一步分组处理)具有性能上的优势。此外，所公开的在其中执行若干条这些指令的方式改进了许多多媒体应用的性能。

[0099] 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。

[0100] 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。

[0101] 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。

[0102] 计算机系统100也可以是计算机网络(诸如LAN)中的终端。计算机系统100这时便是计算机网络的一个计算机子系统。计算机系统100可选地包含视频数字化设备126。视频数字化设备126能用来捕捉能在计算机网络上传输给其它计算机的视频图象。

[0103] 在一个实施例中,处理器109附加支持与X86指令集(诸如加州Santa Clara的Intel公司制造的Pentium®处理器等现有微处理器所使用的指令集)兼容的指令集。从而,在一个实施例中,处理器109支持加州Santa Clara的Intel公司所定义的IA™-Intel体系结构所支持的所有操作(见"微处理器",Intel资料集卷1与卷2,1992与1993,可从加州Santa Clara的Intel购得)。结果,除了本发明的操作外,处理器109还能支持现有的X86操作。虽然本发明是描述为包含在基于X86指令集中的,替代实施例可将本发明包含在其它指令集中。例如,可将本发明包含在采用新指令集的64位处理器中。

[0104] 图2说明按照本发明的一个实施例的处理器的寄存器文件。寄存器文件150用来存储信息,包括控制/状态信息、整数数据、浮点数据及分组数据。在图2所示的实施例中,寄存器文件150包括整数寄存器201、寄存器209、状态寄存器208及指令指针寄存器211。状态寄存器208指示处理器109的状态。指令指针寄存器211存储要执行的下一条指令的地址。整数寄存器201、寄存器209、状态寄存器208及指令指针寄存器211全都耦合在内部总线170上。任何附加的寄存器也耦合在内部总线170上。

[0105] 在一个实施例中,寄存器209既用于分组数据又用于浮点数据。在这一实施例中,处理器109在任何给定时刻都必须将寄存器209作为栈定位的浮点寄存器或作为非栈定位的分组数据寄存器对待。在本实施例中,包括了一种机制允许处理器109在作为栈定位的浮点寄存器与非栈定位的分组数据寄存器的寄存器209上操作之间切换。在另一实施例中,处理器109可同时在作为非栈定位的浮点与分组数据寄存器的寄存器209上操作。作为另一实例,在另一实施例中,这些相同的寄存器可用来存储整数数据。

[0106] 当然,可以实现替代的实施例来包括或多或少的寄存器组。例如,替代实施例可包含独立的浮点寄存器组用于存储浮点数据。作为另一实例,替代实施例可包含第一组寄存器,各用于存储控制/状态信息,及第二组寄存器,各能存储整数、浮点及分组数据。为了清楚起见,不应将一个实施例的寄存器的意义限制在特定类型的电路上。而是,一个实施例的寄存器只需要能存储与提供数据,并执行这里所描述的功能。

[0107] 可将各种寄存器组(诸如整数寄存器201、寄存器209)实现成包含不同数目的寄存器与/或不同大小的寄存器。例如,在一个实施例中,将整数寄存器201实现为存储32位,而将寄存器209实现为存储80位(全部80位用来存储浮点数据而只用64位存储分组数据)。此外,寄存器209包含8个寄存器,$R_0$212a至$R_7$212h。$R_1$212a、$R_2$212b及$R_3$212c为寄存器209中各个寄存器的实例。可将寄存器209中的一个寄存器的32位移到整数寄存器201中的一个整数寄存器中。类似地,可将整数寄存器中的值移入寄存器209中的一个寄存器32位中。在另一实施例中,整数寄存器201各包含64位,并且64位数据可在整数寄存器201与寄存器209之间传送。

[0108] 图3为说明按照本发明的一个实施例的处理器用来处理数据的通用步骤的流程图。例如,这些操作中包含加载操作,将来自高速缓冲存储器160、主存储器104、只读存储器(ROM)104或数据存储设备107的数据加载到寄存器文件150中的寄存器。

[0109] 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。

[0110] 在步骤302，解码器202存取寄存器文件150或存储器中的单元。取决于控制信号207中指定的寄存器地址存取寄存器文件150中的寄存器或存储器中的存储器单元。例如，对于分组数据上的操作，控制信号207可包含SRC1、SRC2及DEST寄存器地址。SRC1是第一源寄存器的地址。SRC2是第二源寄存器的地址。由于不是所有操作都需要两个源地址，在某些情况中SRC2地址是可选的。如果一种操作不需要SRC2地址，便只使用SRC1地址。DEST是存储结果数据的目的地寄存器的地址。在一个实施例中，SRC1或SRC2也用作DEST。相对于图6a与图6b更全面地描述SRC1、SRC2及DEST。存储在对应寄存器中的数据分别称作源1(Source1)、源2(Source2)与结果(Result)。每一个这种数据的长度都是64位。

[0111] 在本发明的另一实施例中，SRC1、SRC2及DEST中任何一个或全部能定义处理器109的可寻址存储器空间中的一个存储器单元。例如，SRC1可标识主存储器104中的存储器单元，而SRC2标识整数寄存器201中的第一寄存器及DEST标识寄存器209中的第二寄存器。这里为了简化描述，本发明将相对于存取寄存器文件150描述。然而，这些存取能对存储器进行。

[0112] 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。

[0113] 数据与存储格式

[0114] 图4说明按照本发明的一个实施例的分组数据类型。示出了三种分组数据格式：分组字节401、分组字402及分组双字403。在本发明的一个实施例中，分组字节为包含8个数据元素的64位长。各数据元素为一个字节长。通常，数据元素是与其它相同长度的数据元素一起存储在单一寄存器(或存储器单元)中的一个单独的数据片段。在本发明的一个实施例中，存储在寄存器中的数据元素的数目是64位除以一个数据元素的位长度。

[0115] 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。

[0116] 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。

[0117] 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_0$212a至$R_7$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、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。

[0118] 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。

[0119] 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。

[0120] 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_0$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。

[0121] 控制信号格式

[0122] 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。

[0123] 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，SRC2603(位19至位14)包含寄存器209中的寄存器的地址。这一第二源寄存器包含在该操作执行期间要使用的分组数据Source2。DEST 605(位5至位0)包含寄存器209中的寄存器地址。这一目的地寄存器将存储分组数据操作的结果分组数据Result。

[0124] 控制位SZ 610(位12与位13)指示在第一与第二分组数据源寄存器中的数据元素的长度。如果SZ 610等于01_2，则将分组数据格式化为分组字节401。如果SZ 610等于10_2，则将分组数据格式化为分组字402。SZ 610等于00_2或11_2保留不用，然而在另一实施例中，这些值之一可用来指示分组双字403。

[0125] 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。

[0126] 控制位S612(位10)指示使用带符号操作。如果S612等于1，则执行带符号操作。如果S612等于0，则执行无符号操作。

[0127] 图6b说明按照本发明的一个实施例指示采用分组数据的第二种控制信号格式。这一格式对应于可从Intel公司，文献销售处(P.O.Box7641,Mt.Prospect,IL,60056-7641)购得的“Pentium处理器系列用户手册”中描述的通用整数操作码格式。注意将OP 601、SZ 610、T 611与S 612全部合并成一个大字段。对于某些控制信号，位3至5为SRC1602。在一个实施例中，当存在一个SRC1602地址时，则位3至5也对应于DEST 605。在一个替代实施例中，当存在SRC2603地址时，则位0至2也对应于DEST 605。对于其它控制信号，如分组移位立即操作，位3至5表示对操作码字段的扩展。在一个实施例中，这一扩展允许程序员包含一个具有控制信号的立即值，诸如移位计数值。在一个实施例中，立即值在控制信号后面。这在“Pentium处理器系列用户手册”附录F，页F-1至F-3中有更详细的描述。位0至2表示SRC2603。这一通用格式允许寄存器到寄存器、存储器到寄存器、寄存器被存储器、寄存器被寄存器、寄存器被立即数、寄存器到存储器的寻址。同时，在一个实施例中，这一通用格式能支持整数寄存器到寄存器及寄存器到整数寄存器寻址。

[0128] 饱和/不饱和的说明

[0129] 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。

[0130]

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}	$2^{63}-1$

[0131] 表7

[0132] 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。

[0133] 分组加法

[0134] 分组加法运算

[0135] 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。

[0136] 图7a说明按照本发明的一个实施例执行分组加法的方法。在步骤701，解码器202解码处理器109接收的控制信号207。从而，解码器202解码出：分组加法的操作码；寄存器209中的SRC1602、SRC2603及DEST 605地址；饱和/不饱和、带符号/无符号及分组数据中的数据元素的长度。在步骤702，解码器202通过内部总线170存取寄存器文件150中给出SRC1602与SRC2603地址的寄存器209。寄存器209向执行单元130提供分别存储在这些地址上的寄存器中的分组数据Source1与Source2。即，寄存器209通过内部总线170将分组数据传递给执行单元130。

[0137] 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。

[0138] 假定数据元素长度为8位，则执行步骤705a。执行单元130将Source1的位7至位0加在SRC2的位7至位0上，生成Result分组数据的位7至位0。与这一加法并行，执行单元130将

Source1的位15至位8加在Source2的位15至位8上，产生Result分组数据的位15至位8。与这些加法并行，执行单元130将Source1的位23至位16加在Source2的位23至位16上，产生Result分组数据的位23至位16。与这些加法并行，执行单元130将Source1的位31至位24加在Source2的位31至位24上，产生Result分组数据的位31至位24。与这些加法并行，执行单元将Source1的位39至位32加在Source2的位39至位32上，产生Result分组数据的位39至位32。与这些加法并行，执行单元130将Source1的位47至40加在Source2的位47至位40上，产生Result分组数据的位47至位40。与这些加法并行，执行单元130将Source1的位55至位48加在Source2的位55至位48上，产生Result分组数据的位55至位48。与这些加法并行，执行单元130将Source1的位63至位56加在Source2的位63至位56上，产生Result分组数据的位63至位56。

[0139] 假定数据元素长度为16位，则执行步骤705b。执行单元130将Source1的位15至位0加在SRC2的位15至位0上，产生Result分组数据的位15至位0。与这一加法并行，执行单元130将Source1的位31至位16加在Source2的位31至位16上，产生Result分组数据的位31至位16。与这些加法并行，执行单元130将Source1的位47至位32加在Source2的位47至位32上，产生Result分组数据的位47至位32。与这些加法并行，执行单元130将Source1的位63至位48加在Source2的位63至位48上，产生Result分组数据的位63至位48。

[0140] 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。

[0141] 表8a说明分组加法运算的寄存器表示。第一行的位是Source1分组数据的分组数据表示。第二行的位是Source2分组数据的分组数据表示。第三行的位是Result分组数据的分组数据表示。各数据元素位下面的号码是数据元素号码。例如，Source1数据元素0为10001000_2。因此，如果该数据元素是8位长度(字节数据)且执行的是无符号不饱和的加法，则执行单元130产生所示的Result分组数据。

[0142] 注意在本发明的一个实施例中，当结果上溢或下溢且运算采用不饱和时，简单地截位结果。即忽略进位位。例如，在表8a中，结果数据元素1的寄存器表示将是：$10001000_2+10001000_2=00001000_2$。类似地，对于下溢也截位其结果。这一截位形式使程序员能容易地执行模运算。例如，结果数据元素1的公式可表示为：(Source1数据元素1+Source2数据元素1)mod 256＝结果数据元素1。此外，熟悉本技术的人员会从这一描述理解上溢与下溢可通过在状态寄存器中设置出错位来检测。

[0143]

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

[0144] 表8a

[0145] 表8b说明分组字数据加法运算的寄存器表示。因此，如果该数据元素是16位长度(字数据)且所执行的是无符号不饱和加法，执行单元130产生所示的Result分组数据。注意在字数据元素2中，来自位7(见下面强调的位1)的进位传播到位8中，导致数据元素2上溢(见下面强调的“上溢”)。

[0146]

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

[0147] 表8b

[0148] 表8c说明分组双字数据加法运算的寄存器表示。本发明的一个替代实施例支持这一运算。因此，如果该数据元素是32位长度(即双字数据)且执行的是无符号不饱和加法，执行单元130产生所示的Result分组数据。注意来自双字数据元素1的位7与位15的进位分别传播到位8与位16中。

[0149]

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

[0150] 表8c

[0151] 为了更好地说明分组加法与普通加法之间的差别，表9中复制了来自上例的数据。然而，在这一情况中，在数据上执行普通加法(64位)。注意来自位7、位15、位23、位31、位39及位47的进位已分别带到位8、位16、位24、位32、位40及位48中。

[0152]

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

[0153] 表9

[0154] 带符号/不饱和分组加法

[0155] 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。

[0156]

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

[0157] 表10

[0158] 带符号/饱和的分组加法

[0159] 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。

[0160]

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

[0161] 表11

[0162] 分组减法

[0163] 分组减法运算

[0164] 本发明的一个实施例使在执行单元130中能执行分组减法运算。即，本发明使第二分组数据的各数据元素能从第一分组数据的各数据元素中分别地减去。

[0165] 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，

[0166] 步骤710-713类似于步骤701-704。

[0167] 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。

[0168] 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。

[0169] 假定数据元素长度是16位，则执行步骤714b与715b。执行单元130求Source2的位15至位0的2的补码。与这一求2的补码并行，执行单元130求Source2的位31至位16的2的补码。与这些求2的补码并行，执行单元130求Source2的位47至位32的2的补码。与这些求2的补码并行，执行单元130求Source2的位63至位48的2的补码。在步骤715b，执行单元130执行Source2的2的补码位与Source1的位的加法，如对步骤705b总的描述。

[0170] 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。

[0171] 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。

[0172] 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。

[0173]

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

[0174] 表12

[0175] 分组数据加法/减法电路

[0176] 图8说明按照本发明的一个实施例在分组数据的各个位上执行分组加法与分组减法的电路。图8示出修改过的位片加法器/减法器800。加法器/减法器801a-b能在Source1加上或减去来自Source2的两位。运算与进位控制803向控制线路809a传输控制信号启动加法或减法运算。从而，加法器/减法器801a在$Source1_i$ 804a上所接收的位i上加上或减去$Source2_i$ 805a上接收的位i，产生在$Result_i$ 806a上传输的结果位。Cin 807a-b与Cout 808a-b表示在加法器/减法器上经常见到的进位控制电路。

[0177] 从运算与进位控制803通过分组数据使能811启动位控制802来控制Cin_{i+1}807b及$Cout_i$。例如，在表13a中，执行无符号分组字节加法。如果加法器/减法器801a相加Source1

位7与Source2位7，则运算与进位控制803将启动位控制802，停止将进位从位7传播到位8。

[0178]

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

[0179] 表13a

[0180] 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。

[0181]

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

[0182] 表13b

[0183] 加法器/减法器801a通过首先反相Source2i 805a及加1形成$Source2_i$ 805a的2的补码，从Source 1_i 804a减去位$Source2_i$ 805a。然后，加法器/减法器801a将这一结果加在$Source1_i$ 804a上。位片的2的补码运算技术是本技术中众所周知的，熟悉本技术的人员会理解如何设计这一位片的2的补码运算电路。注意进位的传播是受位控制802及运算与进位控制803控制的。

[0184] 图9说明按照本发明一个实施例在分组字节数据上执行分组加法与分组减法的电路。Source1总线901与Source2总线902分别通过$Source1_{in}$ 906a-h与$Source2_{in}$ 905a-h将信息信号带到加法器/减法器908a-h中。从而，加法器/减法器908a在Source1位7至位0上加上/减去Source2位7至位0；加法器/减法器908b在Source1位15至位8上加上/减去Source2位15至位8，等等。CTRL 904a-h通过分组控制911接收来自运算控制903的禁止进位传播、允许/禁止饱和以及允许/禁止带符号/无符号算术运算的控制信号。运算控制903通过从CTRL904a-h接收进位信息并且不将它传播给次最高位加法器/减法器908a-h而禁止进位传播。从而，运算控制903执行运算与进位控制803及64位分组数据的位控制802的运算。给出了图1-9中的示例及上述描述，熟悉本技术的人员能建立这一电路。

[0185] 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。

[0186] 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。

[0187] 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。

[0188] 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)。

[0189] 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。

[0190] 分组乘法

[0191] 分组乘法运算

[0192] 在本发明的一个实施例中，SRC1寄存器中包含被乘数数据

[0193] (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。

[0194] 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。

[0195] 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。

[0196] 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。

[0197] 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。

[0198] 在步骤1201，解码器202解码处理器109接收的控制信号207。从而，解码器202解码出：适当乘法运算的运算码；寄存器209中的SRC1602、SRC2603及DEST 604地址；带符号/无符号、高/低及分组数据中的数据元素的长度。

[0199] 在步骤1202，解码器202通过内部总线170存取寄存器文件150中给定SRC1602与SRC2603地址的寄存器209。寄存器209向执行单元130提供存储在SRC1603寄存器中的分组数据(Source1)及存储在SRC2603寄存器中的分组数据(Source2)。即，寄存器209通过内部总线170将分组数据传递给执行单元130。

[0200] 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。

[0201] 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。

[0202] 假定数据元素的长度为8位，便执行步骤1212。在步骤1212中，执行下述各操作，将Source1位7至位0乘以Source2位7至位0生成Result位7至位0。将Source1位15至8乘以Source2位15至8生成Result位15至8。将Source1位23至16乘以Source2位23至16生成Result位23至16。将Source1位31至24乘以Source2位31至24生成Result位31至24。将Source1位39至32乘以Source2位39至32生成Result位39至32。将Source1位47至40乘以Source2位47至40生成REsult位47至40。将Source1位55至48乘以Source2位55至48生成Result位55至48。将Source1位63至56乘以Source2位63至56生成Result位63至56。

[0203] 假定数据元素的长度为16位，则执行步骤1214。在步骤1214中，执行下述操作。将Source1位15至0乘以Source2位15至0生成Result位15至0。将Source1位31至16乘以Source2位31至16生成Result位31至16。将Source1位47至32乘以Source2位47至32生成Result位47至32。将Source1位63至48乘以Source2位63至48生成Result位63至48。

[0204] 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。

[0205] 在步骤1220将Result存储在DEST寄存器中。

[0206] 表14示出在分组字数据上的分组乘法无符号高运算的寄存器表示。第一行的位为Source1的分组数据表示。第二行的位为Source2的数据表示。第三行的位为Result的分组

数据表示。各数据元素位下面的号码为数据元素号。例如，Source1数据元素2为1111111100000000$_2$。

[0207]

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

[0208] 表14

[0209] 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。

[0210]

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

[0211] 表15

[0212] 表16示出分组字数据上的分组乘法低运算的寄存器表示。

[0213]

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

[0214] 表16

[0215] 分组数据乘法电路

[0216] 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。

[0217] 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。

[0218] 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。

[0219] 分组字乘法器1301包含4个16×16乘法器电路：16×16乘法器A1310、16×16乘法器B1311、16×16乘法器C 1312及16×16乘法器D1313。16×16乘法器A 1310具有输入Source1[15:0]与Source2[15:0]，16×16乘法器B 1311具有输入Source1[31:16]与Source2[31:16]，16×16乘法器C 1312具有输入Source1[47:32]与Source2[47:32]，16×16乘法器D 1313具有输入Source1[63:48]与Source2[63:48]。各16×16乘法器耦合在符号使能1382上。各16×16乘法器产生32位积。对于各乘法器，多路复用器(分别为Mx01350、Mx11351、Mx21352及Mx31353)接收32位结果。取决于高/低使能1380的值，各多路复用器输出积的16个高阶位或16个低阶位。将四个多路复用器的输出组合成一个64位结果。这一结果可选地存储在结果寄存器11371中。

[0220] 分组字节乘法器1302包含8个8×8乘法器电路：8×8乘法器A1320至8×8乘法器H1337。各8×8乘法器具有来自各Source1[63:0]1331及Source2[63:0]1333的8位输入。例如8×8乘法器A 1320具有输入Source1[7:0]与Source2[7:0]，而8×8乘法器H 1327具有输入Source1[63:56]与Source2[63:56]。各8×8乘法器耦合在符号使能1382上。各8×8乘法器产生一个16位积。对于各乘法器，多路复用器(诸如Mx41360与Mx111367)接收16位结果。取决于高/低使能1380的值。各多路复用器输出积的8个高阶位或8个低阶位。将8个多路复用器的输出组合成一个64位结果。可选地将这一结果存储在结果寄存器21372中。取决于该运算要求的数据元素的长度，字节/字使能1381启动特定的结果寄存器。

[0221] 在一个实施例中，通过制造能乘两个8×8数或一个16×16数两者的电路，减少用于实现乘法的面积。即将两个8×8乘法器及一个16×16乘法器组合成一个8×8与16×16乘法器。运算控制1300将允许乘法的适当长度。在这一实施例中，可以缩小乘法器所使用的物理面积，然而它将难于执行分组字节乘法及分组字乘法。在另一支持分组双字乘法的实施例中，一个乘法器能执行四个8×8乘法、两个16×16乘法或一个32×32乘法。

[0222] 在一个实施例中，只提供分组字乘法运算。在这一实施例中，不包含分组字节乘法器1302及结果寄存器21372。

[0223] 在指令集中包含上述分组乘法运算的优点

[0224] 从而上述分组乘法指令提供了Source1中各数据元素乘以Source2中其对应数据元素的独立乘法。当然，要求Source1各元素乘以同一个数算法可通过将该相同的数存储在Source2的各元素中来执行。此外，这一乘法指令通过断开进位链防止上溢；借此解除程序员的这一责任，不再需要准备数据来防止上溢的指令，并得出更健壮的代码。

[0225] 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。

[0226] 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。

[0227] 乘-加/减

[0228] 乘-加/减运算

[0229] 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。

[0230] Multiply-Add Source1,Source2

[0231]

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

[0232] 表17a

[0233]

11111111 11111111	11111111 00000000	01110001 11000111	01110001 11000111
乘 3	乘 2	乘 1	乘 0
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	

[0234] 表17b

[0235] 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。

[0236] Multiply-Subtract Source1。Source2

[0237]

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

[0238] 表12

[0239] 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。

[0240] 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。

[0241] 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。

[0242] 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。

[0243] 在步骤1402,解码器202通过内部总线170存取给出SRC1602与SRC2603地址的寄存器文件150中的寄存器209。寄存器209向执行单元130提供存储在SRC1602寄存器中的分组数据(Source1)及存储在SRC2603寄存器中的分组数据(Source2)。即寄存器209通过内部总线170将分组数据传递给执行单元130。

[0244] 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。

[0245] 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。

[0246] 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。

[0247] 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。

[0248] 在步骤1420,将Result存储在DEST寄存器中。

[0249] 分组数据乘-加/减电路

[0250] 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。

[0251] 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。

[0252] 分组乘-加法器/减法器1501具有下列输入:Source1[63:0]1531、Source2[63:0]

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

[0253] 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。

[0254] 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。

[0255] 为了在分解数据上操作的先有技术处理器上执行这些乘-加或乘-减指令的等效指令,将需要四次独立的64位乘法运算与两次64位加或减法运算以及必要的加载与存储操作。这浪费用于Source1与Source2的高于位16及Result的高于位32的数据线及电路。并且,这种先有技术处理器生成的整个64位结果对于程序员可能是无用的。因此,程序员将必须截断各结果。

[0256] 在指令集中包含上述乘-加运算的优点

[0257] 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。

[0258] 从而通过在处理器109支持的指令集中加入上述乘-加与/或乘-减指令,便能用比缺少这些指令的先有技术通用处理器较少的指令执行许多功能。

[0259] 分组移位

[0260] 分组移位操作

[0261] 在本发明的一个实施例中,SCR1寄存器中包含要移位的数据(Source1),SRC2寄存器中包含表示移位计数的数据(Source2),而DEST寄存器中将包含移位的结果(Result)。即Source1中的各数据元素独立地移位该移位计数。在一个实施例中,将Source2解释为一个无符号的64位标量。在另一实施例中,Source2为分组数据并包含Source1中各对应数据元素的移位计数。

[0262] 在本发明的一个实施例中,支持算术移位与逻辑移位两者。算术移位将各数据元素的位向下移位指定的数目,并用符号位的初始值填充各数据元素的高阶位。对于分组字节数据大于7的移位计数、对于分组字数据,大于15的移位计数、或对于分组双字大于31的移位计数导致用符号位的初始值来填充各Result数据元素。逻辑移位可用向上或向下移位来操作。在逻辑向右移位中,用0来填充各数据元素的高阶位。在逻辑向左移位中,用零来填充各数据元素的最低有效位。

[0263] 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。

[0264] 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。

[0265] 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。

[0266] 在步骤1602，解码器202通过内部总线170存取寄存器文件150中给出SRC1602及SRC2603地址的寄存器209。寄存器209向执行单元130提供SRC1602寄存器中所存储的分组数据(Source1)及存储在SRC2603寄存器中的标量移位计数(Source2)。即寄存器209通过内部总线170将分组数据传递给执行单元130。

[0267] 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)。

[0268] 在步骤1610，数据元素的长度确定下面要执行哪一步骤。如果数据元素的长度是8位(字节数据)，则执行单元130执行步骤1612。然而，如果分组数据中的数据元素的长度为16位(字数据)，则执行单元130执行步骤1614。在一个实施例中，只支持8位与16位数据元素长度的分组移位。然而，在另一实施例中，也支持32位数据元素长度的分组移位。

[0269] 假定数据元素的长度为8位，便执行步骤1612。在步骤1612中，执行下述操作。将Source1位7至0移位移位计数(Source2位63至0)生成Result位7至0。Source1位15至8移位移位计数生成Result位15至8。将Source1位23至16移位移位计数生成Result位23至16。将Source1位31至24移位移位计数生成Result位31至24。将Source1位39至32移位移位计数生成Result位39至32。将Source1位47至40移位移位计数生成Result位47至40。将Source1位55至48移位移位计数生成Result位55至48。将Source1位63至56移位移位计数生成Result位63至56。

[0270] 假定数据元素的长度为16位，便执行步骤1614。在步骤1614中执行下述操作。将Source1位15至0移位移位计数生成Result位15至0。将Source1位31至16移位移位计数生成Result位31至16。将Source1位47至32移位移位计数生成Result位47至32。将Source1位63至48移位移位计数生成Result位63至48。

[0271] 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。

[0272] 在步骤1620，将Result存储在DEST寄存器中。

[0273] 表19说明字节分组算术向右移位操作的寄存器表示。第一行的位为Source1的分组数据表示。第二行的位为Source2的数据表示。第三行的位为Result的分组数据表示。各数据元素位下面的数字为数据元素号。例如，Source1数据元素3为10000000_2。

[0274]

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

[0275] 表19

[0276] 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

[0277]

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

[0278] 表20

[0279] 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。

[0280]

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

[0281] 表21

[0282] 分组数据移位电路

[0283] 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。

[0284] 图17说明按照本发明的一个实施例在分组数据的各个字节上执行分组移位的电路。图17示出经修改的字节片移位电路、字节片级i1799的使用。各字节片(除最高位数据元素字节片外)包含一个移位单元及位控制。最高位数据元素字节片只需一个移位单元。

[0285] 移位单元i1711及移位单元i+11771各允许将来自Source1的8位移位该移位计数。在一个实施例中，各移位单元象已知的8位移位电路那样操作。各移位单元具有一个Source1输入、一个Source2输入、一个控制输入、一个下一级信号、一个上一级信号及一个

结果输出。因此，移位单元i1711具有Source1$_i$ 1731输入、Source2[63:0]1733输入、控制i1701输入、下一级i1713信号、上一级i1712输入及存储在结果寄存器i 1751中的结果。因此，移位单元i+11771具有Source1$_{i+1}$1732输入、Source2[63:0]1733输入、控制i+11702输入、下一级i+11773信号、上一级i+11772输入及存储在结果寄存器i+11752中的结果。

[0286] Source1输入通常是Source1的一个8位部分。8位表示数据元素的最小类型，一个分组字节数据元素。Source2输入表示移位计数。在一个实施例中，各移位单元从Source2[63:0]1733接收相同的移位计数。操作控制1700传输控制信号启动各移位单元执行要求的移位。控制信号是从移位类型(算术/逻辑)及移位方向确定的。下一级信号是从该移位单元的位控制接收的。取决于移位的方向(左/右)，在下一级信号上移位单元将最高位移出/进。类似地，取决于移位的方向(右/左)，各移位单元在上一级信号上将最低位移出/进。上一级信号是从前一级的位控制单元接收的。结果输出表示移位单元在其上操作的Source1的部分上的移位操作的结果。

[0287] 位控制i1720是从操作控制1700通过分组数据启动i1706启动的。位控制i1720控制下一级i1713及上一级i+11772。例如，假定移位单元i1711负责Source1的8个最低位，而移位单元i+11771负责Source1的下一个8位。如果执行在分组字节上的移位，位控制i1720将不允许来自移位单元i+11771的最低位与移位单元i1711的最高位连通。然而，执行分组字上的移位时，则位控制i1720将允许来自移位单元i+11771的最低位与移位单元i1711的最高位连通。

[0288] 例如，在表22中，执行分组字节算术向右移位。假定移位单元i+11771在数据元素1上操作，而移位单元i1711在数据元素0上操作。移位单元i+11771将其最低位移出。然而操作控制1700将导致位控制i1720停止从上一级i+11721接收的该位传播到下一级i1713。反之，移位单元i1711以符号位填充最高阶位Source1[7]。

[0289]

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

[0290] 表22

[0291] 然而，如果执行分组字算术移位，则将移位单元i+11771的最低位传递给移位单元i 1711的最高位。表23示出这一结果。这一传递对于分组双字移位同样允许。

[0292]

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

[0293] 表23

[0294] 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。

[0295] 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。

[0296] 在指令集中包含上述移位操作的优点

[0297] 上述分组移位指令导致Source1的各元素移位指定的移位计数。通过在指令集中加入这一指令，便可使用单一指令移位一个分组数据的各元素。反之，不支持这一操作的先有技术通用处理器必须执行许多指令来分解Source1，单个地移位各分解的数据元素，然后将结果组装成分组数据格式供进一步分组处理。

[0298] 传送操作

[0299] 传送操作向或从寄存器209传送数据。在一个实施例中，SRC2603为包含源数据的地址而DEST 605则是数据要传送到的地址。在这一实施例中，不使用SRC1602。在另一实施例中，SRC1602等于DEST 605。

[0300] 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。

[0301] 传送操作可将数据从存储器传送到寄存器209，从寄存器209到存储器，及从寄存器209中的一个寄存器到寄存器209中的第二寄存器。在一个实施例中，分组数据是存储在与存储整数数据不同的寄存器中的。在这一实施例中，传送操作能将数据从整数寄存器201传送到寄存器209。例如，在处理器109中，如果分组数据存储在寄存器209中而整数数据存储在整数寄存器201中，则传送指令能用来将数据从整数寄存器201传送到寄存器209，反之亦然。

[0302] 在一个实施例中，当为传送指定了一个存储器地址时，在该存储单元的8个字节数据(该存储单元包含最低字节)被加载到寄存器209中的一个寄存器或从该寄存器存储到指定存储器单元中。当指定寄存器209中的一个寄存器时，便将该寄存器的内容传送到或加载自寄存器209中的第二寄存器或者从第二寄存器向指定寄存器加载。如果整数寄存器201的长度为64位，并指定了一个整数寄存器，则将该整数寄存器中的8个字节数据加载到寄存器209中的寄存器或从该寄存器存储到指定的整数寄存器中。

[0303] 在一个实施例中，整数是表示为32位的。在执行从寄存器209到寄存器201的传送

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

[0304] 组装操作

[0305] 在本发明的一个实施例中，SRC1602寄存器包含数据(Source1)，SRC2603寄存器包含数据(Source2)，而DEST 605寄存器将包含操作的结果数据(Result)。这便是，将Source1的部分与Source2的部分组装在一起生成Result。

[0306] 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。

[0307] 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。

[0308] 在步骤1801，解码器202解码处理器109接收的控制信号207。从而，解码器202解码出：适当的组装操作的操作码；寄存器209中的SRC1602、SRC2603及DEST 605地址；饱和/不饱和，带符号/无符号及分组数据中的数据元素长度。如上所述，SRC1602(或SRC2

[0309] 603)可用作DEST 605。

[0310] 在步骤1802，解码器202通过内部总线170存取寄存器文件150中给定SRC1602与SRC2603地址的寄存器209。寄存器209向执行单元130提供存储在SRC1602寄存器中的分组数据(Source1)及存储在SRC2603寄存器中的分组数据(Source2)。即寄存器209通过内部总线170将分组数据传递给执行单元130。

[0311] 在步骤1803，解码器202启动执行单元130去执行适当的组装操作。解码器202还通过内部总线170传递饱和及Source1和Source2中的数据元素的长度。可选用饱和来使结果数据元素中的数据的值成为最大。如果Source1或Source2中的数据元素的值大于或小于Result的数据元素所能表示的值的范围，则将对应的结果数据元素设定在其最高或最低值上。例如，如果Source1与Source2的字数据元素中的带符号值小于0x80(或对于双字0x8000)，则将结果字节(或字)数据元素箝位到0x80(或对于双字0x8000)上。如果Source1与Source2的字数据元素中的带符号值大于0x7F(或对于双字0x7FFF)，则将结果字节(或字)数据元素箝位到0x7F(或0x7FFF)上。

[0312] 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。

[0313] 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。

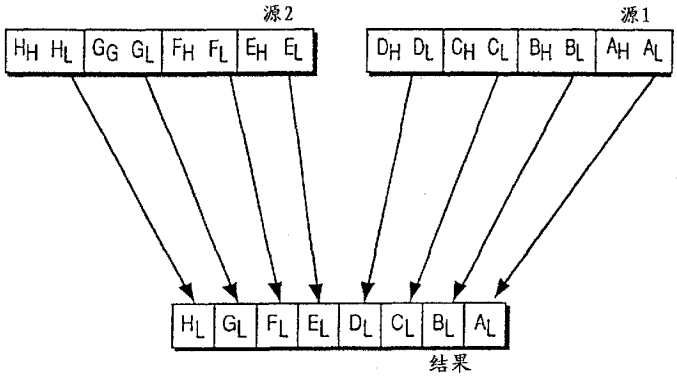
[0314] 假定源数据元素的长度为32位，便执行步骤1814。在步骤1814中，执行下述操作。Source1位15至0为Result位15至0。Source1位47至32为Result位31至16。Source2位15至0为Result位47至32。Source2位47至32为Result位63至48。如果设定了饱和，则测试各双字的高阶位来确定是否应将Result数据元素箝位。

[0315] 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。

[0316] 在步骤1820，将Result存储在DEST 605寄存器中。

[0317] 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_L表示Source1中的数据元素A的低阶8位。

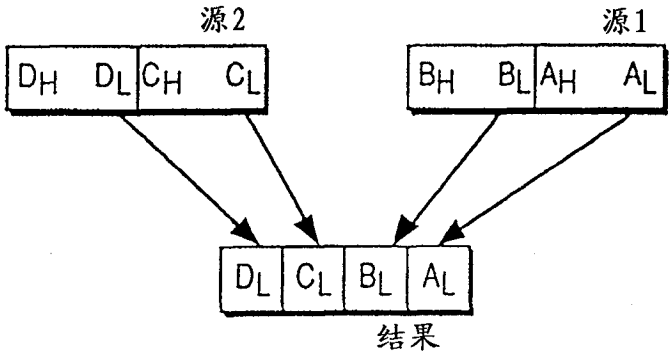
[0318]



[0319] 表24

[0320] 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。

[0321]



[0322] 表25

[0323] 组装电路

[0324] 在本发明的一个实施例中，为了达到组装操作的高效执行，采用了并行性。图19a与19b示出按照本发明的一个实施例在分组数据上执行组装操作的电路。该电路能有选择地执行带饱和的组装操作。

[0325] 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。

[0326] 操作控制1900接收来自解码器202的信息来启动组装操作。操作控制1900使用饱和值为各测试饱和电路启动饱和测试。如果源分组数据的长度为字分组数据503，则操作控

制1900设定输出使能1931。这便启动结果寄存器1952的输出。如果源分组数据的长度为双字分组数据504，则操作控制1900设定输出使能1932。这便启动输出寄存器1953的输出。

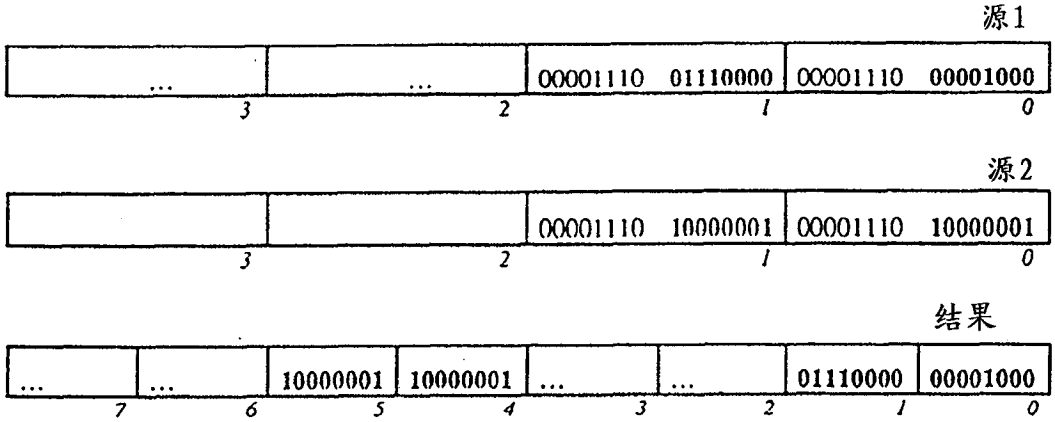
[0327] 各测试饱和电路能有选择地测试饱和。如果禁止饱和测试，则各测试饱和电路只将低阶位传递到结果寄存器中对应的位置上。如果允许测试饱和，则各测试饱和电路测试高阶位来确定是否应箝位结果。

[0328] 测试饱和1910至测试饱和1917具有16位输入与8位输出。8位输出为输入的低8位，或可选地是一个箝位的值(0x80、0x7F、或0xFF)。测试饱和1910接收Source1位15至0并向结果寄存器1952输出位7至0。测试饱和1911接收Source1位31至16并向结果寄存器1952输出位15至8。测试饱和1912接收Source1位47至32并向结果寄存器1952输出位23至16。测试饱和1913接收Source1位63至48并向结果寄存器1952输出位31至24。测试饱和1914接收Source2位15至0并向结果寄存器1952输出位39至32。测试结果1915接收Source2位31至16并向结果寄存器1952输出位47至40。测试饱和1916接收Source2位47至32并向结果寄存器1952输出位55至48。测试饱和1917接收Source2位63至48并向结果寄存器1952输出位63至56。

[0329] 测试饱和1920至测试饱和1923具有32位输入与16位输出。16位输出为输入的低16位，或可选地为一个箝位的值(0x8000、0x7FFF或0xFFFF)。测试饱和1920接收Source1位31至0并为结果寄存器1953输出位15至0。测试饱和1921接收Source1位63至32并为结果寄存器1953输出位31至16。测试饱和1922接收Source2位31至0并为结果寄存器1953输出位47至32。测试饱和1923接收Source2位63至32并为结果寄存器1953输出位63至48。

[0330] 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。

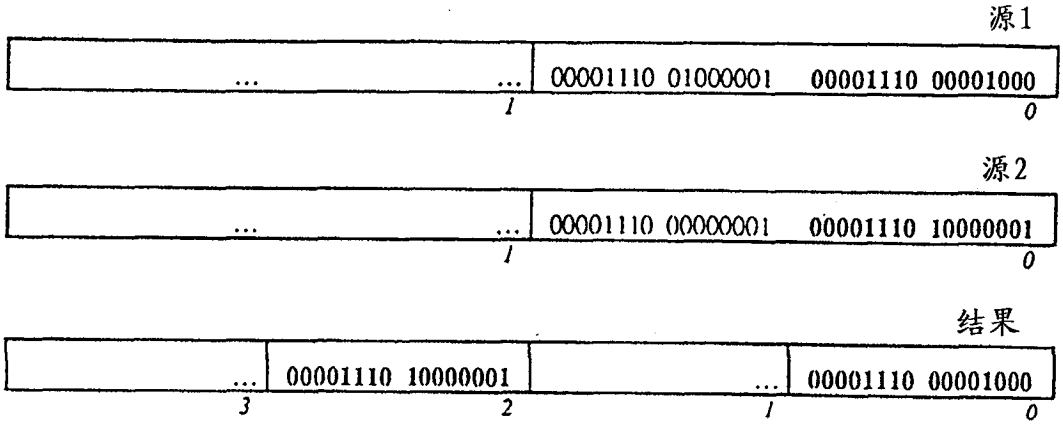
[0331]



[0332] 表26

[0333] 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。

[0334]



[0335] 表27

[0336] 在指令集中包含上述组装操作的优点

[0337] 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：

[0338] Pack.High Source1,Source2

[0339]

A0.	.A0	C0.	.C0	源1
G0.	.G0	B0.	.B0	源2
		=		
A0.	C0.	G0.	B0.	结果1

[0340] 表28

[0341] 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。

[0342] 分解操作

[0343] 分解操作

[0344] 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)。

[0345] 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。

[0346] 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。

[0347] 在步骤2010，数据元素的长度确定下面要执行哪一步骤。如果数据元素的长度为8位(分组字节401数据)，则执行单元130执行步骤2012。然而，如果分组数据中的数据元素的长度为16位(分组字402数据)则执行单元130执行步骤2014。然而，如果分组数据中的数据元素的长度为32位(分组双字503数据)，则执行单元130执行步骤2016。

[0348] 假定源数据元素长度为8位，便执行步骤2012。在步骤2012中，执行下述操作。

Source1位7至0为Result位7至0。Source2位7至0为Result位15至8。Source1位15至8为Result位23至16。Source2位15至8为Result位31至24。Source1位23至16为Result位39至32。Source2位23至16为Result位47至40。Source1位31至24为Result位55至48。Source2位31至24为Result位63至56。

[0349] 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。

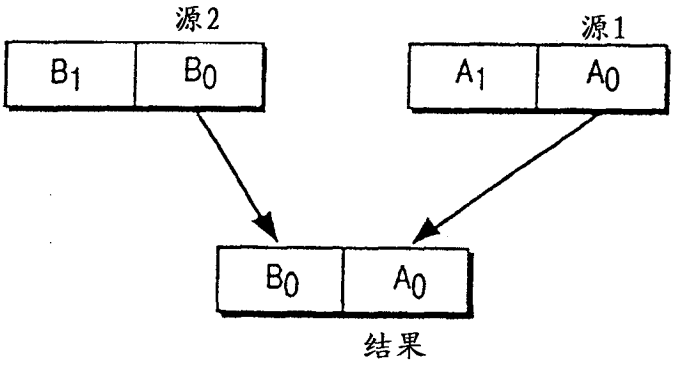
[0350] 假定源数据元素长度为32位，则执行步骤2016。在步骤2016中，执行下述操作。Source1位31至0为Result位31至0。Source2位31至0为Result位63至32。

[0351] 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。

[0352] 在步骤2020，将Result存储在DEST 605寄存器中。

[0353] 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。

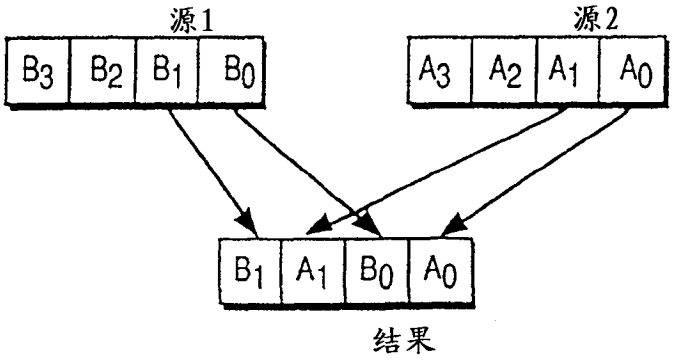
[0354]



[0355] 表29

[0356] 表30说明分解字操作(各数据元素A_{0-3}及B_{0-3}包含16位)的寄存器表示。

[0357]



[0358] 表30

[0359] 表31说明分解字节操作(各数据元素A_{0-7}及B_{0-7}包含8位)的寄存器表示。

[0360]

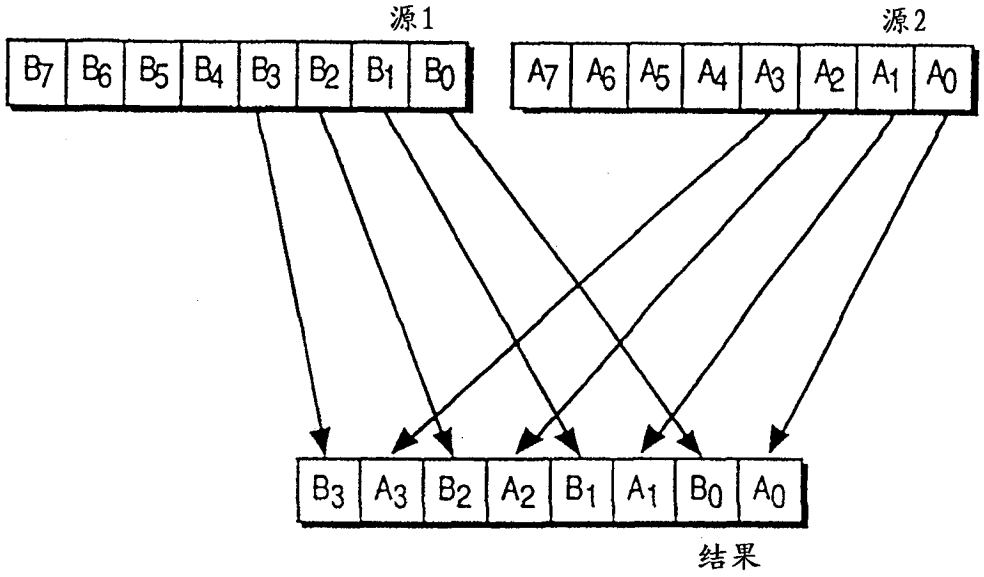


表31

[0361] 分解电路

[0362] 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。

[0363] 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。

[0364] 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。

[0365] 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。

[0366] 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。

[0367] 例如，在表32中，执行了分解字操作。操作控制2100将启动结果寄存器2153输出结果[63:0]2160。

源1

...	...	00001110 01110000	00001110 00001000
3	2	1	0

源2

[0368]

...	...	00001110 00000001	00001110 10000001
3	2	1	0

结果

00001110 00000001	00001110 01110000	00001110 10000001	00001110 00001000
3	2	1	0

[0369] 表32

[0370] 然而，如果执行分解双字，操作控制2100将启动结果寄存器2154输出Result[63:0]2160。表33示出这一结果。

源1

...	00001110 01000001	00001110 00001000
1	0	

源2

[0371]

...	00001110 00000001	00001110 10000001
1	0	

结果

00001110 00000001	00001110 10000001	00001110 01000001	00001110 00001000
1	0		

[0372] 表33

[0373] 在指令集中包含上述分解指令的优点

[0374] 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。

[0375]

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

[0376] 表34a

[0377] 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。

[0378]

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

[0379] 表34b

[0380] 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。

[0381] 个数计算

[0382] 个数计算

[0383] 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。

[0384] 表35a说明在分组数据上的个数计数操作的寄存器表示。第一行的位是Source1分组数据的分组数据表示。第二行的位是Result分组数据的分组数据表示。各数据元素位下面的数据字为数据元素号。例如，Source1数据元素0为1000111110001000_2。因此，如果数据元素长度为16位(字数据)，并且执行个数计数操作，执行单元130生成所示的Result分组数据。

[0385]

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

[0386] 表35a

[0387] 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。

[0388]

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

[0389] 表35b

[0390] 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。

[0391]

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

[0392] 表35c

[0393] 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。

[0394]

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

[0395] 表35d

[0396] 执行个数计数的方法

[0397] 图22为说明按照本发明的一个实施例在分组数据上执行个数计数操作的方法的流程图。在步骤2201，解码器202响应一个控制信号207的接收，解码该控制信号207。在一个实施例中，控制信号207是通过总线101供给的。在另一实施例中，控制信号207是高速缓冲存储器160供给的。从而，解码器202解码出：个数计数的操作码、以及寄存器209中的SRC1602及DEST 605地址。注意在当前本发明的实施例中不使用SRC2603。在这一实施例中也不使用饱和/不饱和、带符号/无符号及分组数据中的数据元素长度。在本发明的当前实施例中，只支持16位数据元素长度的分组加法。然而，熟悉本技术的人员会理解能在具有8个分组字节数据元素或两个分组双字数据元素的分组数据上执行个数计数。

[0398] 在步骤2202，解码器202通过内部总线170存取寄存器文件150中给出SRC1602地址的寄存器209。寄存器209向执行单元130提供存储在这一地址上的寄存器中的分组数据Source1，即寄存器209通过内部总线170将分组数据传递给执行单元130。

[0399] 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。

[0400] 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。

[0401] 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。

[0402] 在一个数据元素上执行个数计数的方法

[0403] 图23为说明按照本发明的一个实施例在一个分组数据的一个数据元素上执行个数计数操作及生成一个结果分组数据的单个结果数据元素的方法。在步骤2310a，从Source1位15、14、13与12生成一个列和CSum 1a及列进位CCarry 1a。在步骤2310b，从Source1位11、10、9与8生成列和CSum1b及列进位CCarry1b。在步骤2310C从Source1位7、6、5与4生成列和CSum1c及列进位CCarry1c。在步骤2310d，从Source1位3、2、1与0生成列和CSum1d及列进位CCarry1d。在本发明的一个实施例中，步骤2310a-d是并行执行的。在步骤2320a，从CSum1a、CCarry 1a、CSum1b与CCarry 1b生成列和CSum2a及列进位CCarry 2b。在步骤2320b，从CSum1c、CCarry1、CSum1d与CCarry1d生成列和CSum2b及列进位CCarry2b。在本发明的一个实施例中，步骤2320a-b是并行执行的。在步骤2330，从CSumm2a、CCarry 2a、CSum2b及CCarry 2b生成列和CSum3及列进位CCarry3。在步骤2340，从CSum3与CCarry3生成Result结果。在一个实施例中，Result是以16位表示的。在这一实施例中，由于只需要位4至位0来表示Source1中置位的位的最大数目，将位15至5设定为0。Source1的最大位数为16。这出现在Source1等于1111111111111111_2时。Result将为16并用0000000000010000_2表示。

[0404] 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。

[0405] 执行个数计数的电路

[0406] 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。

[0407] 图24示出一个电路，其中Source1总线2401通过$Source1_{IN}$2406a-d将信息信号带到popcnt电路2408a-d。从而popcnt电路2408a求出Source1的位16至位0中置位的位的总数，生成Result的位15至位0。popcnt电路2408b求出Source1的位31至位16中置位的位的总数，生成Result的位31至位16。popcnt电路2408c求出Source1的位47至位32中置位的位的总数，生成Result的位47至位32。popcnt电路2408d求出Source1的位63至位48中的置位的位的总数，生成Result的位63至位48。启动2404a-d通过控制2403从操作控制2410接收启动popcnt电路2408a-d执行个数计数操作的控制信号，及将Result放置在Result总线2409上。给予了上面描述及图1-6b及22-25中的描述与说明，熟悉本技术的人员将能建立这一电路。

[0408] 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。

[0409] 在一个数据元素上执行个数

[0410] 计数的电路

[0411] 图25示出在分组数据的一个字数据元素上执行个数计数操作的详细电路。具体地，图25示出popcnt电路2408a的一部分。为了达到采用个数计数操作的应用的最大性能，应在一个时钟周期内完成这个操作。因此，假定存取寄存器及存储结果需要时钟周期的一定百分比，图24的电路在一个时钟周期大约80％的时间内完成其操作。这一电路具有允许处理器109在一个时钟周期中在四个16位数据元素上执行个数计数操作的优点。

[0412] popcnt电路2408a采用4->2进位保留加法器(除非另有指定，CSA将指4->2进位保留加法器)，popcnt电路2408a-d中可能采用的4->2进位保留加法器是本技术中众所周知的。4->2进位保留加法器为将4个操作数相加得出两个和的加法器。由于popcnt电路2408a中的个数计数操作包含16位，第一级包含4个4->2进位保留加法器。这四个4->2进位保留加法器将16个一位操作数变换成8个2位和。第二级将8个2位和变换成4个3位和，而第三级将4个3位和变换成两个4位和。然后一个4位全加法器将两个四位和相加生成最终结果。

[0413] 虽然采用了4->2进位保留加法器，替代实施例中可采用3->2进位保留加法器。另外，也可使用若干个全加法器；然而，这种配置不能象图25中所示的实施例那样快地提供结果。

[0414] $Source1_{IN15-0}$2406a携带Source1的位15至位0。第一个四位耦合在4->2进位保留加法器(CSA 2510a)的输入上。下面的四位耦合在CSA 2510b的输入上。再下面的四位耦合在CSA 2510c的输入上。最后四位耦合在CSA 2510d的输入上。各CSA 2510a-d生成两个2位输出。将CSA 2510a的两个2位输出耦合到CSA 2520a的两个输入上。将CSA 2510b的两个2位输出耦合到CSA 2520a的其它两个输入上。将CSA 2510c的两个2位输出耦合到CSA 2520b的两个输入上。将CSA 2510d的两个2位输出耦合到CSA 2520b其余两个输入上。各CSA 2520a-b生成两个3位输出。将2520a的两个3位输出耦合到CSA2530的两个输入上。将2520b的两个3位输出耦合到CSA 2530的其余两个输入上。CSA 2530生成两个4位输出。

[0415] 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。

[0416] 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。

[0417] 在指令集中加入上述个数计数操作的优点

[0418] 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。

[0419] 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。

[0420] 逻辑运算

[0421] 逻辑运算

[0422] 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”。

[0423] 在本发明的一个实施例中，支持下述逻辑运算：逻辑“与”、逻辑“与非”(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“与非”(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。

[0424] 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。

[0425] 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”、“与非”、“或”或“异或”)的操作码；寄存器209中的SRC1602、SRC2603及DEST604地址。

[0426] 在步骤2602，解码器202通过内部总线170存取寄存器文件150中给出SRC1602及SRC2603地址的寄存器209。寄存器209向执行单元130提供存储在SRC1602寄存器中的分组数据(Source1)及存储在SRC2603寄存器中的分组数据(Source2)。即，寄存器209通过内部总线170将分组数据传递给执行单元130。

[0427] 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。

[0428] 在步骤2610，分组逻辑运算中所选择的一种确定下面执行哪一步骤。如果选择了逻辑“与”运算，执行单元130执行步骤2612；如果选择了逻辑“与非”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。

[0429] 假定选择了逻辑“与”运算，便执行步骤2612。在步骤2612中，Source1位63至0和Source2位63至0进行“与”运算生成Result位63至0。

[0430] 假定选择了逻辑“与非”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“与非”运算生成Result位63至0。

[0431] 假定选择了逻辑“或”运算，便执行步骤2614。在步骤2614中，Source1位63至0和Source2位63至0进行“或”运算生成Result位63至0。

[0432] 假定选择了逻辑“异或”运算，便执行步骤2615。在步骤2615中，Source1位63至0和Source2位63至0进行“异或”运算生成Result位63至0。

[0433] 在步骤2620，将Result存储在DEST寄存器中。

[0434] 表36说明分组数据上的逻辑“与非”运算的寄存器表示。第一行的位是Source1的分组数据表示。第二行的位是Source2的分组数据表示。第三行的位是Result的分组数据表示。各数据元素位下方的数字为数据元素号。例如，Source1数据元素2为1111111100000000_2。

[0435]

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

[0436] 表36

[0437] 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。

[0438] 分组数据逻辑电路

[0439] 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。

[0440] 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”、“与非”、“或”及“异或”运算中选择的一种。

[0441] 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。

[0442] 在指令集中加入上述逻辑运算的优点

[0443] 上述逻辑指令执行逻辑“与”、逻辑“与非”、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。

[0444] 分组比较

[0445] 分组比较操作

[0446] 在本发明的一个实施例中，SRC1602寄存器中包含要比较的数据(Source1)，SRC2603寄存器中包含要相对于它进行比较的数据(Source2)，而DEST 605寄存器中将包含比较的结果(Result)。即，用Source2的各数据元素按指定的关系独立地与Source1中各数据元素比较。

[0447] 在本发明的一个实施例中，支持下述比较关系：等于；带符号的大于；带符号的大于或等于；无符号大于；或无符号大于或等于。在每对对应的数据元素中测试这种关系。例如，Source1[7:0]大于Source2[7:0]，结果为Result[7:0]。如果比较结果满足该关系，则在一个实施例中将Result中的对应数据元素设置成全1。如果比较的结果不满足该关系，则将Result中的对应数据元素设置成全0。

[0448] 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。

[0449] 在步骤2801，解码器202解码处理器109接收的控制信号207。从而，解码器202解码出：适当比较操作的操作码；寄存器209中的SRC1602、SRC2603及DEST 605地址；饱和/不饱和(对比较操作没有必要)，带符号/无符号及分组数据中的数据元素的长度。如上所述，SRC1602(或SRC2603)可用作DEST 605。

[0450] 在步骤2802，解码器202通过内部总线170存取寄存器文件150中给定SRC1602与SRC2603地址的寄存器209。寄存器209向执行单元130提供存储在SRC1602寄存器中的分组数据(Source1)及存储在SRC2603寄存器中的分组数据(Source2)。即，寄存器209通过内部总线170将分组数据传递给执行单元130。

[0451] 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。

[0452] 在步骤2810，数据元素的长度确定下面要执行的步骤。如果数据元素的长度为8位(分组字节401数据)，则执行单元130执行步骤2812。然而，如果分组数据中的数据元素的长度为16位(分组字402数据)，则执行单元130执行步骤2814。在一个实施例中，只支持8位与16位数据元素长度的分组比较。然而，在另一实施例中，也支持32位数据元素长度的分组比较(分组双字403)。

[0453] 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。

[0454] 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。

[0455] 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。

[0456] 在步骤2820，将Result存储在DEST 605寄存器中。

[0457] 表37说明分组比较无符号大于操作的寄存器表示。第一行的位是Source1的分组数据表示。第二行的位是Source2的数据表示。第三行的位是Result的分组数据表示。各数据元素位下面的数字是数据元素号。例如，Source1数据元素3为10000000_2。

[0458]

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

[0459] 表37

[0460] 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。

[0461]

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

[0462] 表38

[0463] 分组数据比较电路

[0464] 在一个实施例中，在和非分组数据上的单个比较操作相同数目的时钟周期中能在多个数据元素上产生比较操作。为了达到相同数目的时钟周期中的执行，采用了并行性。即，同时指示寄存器在数据元素上执行比较操作。下面更详细地讨论这一点。

[0465] 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i2999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。

[0466] 比较单元i 2911及比较单元i+12971各允许来自Source1的8位与来自Source2的对应的8位进行比较。在一个实施例中，各比较单元象已知的8位比较单元一样操作。这一已知的8位比较电路包含允许从Source1减去Source2的字节片电路。处理减法的结果来确定比较操作的结果。在一个实施例中，减法结果包含一个溢出信息。测试这一溢出信息来判定比较操作的结果是否为真。

[0467] 各比较单元具有一个Source1输入、一个Source2输入、一个控制输入、一个下一级信号、一个上一级信号及一个结果输出。因此，比较单元i 2911具有$Source1_i$2931输入、$Source2_i$2933输入、控制i2901输入、下一级i 2913信号、上一级i 2912输入及存储在结果寄存器i 2951中的结果。因此，比较单元i+12971具有$Source1_{i+1}$2932输入、$Source2_{i+1}$ 2934输入、控制i+12902输入、下一级i+12973信号、上一级i+12972输入及存储在结果寄存器i+

12952中的结果。

[0468] Source1n输入通常是Source1的一个8位部分。8位表示最小类型的数据元素，一个分组字节401的数据元素。Source2输入是Source2的对应8位部分。操作控制2900传输控制信号来启动各比较单元执行所要求的比较。控制信号是从比较的关系(诸如带符号大于)及数据元素的长度(诸如字节或字)确定的。下一级信号是从该比较单元的位控制接收的。当使用大于字节长度的数据元素时，位控制单元有效地组合比较单元。例如，当比较字分组数据时，第一比较单元与第二比较单元之间的位控制单元将使得这两个比较单元作为一个16位比较单元工作。类似地，第三与第四比较单元之间的控制单元将使得这两个比较单元作为一个比较单元工作。这可以继续到四个分组字数据元素。

[0469] 取决于所要求的关系及Source1与Source2的值，比较单元通过允许较高阶比较单元的结果向下传播到较低阶比较单元或反过来以执行比较。这便是，各比较单元将利用位控制i 2920传递的信息来提供比较结果。如果使用双字分组数据，则四个比较单元一起工作以形成用于各数据元素的一个32位长的比较单元。各比较单元的结果输出表示该比较单元在其上操作的Source1与Source2的部分上的比较操作的结果。

[0470] 位控制i 2920是从操作控制2900通过分组数据使能i 2906启动的。位控制i 2920控制下一级i 2913与上一级i+12972。例如，假定比较单元i 2911负责Source1与Source2的8个最低位，而比较单元i+12971负责Source1与Source2的下一8位。如果在分组字节数据上执行比较，位控制i 2920将不允许来自比较单元i+12971的结果信息传递到比较单元i 2911，反之亦然。然而，如果在分组字上执行比较，则位控制i 2920将允许来自比较单元i 2911的结果(在一个实施例中为溢出)信息传递到比较单元i+1，以及来自比较单元i+12971的结果(在一个实施例中为溢出)信息传递给比较单元i 2911。

[0471] 例如，在表39中，执行分组字节带符号大于比较。假定比较单元i+12971在数据元素1上操作，而比较单元i 2911在数据元素0上操作。比较单元i+12971比较一个字的最高8位并通过上一级i+12972传递该结果信息。比较单元i 2911比较该字的最低8位并通过下一级i 2913传递该结果信息。然而操作控制2900将使得位控制i 2920停止在比较单元之间传播从上一级i+12972及下一级i 2913接收的结果信息。

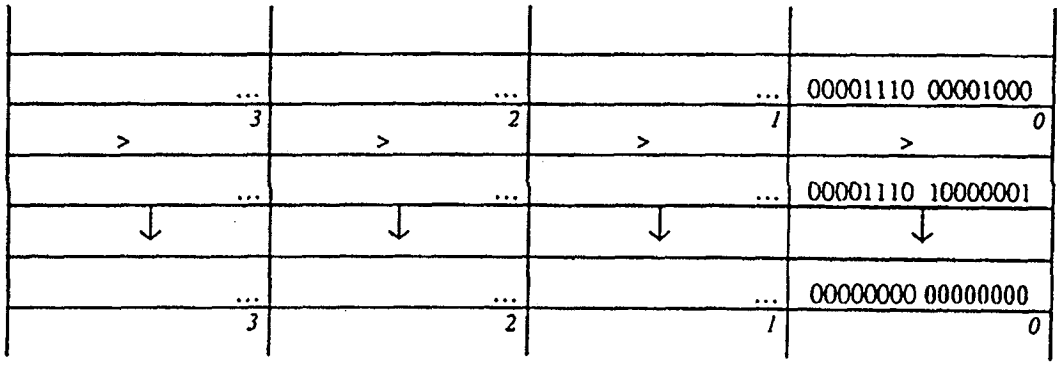
[0472]

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

[0473] 表39

[0474] 然而，如果执行分组字带符号大于比较，则比较单元i+12971的结果将传递到比较单元i 2911，反之亦然。表40示出这一结果。这在类型的传递对分组双字比较同样允许。

[0475]



[0476] 表40

[0477] 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。

[0478] 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。

[0479] 在指令集中加入上述分组

[0480] 比较操作的优点

[0481] 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(ifY>A then X=X+B;else X=X)，如下面表41中所示(表41中所示的值为以16进制符号示出的)。

[0482] Compare.Greater_Than Source1,Source2

[0483]

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

[0484] Packed AND Source3,Mask

[0485]

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

[0486] Packed Add Source4,Result

[0487]

00000010	00000020	Source4=X0-1
>	>	
00000005	00000000	结果
=		
00000015	00000020	新 X0-1值

[0488] 表41

[0489] 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。

[0490] 多媒体算法示例

[0491] 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。

[0492] 1)复数乘法

[0493] 所公开的乘-加指令能用来在单一的指令中将两个复数相乘，如表42a中所示。两个复数(诸如，r1i1与r2i2)的乘法是按照下列等式执行的：

[0494] 实部＝r1 • r2-i1 • i2

[0495] 虚部＝r1 • i2+r2 • i1

[0496] 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。

[0497] Multiply-Add Source1,Source2

[0498]

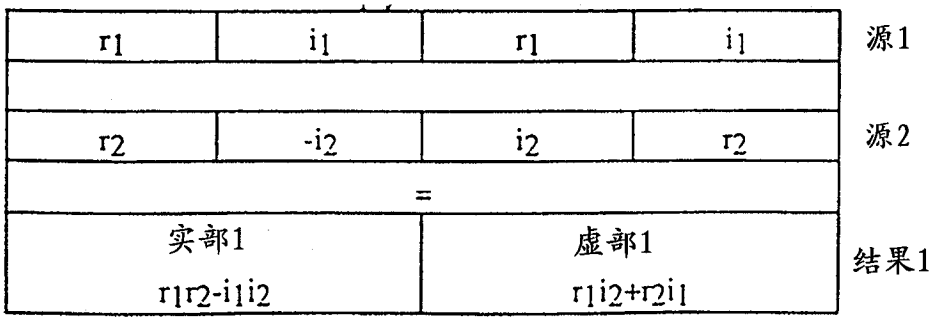
r_1	i_2	r_1	i_1	源1
r_2	$-i_2$	i_2	r_2	源2
=				
实部 $r_1r_2-i_1i_2$		虚部 $r_1i_2+r_2i_1$		结果1

[0499] 表42a

[0500] 作为另一示例，表42b示出用来将三个复数一起乘的指令。

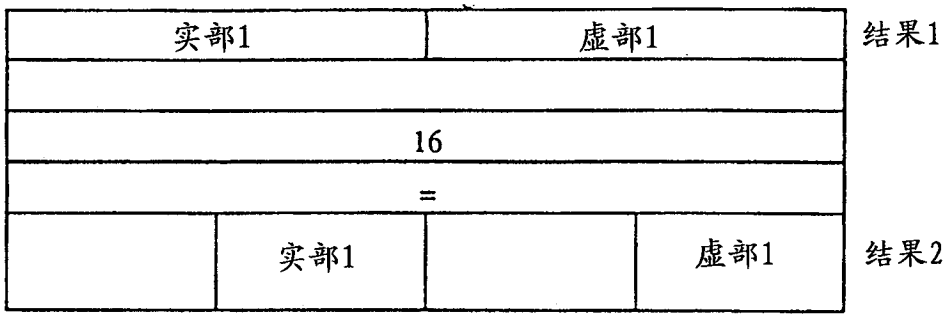
[0501] Multiply-Add Source1,Source2

[0502]



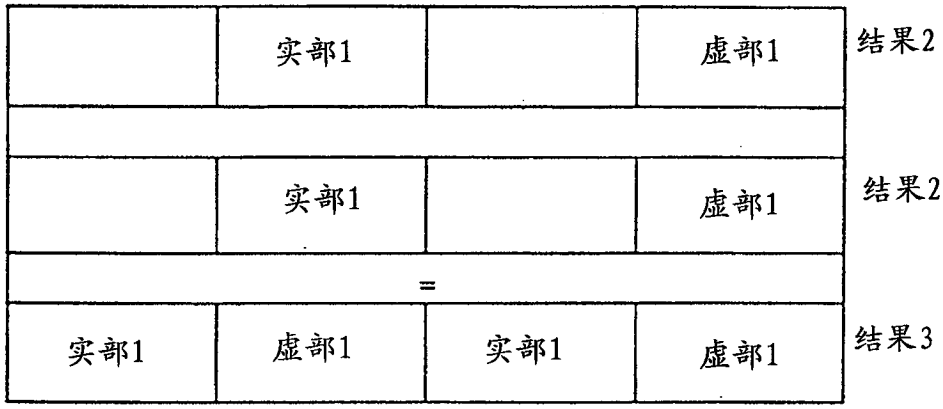
[0503] Packed Shift Right Source1,Source2

[0504]



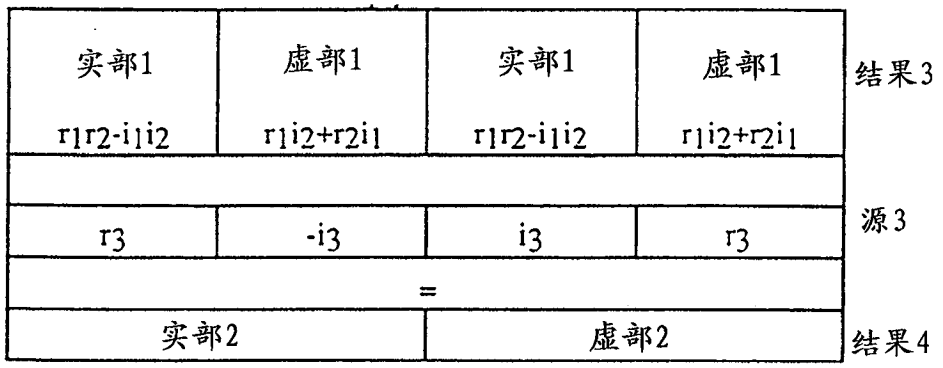
[0505] Pack Result2,Result2

[0506]



[0507] Multiply-Add Result3,Source3

[0508]



[0509] 表42b

[0510] 2)乘累加运算

[0511] 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。

[0512] Multiply-Add Source1,Source2

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

[0513]

[0514] Multiply-Add Source3,Source4

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

[0515]

[0516] Unpacked Add Result1,Result2

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

[0517]

[0518] 表43

[0519] 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。

[0520] Multiply-Add Source 1,Source2

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

[0521]

[0522] Multiply-Add Source3,Source4

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

[0523]

[0524] Packed Add Result1,Result2

[0525]

$A_1B_1+A_2B_2$	$A_3B_3+A_4B_4$
$A_5B_5+A_6B_6$	$A_7B_7+A_8B_8$
=	
$A_1B_1+A_2B_2+A_5B_5+A_6B_6$	$A_3B_3+A_4B_4+A_7B_7+A_8B_8$

结果1
结果2
结果3

[0526] Unpack High Result3,Source5

[0527]

$A_1B_1+A_2B_2+A_5B_5+A_6B_6$	$A_3B_3+A_4B_4+A_7B_7+A_8B_8$
0	0
=	
0	$A_1B_1+A_2B_2+A_5B_5+A_6B_6$

结果3
源5
结果4

[0528] Untpack Low Result3,Source5

[0529]

$A_1B_1+A_2B_2+A_5B_5+A_6B_6$	$A_3B_3+A_4B_4+A_7B_7+A_8B_8$
0	0
=	
0	$A_3B_3+A_4B_4+A_7B_7+A_8B_8$

结果3
源5
结果5

[0530] Packed Add Result4,Result5

[0531]

0	$A_1B_1+A_2B_2+A_5B_5+A_6B_6$
0	$A_3B_3+A_4B_4+A_7B_7+A_8B_8$
=	
0	总计

结果4
结果5
结果6

[0532] 表44

[0533] 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。

[0534] Multiply-Add Source1,Source2

[0535]

A_1	A_2	C_1	C_2
B_1	B_2	D_1	D_2
=			
$A_1B_1+A_2B_2$		$C_1D_1+C_2D_2$	

源1
源2
结果1

[0536] 表45

[0537] 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。

[0538] Multirly-Add Source 1,Source2

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

[0539]

[0540] Multiply-Add Source3,Source4

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

[0541]

[0542] Packed Add Result1,Result2

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

[0543]

[0544] 表46

[0545] 3)点积算法

[0546] 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。

[0547] 两个长度N的序列A与B的点积定义为:

[0548] $$\mathrm{Re}sult=\sum_{i=0}^{N-1}Ai\bullet Bi$$

[0549] 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。

[0550] 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:

[0551] 1)使用传送指令从A序列取4个16位值来生成Source1;

[0552] 2)使用传送指令从B序列取4个16位值来生成Source2;以及

[0553] 3)使用乘-加、分组加法及移位指令如上所述相乘与累加。

[0554] 对于带有稍多元素的矢量,使用表46中所示的方法并在最后将最终结果加在一

起。其它支持指令包含用于初始化累加器寄存器的分组OR与XOR指令，用于在计算的最后级上移出不需要的值的分组移位指令。利用已存在处理器109的指令集中的指令完成循环控制操作。

[0555] 4)二维环路滤波器

[0556] 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。

[0557]
$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

[0558] 表47

[0559] 为了计算位置(x,y)上的象素的新值，使用下述等式：

[0560] 结果象素＝(x-1,y-1)+2(x,y-1)+(x+1,y-1)+2(x-1,y)+4(x,y)十2(x+1,y)+(x-1,y+1)+2(x,y-1)+(x+1,y+1)

[0561] 通过包含组装、分组、传送、分组移位及分组加法，本发明允许使用分组数据执行二维环路滤波器。按照上述环路滤波器的一种实现，这一环路滤波器是作为两个简单的一维滤波器应用的-即，上述二维滤波器能用作两个121滤波器。第一滤波器在水平方向上，而第二滤波器则在垂直方向上。

[0562] 表48示出象素数据的一个8×8块的表示。

[0563] ←8→

[0564]

A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7
B_0	B_1	B_2	B_3	B_4	B_5	B_6	B_7
C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7

[0565] 表48

[0566] 执行下述步骤来实现象素数据的这一8×8块上的滤波器的水平通过：

[0567] 1)使用传送指令存取8个8位象素值作为分组数据；

[0568] 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；

[0569] 3)复制Source 1两次生成Source2与Source3；

[0570] 4)在Source1上执行非分组向右移位16位；

[0571] 5)在Source3上执行非分组向左移位16位；

[0572] 6)通过执行下述分组加法生成(Source1+2*Source2+Source3)；

[0573] a)Source1＝Source1+Source2；

[0574] b)Source1＝Source1+Source2；

[0575] c)Source1＝Source1+Source3；

[0576] 7)作为一个8×8中间结果数组的一部分存储得出的分组字数据；以及

[0577] 8)重复这些步骤直到生成如下面表49中所示的整个8×8中间结果数组(如IA_0表示来自表49的A_0的中间结果)。

[0578] ←16→

[0579]

IA_0	IA_1	IA_2	IA_3	IA_4	IA_5	IA_6	IA_7
IB_0	IB_1	IB_2	IB_3	IB_4	IB_5	IB_6	IB_7
IC_0	IC_1	IC_2	IC_3	IC_4	IC_5	IC_6	IC_7
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
II_0	II_1	II_2	II_3	II_4	II_5	II_6	II_7

[0580] 表49

[0581] 执行下述步骤来实现8×8中间结果数组上的滤波器的垂直通过：

[0582] 1)使用传送指令存取来自该中间结果数组的4×4的数据块作为分组数据以生成Source1、Source2及Source3(如见作为示例的表50)；

[0583] ←6→

[0584]

IA_0	IA_1	IA_2	IA_3	源1
IB_0	IB_1	IB_2	IB_3	源2
IC_0	IC_1	IC_2	IC_3	源3

[0585] 表50

[0586] 2)通过执行下述分组加法生成(Source1+2*Source2+Source3)：

[0587] a)Source1＝Source1+Source2；

[0588] b)Source1＝Source1+Source2；

[0589] c)Source1＝Source1+Source3；

[0590] 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；

[0591] 4)组装带有饱和的结果Source1，将16位值转换回8位象素值；

[0592] 5)将得出的分组字节数据作为一个8×8结果数组的一部分存储(对于表50中所示的例子，这四个字节表示B0、B1、B2与B3的新象素值)；以及

[0593] 6)重复这些步骤直到生成整个8×8结果数组为止。

[0594] 值得指出的是,8×8结果数组的顶与底行是用不同的算法确定的,为了不冲淡本发明在这里未描述该算法。

[0595] 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。

[0596] 5)运动估计(Motion Fstimation)

[0597] 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)。

[0598] 通过包含传送、分解、分组加法、带饱和的分组减法及逻辑运算,本发明允许用分组数据执行运动估计。例如,如果两个16×16的视频块是用作为分组数据存储的两个8位象素值的数组表示的,可用下述步骤计算出这两块中的象素值的绝对差值:

[0599] 1)利用传送指令从块A中取8个8位值生成Source1;

[0600] 2)利用传送指令从块B中取8个8位值生成Source2;

[0601] 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);

[0602] 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);

[0603] 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;

[0604] 6)重复这些步骤直到处理完16×16块。

[0605] 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。

[0606] 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。

[0607] 6)离散余弦变换

[0608] 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。

[0609] 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。

[0610] 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：

[0611] X＝a*x+b*y

[0612] Y＝c*x-d*y

[0613] 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。

[0614] 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：

[0615] 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)；

[0616] 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

[0617] 3)利用Source1与Source2执行乘-加指令生成Result(见下面表51)。

[0618]

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

[0619] 表51

[0620] 在一些情况中，蝶形运算的系数为1。对于这些情况，蝶形运算退化成只有加与减，这可以利用分组加法与分组减法指令来执行。

[0621] IEEE文件规定了电视会议必须执行的逆DCT所用的精度。(见IEEE电路与系统协会，“8×8逆离散余弦变换的实现的IEEE标准规范”，IEEE Std.1180-1990，IEEE Inc.345East 47th st.,NY,NY 10017,USA,1991年3月18日)。公开的乘-加指令满足这一要求的精度因为它使用16位输入来生成32位输出。

[0622] 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。

[0623] 替代实施例

[0624] 虽然已将本发明描述成其中各个不同的运算具有独立的电路，但也能实现替代的实施例使不同运算共同某些电路。例如，在一个实施例中使用下列电路：1)单个算术逻辑单元(ALU)来执行分组加法、分组减法、分组比较及分组逻辑运算；2)一个电路单元来执行组装、分解及分组移位操作；3)一个电路单元来执行分组乘法及乘-加运算；以及4)一个电路单元来执行个数计数操作。

[0625] 这里使用了对应与相应的名词来指称存储在两个或更多分组数据中的数据元素之间的预定关系。在一个实施例中，这一关系是基于分组数据中的数据元素的位位置的。例如，第一分组数据的数据元素0(例如以分组字节格式存储在位位置0-7中)对应于第二分组

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

[0626] 虽已用若干实施例描述了本发明,熟悉本技术者将理解本发明不限于所述实施例。可在所附权利要求书的精神与范围内修改或改变来实现本发明的方法与装置。因此,该说明书应认为是示例性的而不是对本发明的限制。

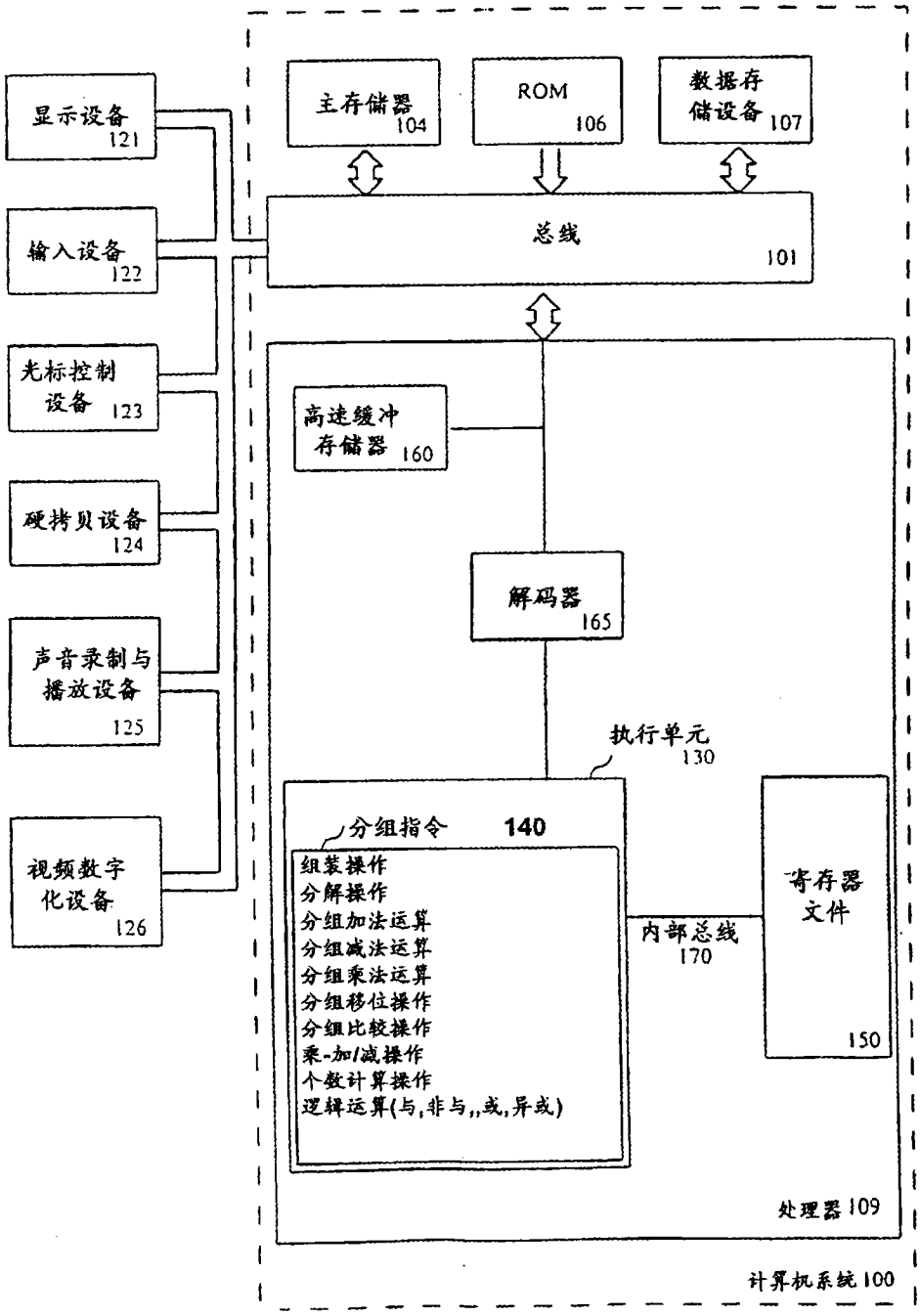

显示设备 121
输入设备 122
光标控制设备 123
硬拷贝设备 124
声音录制与播放设备 125
视频数字化设备 126
主存储器 104
ROM 106
数据存储设备 107
总线 101
高速缓冲存储器 160
解码器 165
执行单元 130
分组指令 140
组装操作
分解操作
分组加法运算
分组减法运算
分组乘法运算
分组移位操作
分组比较操作
乘-加/减操作
个数计算操作
逻辑运算(与,非与,,或,异或)
内部总线 170
寄存器文件 150
处理器 109
计算机系统 100

图1

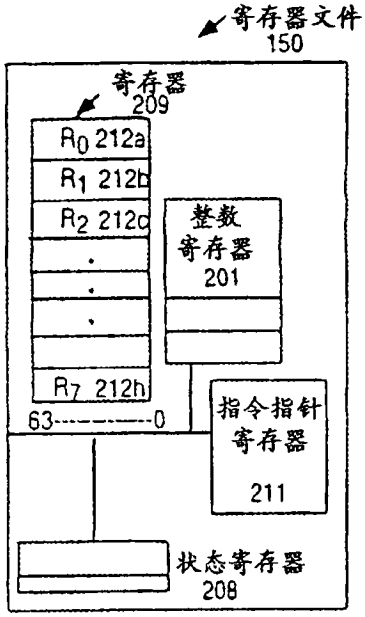

寄存器文件
150
寄存器
209
R0 212a
R1 212b
R2 212c
R7 212h
整数
寄存器
201
63----------0
指令指针
寄存器
211
状态寄存器
208

图2

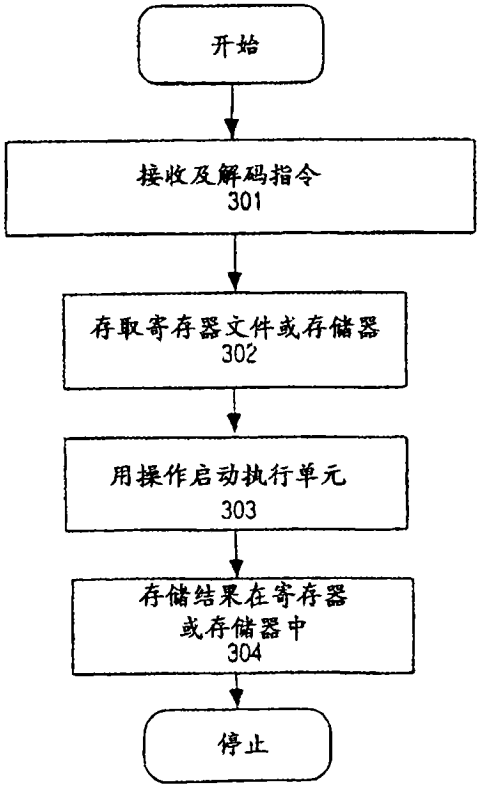

开始
接收及解码指令
301
存取寄存器文件或存储器
302
用操作启动执行单元
303
存储结果在寄存器
或存储器中
304
停止

图3

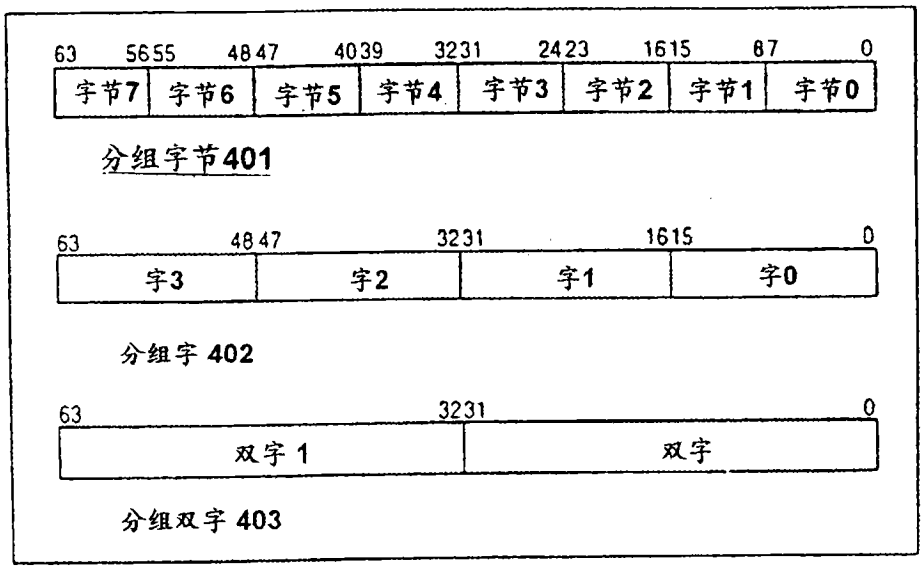

63 5655 4847 4039 3231 2423 1615 87 0
字节7 字节6 字节5 字节4 字节3 字节2 字节1 字节0
分组字节401
63 4847 3231 1615 0
字3 字2 字1 字0
分组字 402
63 3231 0
双字 1 双字
分组双字 403

图4

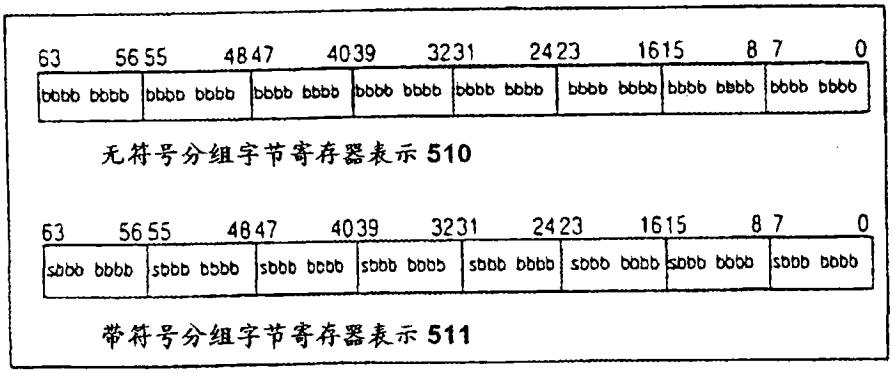

63 56 55 48 47 40 39 32 31 24 23 16 15 8 7 0
bbbb bbbb bbbb bbbb bbbb bbbb bbbb bbbb bbbb bbbb bbbb bbbb bbbb bbbb bbbb bbbb
无符号分组字节寄存器表示 510
63 56 55 48 47 40 39 32 31 24 23 16 15 8 7 0
sbbb bbbb sbbb bbbb sbbb bbbb sbbb bbbb sbbb bbbb sbbb bbbb sbbb bbbb sbbb bbbb
带符号分组字节寄存器表示 511

图5a

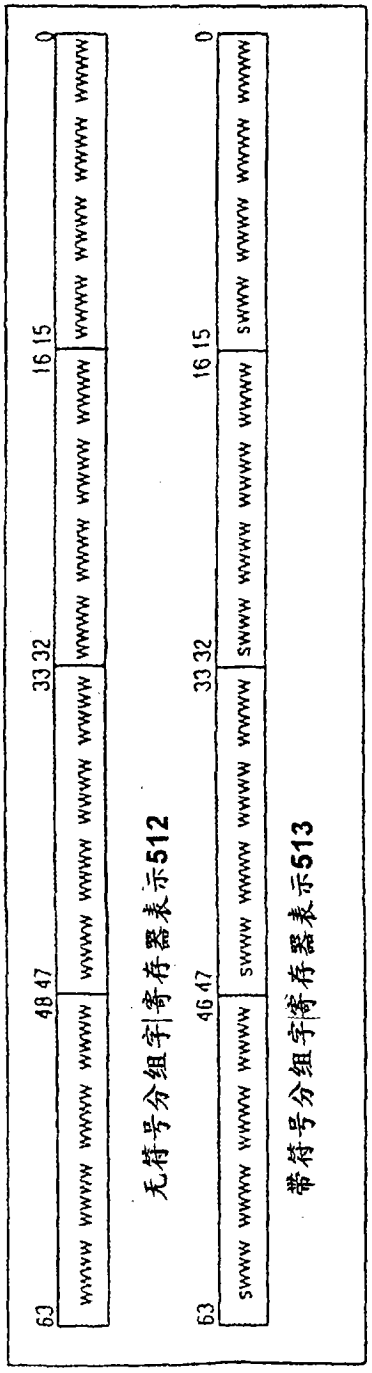

63
48 47
33 32
16 15
0
wwww wwww wwww wwww
wwww wwww wwww wwww
wwww wwww wwww wwww
wwww wwww wwww wwww
无符号分组字寄存器表示512
63
46 47
33 32
16 15
0
swww wwww wwww wwww
swww wwww wwww wwww
swww wwww wwww wwww
swww wwww wwww wwww
带符号分组字寄存器表示513

图5b

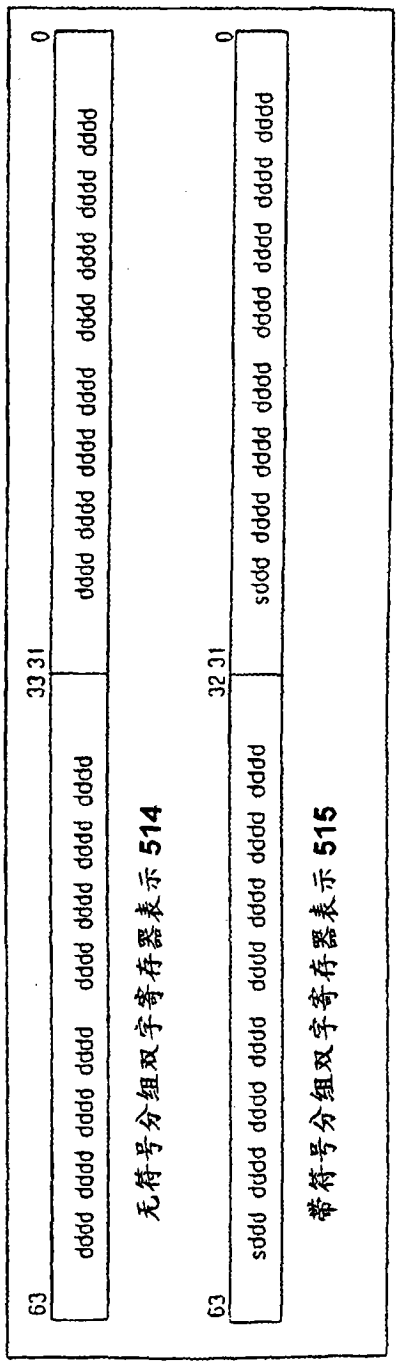

63
33 31
0
dddd dddd dddd dddd
dddd dddd dddd dddd
无符号分组双字寄存器表示 514
63
32 31
0
sddd dddd dddd dddd
sddd dddd dddd dddd
带符号分组双字寄存器表示 515

图5c

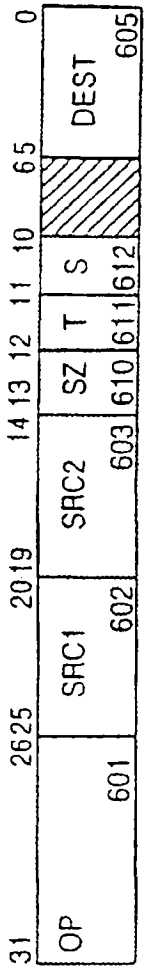

31
26 25
20 19
14 13 12 11 10
6 5
0
OP
601
SRC1
602
SRC2
603
SZ
610
T
611
S
612
DEST
605

图6a

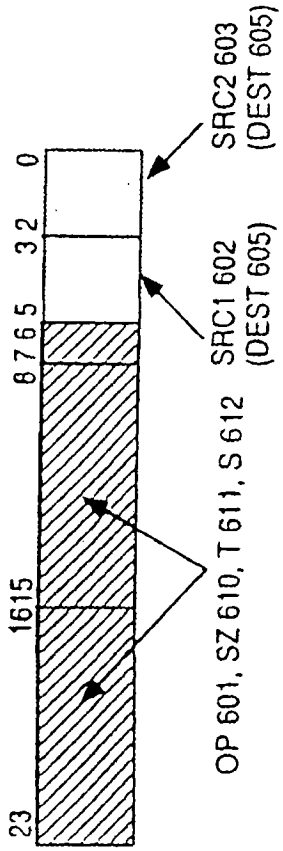

SRC2 603
(DEST 605)
SRC1 602
(DEST 605)
OP 601, SZ 610, T 611, S 612
0
3 2
8 7 6 5
16 15
23

图6b

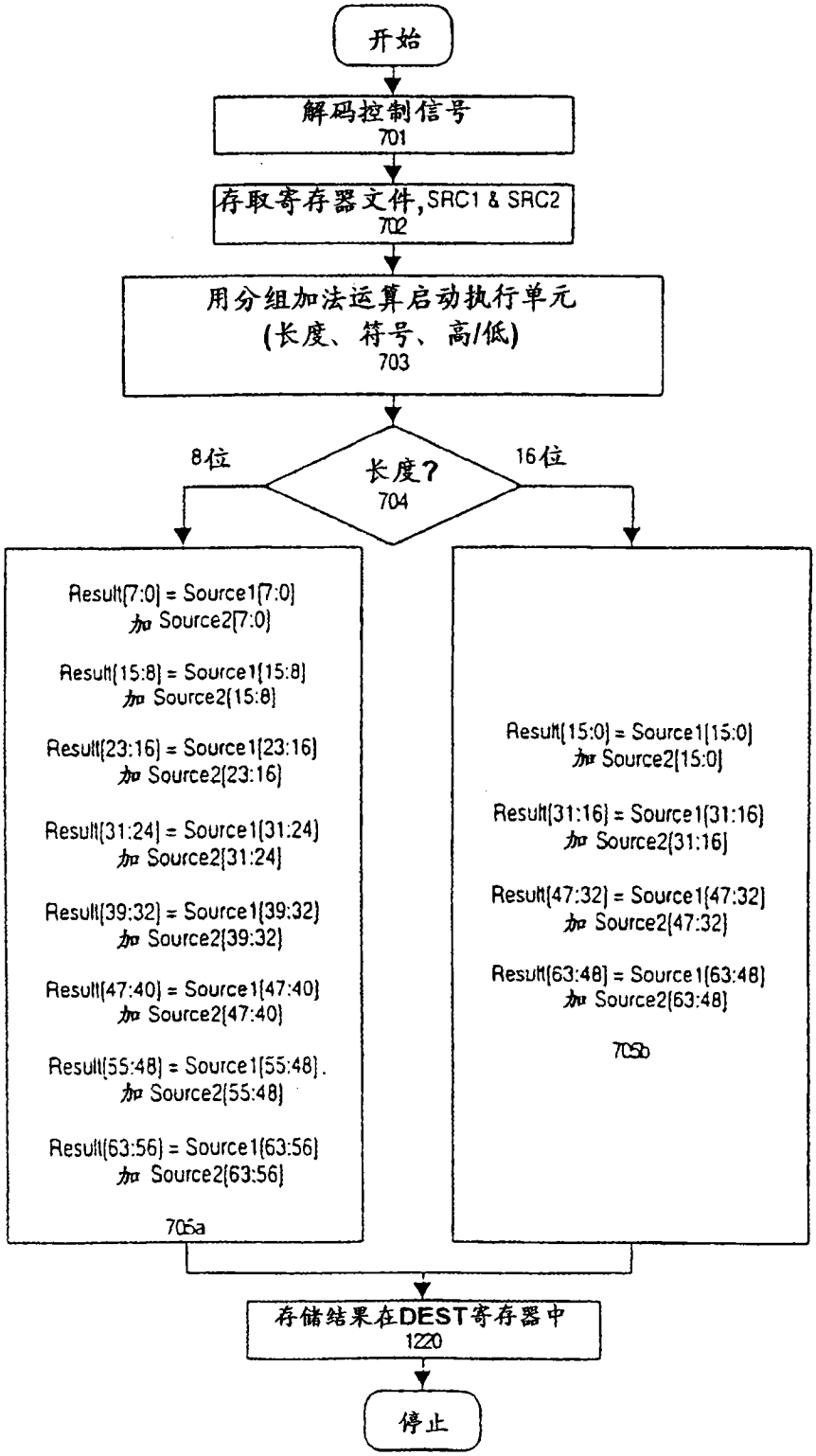

开始
解码控制信号
701
存取寄存器文件,SRC1 & SRC2
702
用分组加法运算启动执行单元
(长度、符号、高/低)
703
长度?
704
8位
16位
Result[7:0] = Source1[7:0]
加 Source2[7:0]
Result[15:8] = Source1[15:8]
加 Source2[15:8]
Result[23:16] = Source1[23:16]
加 Source2[23:16]
Result[31:24] = Source1[31:24]
加 Source2[31:24]
Result[39:32] = Source1[39:32]
加 Source2[39:32]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

图7a

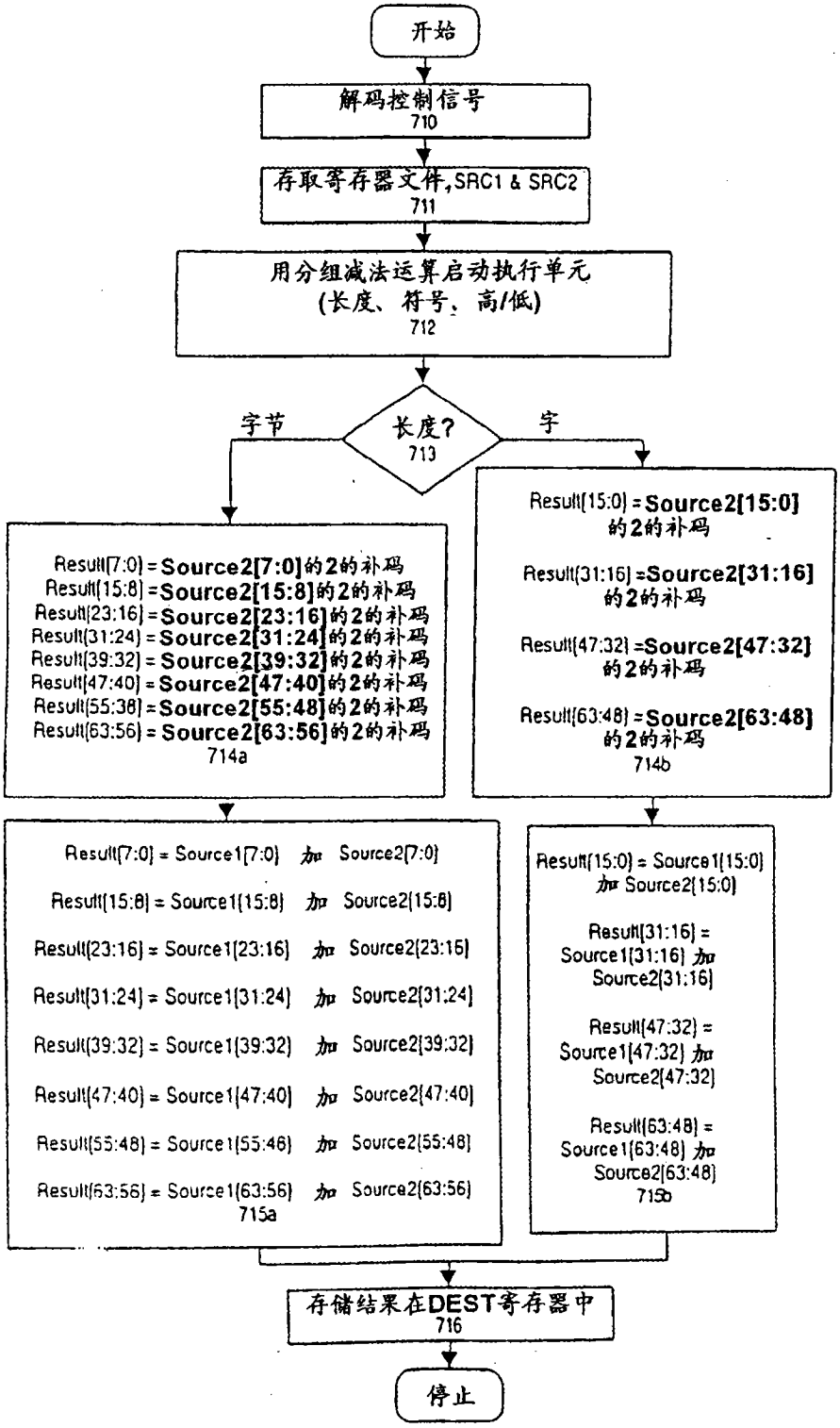

开始
解码控制信号
710
存取寄存器文件,SRC1 & SRC2
711
用分组减法运算启动执行单元
(长度、符号、高/低)
712
长度?
713
字节
字
Result[7:0] = Source2[7:0]的2的补码
Result[15:8] = Source2[15:8]的2的补码
Result[23:16] = Source2[23:16]的2的补码
Result[31:24] = Source2[31:24]的2的补码
Result[39:32] = Source2[39:32]的2的补码
Result[47:40] = Source2[47:40]的2的补码
Result[55:48]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

图7b

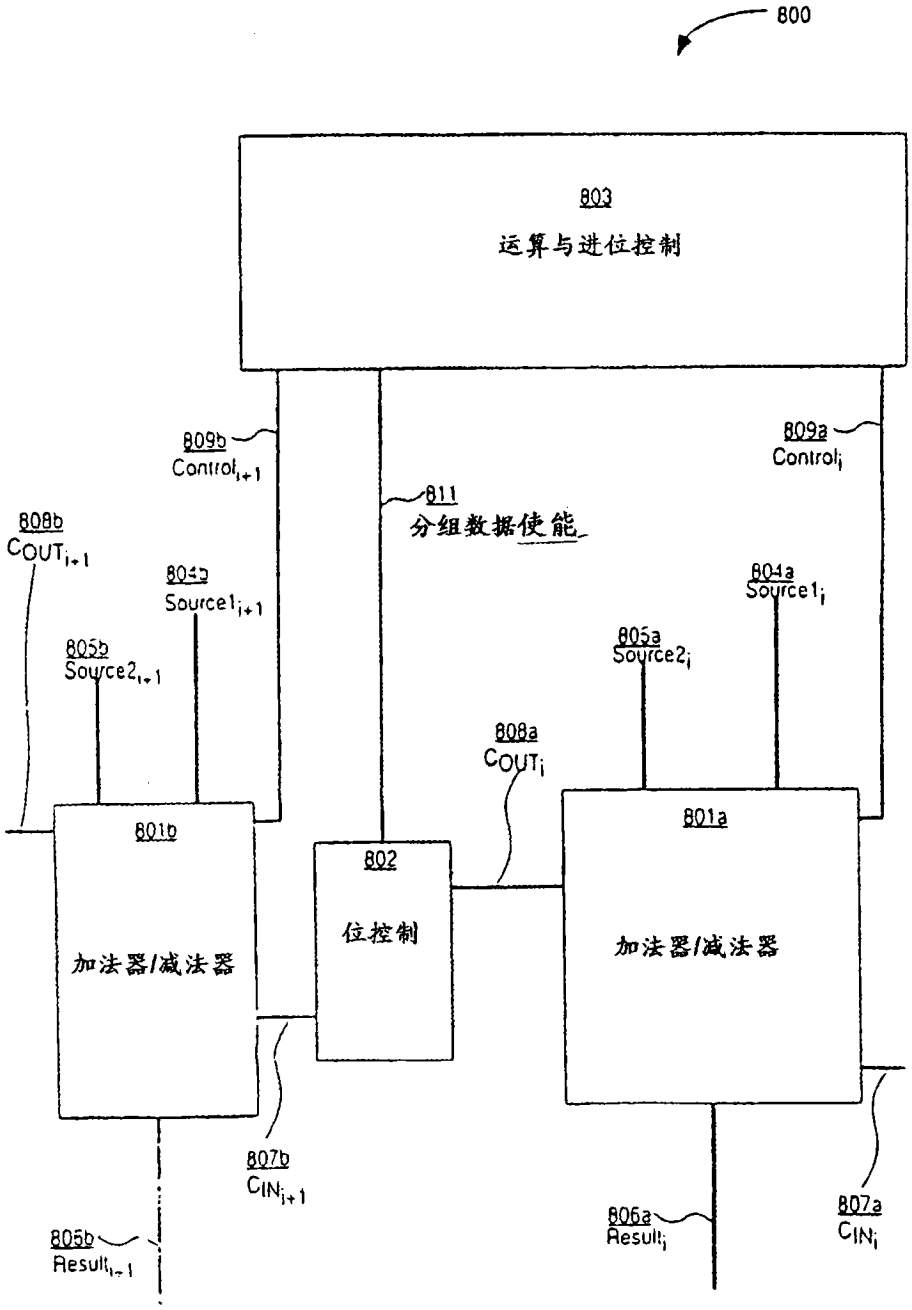

800
803
运算与进位控制
809b
Control$_{i+1}$
809a
Control$_i$
811
分组数据使能
808b
COUT$_{i+1}$
804b
Source1$_{i+1}$
804a
Source1$_i$
805b
Source2$_{i+1}$
805a
Source2$_i$
808a
COUT$_i$
801b
801a
802
位控制
加法器/减法器
加法器/减法器
807b
CIN$_{i+1}$
805b
Result$_{i+1}$
806a
Result$_i$
807a
CIN$_i$

图8

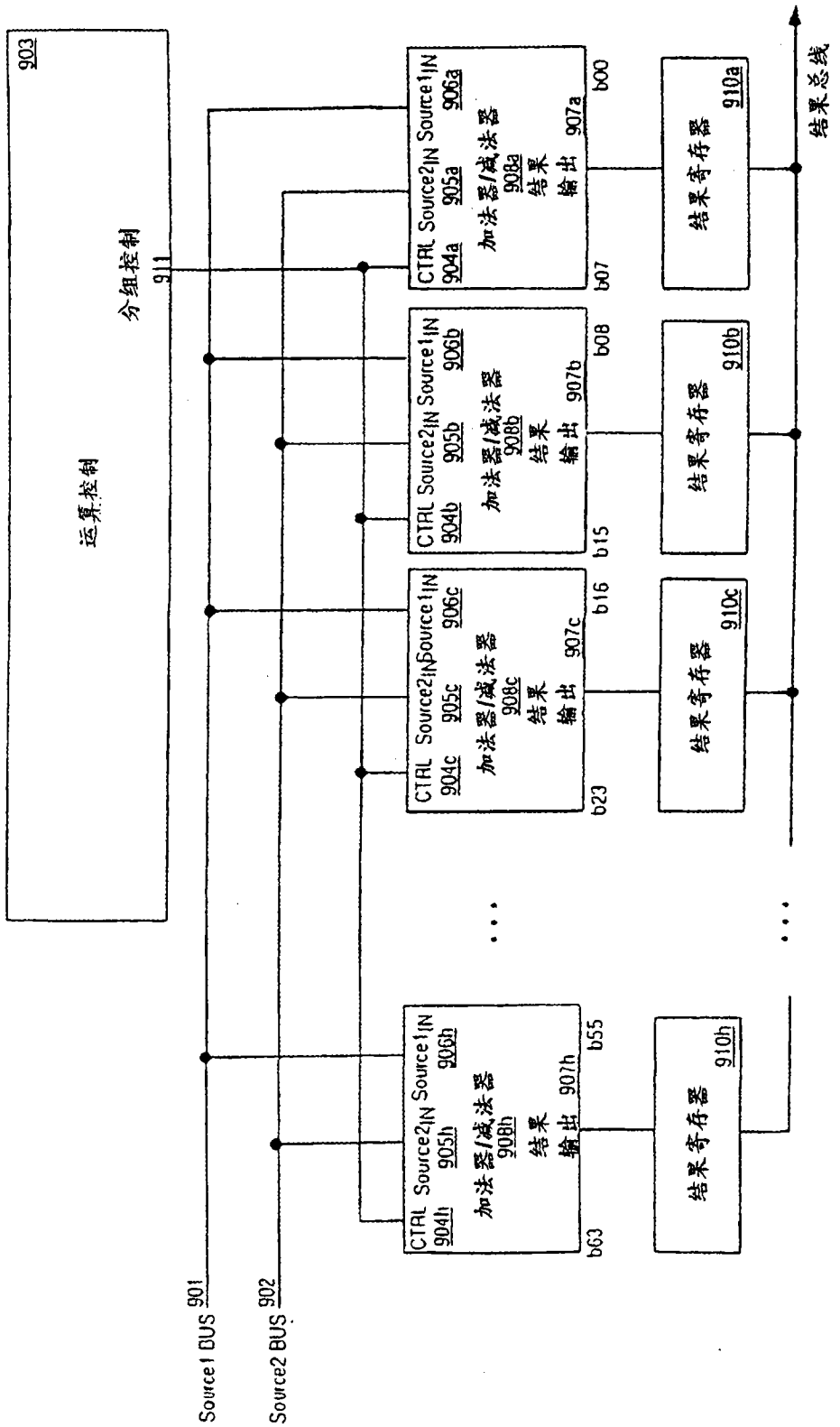

903
运算控制
分组控制
911
CTRL 904a
Source2IN 905a
Source1IN 906a
加法器/减法器 908a
结果输出 907a
b00
b07
结果寄存器 910a
CTRL 904b
Source2IN 905b
Source1IN 906b
加法器/减法器 908b
结果输出 907b
b08
b15
结果寄存器 910b
CTRL 904c
Source2IN 905c
Source1IN 906c
加法器/减法器 908c
结果输出 907c
b16
b23
结果寄存器 910c
CTRL 904h
Source2IN 905h
Source1IN 906h
加法器/减法器 908h
结果输出 907h
b55
b63
结果寄存器 910h
结果总线
Source1 BUS 901
Source2 BUS 902

图9

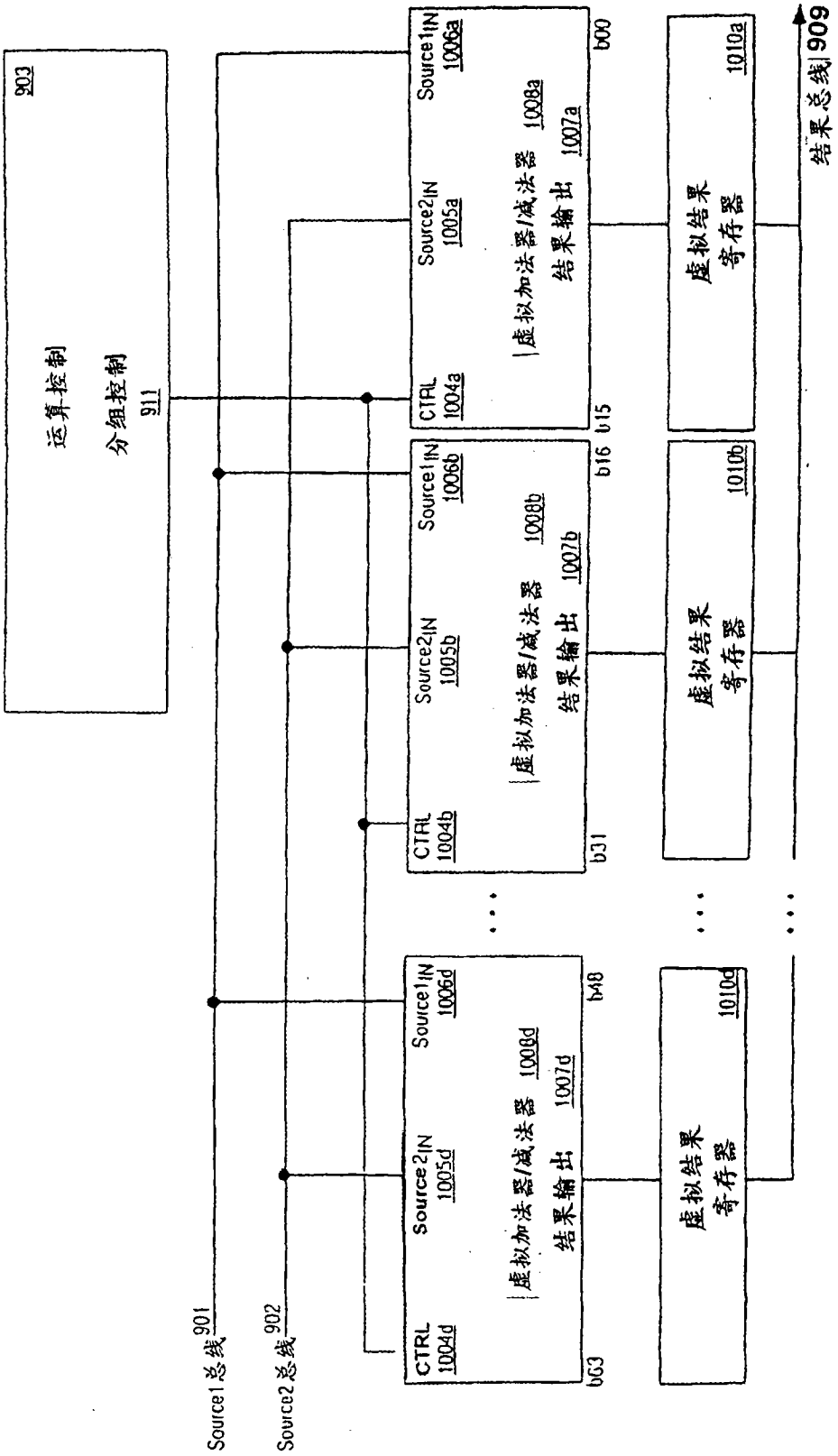

903
运算控制
分组控制
911
Source1总线 901
Source2总线 902
CTRL 1004a
Source2IN 1005a
Source1IN 1006a
虚拟加法器/减法器 1008a
结果输出 1007a
b00
b15
CTRL 1004b
Source2IN 1005b
Source1IN 1006b
虚拟加法器/减法器 1008b
结果输出 1007b
b16
b31
CTRL 1004d
Source2IN 1005d
Source1IN 1006d
虚拟加法器/减法器 1008d
结果输出 1007d
b48
b63
虚拟结果寄存器 1010a
虚拟结果寄存器 1010b
虚拟结果寄存器 1010d
结果总线 909

图10

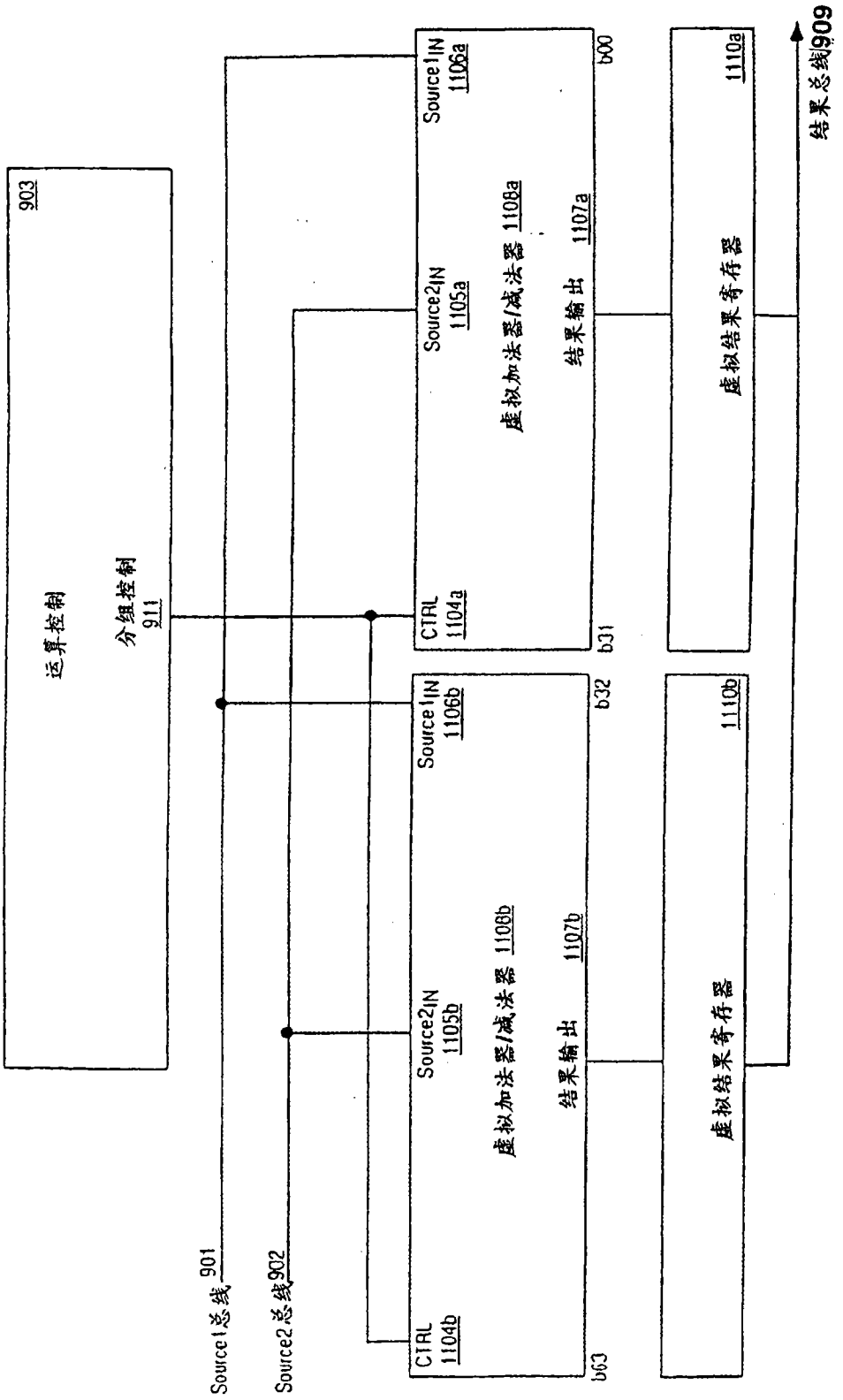

903
运算控制
分组控制
911
Source1总线901
Source2总线902
Source1IN
1106a
Source2IN
1105a
虚拟加法器/减法器 1108a
结果输出 1107a
CTRL
1104a
b00
b31
Source1IN
1106b
Source2IN
1105b
虚拟加法器/减法器 1108b
结果输出 1107b
CTRL
1104b
b32
b63
虚拟结果寄存器 1110a
虚拟结果寄存器 1110b
结果总线909

图11

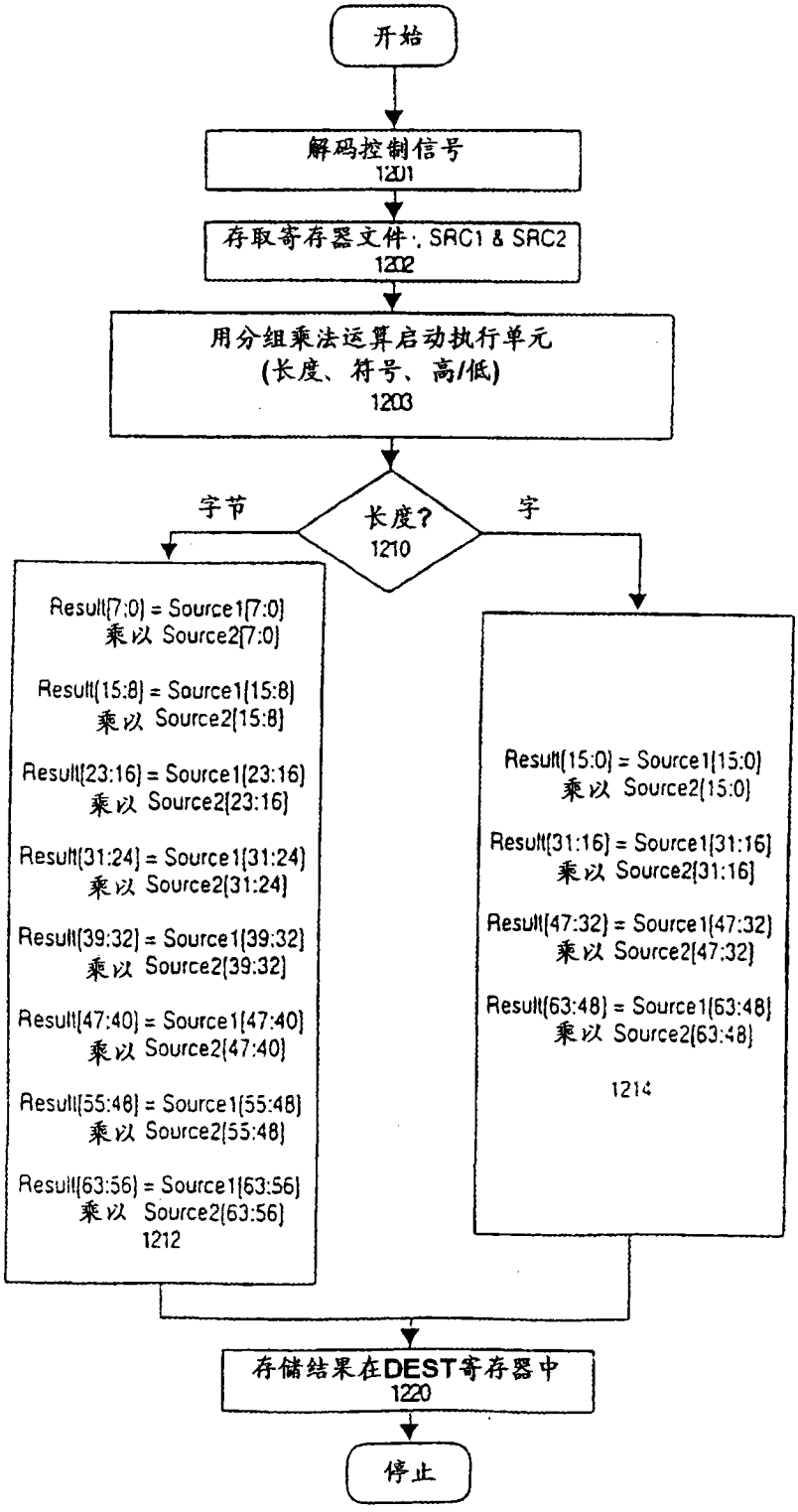

开始
解码控制信号
1201
存取寄存器文件：SRC1 & SRC2
1202
用分组乘法运算启动执行单元
(长度、符号、高/低)
1203
长度?
1210
字节
字
Result[7:0] = Source1[7:0]
乘以 Source2[7:0]
Result[15:8] = Source1[15:8]
乘以 Source2[15:8]
Result[23:16] = Source1[23:16]
乘以 Source2[23:16]
Result[31:24] = Source1[31:24]
乘以 Source2[31:24]
Result[39:32] = Source1[39:32]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

图12

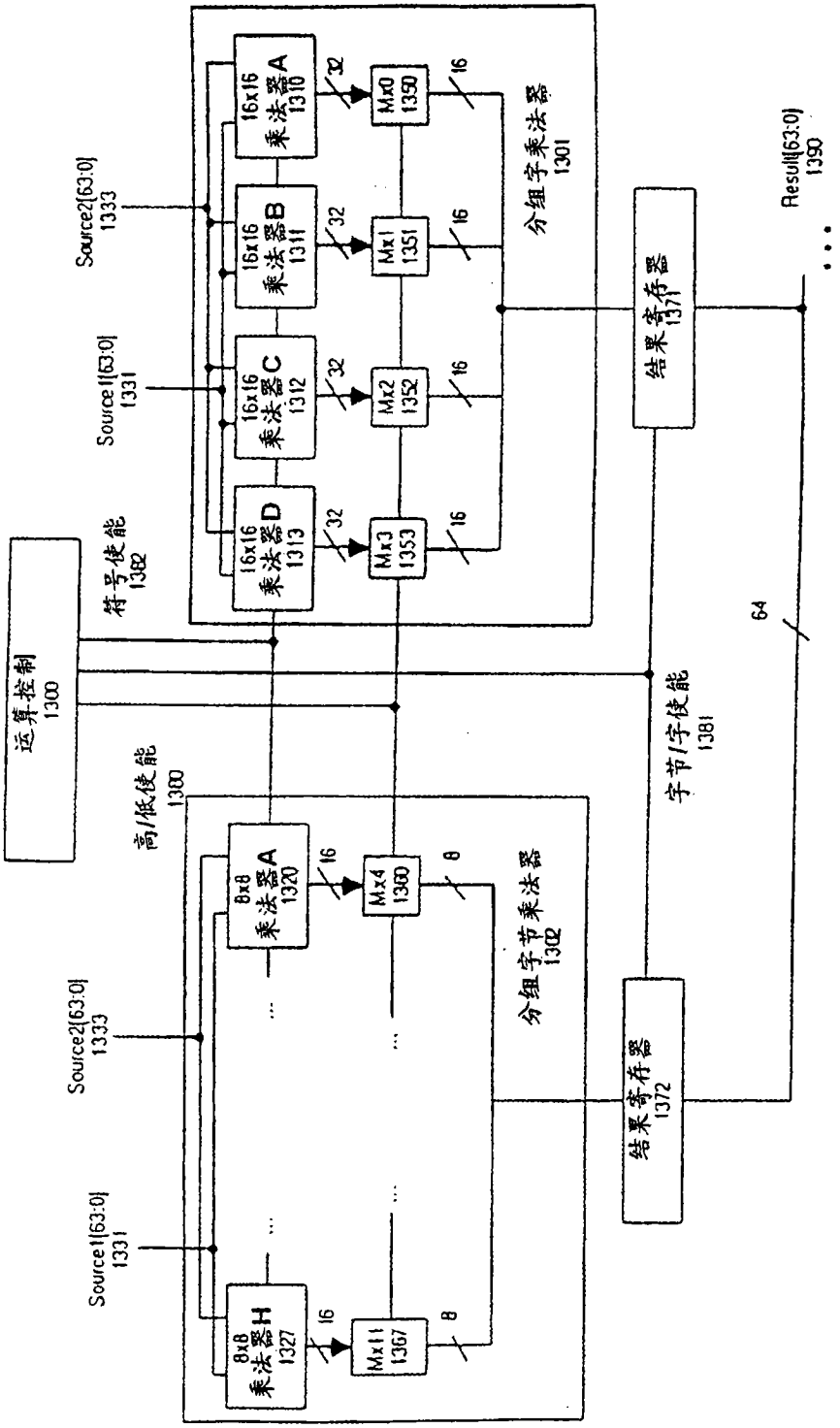

运算控制
1300
符号使能
1382
高/低使能
1300
分组字乘法器
1301
分组字节乘法器
1302
16x16
乘法器A
1310
16x16
乘法器B
1311
16x16
乘法器C
1312
16x16
乘法器D
1313
8x8
乘法器A
1320
8x8
乘法器H
1327
Mx0
1350
Mx1
1351
Mx2
1352
Mx3
1353
Mx4
1360
Mx11
1367
Source2[63:0]
1333
Source1[63:0]
1331
结果寄存器
1371
结果寄存器
1372
字节/字使能
1381
Result[63:0]
1390
64
32
16
8

图13

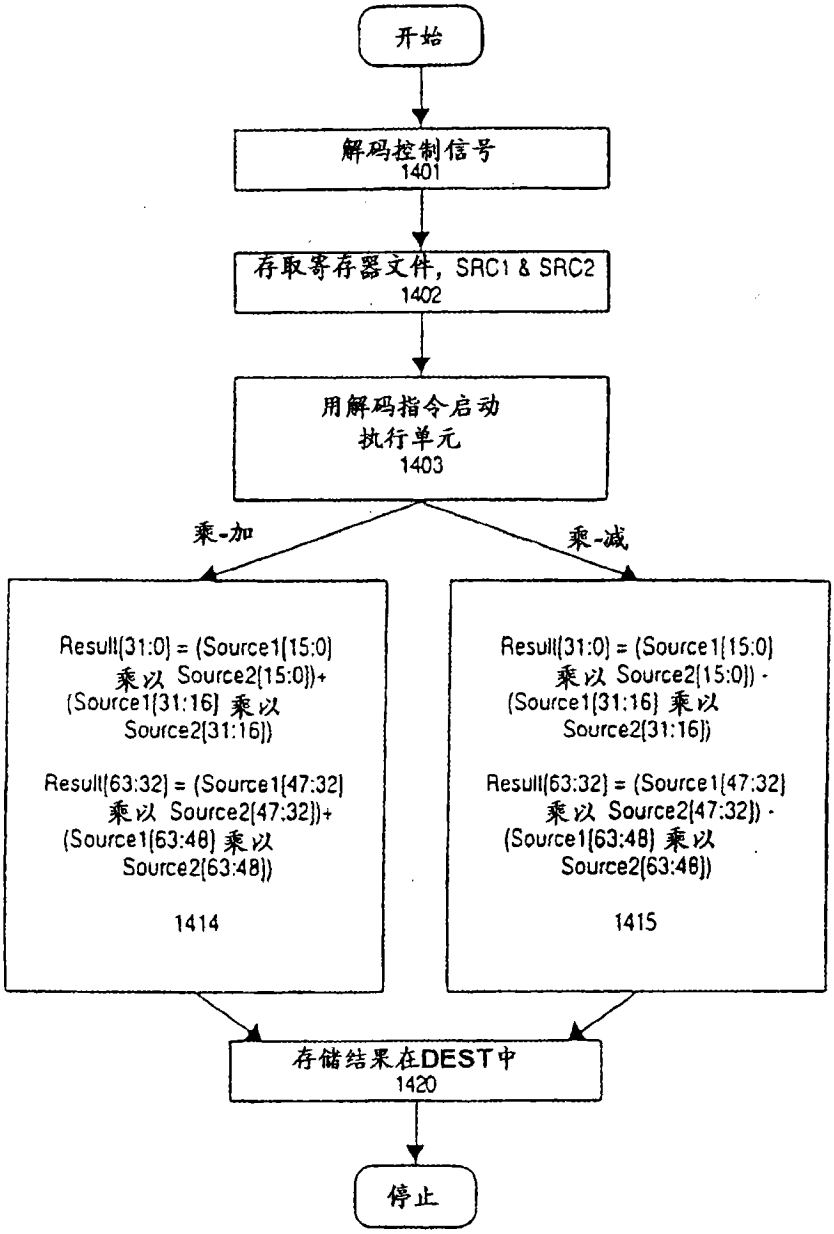

开始
解码控制信号
1401
存取寄存器文件，SRC1 & SRC2
1402
用解码指令启动
执行单元
1403
乘-加
乘-减
Result[31:0] = (Source1[15:0]
乘以 Source2[15:0])+
(Source1[31:16] 乘以
Source2[31:16])
Result[63:32] = (Source1[47:32]
乘以 Source2[47:32])+
(Source1[63:48] 乘以
Source2[63:48])
1414
Result[31:0] = (Source1[15:0]
乘以 Source2[15:0]) -
(Source1[31:16]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

图14

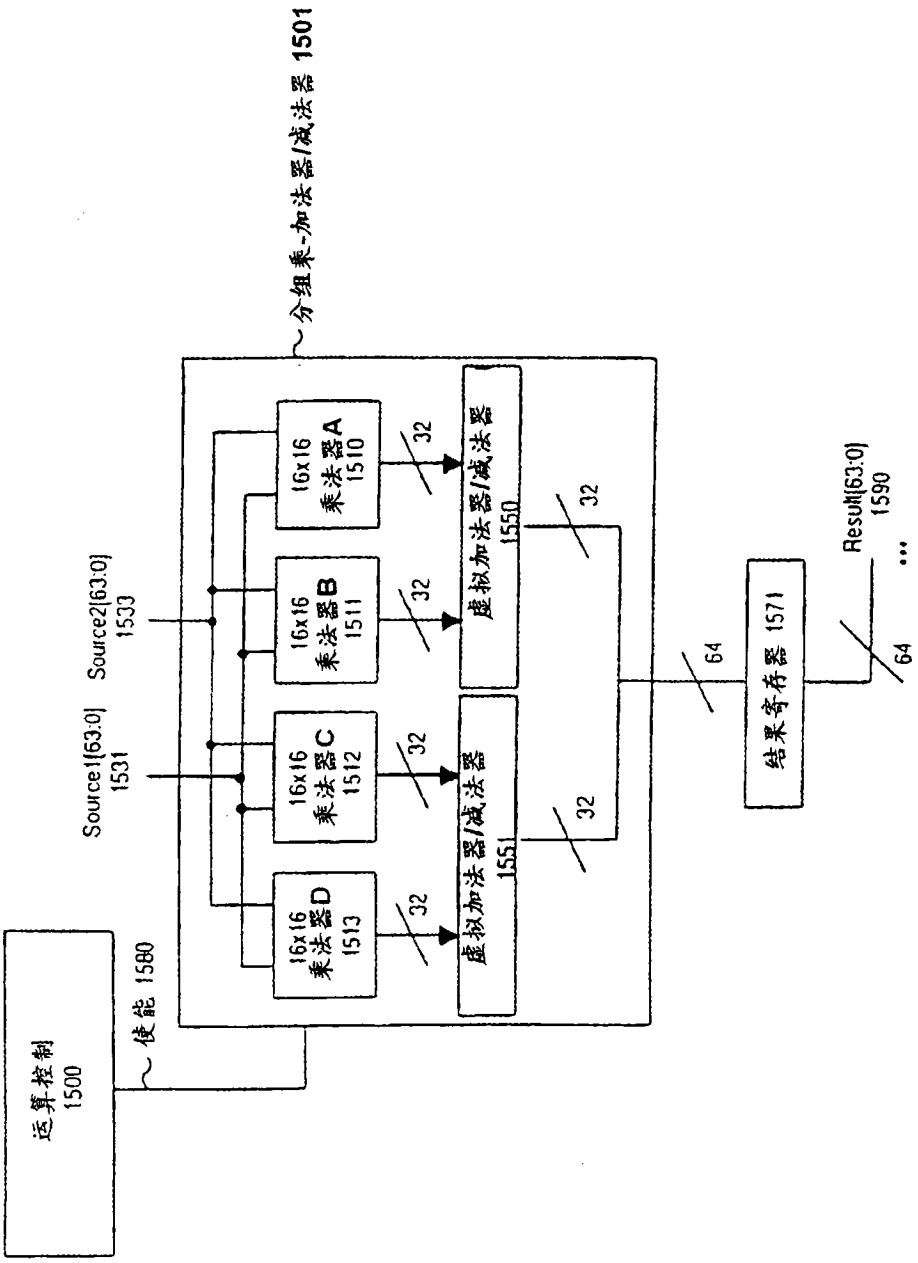

分组乘-加法器/减法器 1501
运算控制 1500
使能 1580
Source1[63:0] 1531
Source2[63:0] 1533
16x16 乘法器A 1510
16x16 乘法器B 1511
16x16 乘法器C 1512
16x16 乘法器D 1513
虚拟加法器/减法器 1550
虚拟加法器/减法器 1551
32
64
结果寄存器 1571
Result[63:0] 1590

图15

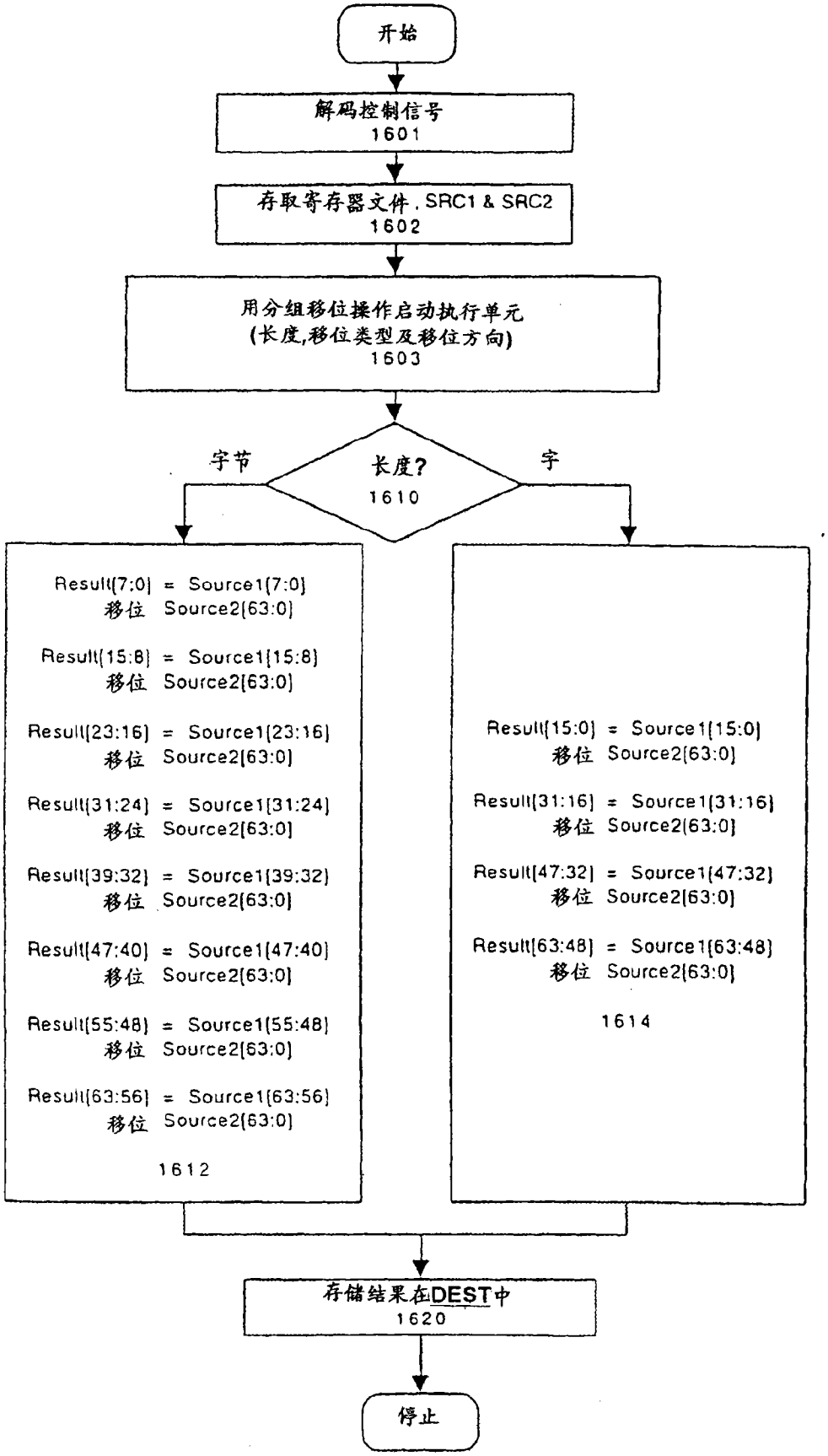

开始
解码控制信号
1601
存取寄存器文件，SRC1 & SRC2
1602
用分组移位操作启动执行单元
(长度,移位类型及移位方向)
1603
长度?
1610
字节
字
Result[7:0] = Source1[7:0]
移位 Source2[63:0]
Result[15:8] = Source1[15:8]
移位 Source2[63:0]
Result[23:16] = Source1[23:16]
移位 Source2[63:0]
Result[31:24] = Source1[31:24]
移位 Source2[63:0]
Result[39:32] = Source1[39:32]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

图16

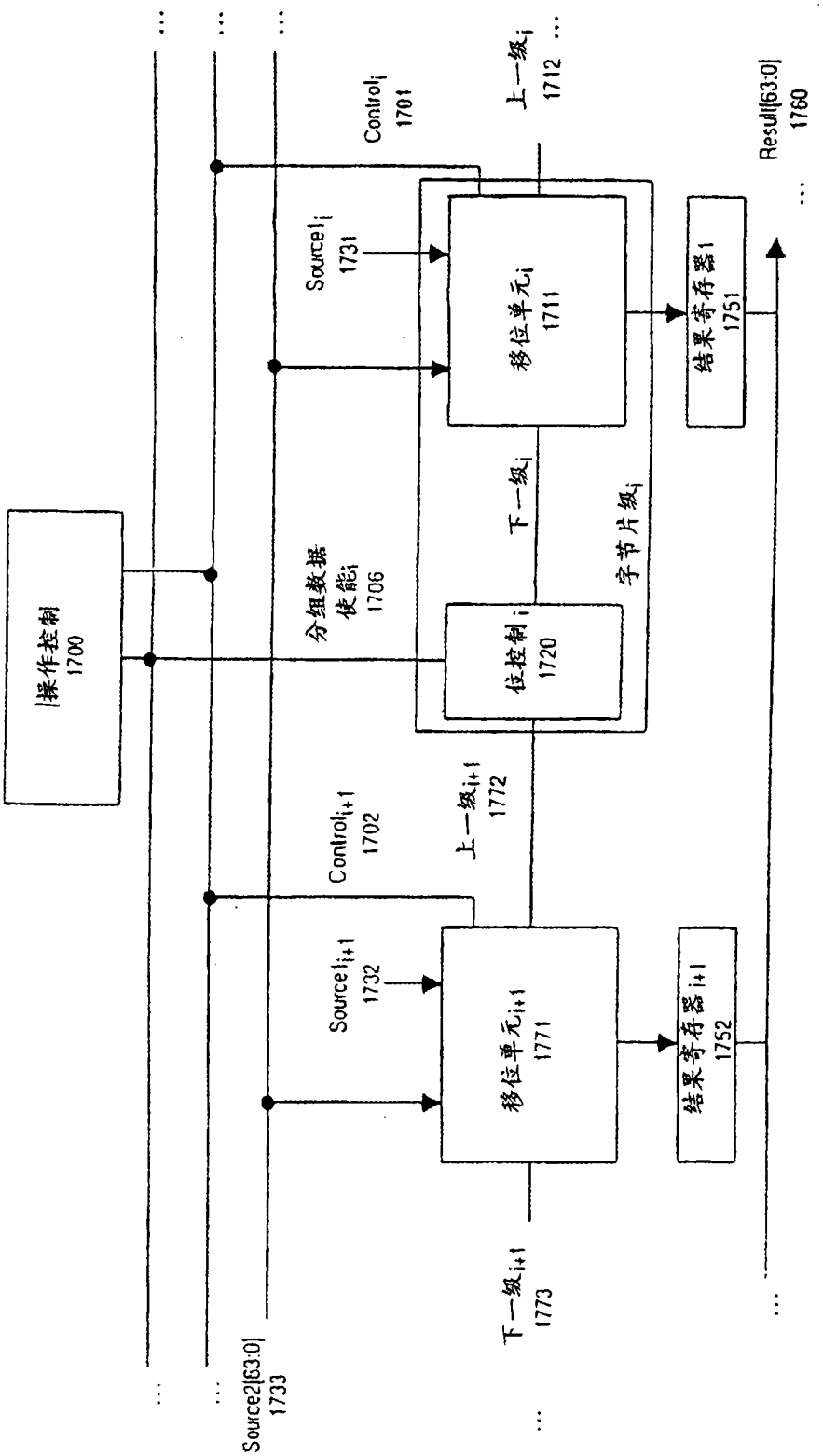

操作控制
1700
Control$_i$
1701
Source1$_i$
1731
分组数据使能$_i$
1706
上一级$_i$
1712
移位单元$_i$
1711
结果寄存器1
1751
Result[63:0]
1760
下一级$_i$
字节片级$_i$
位控制$_i$
1720
Control$_{i+1}$
1702
上一级$_{i+1}$
1772
Source1$_{i+1}$
1732
移位单元$_{i+1}$
1771
结果寄存器$_{i+1}$
1752
下一级$_{i+1}$
1773
Source2[63:0]
1733

图17

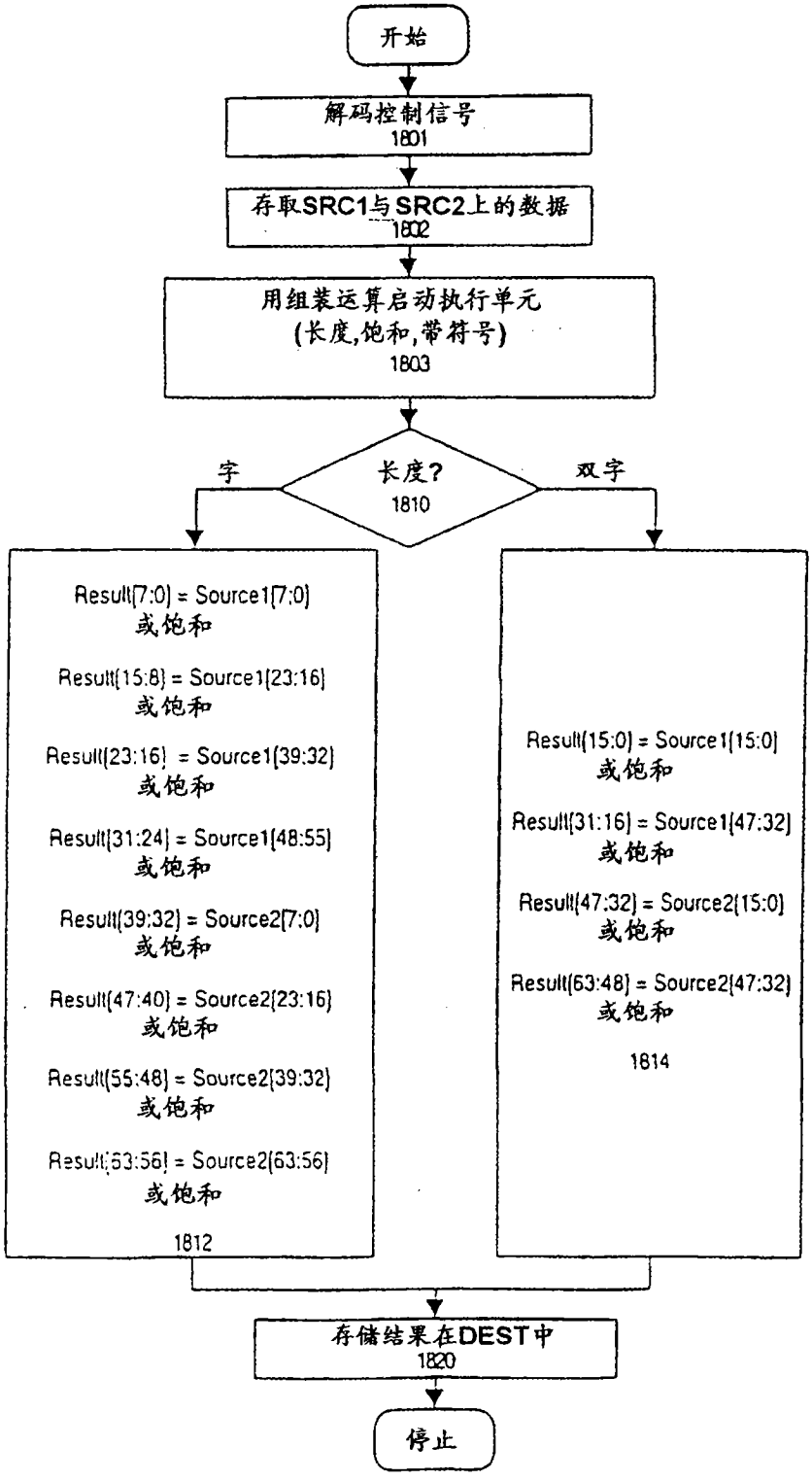

开始
解码控制信号
1801
存取SRC1与SRC2上的数据
1802
用组装运算启动执行单元
(长度,饱和,带符号)
1803
长度?
1810
字
双字
Result[7:0] = Source1[7:0]
或饱和
Result[15:8] = Source1[23:16]
或饱和
Result[23:16] = Source1[39:32]
或饱和
Result[31:24] = Source1[48:55]
或饱和
Result[39:32] = Source2[7:0]
或饱和
Result[47:40] = Source2[23:16]
或饱和
Result[55:48] = Source2[39:32]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

图18

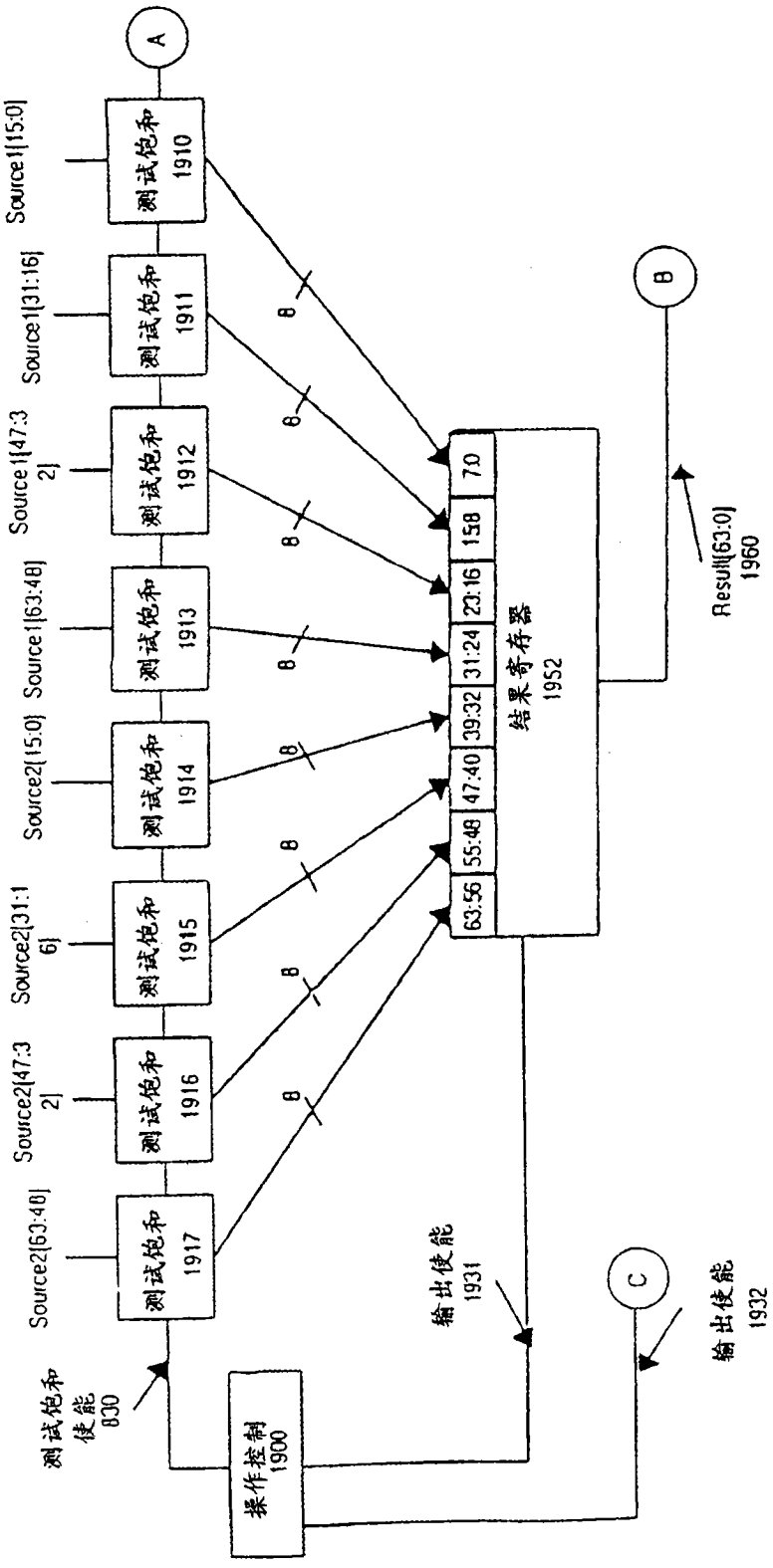

A
B
C
测试饱和 1910
测试饱和 1911
测试饱和 1912
测试饱和 1913
测试饱和 1914
测试饱和 1915
测试饱和 1916
测试饱和 1917
Source1[15:0]
Source1[31:16]
Source1[47:32]
Source1[63:48]
Source2[15:0]
Source2[31:16]
Source2[47:32]
Source2[63:48]
8
结果寄存器 1952
7:0
15:8
23:16
31:24
39:32
47:40
55:48
63:56
Result[63:0] 1960
输出使能 1931
输出使能 1932
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

图19a

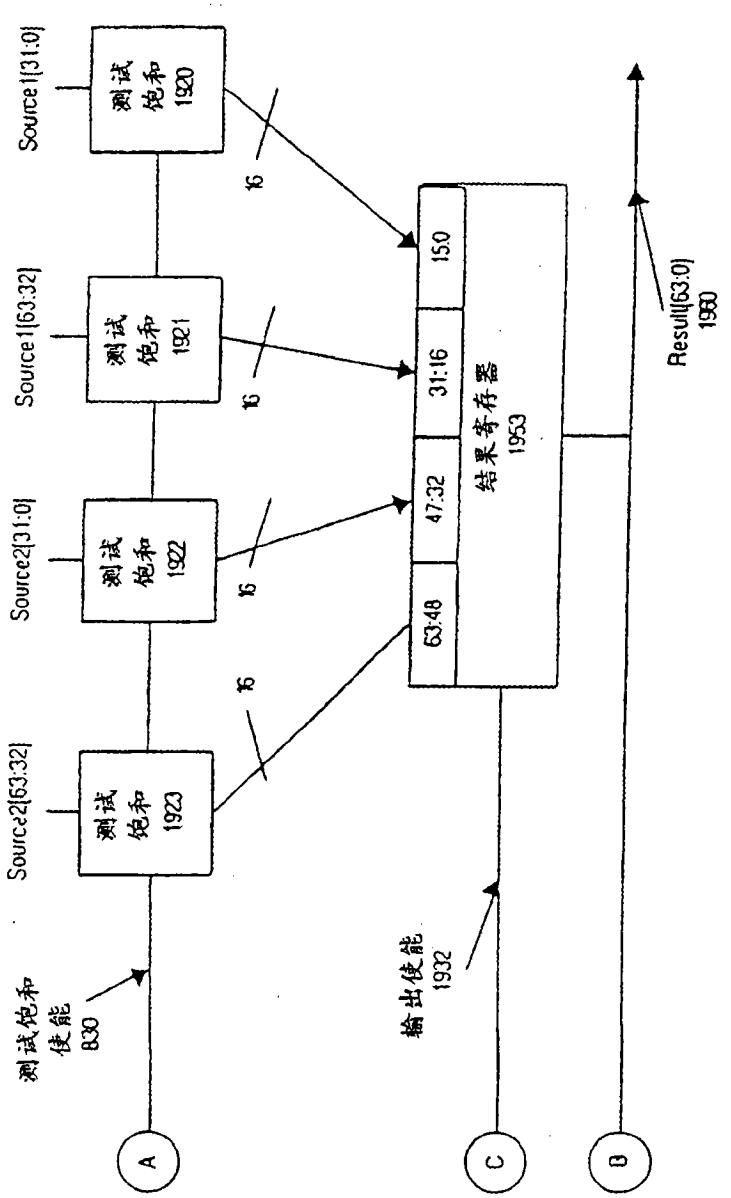

Source1[31:0]
测试饱和 1920
Source1[63:32]
测试饱和 1921
Source2[31:0]
测试饱和 1922
Source2[63:32]
测试饱和 1923
16
16
16
16
150
31:16
47:32
63:48
结果寄存器 1953
Result[63:0] 1950
测试饱和使能 830
输出使能 1932
A
C
B

图19b

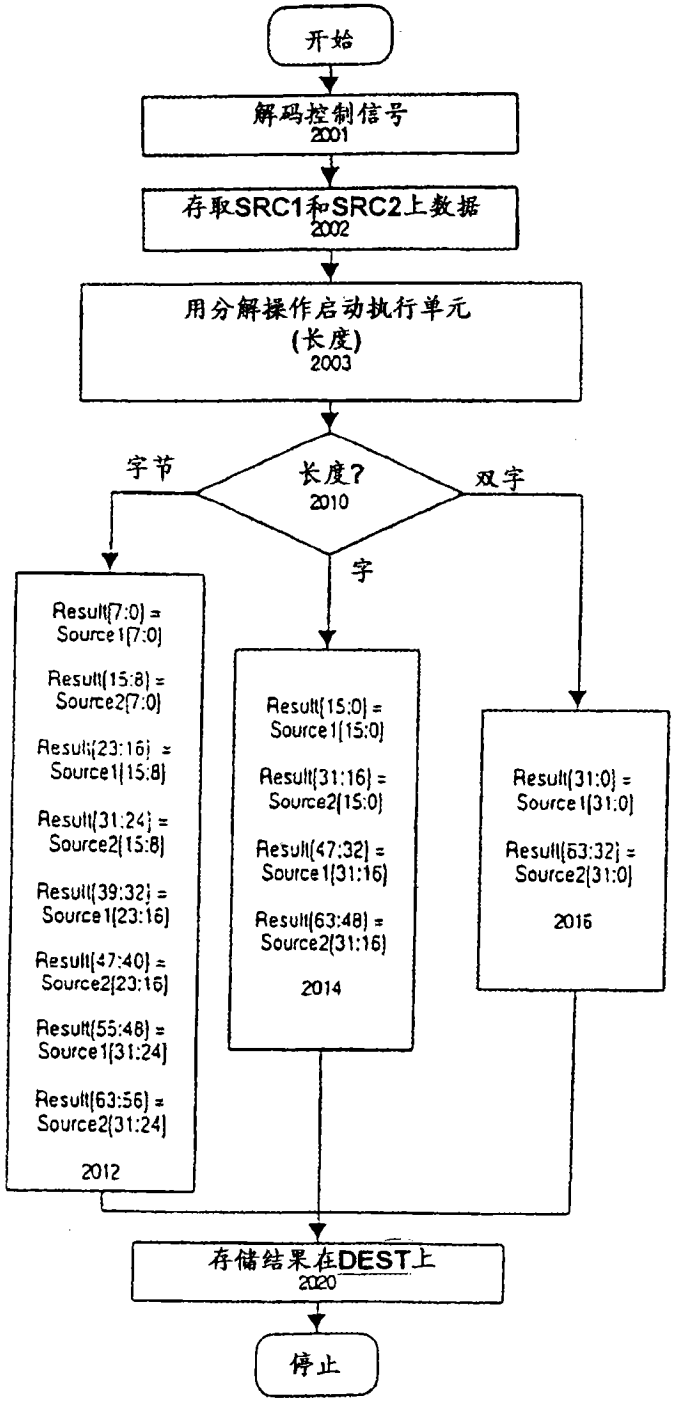

开始
解码控制信号
2001
存取SRC1和SRC2上数据
2002
用分解操作启动执行单元
(长度)
2003
长度?
2010
字节
字
双字
Result[7:0] = Source1[7:0]
Result[15:8] = Source2[7:0]
Result[23:16] = Source1[15:8]
Result[31:24] = Source2[15:8]
Result[39:32] = Source1[23:16]
Result[47:40] = Source2[23:16]
Result[55:48] = Source1[31:24]
Result[63:56] = Source2[31:24]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

图20

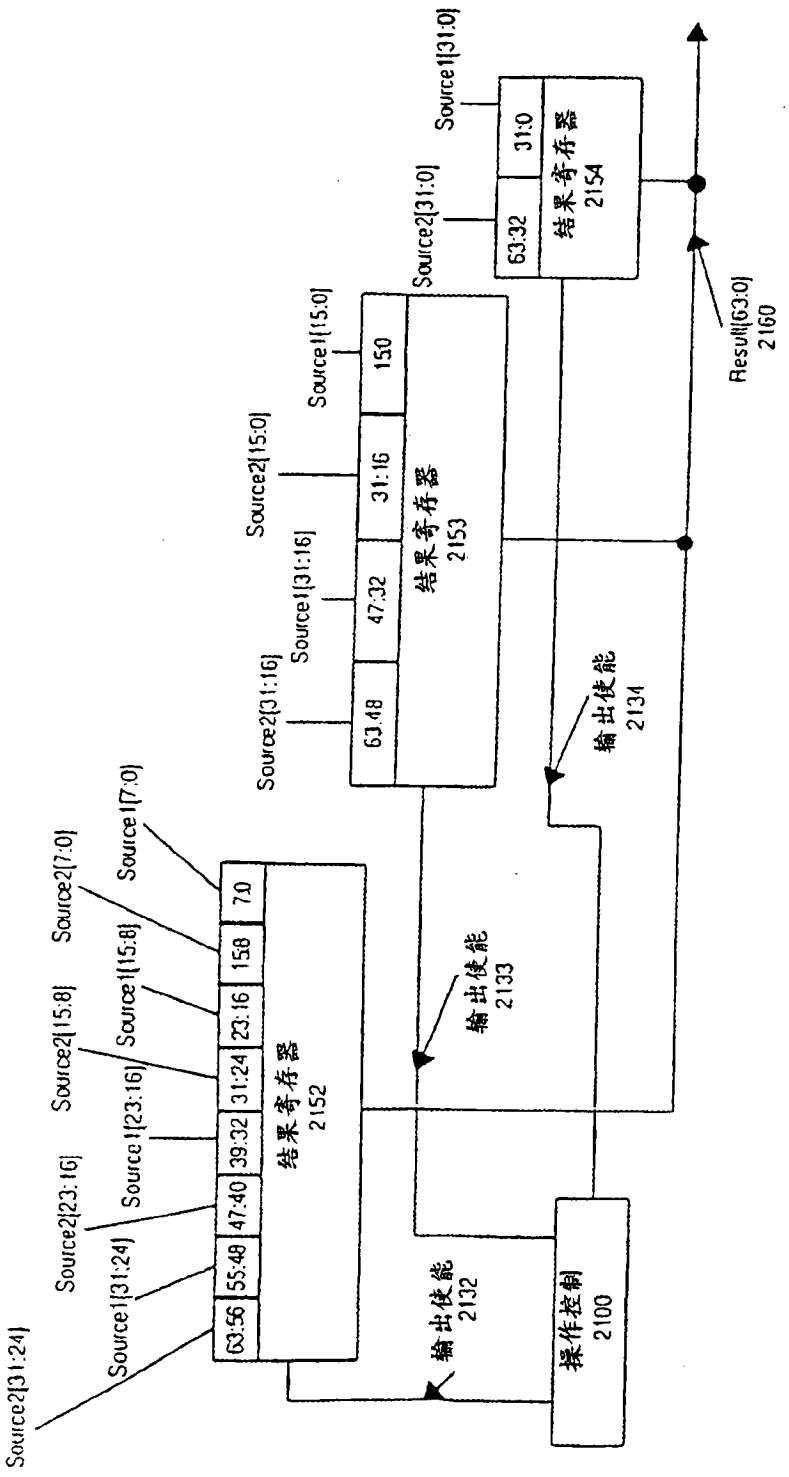

Source1[31:0]
Source2[31:0]
63:32
31:0
结果寄存器
2154
Source1[15:0]
Source2[15:0]
Source1[31:16]
Source2[31:16]
63:48
47:32
31:16
15:0
结果寄存器
2153
Result[63:0]
2160
输出使能
2134
输出使能
2133
输出使能
2132
操作控制
2100
Source1[7:0]
Source2[7:0]
Source1[15:8]
Source2[15:8]
Source1[23:16]
Source2[23:16]
Source1[31:24]
Source2[31:24]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

图21

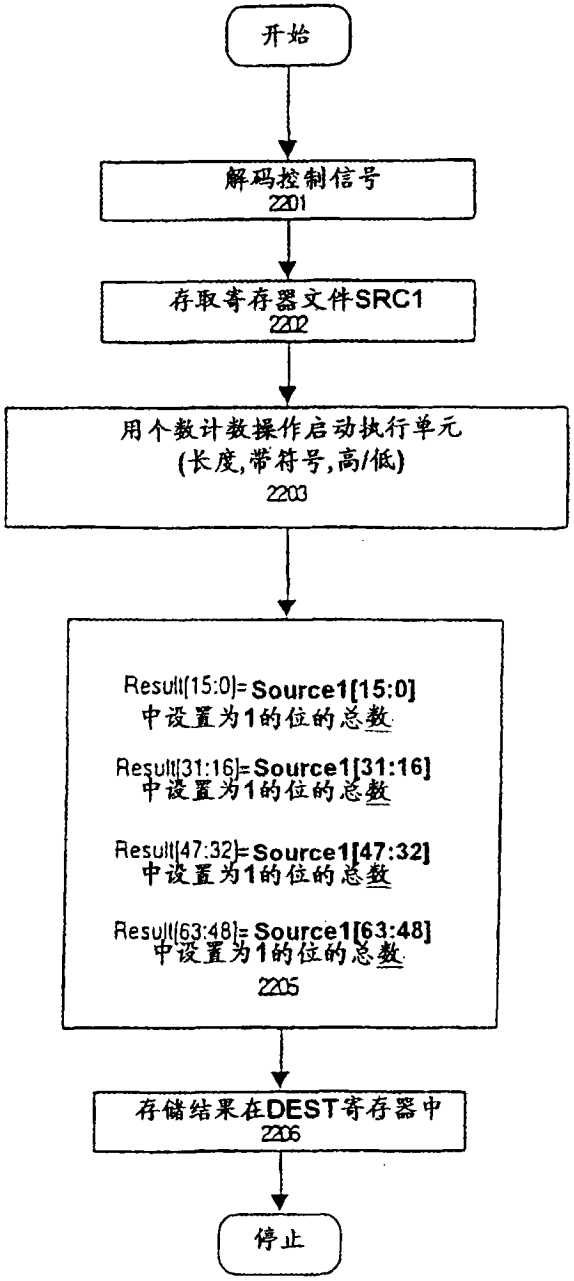

开始
解码控制信号
2201
存取寄存器文件SRC1
2202
用个数计数操作启动执行单元
(长度,带符号,高/低)
2203
Result[15:0]=Source1[15:0]
中设置为1的位的总数
Result[31:16]=Source1[31:16]
中设置为1的位的总数
Result[47:32]=Source1[47:32]
中设置为1的位的总数
Result[63:48]=Source1[63:48]
中设置为1的位的总数
2205
存储结果在DEST寄存器中
2206
停止

图22

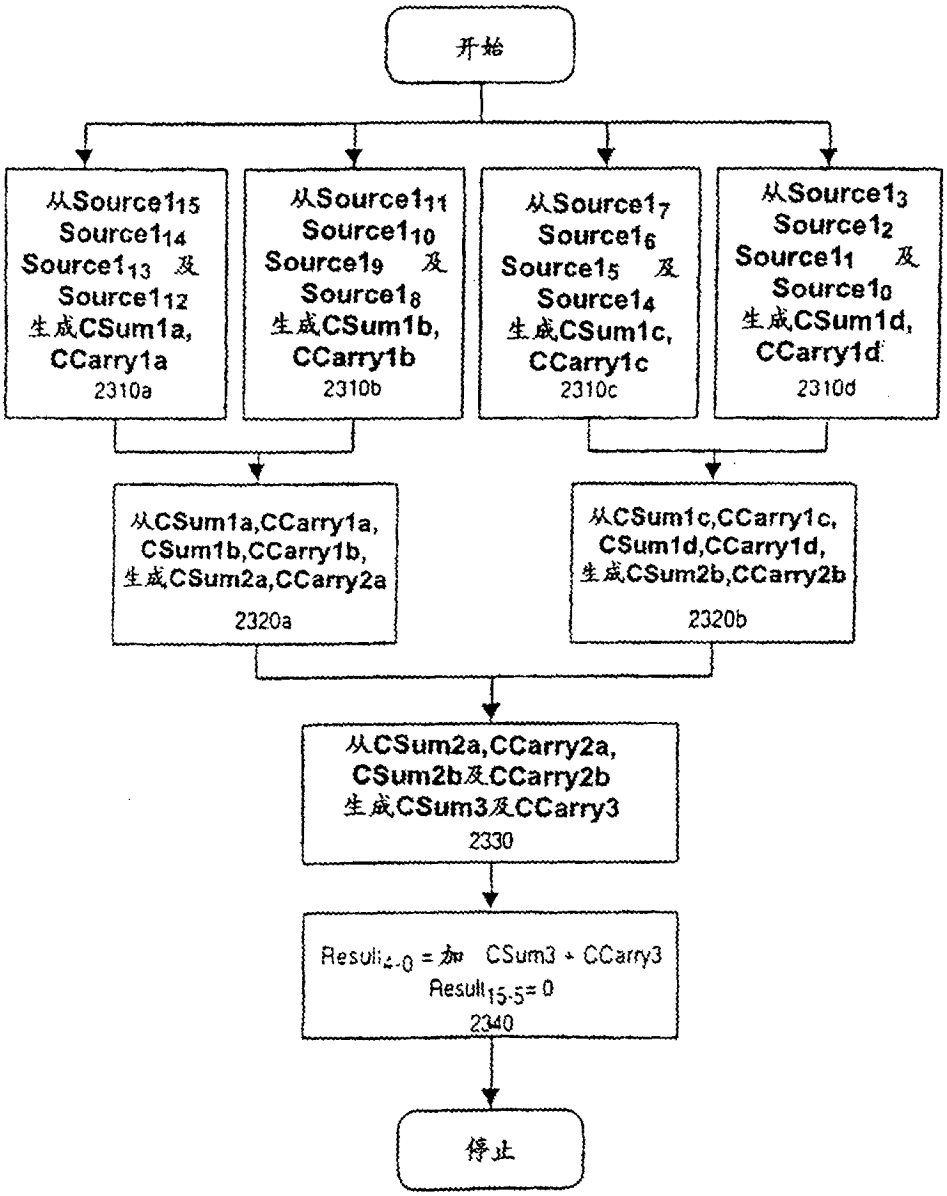

开始
从Source1$_{15}$ Source1$_{14}$ Source1$_{13}$ 及 Source1$_{12}$ 生成CSum1a, CCarry1a
2310a
从Source1$_{11}$ Source1$_{10}$ Source1$_{9}$ 及 Source1$_{8}$ 生成CSum1b, CCarry1b
2310b
从Source1$_{7}$ Source1$_{6}$ Source1$_{5}$ 及 Source1$_{4}$ 生成CSum1c, CCarry1c
2310c
从Source1$_{3}$ Source1$_{2}$ Source1$_{1}$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

图23

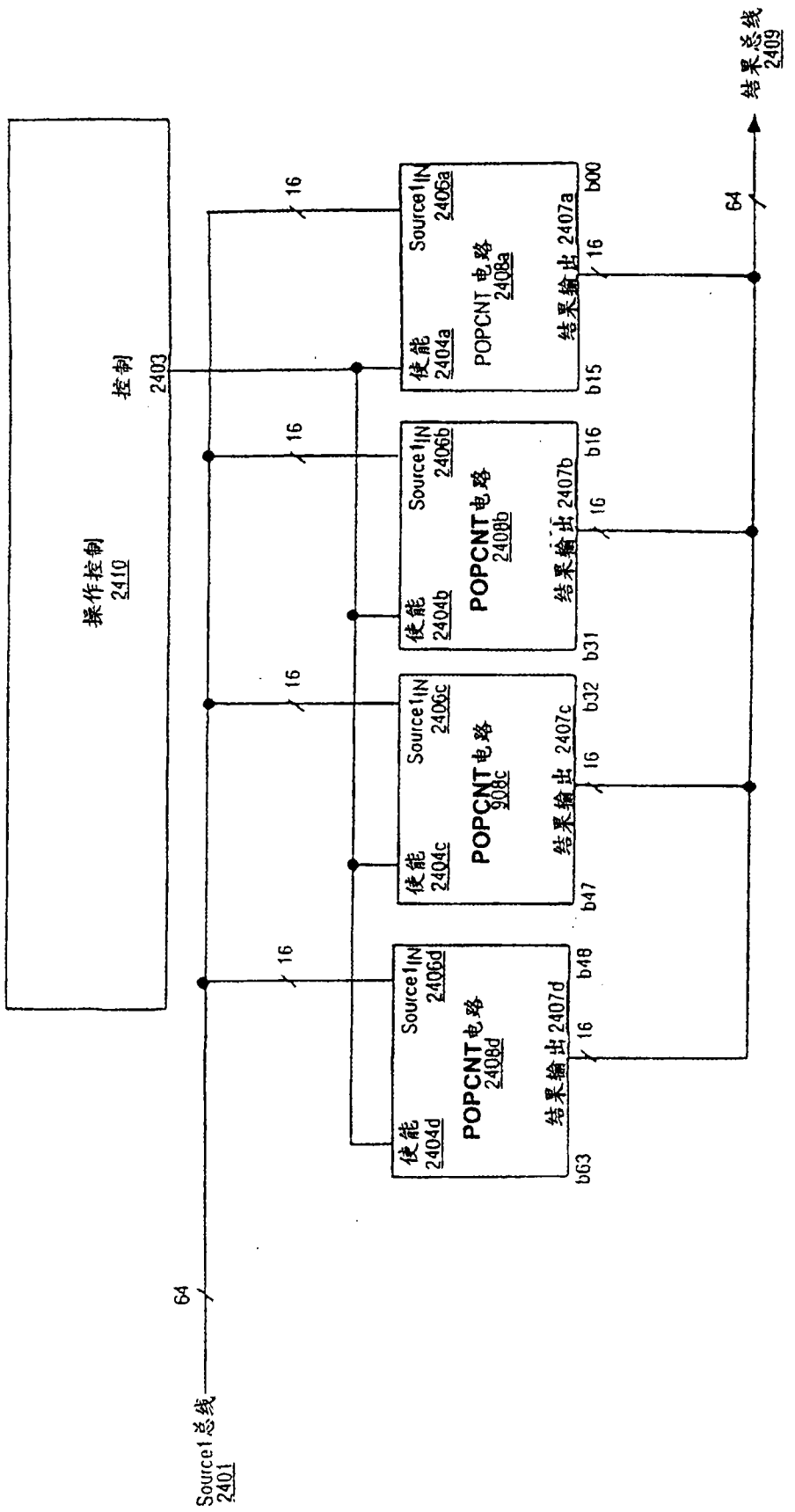

操作控制
2410
控制
2403
Source1总线
2401
64
16
Source1IN
2406a
使能
2404a
POPCNT电路
2408a
结果输出 2407a
b00
b15
Source1IN
2406b
使能
2404b
POPCNT电路
2408b
结果输出 2407b
b16
b31
Source1IN
2406c
使能
2404c
POPCNT电路
2408c
结果输出 2407c
b32
b47
Source1IN
2406d
使能
2404d
POPCNT电路
2408d
结果输出 2407d
b48
b63
结果总线
2409

图24

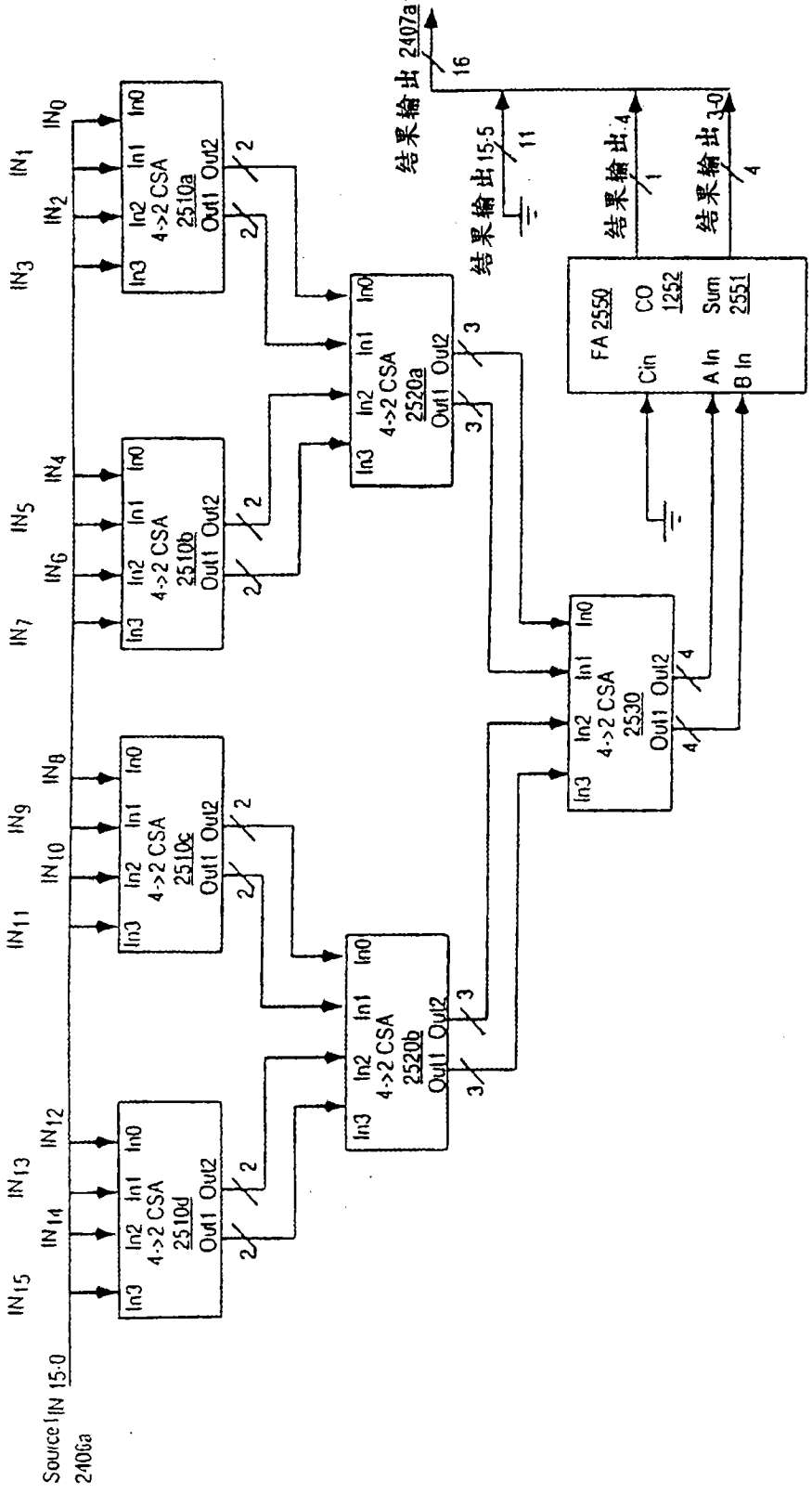

结果输出 2407a
16
结果输出 15:5
11
结果输出 4
结果输出 3:0
4
FA 2550
Cin
CO 1252
A In
Sum 2551
B In
4->2 CSA 2510a
4->2 CSA 2510b
4->2 CSA 2510c
4->2 CSA 2510d
4->2 CSA 2520a
4->2 CSA 2520b
4->2 CSA 2530
In0
In1
In2
In3
Out1
Out2
IN0
IN1
IN2
IN3
IN4
IN5
IN6
IN7
IN8
IN9
IN10
IN11
IN12
IN13
IN14
IN15
Source IN 15:0
2406a

图25

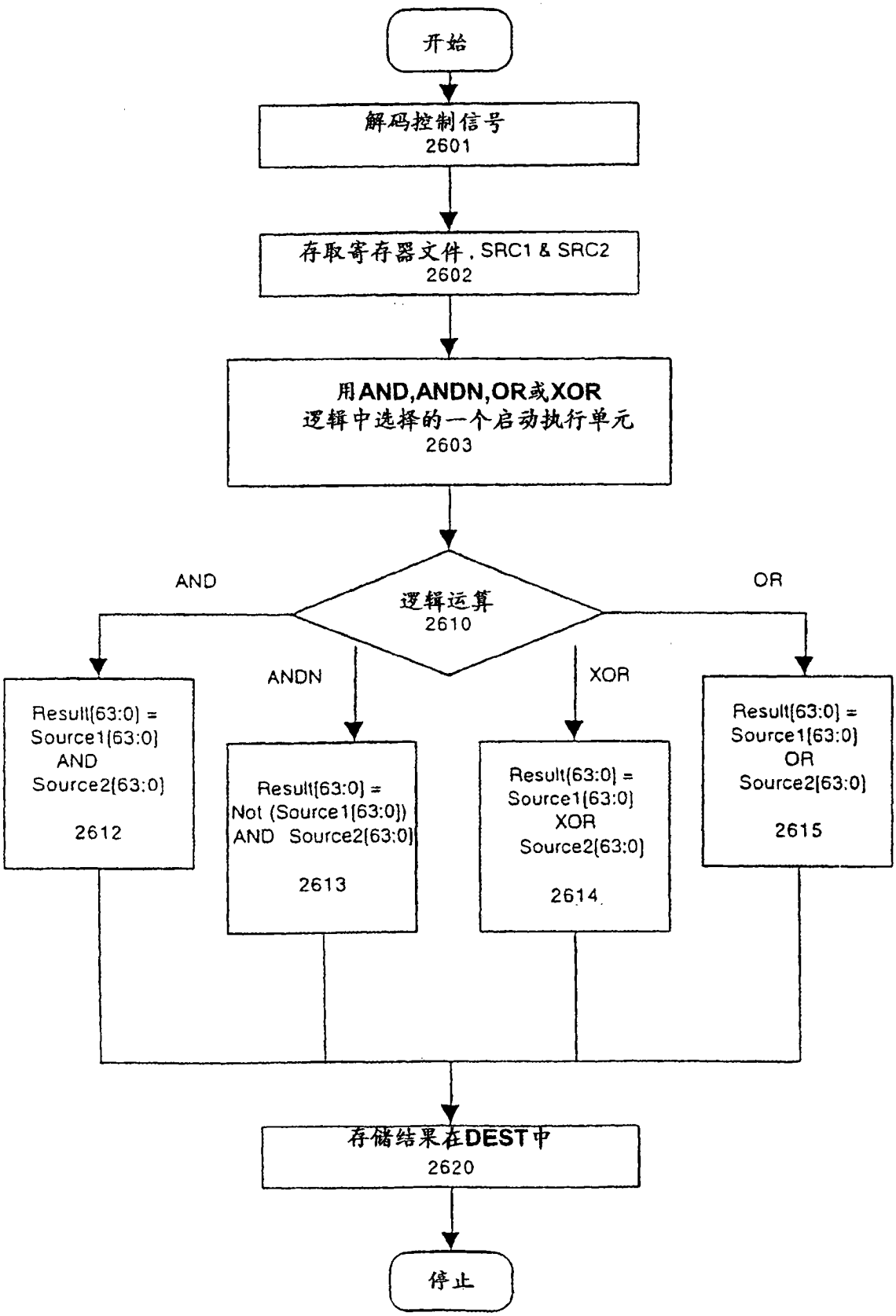

开始
解码控制信号
2601
存取寄存器文件，SRC1 & SRC2
2602
用AND,ANDN,OR或XOR
逻辑中选择的一个启动执行单元
2603
逻辑运算
2610
AND
ANDN
XOR
OR
Result[63:0] = Source1[63:0] AND Source2[63:0]
2612
Result[63:0] = Not (Source1[63:0]) AND Source2[63:0]
2613
Result[63:0] = Source1[63:0] XOR Source2[63:0]
2614
Result[63:0] = Source1[63:0] OR Source2[63:0]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

图26

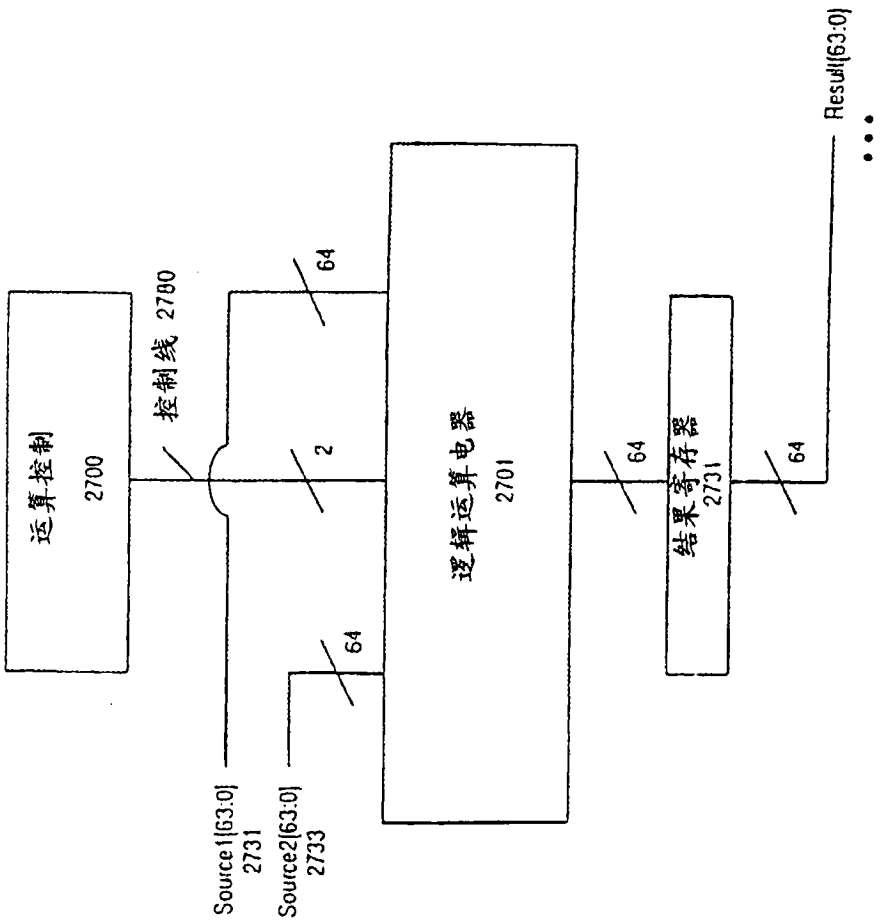

运算控制
2700
控制线 2780
64
2
64
逻辑运算电器
2701
64
结果寄存器
2731
64
Result[63:0]
Source1[63:0]
2731
Source2[63:0]
2733

图27

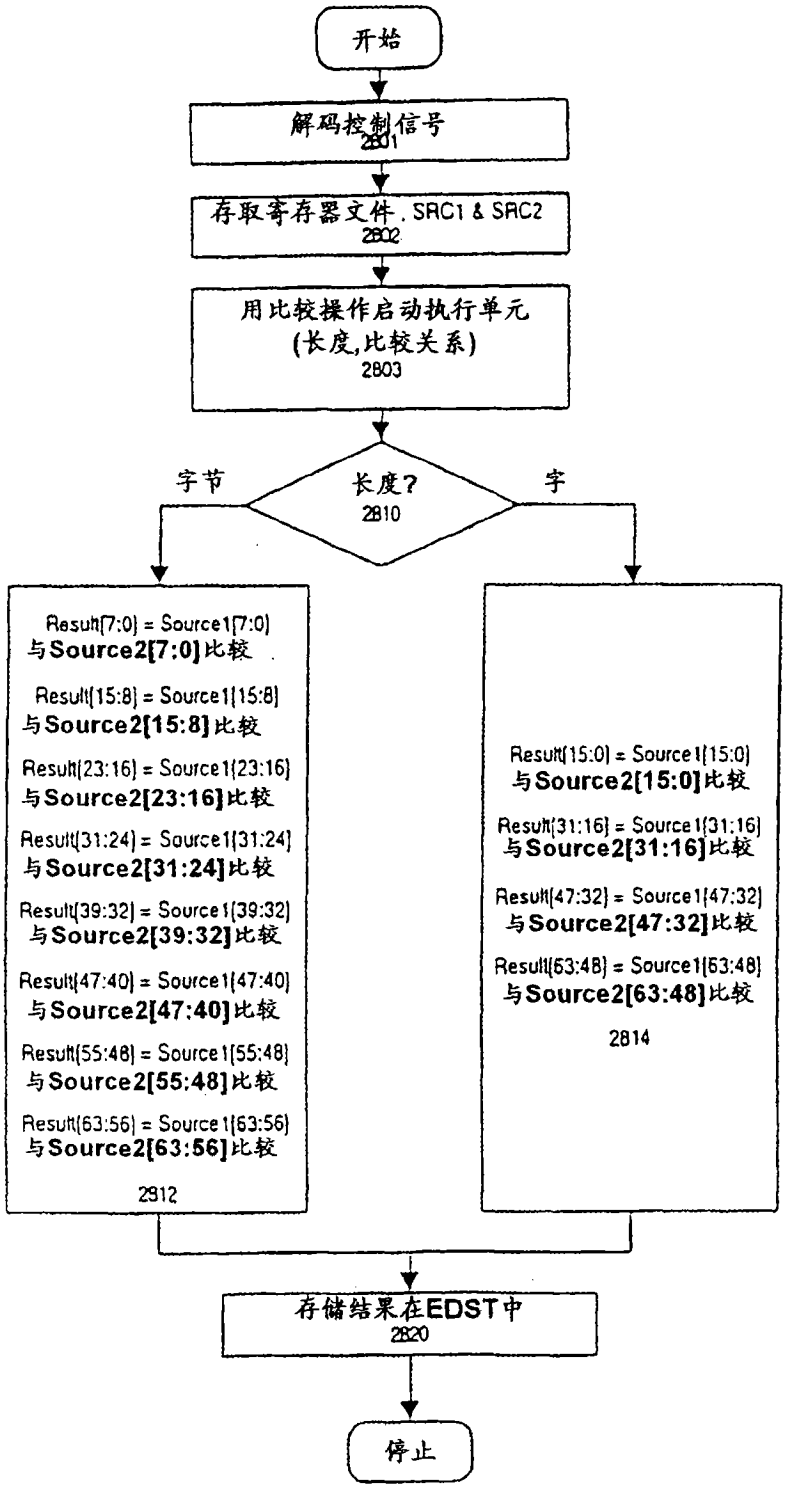

开始
解码控制信号
2801
存取寄存器文件，SRC1 & SRC2
2802
用比较操作启动执行单元
(长度,比较关系)
2803
长度?
2810
字节
字
Result[7:0] = Source1[7:0]
与Source2[7:0]比较
Result[15:8] = Source1[15:8]
与Source2[15:8]比较
Result[23:16] = Source1[23:16]
与Source2[23:16]比较
Result[31:24] = Source1[31:24]
与Source2[31:24]比较
Result[39:32] = Source1[39:32]
与Source2[39:32]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

图28

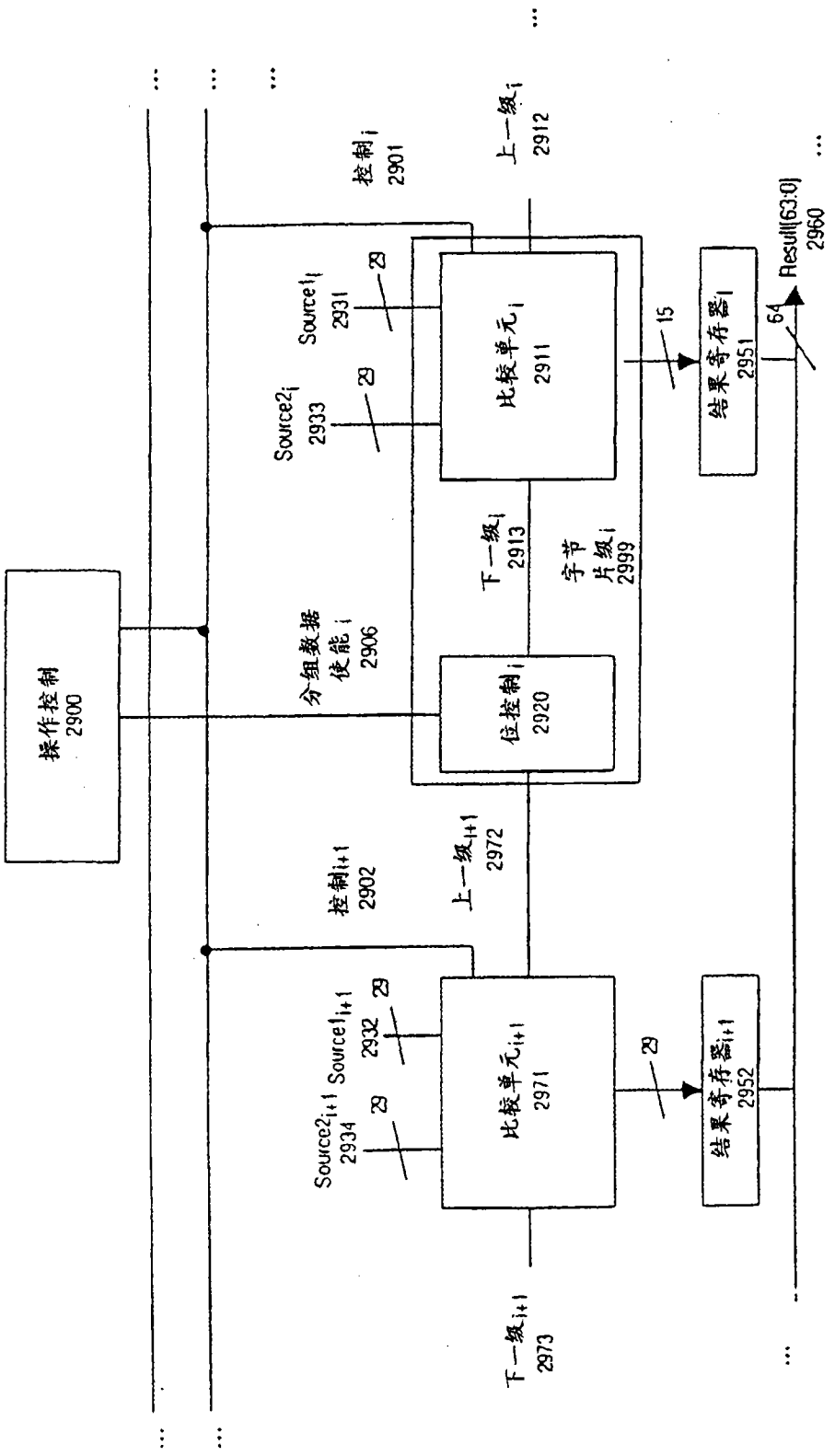

操作控制
2900
控制$_i$
2901
上一级$_i$
2912
Source1$_i$
2931
29
Source2$_i$
2933
29
比较单元$_i$
2911
15
结果寄存器$_i$
2951
64
Result[63:0]
2960
分组数据
使能$_i$
2906
下一级$_i$
2913
字节
片级$_i$
2999
位控制$_i$
2920
控制$_{i+1}$
2902
上一级$_{i+1}$
2972
Source1$_{i+1}$
2932
29
Source2$_{i+1}$
2934
29
比较单元$_{i+1}$
2971
29
结果寄存器$_{i+1}$
2952
下一级$_{i+1}$
2973

图29