[19]中华人民共和国专利局

[51]Int.Cl[6]
G06F 7/38
G06F 7/42 G06F 7/44
G06F 9/345

[12] 发明专利申请公开说明书

[21] 申请号 96191245.6

[43]公开日 1998 年 2 月 18 日

[11] 公开号 CN 1173931A

[22]申请日 96.8.30

[30]优先权

[32]95.9.1 [33]US[31]60/003,140

[32]95.9.25 [33]US[31]60/004,642

[86]国际申请 PCT/US96/13900 96.8.30

[87]国际公布 WO97/09671 英 97.3.13

[85]进入国家阶段日期 97.6.20

[71]申请人 菲利浦电子北美公司

地址 美国纽约州

[72]发明人 G·斯莱芬伯格 P·范德穆伦 赵镛显
V·梅拉
李彦志

[74]专利代理机构 中国专利代理(香港)有限公司

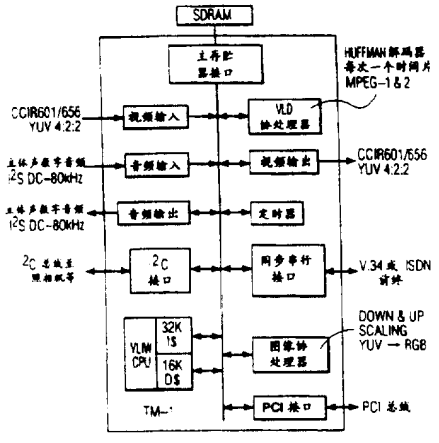
代理人 王 勇 陈景峻

权利要求书 6 页 说明书 24 页 附图页数 26 页

[54]发明名称 处理器的定制的基本操作的方法与设备

[57]摘要

定制操作可用于执行包括多媒体功能在内等功能的处理器系统。这些定制操作在保留专用嵌入方案的成本低廉芯片数少等优点以及通用处理器可编程性等优点不变的同时，提高系统（诸如 PC 系统）提供实时多媒体功能的能力。这些定制操作所作用的计算机系统，提供带操作数的输入数据，对操作数进行操作，并将结果送到目的寄存器。所执行的操作包括包含剪切或饱和操作在内的视频与音频信号处理。本发明也以来自输入寄存器（rscr）的选定操作数为运算对象进行并行操作并将操作结果存入目的寄存器（rdes）。



权　利　要　求　书

1、一个计算机系统，包括：

接收输入数据的输入寄存器，每个输入数据含有 M 位，每个输入数据中具有操作数，每个操作数包含 M 位输入数据中 N 位，其中 N 小于或等于 M；

一个处理器，该处理器在某指令集中的一个指令的控制下，并行执行 Q 个以输入数据的操作数为对象的运算操作产生 N 位的结果数据；

一个目的寄存器，用于存放作为 M 位输出的 Q 组运算结果数据。

2、一个计算机系统，包括：

提供 M 位输入数据的输入寄存器，每个输入数据至少包含两个操作数，每个操作数包含 N 位，其中 N 小于或等于 M；

一个专用处理器，并行执行一组以输入数据中选定操作数为操作对象的运算操作，每组操作产生 N 位的结果数据；该处理器响应某指令集中的某个指令而执行操作；

一个 M 位目的寄存器，用于存放 N 位结果数据，用作一个 M 位输出数据。

3、如权利要求 2 中所述的计算机系统，其中：

各 M 位输入寄存器的每个输入数据包括一个 N 位的第一操作数和一个 N 位的第二操作数；

该处理器包括：

将第一输入数据中的第一操作数与第二输入数据中的第一操作数相加产生一个 N 位的第一操作结果数据以及将第一输入数据中的第二操作数与第二输入数据中的第二操作数相加产生一个 N 位的第二操作结果数据的装置，以及将各个结果数据在某限定范围内分别剪切成长度分别为 N 位的剪切结果数据的装置；

由所述目的寄存器将该分别剪切的结果数据存储在一起。

4、如权利要求 2 中所述的计算机系统，其中：

M 位的输入数据包括长度各为 N 位的两个操作数；

该处理器包括：

计算每个操作数的绝对值并分别产生 N 位长度的绝对值的结果数

据的装置，以及将各个绝对值在某限定范围内分别剪切成长度各为 N 位的剪切结果数据的装置；

由所述目的寄存器将该分别剪切的结果数据存储在一起。

5、如权利要求 2 中所述的计算机系统，其中：

M 位的输入数据包含一个 N 位的第一操作数和一个 N 位的第二操作数；

该处理器包括：

将第一输入数据中的第一操作数与第二输入数据中的第一操作数相乘产生第一个积以及将第一输入数据中的第二操作数与第二输入数据中的第二操作数相乘产生第二个积、两个积的长度均为 N 位的装置，以及将各个积在某限定范围内分别剪切成长度各为 N 位的剪切结果数据的装置：

由所述目的寄存器将该分别剪切的结果数据存储在一起。

6、如权利要求 2 中所述的计算机系统，其中：

M 位的输入数据包含一个 N 位的第一操作数和一个 N 位的第二操作数；

该处理器包含：

将第一输入数据中的第一操作数与第二输入数据中的第一操作数相减产生第一个差、将第一输入数据中的第二操作数与第二输入数据中的第二操作数相减产生第二个差、两个差的长度均为 N 位的装置，以及将第一个差与第二个差在某限定范围内分别剪切成长度各为 N 位的剪切结果数据的装置；

由所述目的寄存器将该分别剪切的结果数据存储在一起。

7、如权利要求 2 中所述的计算机系统，其中：

M 位的输入数据包含 P 个长度各为 N 位的操作数，其中 P 至少是 2；

该处理器包含：

将第一输入数据 P 个操作数中的每个操作数依次与第二输入数据 P 个操作数中对应的操作数相加、每次相加产生一个对应的 N 位的和的装置，以及将各个和在某限定范围内分别剪切成 P 个 N 位的剪切结果数据的装置；

由所述目的寄存器将该 P 个剪切结果数据存储在一起。

8、如权利要求2中所述的计算机系统，其中：

M位的输入数据包含P个长度各为N位的操作数，其中P至少是2；

该处理器包含：

将第一输入数据P个操作数中的每个操作数依次与第二输入数据P个操作数中对应的操作数相减、每次相减产生一个对应的N位的差的装置，以及将各个差在某限定范围内分别剪切成P个N位的剪切结果数据的装置；

由所述目的寄存器将该P个剪切结果数据存储在一起。

9、如权利要求2中所述的计算机系统，其中：

M位的输入数据包含P个长度各为N位的操作数，其中P至少是2；

该处理器包含：

将第一输入数据P个操作数中的每个操作数依次与第二输入数据P个操作数中对应的操作数相乘、每次相乘产生一个对应的N位的积的装置，以及将各个积在某限定范围内分别剪切成P个N位的剪切结果数据的装置；

由所述目的寄存器将该P个剪切结果数据存储在一起。

10、如权利要求2中所述的计算机系统，其中：

M位的输入数据包含P个长度各为N位的操作数，其中P至少是2；

该处理器包含依次计算第一输入数据与第二输入数据内P对操作数中的每对操作数的平均值、每次计算产生一个对应的N位的平均值的装置；

由所述目的寄存器存储该P个平均值。

11、如权利要求2中所述的计算机系统，其中：

M位的输入数据包含P个长度各为N位的操作数，其中P至少是2；

该处理器包含将第一输入数据P个操作数中的每个操作数依次与第二输入数据P个操作数中对应的操作数相乘、每次相乘产生一个对应的2*N位的积的装置；

由所述目的寄存器从各个积中读取共N个选定位，形成对应的中

间数据，将该 P 个中间数据存储到某目的寄存器，所述目的寄存器长度为 M 位。

12、一个计算机系统，包括：

一个输入寄存器，用于提供 M 位的输入数据；

一个处理器，用于从输入数据读取 N 个位的数据值用于 P 输入数据 N<M；

一个目的寄存器，用于存放从 P 输入数据的读取的各组 N 位数据值。

13、一个计算机系统，包括：

一个输入寄存器，用于提供 M 位的输入数据；

一个处理器，用于从输入数据中每个输入数据读取 N 个位的数据值用于 P 数据，N<M，该处理器包含将该从 P 输入数据的读取的各组 N 位数据值组合存入一个目的寄存器的装置。

14、如权利要求 12 中所述的计算机系统，其中：

P 为 2，N 为 M 的一半，并且所述处理器读取输入数据的最高有效位组（bits）或输入数据的最低有效位组之一。

15、如权利要求 12 中所述的计算机系统，其中：

提供一个第一输入数据和一个第二输入数据；

所述处理器读取各个输入数据的最高有效位组（msb），各个最高有效位组被分别用作最高有效位组内的最高有效位组（mmsb）和最高有效位组内的最低有效位组（lmsb）；

所述打包装置将第一输入数据中的最高有效位组内的最高有效位组（mmsb）组合成目的寄存器中的最高有效位组；

所述打包装置将第二输入数据中的最高有效位组内的最高有效位组（mmsb）组合成寄存器中的次高有效位组；

所述打包装置将第二输入数据中的最高有效位组内的最低有效位组（lmsb）组合成寄存器中的最低有效位组；

所述打包装置将第一输入数据中的最高有效位组内的最低有效位组（lmsb）组合成寄存器中的次低有效位组。

16、如权利要求 12 中所述的计算机系统，其中：

提供一个第一输入数据和一个第二输入数据；

所述处理器读取各个输入数据的最低有效位组（lsb），各个最低

有效位组被分别用作最低有效位组内的最高有效位组（ mlsb ）和最低有效位组内的最低有效位组（ llsb ）；

所述打包装置将第一输入数据中的最低有效位组内的最高有效位组（ mlsb ）组合成寄存器中的最高有效位组；

所述打包装置将第二输入数据中的最低有效位组内的最高有效位组（ mlsb ）组合成寄存器中的次高有效位组；

所述打包装置将第二输入数据中的最低有效位组内的最低有效位组（ llsb ）组合成寄存器中的最低有效位组；

所述打包装置将第一输入数据中的最低有效位组内的最低有效位组（ llsb ）组合成寄存器中的次低有效位组。

17、如权利要求 12 中所述的计算机系统，其中：

提供一个第一输入数据和一个第二输入数据；

所述处理器读取各个输入数据的最低有效位组（ lsb ）；

所述打包装置将第二输入数据中的最低有效位组（ lsb ）组合成目的寄存器中的最低有效位组；

所述打包装置将第一输入数据中的最低有效位组（ lsb ）组合成目的寄存器中的次低有效位组；

所述打包装置用预定的位值填充目的寄存器中的最高有效位组。

18、一个计算机系统，包含：

一个用于接收输入数据的输入寄存器，每个输入数据有 M 位，包含 Q 个操作数，每个操作数包含 N 位，其中 N 小于或等于 M；

一个用于对所述输入数据进行数据处理的处理器，所述数据处理包括并行执行以选定操作数为操作对象的一组运算操作，每组操作产生对应于各个操作数的 N 位长度的结果数据；

一个用于存放作为包括 M 位的输出数据的 Q 个结果数据的目的寄存器。

19、如权利要求 18 中所述的计算机系统，其中所述数据处理至少是音频处理和视频处理之一。

20、如权利要求 18 中所述的计算机系统，其中所述计算机系统集成在半导体基片上。

21、一个计算机系统，包含：

一个用于提供输入数据的输入寄存器，输入数据包含操作数；

一个用于执行以操作数为操作对象的一组运算操作的处理器，所述操作包含剪切功能，所述处理器产生结果数据；

一个用于存放结果数据中选定数据的目的寄存器。

22、如权利要求21中所述的计算机系统，其中

每个输入数据至少包含两个操作数；

所述处理器响应某指令集中的某个指令而并行执行选定的操作，所述操作包含剪切功能并产生结果数据。

23、一个计算机系统，包含：

一组输入寄存器，每个存放多个操作数的连接；

从输入寄存器中接收操作数、响应某指令并行执行以该操作数中特定操作数为操作对象的一组操作的数据处理装置，所述操作包括单操作数的第一种操作和多操作数的第二种操作，每个操作均产生结果数据，每一组操作产生各自的输出数据，并且至少包含一个操作；

一个与该数据处理装置相连、用于将各个输出结果数据存放一起的目的寄存器。

24、一个用于信号数据处理的信号处理系统，包含：

至少一个用于存储和提供信号数据的输入寄存器；

一个处理器，用于在指令控制下，执行系统硬件提供的一组机器指令，每个指令指示该处理器至少执行一个操作并产生结果数据，该组指令至少包含一个剪切指令，用于在将对信号数据操作的结果数据送到目的寄存器之前对该结果数据进行剪切。

25、权利要求24中的计算机系统，其中所述计算机系统集成在半导体基片上。

说　明　书

处理器的定制的基本操作的方法与设备

引用相关申请文件

本发明要求 1995 年 9 月 1 日提交的美国临时申请 60/003,140 号和 1995 年 9 月 25 日提交的美国临时申请 60/004,642 号的优先权。

本说明书中为讨论 VLIM 处理系统引用了如下专利申请:

5,103,311 号美国专利: 数据处理模块及包括该模块的视频处理系统;

5,450,556 号美国专利: 应用由分支控制单元生成的路径数据阻止非正确路径上操作的 VLIW 处理器;

5,313,551 号美国专利: 软件控制下的多端口存储器旁路;

1992 年 12 月 29 日申请的序列号 07/998,080 的美国专利申请: 指令发布槽位数少于功能单元的 VLIW 处理器;

1990 年 10 月 5 日申请的序列号 07/594,534 的美国专利申请: 包括一存储电路和一组功能单元的处理器;

1994 年 12 月 16 日申请的序列号 08/358,127 的美国专利申请: 数据处理系统中的例外恢复（ exception recovery ）; 以及

同时提交的申请: 应用剪切功能的用于多媒体应用的定制操作的方法与设备; 应用多操作数进行单一指令控制下并行处理的定制操作的方法与设备。

发明背景

1、发明领域

本发明涉及应用于执行包括多媒体功能等功能的处理器系统—诸如具有处理高质量视频和音频信号能力、执行专业高级功能操作的系统—的定制操作。

2、相关技术描述

系统可使用一个通用 CPU 和附加单元来作为多功能 PC 增强器。一般来说, PC 必须处理有多重标准的视频和音频信号流, 用户还要求尽可能有解压和压缩这两种功能。尽管 PC 机中使用的 CPU 芯片功能正朝着能解压低分辨率实时视频信号的方向发展, 它们还不可能对视频

信号进行高质量的解压和压缩。此外，用户还要求系统提供生动的视频和音频信号，同时又不降低系统的响应灵敏度。

无论是通用应用软件还是嵌入基于微处理器的应用软件，人们都希望用高级语言进行程序设计。为了有效地支持优化编译器和简单化的编程模型，要求微处理器的体系结构具有一定的特点，例如有较大的线性地址空间、通用寄存器以及直接支持使用线性地址指针的寄存器对寄存器操作。近来微处理器体系结构普遍采用32位线性地址、32位寄存器和32位整数操作，而64位及128位的处理器系统正处于研究开发之中。

不过，对于许多算法中进行的数据处理来说，如果数据运算占用全部位数（例如，32位系统中的32位），是对昂贵的硅片资源的一种浪费。有些重要的多媒体应用（如MPEG视频信号流的解压缩），对长度8位的数据项的处理要花费相当长的运算时间。用32、64、128…位类型的运算方式来处理长度短的数据项，不能发挥系统应用的32、64、128…位类型的运算硬件的效率。因而，定制的操作可以同时对数据项进行操作，只要增加少许硬件花费，就能大大地改善运算的效能。

其它方法也能达到改善运算效能的类似效果，例如在每个时钟周期执行大量的传统微处理器指令。然而，所述的其它方法对于要求低成本的目标应用，一般来说成本惊人地昂贵。此外，用M位类型的运算（例如32位类型的运算）来处理长度短的N位数据项（N<M）不能发挥系统应用的M位类型的运算硬件的效率。

常规的数据信号处理（DSP）运算的逻辑计算模数值。本发明所述的剪切或饱和运算方法在数据处理生成的数据超过寄存器物理限度的信号处理应用中具有特殊的价值。按照常规，当生成的数据超过寄存器物理范围限度时，数据就被映射到有效物理范围的另一端。在进行信号处理时，这种周期性的数据映射会产生恶劣的后果。例如，应用常规方法，一组音量很低的音频信号数据可能会被映射到最高音量对应的位置上。在控制应用和视频/音频信号处理应用系统中，当控制范围或强度范围饱和时，模数值对系统是不利的。

发明概述

本发明旨在在保留专用嵌入方式的优点，即芯片成本低廉和芯片数

少，以及通用处理器可再编程性等优点不变的同时，对系统（例如 PC 系统）进行增强以提供实时多媒体功能。在 PC 应用中，本发明超越了固定功能多媒体芯片的功能。

与此对应，本发明的一个目的是，以低廉的成本获得极高的多媒体性能。

本发明的另一个目的是，提高应用软件的小内核程序的处理速度。

本发明的另一个目的是，充分发挥高速缓冲存储器/存储器带宽的优点，同时又不需要过度数量的字节操作指令。

本发明的另一个目的是提供功能强大的专门操作，以改善多媒体应用软件的性能。

本发明的另一个目的是提供执行中有效地利用专门的位操作硬件的定制操作。

本发明的另一个目的是提供专用于应用系统（如多媒体应用软件）的定制操作。

本发明的另一个目的是运用能储存多个操作数的多操作数寄存器来进行单指令的控制下的并行处理。这对当前信号样本为 8 位或 16 位格式的音频和/或视频处理应用软件具有特殊的好处。

本发明的另一个目的是，利用剪切操作在被截断的范围的正确一端保存接收到的信号，如音频或视频信号。

本发明的应用范围是从如视频电话这样的低价格、单用途系统到传统个人计算机用的可重新编程的多用途插件卡。另外，本发明还可以应用于容易实行通用多媒体标准—如 MPEG－1 和 MPEG－2 的系统。此外，应用本发明能使功能强大的多用途 CPU 执行各种无论是开放式的还是专有的多媒体算法。

在源码级定义软件兼容性的好处是能自由地掌握成本与性能间的最佳平衡。功能强大的编译器保证程序员不需要用繁琐的汇编语言进行程序设计。本发明允许程序员从源码中选用功能强大的低级操作，用熟悉的函数调用文法调用数字信号处理那样的操作。

计算机系统包括，1、输入寄存器，每个均用于接收输入数据，每个输入数据含有 M 位，每个操作数包含 N 位，其中 N 小于或等于 M；2、一个处理器，该处理器执行一组以选定操作数为对象的操作，每组操作中至少含有一个操作，操作结果产生 N 位长度的数据；3、一个

目的寄存器，用于存放每组操作的运算结果数据。该组操作可以包含一个剪切或饱和操作。此外，该组操作可以并行运行，并且响应某命令集中的一个命令。

通过以下的详细说明，本发明还有一些目的和优点，对熟悉本领域的人来说是显而易见的。以下的说明中，只是仅仅通过对本发明最佳实施方式思想的描述，叙述和说明了本发明的较佳实施例。正如将会认识到的，本发明还可以有其它不同的实施方式，在许多明显的方面，可以对本发明的一些细节上作出改进，但都不会偏离本发明的本质。同样，附图及说明的性质也应被视作是说明性的，而不是限制性的。

附图简要说明

本发明的上述目的以及其它目的的显而易见性，体现在对本发明进行的、包括以下附图的说明中：

附图 1 是本发明使用的一示例系统的方框图；

附图 2 所示是 CPU 寄存器结构的一个例子；

附图 3（a）所示是存储器中矩阵组织的一个例子；

附图 3（b）所示是该例中要被执行的一个任务；

附图 4 所示是一例用定制操作进行一个字节矩阵置换的应用；

附图 5（a）和 5（b）所示是执行附图 4 所示字节矩阵置换的一列操作；

附图 6 所示是一个 dspiadd 操作；

附图 7 所示是一个 dspuadd 操作；

附图 8 所示是一个 dspidualadd 操作；

附图 9 所示是一个 dspuquadaddui 操作；

附图 10 所示是一个 dspimul 操作；

附图 11 所示是一个 dspumul 操作；

附图 12 所示是一个 dspidualmul 操作；

附图 13 所示是一个 dspisub 操作；

附图 14 所示是一个 dspusub 操作；

附图 15 所示是一个 dspidualsub 操作；

附图 16 所示是一个 ifir16 操作；

附图 17 所示是一个 ifir8ii 操作；

附图 18 所示是一个 ifir8ui 操作；

附图 19 所示是一个 ufir16 操作;

附图 20 所示是一个 ufir8uu 操作;

附图 21 所示是一个 mergelsb 操作;

附图 22 所示是一个 mergemsb 操作;

附图 23 所示是一个 pack16lsb 操作;

附图 24 所示是一个 pack16msb 操作;

附图 25 所示是一个 packbytes 操作;

附图 26 所示是一个 quadavg 操作;

附图 27 所示是一个 quadumulmsb 操作;

附图 28 所示是一个 ume8ii 操作;

附图 29 所示是一个 ume8uu 操作;

附图 30 所示是一个 iclipi 操作;

附图 31 所示是一个 uclipi 操作;

附图 32 所示是一个 uclipu 操作;

较佳实施例说明

附图 1 是本发明使用的一示例系统的方框图。该系统包括一个微处理器、一个同步动态随机存取寄存器（ SDRAM ）和作为与多媒体入、出数据流通路接口所需的外部线路。

本例中，由一个 32 位 CPU 组成 VLIW 处理器的内核。该 CPU 使用一个 32 位线性地址空间和 128 个全部为通用型的 32 位寄存器。本系统中没有将寄存器分类归组，相反，任何操作可以使用任何寄存器来存放任何操作数。

在本系统中， CPU 使用 VLIW 指令集结构，允许同时发出多达五个同时操作。本例中，这些操作可以指向 CPU 内 27 个功能单元中的任何五个，包括整数和浮点算术单元和数据并行的的数字信号处理（ DSP ）类单元。

运用本发明的 CPU 的操作集中，包括传统的微处理器操作，此外还有多媒体专用操作，它们能极大地加快标准视频信号压缩和解压算法的速度，一个专门操作或者定制操作，即一条指令内的数个操作（本例中是 5 个）中的一个操作，能执行多达 11 个传统的微处理器操作。多媒体专用操作与 VLIW 、 RISC 或其它体系结构相结合，在多媒体应用中产生巨大的信息通量。本发明允许使用 32 、 64 、 128 …位的寄存器

存放数据，执行这些“多媒体”运算。

附图 2 所示是 CPU 寄存器结构的一个例子。本实施例中的 CPU 有 128 个 32 位通用寄存器，分别以 r0 … r127 记之。本实施例中寄存器 r0 和 r1 实际上作为专用寄存器使用，寄存器 r2 至 r127 才真正被当作通用寄存器。

本系统中，处理器每隔一个时钟周期发出一条长指令，每个这种指令包括几个操作（本实施例中有 5 个操作），每个操作都类似一条 RISC 机器指令，所不同的是操作的执行是有条件的，取决于某通用寄存器的内容。

寄存器中的数据可以是一例如一整数表示或浮点表示。

本实施例中，整数分为‘带符号整数’和‘无符号整数’两种，以二进制数 2 的补码位模式表示。整数算术运算不会产生陷阱。如果运算结果不可被表示，则返回的位模式是特定于操作的，如在对各运算的说明部分所定义的。典型的情况有：对常规加减类操作实行回绕式处理，DSP 类型的操作返回值为可表示的最高值或最低值，64 位结果(例如无符号整数积)的返回值为其中的最低 32 位。

因为本实施例是 32 位结构的，它不用浮点表示法表示多媒体运算中使用的数据的值。不过很显然，对 64 位，128 位…结构的系统而言，可以用浮点表示法来表示多媒体运算中使用的数据值。例如，单精度（32 位）IEEE-754 浮点算术表示法和/或双精度（64 位）IEEE-754 浮点表示法就可以用来表示数据值。

在本发明的结构体系中，所有运算均可选用“保护”（guarded）功能。保护运算有条件地执行运算，其条件值存放在保护寄存器（rguard）中。例如，一个保护的整数加法指令的写法是：

IF r23 iadd r14 r10->r13

本例的意思是，“if r23 then r13 = r14 + r10”。语句“if r23”根据 r23 中值的最低有效位（LSB）的内容来判断“真”与“假”。因此，根据 r23 中值的最低有效位（LSB）的内容，r13 要么不变，要么被设置为等于 r14 与 r10 的整数和。例如，在本发明的本实施例中，如果 LSB 被判断为 1，就对目的寄存器(rdest)（本例中为 r13）进行写操作。保护用于控制对程序员可见的系统状态一如寄存器值、存储器内容及设备状态一等的影响。

本发明中的存储器是字节可寻址的。数据的寄存或存储采用“自然对齐”方式，就是说，寄存或存储一个16位数据时，寻找的地址数是2的倍数；寄存或存储一个32位数据时，寻找的地址数是4的倍数。熟悉本领域的人很容易据此修改。

计算操作是寄存器对寄存器的操作。一个特定的操作，要有一个或两个寄存器作为操作对象，操作的结果被写入一个目的寄存器（rdest）。

定制操作属于特殊的计算操作范畴，象是正常的计算操作，而通用CPU中又没有这些定制操作。定制操作的优点在于允许执行例如多媒体应用软件中的运算。本发明的定制操作是为显著提高重要的多媒体应用软件及其它应用软件的性能而设计的、功能强大的专用操作。如果在应用程序的源码中适当地结合定制操作，会能使该应用程序充分利用本发明的微处理器（例如Philips公司的Trimedia TM-1芯片）的高度并行操作的潜力。

无论是通用应用软件还是基于微处理器的嵌入应用，人们都希望用高级语言进行程序设计。为了有效地支持优化编译器和简单化的编程模型，要求微处理器的体系结构具有一定的特点，例如有较大的线性地址空间、通用寄存器以及直接支持使用线性地址指针的寄存器对寄存器操作功能。

本发明允许利用系统的全部资源。举例来说，利用32位资源同时处理两个16位数据项或4个8位数据项。这种资源利用只需增加极少的的费用投入就能极大地改进系统的性能。而且，这种资源利用能从在标准的微处理器资源中获得较高的执行速度。

有些功能强的定制操作不使用条件转移语句，这有助于调度程序有效地利用目前系统中（例如使用TM-1指令的Philips TM-1芯片）每一条指令的五个操作槽（slot）。充满所有五个操作槽，对于计算密集的多媒体应用软件中的内部循环过程，具有特别重要的意义。定制操作帮助本发明以尽可能低的代价换取极高的多媒体操作性能。

表1是本发明中定制操作一览表。有些定制操作有不同的格式，其区别表现于对操作数及结果的不同处理上。表示不同格式命令的助记符，试图对上述不同处理加以区分，以便于用户选择合适的操作。当然，每个操作也可以使用完全不同的助记符或名称。

表 1：按功能分类的定制操作一览表

功能	助记符	说明
DSP 绝对值	dspiabs	带符号 32 位值的绝对值的剪切值
	dspidualabs	两个带符号 16 位半字的绝对值的剪切值
DSP 加	dspiadd	带符号 32 位值之和的剪切值
	dspuadd	无符号 32 位值之和的剪切值
	dspidualadd	两个带符号 16 位半字值之和的剪切值
	dspuquadaddui	四个带符号/无符号字节值之和的剪切值
DSP 积	dspimul	带符号 32 位值乘积的剪切值
	dspumul	无符号 32 位值乘积的剪切值
	dspidualmul	两个带符号 16 位半字值乘积的剪切值
DSP 差	dspisub	带符号 32 位值之差的剪切值
	dspusub	无符号 32 位值之差的剪切值
	dspidualsub	两个带符号 16 位半字值之差的剪切值
积之和	ifir16	带符号 16 位半字值乘积之和
	ifir8ii	带符号字节值乘积之和
	ifir8ui	带符号与无符号字节值乘积之和
	ufir16	无符号 16 位半字值乘积之无符号和
	ufir8uu	无符号字节值乘积之无符号和
合并	mergelsb	合并最低有效字节
	mergemsb	合并最高有效字节
打包	pack16lsb	打包最低有效 16 位半字
	pack16msb	打包最高有效 16 位半字
	packbytes	打包最低有效字节
字节平均值	quadavg	四个无符号字节的平均值
字节积	quadumulmsb	四个 8 位无符号字节对的积的最高字节
动态评估	ume8ii	带符号 8 位之差的绝对值的无符号的和
	ume8uu	无符号 8 位之差的绝对值之无符号和

剪切	iclipi	将一带符号值剪切为另一带符号值
	uclipi	将一带符号值剪切为另一无符号值
	uclipu	将一无符号值剪切为另一无符号值

下面举例说明本发明所述的一个定制操作的使用方法。本例通过一个字节矩阵变换，简要说明了定制操作是如何显著提高应用软件小型内核的处理速度的。正如大多数定制操作应用所反映的那样，本例中定制操作的效能体现在其对多个数据项进行并行处理的能力上。

本例的任务是对存储器中的一个 4 × 4 字节压缩矩阵进行转置。该矩阵中的数据可以是例如 8 位象素值。附图 3（ a ）表示转置前后矩阵数据在存储器中的位置。附图 3（ b ）用标准的数学语言叙述了本例要执行的任务。

用传统微处理器指令执行这个过程，其指令直截了当，但却耗费时间。一种方法是，执行 12 个字节加载指令来加载字节（ 16 个字节中只有 12 个需要易位），再用 12 个字节存储指令将它们存放回存储器中的新位置上。另一种方法是执行 4 个字（ Word ）加载指令，将已加载各字按字节在寄存器中重新排列，然后在执行 4 个字（ Word ）存储指令。不幸的是，在寄存器中将个字节重新排列，需要用大量的指令来进行适当的字节屏蔽和字节移位操作，而第一种方法总共进行 24 次加载和存储操作，实质上使用了加载/存储单元中的数据移位和数据屏蔽硬件，因而，该方法所需的指令序列长度较短。

进行 24 次加载和存储操作的问题是，加载和存储操作的本身速度并不快：至少要访问高速缓冲存储器，可能还要访问存储器体系中速度更慢的某些层次；再者说，进行字节加载和存储操作，通过字长 32 位的数据接口访问高速缓冲存储器或存储器，这是对高速缓冲存储器或存储器数据接口资源的一种浪费。因此，需要有一种充分发挥高速缓冲存储器/存储器带宽的优点、同时又不需要过度数量的字节操作指令的快速算法。

本发明含有对字节和 16 位半字（ halfwords ）直接和并行地进行合并（ mergemsb 和 mergelsb ）和打包（ pack16msb 和 pack16lsb ）的操作指令。这些指令中有四条可应用于本例中来加快将字节打包成字的处理速度。

附图 4 所示的是这些指令在字节矩阵置换例中的应用。附图 5

（a）所示是进行字节矩阵置换所需的一列操作命令，将其汇编成实际指令时，这些定制操作将尽可能地被相互组合，例如组合成每个指令含有5个操作。附图5（a）中的低级代码在此仅作示意之用。

附图5（a）中第一组4个字加载操作（ld32d）将输入矩阵中打包的字送入寄存器r10、r11、r12和r13；第二组4个合并操作（mergemsb和mergelsb）在寄存器r14、r15、r16和r17中生成中间结果；第三组4个打包操作命令（pack16msb和pack16lsb）对源操作数进行替换，假如为了进行其它计算需要保留原矩阵操作数（有一种TM－1优化C编译器可以自动执行这种分析），该组操作可以将转置后的矩阵另存到各个独立的寄存器中。本例中，转置矩阵被用st32d存入四个寄存器r18、r19、r20和r21中。最后4个字存储操作命令将置换后的矩阵存回到存储器中。

因此，使用本发明的定制操作，进行该字节矩阵置换需要4个字操作和4个字存储操作（至少这个数）以及8个寄存器到对寄存器的数据处理操作，总共是16次操作，换言之，字节矩阵置换的速度是平均每个字节一次操作。附图5（b）是一段对应的C语言程序。

就本例而言，运用基于定制操作的算法与需要24次字节加载和存储操作的简单算法相比，表面上看其优点也就是减少了8次操作（减少了33％），而实际上，其优点远不止于此。首先，运用定制操作时，对存储器的访问次数从24次减少到8次，即减少到原需次数的三分之一，而存储器操作速度要比寄存器间操作速度慢（本例中所用定制操作就是后一种操作），所以，减少访问存储器的次数非常重要。

第二，基于定制操作的代码的使用，使本系统采用的编译系统（TM－1系统）利用TM－1微处理器硬件潜在功能的能力得到加强。具体地说，当访问存储器的次数与寄存器间操作的次数相当时，编译系统更容易生成一个对代码的优化调度方案。一般来说，高性能的微处理器在一个时钟周期内访问存储器的次数有一定的限度，如果代码很长，又都是访问存储器的语句，就会在长的TM－1指令中产生空操作槽，从而浪费了硬件的潜在功能。

如本例所示，使用本发明的定制操作，可以减少进行一次计算所需操作的绝对次数，也有助于编译系统生成充分利用所用CPU潜能的程序码。对于诸如用于例如一个完整MPEG视频解码算法的MPEG图像

恢复系统以及动态评估（ motion-estimation ）内核等其它应用软件，使用本发明的定制操作也有益处。当然，本发明的定制操作的作用并不仅仅限于这些。

表 1 所列的为本发明包含的定制操作。以下将对这些定制操作逐一地进行详细说明。在以下的功能语句代码中，使用了标准符号、文法等。例如， temp1 和 temp2 代表两个临时寄存器；再例如，功能 temp1<-sign_ext16to32(rsrc1<15:0>)表示将寄存器 rsrc1 中 15:0 位（位 0 至位 15 ）的内容装入 temp1 寄存器，其中符号位（本例中为第 15 位）被扩展至 16 至 32 位（符号位扩展）；同理， temp2<-sign_ext16to32(rsrc1<16:31>)表示将寄存器 rsrc1 中的第 16 位至第 31 位的内容被提取（并被置于第 0 位至第 15 位用于计算），其中符号位（本例中为第 31 位）被扩展至 16 至 32 位。扩展符号位用于带符号数值上，本例中的带符号数值为带符号整数。对于无符号数值，使用的是填零方法。表示填零的记法与表示符号扩展的记法类似，例如，zero_ext8to32(rsrc1<15:0>)表示 15 至 0 位的值被用于计算， 8 至 32 位要被填零。源寄存器 rsrc1 与 rsrc2 及目的寄存器 rdest 可以是上述任何空闲的寄存器。

以下所列各操作中，每个操作可以选用保护（ guard ）的方式，其在寄存器 rguard 中定义。如果采用保护方式，则在本例中，其最低有效位（ LSB ）决定了是否对目的寄存器进行修改。在本例中，如果 rguard 中的最低有效位的值为 1 ，则对目的寄存器 rdest 进行写操作，否则就不改变 rdest 的值。

dspiabs: dspiabs 操作计算一个带符号数的绝对值的剪切值，伪码记作 h_dspiabs:（硬件 dspiabs ）。该操作具有以下功能:

```
if rguard then {
  if rsrc1>=0 then
     rdest<-rsrc1
else if rsrc1=0x800000000 then
        rdest<-0x7fffffff
else    rdest<-rsrc1
}
```

dspiabs 操作是一个伪操作，由调度程序转换为 h_dspiabs，其第一

个参数为常量零，第二个参数等于 dspiabs 的参数。在汇编源程序文件中一般不使用伪操作。 h_dspiabs 执行与 dspiabs 相同的功能，只不过前者需要一个零作为其第一个参数。

dspiabs 计算 rsrc1 的绝对值，将结果剪切（ clip ）成[2^{31}-1 … 0]即[0x7fffffff … 0]范围内的值，并将该剪切值存入目的寄存器 rdest 。所有值都是带符号整数。

dspidualabs: dspidualabs 操作计算两个带符号 16 位半字（ halfword ）的绝对值的剪切值，伪码记作 h_dspidualabs:（硬件 dspidualabs ）。该操作具有以下功能:

```
if rguard then {
  temp1<-sign_ext16to32(rsrc1<15:0>)
  temp2<-sign_ext16to32(rsrc1<31:16>)
  if temp1=oxffff8000 then temp1<-0x7fff
  if temp2=oxffff8000 then temp2<-0x7fff
  if temp1<0 then temp1<- -temp1
  if temp2<0 then temp2<- -temp2
  rdest<31:16><-temp2<15:0>
  rdest<15:0<-temp1<15:0>
}
```

dspidualabs 操作是一个伪操作，由调度程序转换为双参数的 h_dspidualabs ，本例中，其第一个参数为常量零，第二个参数等于 dspidualabs 的参数。

dspidualabs 操作功能是分别计算寄存器 rsrc1 内高 16 位半字（ halfword ）和低 16 位半字的两个 16 位带符号数绝对值的剪切值。两个绝对值均被剪切为[0x0 … 0x7fff]范围内的值，并被分别存入目的寄存器 rdest 中的两个对应的半字位置。所有值都是 16 位带符号整数。h_dspidualabs 操作执行与 dspidualabs 相同的功能，只不过前者需要一个零作为其第一参数。

dspiadd: dspiadd 操作计算带符号数之和的剪切值，该操作具有以下功能:

```
if rguard then {
  temp<-sign_ext32to64(rsrc1)+ sign_ext32to64(rsrc2)
```

```
if temp<0xffffffff80000000 then
  rdest<- 0x80000000
else  if temp>0x00000007fffffff then
  rdest<-0x7fffffff
else
rdest<-temp
}
```

如附图 6 所示， dspiadd 操作计算 rsrc1+rsrc2 的带符号和，将结果剪切成[2^{31}-1 … -2^{31}]即[0x7fffffff … 0x80000000]范围内的 32 位带符号值，并将该剪切值存入目的寄存器 rdest。所有值都是带符号整数。

dspuadd: dspuadd 操作计算无符号数之和的剪切值，该操作具有以下功能:

```
if rguard then {
  temp<-zero_ext32to64(rsrc1)+ zero_ext32to64(rsrc2)
  if （无符号） temp>0x00000000ffffffff then
      rdest <- 0xffffffff
  else
      rdest<-temp<31:0>
}
```

如附图 7 所示， dspuadd 操作计算 rsrc1+rsrc2 的无符号和，将结果剪切成[2^{31}-1 … 0]即[0xffffffff … 0]范围内的值，并将该剪切值存入目的寄存器 rdest。

dspidualadd: dspidualadd 操作计算两个带符号 16 位半字之和的剪切值，该操作具有以下功能:

```
if rguard then {
  temp1<-sign_ext16to32(rsrc1<15:0>)+sign_ext16to32(rsrc2<15:0>)
  temp2<sign_ext16to32(rsrc1<31:16>)+sign_ext16to32(rsrc2<31:16>)
  if   temp1<0xffff8000 then temp1 <- 0x8000
  if   temp2=0xffff8000 then temp2 <- 0x8000
  if   temp1>0x7fff then temp1 <- 0x7fff
  if   temp2<0x7fff then temp2 <- 0x7fff
  rdest<31:16> <- temp2<15:0>
```

```
    rdest<15:0> <- temp1<15:0>
}
```

如附图 8 所示， dspidualadd 操作分别计算 rsrc1 与 rsrc2 中两对 16 位半字的 16 位带符号和的剪切值。两个和数值均被剪切（ clip ）成 $[2^{15}-1 \cdots -2^{15}]$即[0x7fff … 0x8000]范围内的值，写入目的寄存器 rdest 中对应的半字位置。所有值均为 16 位带符号整数。

dspuquadaddui: dspuquadaddui 操作计算四个带符号与无符号字节（ byte ）之和的剪切值，该操作具有以下功能:

```
if rguard then {
  for (i<-0,m<-31,n<-24;i<4;i<-i+1,m<-m-8,n<-n-8){
  temp<-zero_ext8to32(rsrc1<m:n>)
  + sign_ext8to32(rsrc2<m:n>)
 if   temp<0 then
     rdest<m:n> <- 0
 else   if   temp > 0xff
        then
            rdest<m:n> <- 0xff
        else
            rdest<m:n> <- temp<7:0>
}
```

如附图 9 所示， dspuquadaddui 操作为 rsrc1 与 rsrc2 的四对 8 位字节分别计算出四个带符号和，其中假定 rsrc1 中的各字节值为无符号值， rsrc2 中的各字节值为带符号值。将四个和数剪切成无符号的 [255 … 0]即[0xff … 0]范围内的值，其结果是，所有四个字节和数均为无符号值。所有各计算均保留原数值精度。

dspimu1: dspimu1 操作计算带符号数乘法的剪切值，该操作具有以下功能:

```
if rguard then {
  temp<-sign_ext32to64(rsrc1)+sign_ext32to64(rsrc2)
  if temp<0xffffffff80000000 then
rdest<- 0x80000000
```

```
    else   if temp>0x000000007fffffff then
  rdest<-0x7fffffff
    else
    rdest<-temp<31:0>
}
```

如附图 10 所示， dspimu1 操作计算 rsrc1 × rsrc2 的积，将结果剪切成[2^{31}-1 … -2^{31}]即[0x7fffffff … 0x80000000]范围内的值，并将该剪切值存入目的寄存器 rdest 。所有值都是带符号整数。

dspumu1: dspumu1 操作计算无符号数乘积的剪切值，该操作具有以下功能:

```
if rguard then {
  temp<-zero_ext32to64(rsrc1) × zero_ext32to64(rsrc2)
  if （无符号值） temp>0x00000000ffffffff then
  rdest <- 0xffffffff
  else
  rdest<-temp<31:0>
}
```

如附图 11 所示， dspumu1 操作计算 rsrc1 × rsrc2 的无符号乘积值，将结果剪切成[2^{31}-1 … 0]即[0xffffffff … 0]范围内的值，并将该剪切值存入目的寄存器 rdest 。

dspidualmu1: dspidualmu1 操作计算两个带符号 16 位半字乘积的剪切值，该操作具有以下功能:

```
if rguard then {
temp1<-sign_ext16to32(rsrc1<15:0>) × sign_ext16to32(rsrc2<15:0>)
temp2<-sign_ext16to32(rsrc1<31:16>) × sign_ext16to32(rsrc2<31:16>)
  if   temp1<0xffff8000 then temp1 <- 0x8000
  if   temp2=0xffff8000 then temp2 <- 0x8000
  if   temp1>0x7fff then temp1 <- 0x7fff
  if   temp2<0x7fff then temp2 <- 0x7fff
  rdest<31:16> <- temp2<15:0>
  rdest<15:0> <- temp1<15:0>
}
```

如附图 12 所示， dspidualmu1 操作分别计算 rsrc1 与 rsrc2 中两对 16 位高和低半字的带符号乘积的 16 位剪切值，两个乘积数值均被剪切成[2^{15}-1 … -2^{15}]即[0x7fff … 0x8000]范围内的值，并被存入目的寄存器 rdest 中对应的半字位置。所有值均为 16 位带符号整数。

dspisub: dspisub 操作计算带符号减的剪切值，该操作具有以下功能:

```
if rguard then {
  temp<-sign_ext32to64(rsrc1) - sign_ext32to64(rsrc2)
  if temp<0xffffffff80000000 then
    rdest<- 0x80000000
  else   if temp>0x000000007fffffff then
    rdest<-0x7fffffff
  else
  rdest<-temp   <31:0>
}
```

如附图 13 所示， dspisub 操作计算 rsrc1-rsrc2 的算术差，将结果剪切成[0x80000000 … 0x7fffffff]范围内的值，并将该剪切值存入目的寄存器 rdest 。所有值都是带符号整数。

dspusub: dspusub 操作计算无符号减的剪切值，该操作具有以下功能:

```
if rguard then {
  temp<-zero_ext32to64(rsrc1) - zero_ext32to64(rsrc2)
  if （带符号值） temp<0 then
       rdest <- 0
  else
       rdest<-temp<31:0>
}
```

如附图 14 所示， dspusub 操作计算 rsrc1-rsrc2 的无符号差，将结果剪切成[0 … 0xffffffff]范围内的值，并将该剪切值存入目的寄存器 rdest 。

dspidualsub: dspidualsub 操作计算两个带符号 16 位半字减的剪切值，该操作具有以下功能:

```
if rguard then {
   temp1<-sign_ext16to32(rsrc1<15:0>)-sign_ext16to32(rsrc2<15:0>)
   temp2<-sign_ext16to32(rsrc1<31:16>)-sign_ext16to32(rsrc2<31:16>)
      if   temp1<0xffff8000 then temp1 <- 0x8000
      if   temp2<0xffff8000 then temp2 <- 0x8000
      if   temp1>0x7fff then temp1 <- 0x7fff
      if   temp2>0x7fff then temp2 <- 0x7fff
      rdest<31:16> <- temp2<15:0>
      rdest<15:0> <- temp1<15:0>
   }
```

如附图 15 所示， dspidualsub 操作分别计算 rsrc1 与 rsrc2 中两对 16 位高和低半字的带符号 16 位算术差数值，两个算术差数值均被剪切成$[2^{15}-1 \cdots -2^{15}]$即[0x7fff … 0x8000]范围内的值，并被存入目的寄存器 rdest 中对应的半字位置。所有值均为 16 位带符号整数。

ifir16: ifir16 操作计算两个带符号 16 位半字乘积之和，该操作具有以下功能:

```
if rguard then
   rdest<- sign_ext16to32(rsrc1<31:16>) ×
           sign_ext16to32(rsrc2<31:16>) +
           sign_ext16to32(rsrc1<15:0>) ×
           sign_ext16to32(rsrc2<15:0>)
```

如附图 16 所示， ifir16 操作先计算 rsrc1 与 rsrc2 中两对 16 位半字的乘积，再将两个乘积数相加后存入目的寄存器 rdest 。假定所有的半字均为带符号值，因此，两个乘积数及其和也均为带符号数值。所有计算均保留原来的数值精度。

ifir8ii: ifir8ii 操作计算带符号字节值乘积的带符号和，该操作具有以下功能:

```
if rguard then
   rdest<- sign_ext8to32(rsrc1<31:24>) ×
           sign_ext8to32(rsrc2<31:24>) +
           sign_ext8to32(rsrc1<23:16>) ×
           sign_ext8to32(rsrc2<23:16>) +
```

sign_ext8to32(rsrc1<15:8>) ×
sign_ext8to32(rsrc2<15:8>) +
sign_ext8to32(rsrc1<7:0>) ×
sign_ext8to32(rsrc2<7:0>)

如附图 17 所示， ifir8ii 操作先分别计算 rsrc1 与 rsrc2 中四个 8 位字节对的乘积，再将四个乘积数相加后写入目的寄存器 rdest。假定所有的字节值均为带符号值，因此，四个乘积数及其和也均为带符号数值。所有计算均保留原来的数值精度。

ifir8ui: ifir8ui 操作计算带符号与无符号字节值乘积的带符号和，该操作具有以下功能:

if rguard then
  rdest<- zero_ext8to32(rsrc1<31:24>) ×
    sign_ext8to32(rsrc2<31:24>) +
    zero_ext8to32(rsrc1<23:16>) ×
    sign_ext8to32(rsrc2<23:16>) +
    zero_ext8to32(rsrc1<15:8>) ×
    sign_ext8to32(rsrc2<15:8>) +
    zero_ext8to32(rsrc1<7:0>) ×
    sign_ext8to32(rsrc2<7:0>)

如附图 18 所示， ifir8ui 操作先分别计算 rsrc1 与 rsrc2 中四个 8 位字节对的乘积，再将四个乘积数相加后存入目的寄存器 rdest。假定 rsrc1 中的字节值为无符号值， rsrc2 中的字节值为带符号值，因此，四个乘积数及其和也均为带符号数值。所有计算均保留原来的数值精度。

ufir16: ufir16 操作计算无符号 16 位半字值乘积之和，该操作具有以下功能:

if rguard then {
  rdest<- zero_ext16to32(rsrc1<31:16>) ×
    zero_ext16to32(rsrc2<31:16>) +
    zero_ext16to32(rsrc1<15:0>) ×
    zero_ext16to32(rsrc2<15:0>)

如附图 19 所示， ufir16 操作先计算 rsrc1 与 rsrc2 中两对 16 位半

字的乘积，再将两个乘积数相加后存入目的寄存器 rdest。假定所有的半字均为无符号值，因此，两个乘积数及其和也均为无符号数值。所有计算均保留原来的数值精度。最后的积之和在被写入 rdest 之前被剪切成[0xffffffff … 0]范围的值。

ufir8uu: ufir8uu 操作计算无符号字节值乘积之和的无符号和，该操作具有以下功能:

```
if rguard then {
  rdest<- zero_ext8to32(rsrc1<31:24>) ×
          zero_ext8to32(rsrc2<31:24>) +
          zero_ext8to32(rsrc1<23:16>) ×
          zero_ext8to32(rsrc2<23:16>) +
          zero_ext8to32(rsrc1<15:8>) ×
          zero_ext8to32(rsrc2<15:8>) +
          zero_ext8to32(rsrc1<7:0>) ×
          zero_ext8to32(rsrc2<7:0>)
```

如附图 20 所示， ufir8uu 操作先分别计算 rsrc1 与 rsrc2 中四个 8 位字节对的乘积，再将四个乘积数相加后存入目的寄存器 rdest。假定所有的字节值均为无符号值。所有计算均保留原精度。

mergelsb: mergelsb 操作为合并最低有效字节，该操作具有以下功能:

```
if rguard then {
  rdest <7:0> <- rsrc2<7:0>
  rdest <15:8> <- rsrc1<7:0>
  rdest <23:16> <- rsrc2<15:8>
  rdest <31:24> <- rsrc1<15:8>
```

如附图 21 所示， mergelsb 操作从 rsrc1 与 rsrc2 各取两对最低有效字节合并存入 rdest 中。其中 rsrc2 中的最低有效字节被组合到 rdest 中的最低有效字节位置， rsrc1 中的最低有效字节被组合到 rdest 中的次低有效字节位置， rsrc2 中的次低有效字节被组合到 rdest 中的次高有效字节位置， rsrc1 中的次低有效字节被组合到 rdest 中的最高有效字节位置。

mergemsb: mergemsb 操作为合并最高有效字节，该操作具有以下

功能:

if rguard then {

rdest <7:0> <- rsrc2<23:15>

rdest <15:8> <- rsrc1<23:15>

rdest <23:16> <- rsrc2<31:24>

rdest <31:24> <- rsrc1<31:24>

如附图 22 所示， mergemsb 操作从 rsrc1 与 rsrc2 各取两对最高有效字节合并存入 rdest 中。其中 rsrc2 中的次高有效字节被放置到 rdest 中的最低有效字节位置， rsrc1 中的次高有效字节被放置到 rdest 中的次低有效字节位置， rsrc2 中的最高有效字节被放置到 rdest 中的次高有效字节位置， rsrc1 中的最高有效字节被放置到 rdest 中的最高有效字节位置。

pack16lsb: pack16lsb 操作为打包最低有效 16 位半字，该操作具有以下功能:

if rguard then {

rdest <15:0> <- rsrc2<15:0>

rdest <31:16> <- rsrc1<15:0>

如附图 23 所示， pack16lsb 操作从 rsrc1 与 rsrc2 各取一个最低有效 16 位半字组合存入 rdest 中。其中 rsrc1 中的最低有效 16 位半字被放置到 rdest 中的最高有效半字位置， rsrc2 中的最低有效 16 位半字被放置到 rdest 中的最低有效半字位置。

pack16msb: pack16msb 操作为打包最高有效 16 位半字，该操作具有以下功能:

if rguard then {

rdest <15:0> <- rsrc2<31:16>

rdest <31:16> <- rsrc1<31:16>

如附图 13 所示， pack16msb 操作从 rsrc1 与 rsrc2 各取一个最高有效 16 位半字组合存入 rdest 中。其中 rsrc1 中的最高有效 16 位半字被放置到 rdest 中的最高位半字位置， rsrc2 中的最高有效 16 位半字被放置到 rdest 中的最低位半字位置。

packbytes: packbytes 操作是打包最低字节，它具有以下功能:

if rguard then {

```
rdest <7:0> <- rsrc2<7:0>
rdest <15:8> <- rsrc1<7:0>
```

如附图 25 所示， packbytes 操作从 rsrc1 与 rsrc2 各取一个最低有效字节组合存入 rdest 中。其中 rsrc1 中的最低有效字节被放置到 rdest 中的次低有效字节位置， rsrc2 中的最低有效字节被放置到 rdest 中的最低有效字节位置。 rdest 中的两个最高有效字节位置被填零。

quadavg: quadavg 操作计算四个无符号字节的平均值。该操作具有以下功能:

```
if rguard then {
  temp<- (zero_ext8to32(rsrc1<7:0>)+zero_ext8to32(rsrc2<7:0>)+1)/2
   rdest<7:0> <-temp<7:0>
  temp <- (zero_ext8to32(rsrc1<15:8>)+
    zero_ext8to32(rsrc2<15:8>)+1)/2
   rdest<15:8> <-temp<7:0>
   temp <-(zero_ext8to32(rsrc1<23:16>)+
     zero_ext8to32(rsrc2<23:16>)+1)/2
   rdest<23:16> <-temp<7:0>
   temp <- (zero_ext8to32(rsrc1<31:24>)+
     zero_ext8to32(rsrc2<31:24>)+1)/2
   rdest<31:24> <-temp<7:0>
   }
```

如附图 26 所示， quadavg 操作对寄存器 rscr1 与 rscr2 中的四对 8 位字节分别计算平均值。假定所有字节均为无符号字节。每对字节的平均值的最低 8 位被写入目的寄存器 rdest 中与该对字节位置相对应的字节位置。不对上溢与下溢情况作检查。

quadumulmsb: quadumulmsb 操作计算四个无符号 8 位字节对的积的最高字节。该操作具有以下功能:

```
if rguard then {
  temp <- (zero_ext8to32(rsrc1<7:0>) × zero_ext8to32(rsrc2<7:0>))
  rdest<7:0> <-temp<15:8>
 temp <- (zero_ext8to32(rsrc1<15:8>) × zero_ext8to32(rsrc2<15:8>))
  rdest<15:8> <-temp<15:8>
```

```
temp <-(zero_ext8to32(rsrc1<23:16>) ×
  zero_ext8to32(rsrc2<23:16>))
 rdest<23:16> <-temp<15:8>
 temp <- (zero_ext8to32(rsrc1<31:24>) ×
  zero_ext8to32(rsrc2<31:24>))
 rdest<31:24> <-temp<15:8>
}
```

如附图27所示， quadumulmsb操作分别计算寄存器rscr1与rscr2中的四对8位字节的积。假定所有字节均为无符号字节。每对字节的16位积的最高8位被写入目的寄存器rdest中与该对字节位置相对应的字节位置。

ume8ii: ume8ii操作计算带符号8位字节之差的绝对值的无符号之和。该操作具有以下功能:

```
if rguard then
  rdest <- abs_val(sign_ext8to32(rsrc1<31:24>)-
                   sign_ext8to32(rsrc2<31:24>))+
           abs_val(sign_ext8to32(rsrc1<23:16>)-
                   sign_ext8to32(rsrc2<23:16>))+
           abs_val(sign_ext8to32(rsrc1<15:8>)-
                   sign_ext8to32(rsrc2<15:8>))+
           abs_val(sign_ext8to32(rsrc1<7:0>)-
                   sign_ext8to32(rsrc2<7:0>))
```

如附图28所示， ume8ii操作分别计算寄存器rscr1与rscr2中的四对带符号8位字节的差，并将这四个差的绝对值的和写入目的寄存器rdest。所有计算均保留原精度。

ume8uu: ume8uu操作计算无符号8位字节之差的绝对值之和。该操作具有以下功能:

```
if rguard then
  rdest <- abs_val(zero_ext8to32(rsrc1<31:24>)-
                   zero_ext8to32(rsrc2<31:24>))+
           abs_val(zero_ext8to32(rsrc1<23:16>)-
                   zero_ext8to32(rsrc2<23:16>))+
```

```
abs_val(zero_ext8to32(rsrc1<15:8>)-
        zero_ext8to32(rsrc2<15:8>))+
abs_val(zero_ext8to32(rsrc1<7:0>)-
        zero_ext8to32(rsrc2<7:0>))
```

如附图29所示，ume8uu操作分别计算寄存器rscr1与rscr2中的四对无符号8位字节的差，并将这四个差的绝对值的和写入目的寄存器rdest。所有计算均保留原精度。

iclipi: iclipi操作将一带符号值剪切为另一带符号值。该操作具有以下功能:

```
if rguard then
  rdest <- min(max(rscr1,-rscr2-1),rscr2)
```

iclipi操作返回scrc1在无符号整数范围-rscr2-1至rscr2内（包含边界值）的剪切值。假定参数rsrc1是带符号整数，rscr2是0至0x7fffffff范围内（包含边界值）的一个无符号整数。

uclipi: uclipi操作将一带符号值剪切为另一无符号值。该操作具有以下功能:

```
if rguard then
  rdest <- min(max(rscr1,0),rscr2)
```

uclipi操作返回scrc1在无符号整数范围0至rscr2内（包含边界值）的剪切值。假定参数rsrc1是无符号整数，rscr2无符号整数。

uclipu: uclipu操作将一无符号值剪切为另一无符号值。该操作具有以下功能:

```
if rguard then {
  if rscr1 > rscr2 then
    rdest <- rscr2
  else
    rdest <- rscr1
}
```

uclipu操作返回scrc1在无符号整数范围0至rscr2内（包含边界值）的剪切值。假定参数rsrc1及rscr2均为带符号整数。

通过运用以上的多媒体定制操作，应用软件具有以较低的成本使微处理器高度并行地执行多媒体功能的优点。

从以上说明应该了解的是，本发明可应用于用 VLIW、 RISC、超标量等指令格式的处理器高度并行处理。此外，熟悉本领域的人也可以容易地根据以上思想添加一些操作。例如，虽然本说明中未对四对字节剪切差的操作予以专门描述，但熟悉本领域的人可以容易地根据以上说明开发出这条操作命令。

本文至此就描述了用于执行多媒体功能的定制操作的方法和系统。

本文仅描述和显示了本发明的较佳实施例，但正如前所述，应该清楚，本发明能被应用于其它各种组合与环境中，并能在所表述的本发明思想的范围内加以修改。

说 明 书 附 图

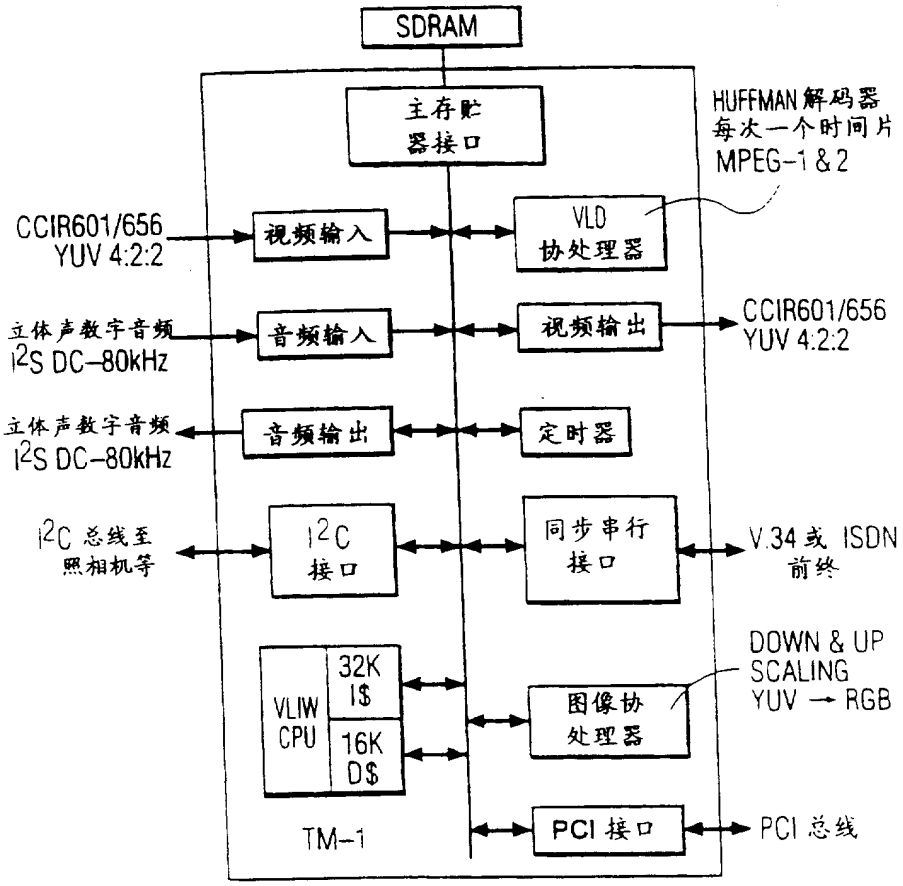

SDRAM
主存贮器接口
HUFFMAN解码器
每次一个时间片
MPEG-1 & 2
CCIR601/656
YUV 4:2:2
视频输入
VLD
协处理器
立体声数字音频
I2S DC-80kHz
音频输入
视频输出
CCIR601/656
YUV 4:2:2
立体声数字音频
I2S DC-80kHz
音频输出
定时器
I2C 总线至照相机等
I2C 接口
同步串行接口
V.34 或 ISDN 前终
DOWN & UP
SCALING
YUV → RGB
VLIW CPU
32K I$
16K D$
图像协处理器
TM-1
PCI 接口
PCI 总线

图 1

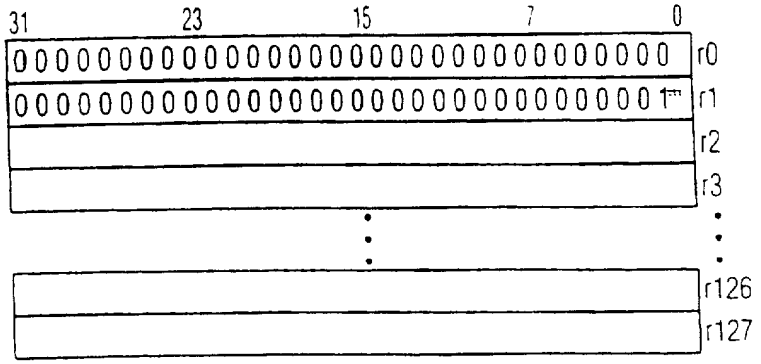

31
23
15
7
0
r0
r1
r2
r3
r126
r127

图 2

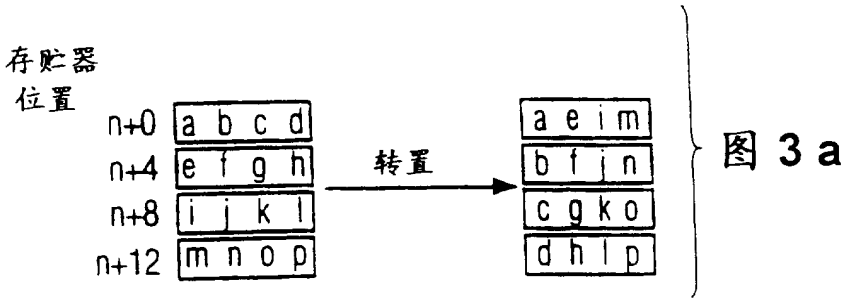

存贮器
位置
n+0
n+4
n+8
n+12
a b c d
e f g h
i j k l
m n o p
转置
a e i m
b f j n
c g k o
d h l p

图 3 a

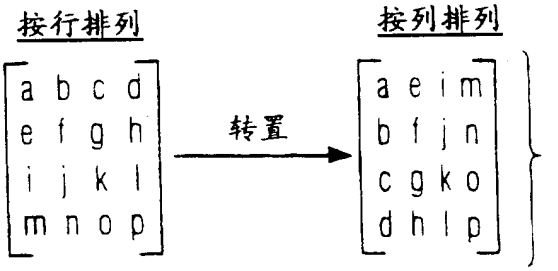

按行排列
按列排列
a b c d
e f g h
i j k l
m n o p
转置
a e i m
b f j n
c g k o
d h l p

图 3 b

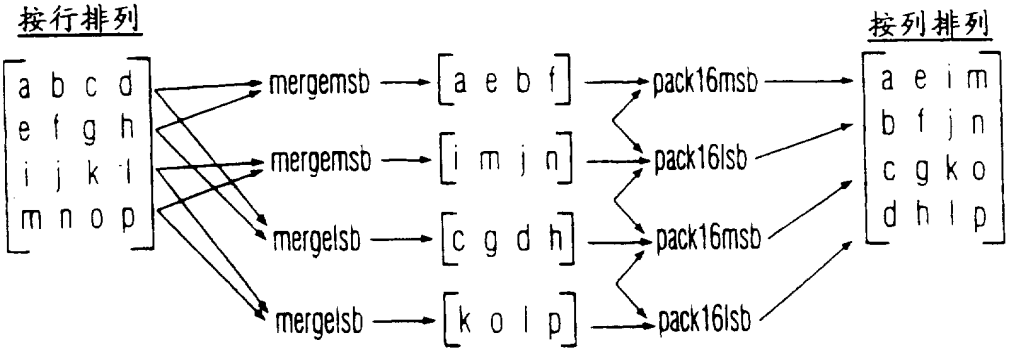

按行排列
按列排列
a b c d
e f g h
i j k l
m n o p
mergemsb
mergemsb
mergelsb
mergelsb
a e b f
i m j n
c g d h
k o l p
pack16msb
pack16lsb
pack16msb
pack16lsb
a e i m
b f j n
c g k o
d h l p

图 4

图 5 a

```
IF r1 ld32d(0)  r100→ r10
IF r1 ld32d(4)  r100→ r11
IF r1 ld32d(8)  r100→ r12
IF r1 ld32d(12)  r100→ r13

IF r1 mergemsb  r10 r11→ r14
IF r1 mergemsb  r12 r13→ r15
IF r1 mergelsb  r10 r11 → r16
IF r1 mergelsb  r12 r13 → r17
IF r1 pack16msb  r14 r15→ r18
IF r1 pack16lsb  r14 r15 → r19
IF r1 pack16msb  r16 r17→ r20
IF r1 pack16lsb  r16 r17 → r21

IF r1 st32d(0)  r101  r18
IF r1 st32d(4)  r101  r19
IF r1 st32d(8)  r101 r20
IF r1 st32d(12)  r101 r21
```

图 5 b

```
char matrix [4] [4] ;
        •
        •
        •
int  *m = (int  *)  matrix;

temp0 = MERGEMSB (m[0] ,  m[1]) ;
temp1 = MERGEMSB (m[2] ,  m[3]) ;
temp2 = MERGELSB (m[0] ,  m[1]) ;
temp3 = MERGELSB (m[2] ,  m[3]) ;

m[0] = PACK16MSB(temp0, temp1) ;
m[1] = PACK16LSB(temp0, temp1) ;
m[2] = PACK16MSB(temp2, temp3) ;
m[3] = PACK16LSB(temp2, temp3) ;
        •
        •
        •
```

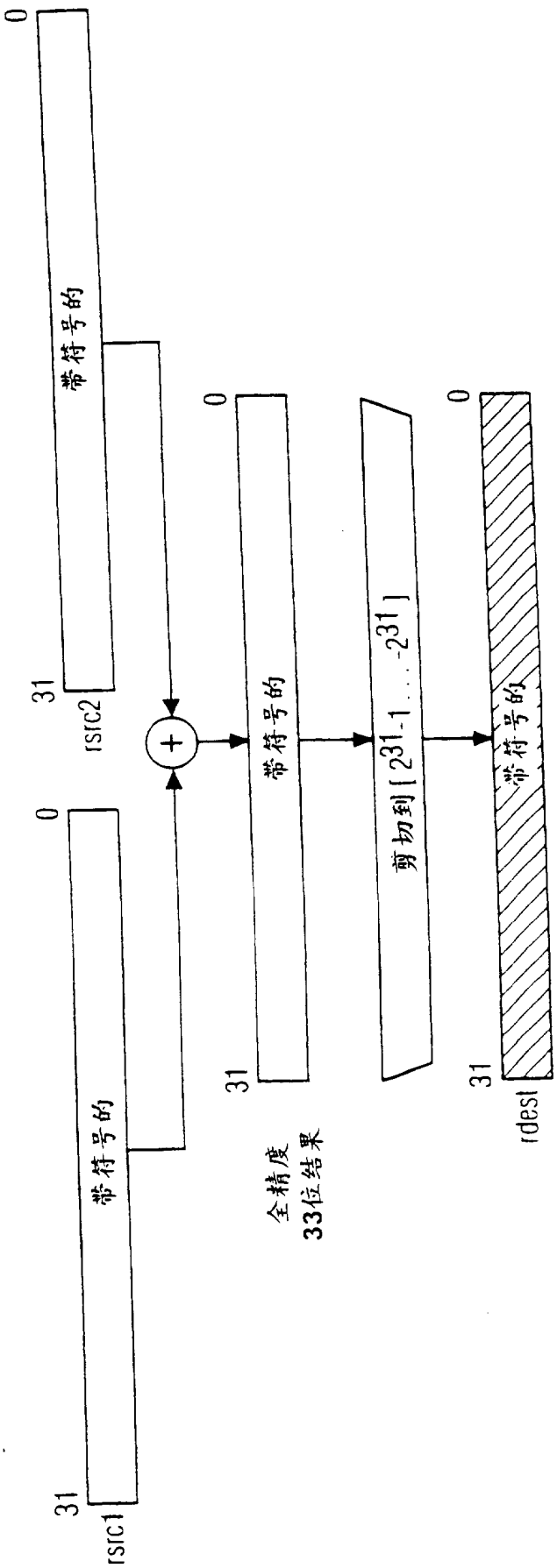

0
带符号的
31
rsrc2
0
带符号的
31
rsrc1
+
0
带符号的
31
全精度
33位结果
剪切到[2^{31}-1 ... -2^{31}]
0
带符号的
31
rdest

图 6

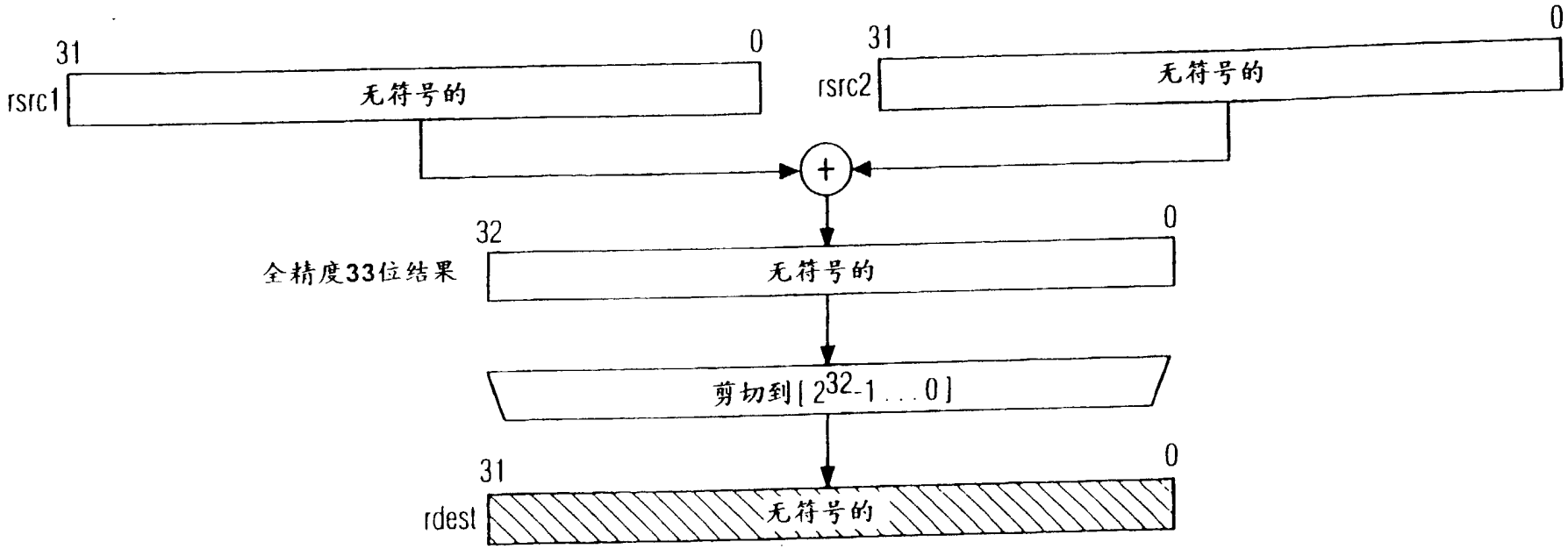

31
0
rsrc1
无符号的
31
0
rsrc2
无符号的
+
32
0
全精度33位结果
无符号的
剪切到[2^32-1...0]
31
0
rdest
无符号的

图 7

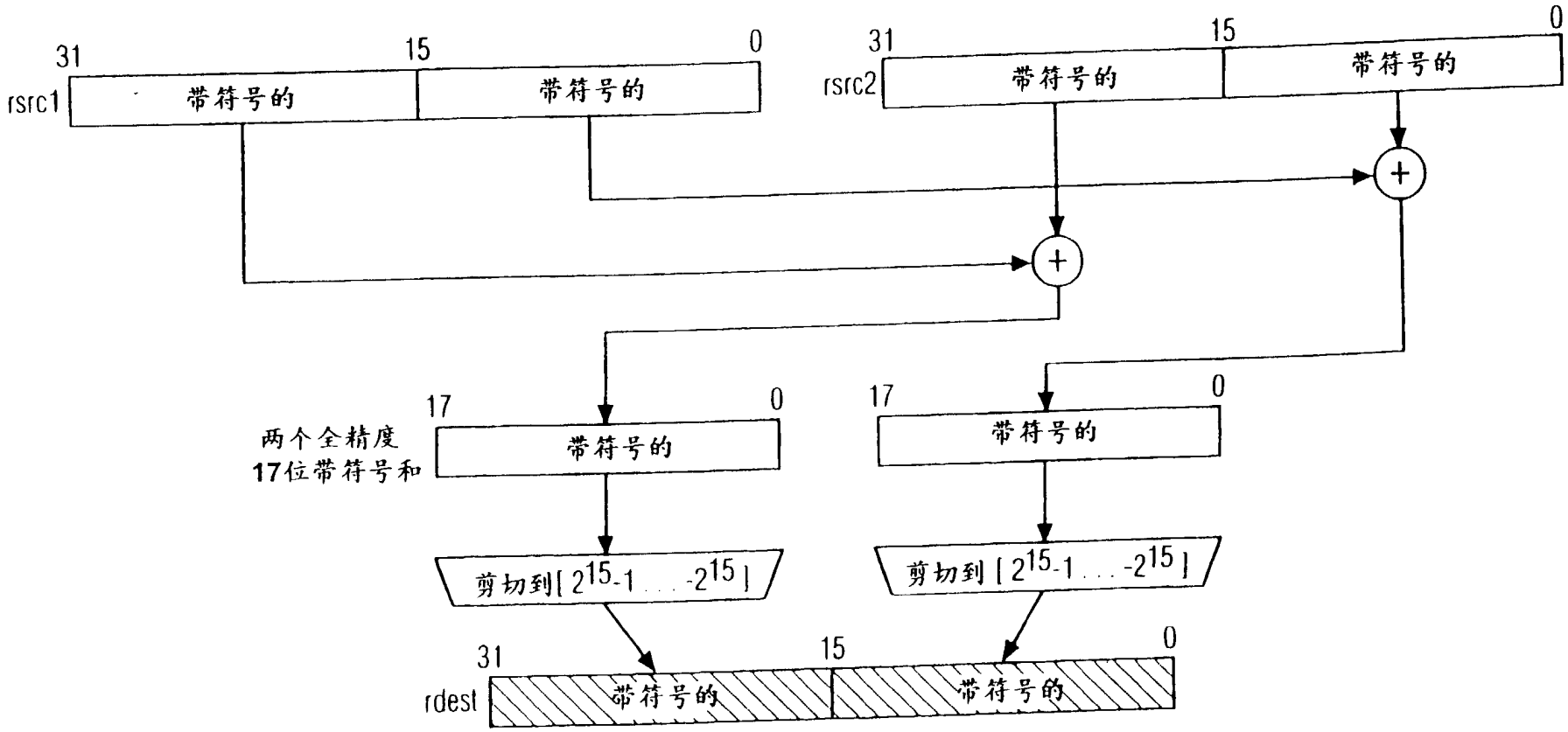

31
15
0
rsrc1
带符号的
带符号的
31
15
0
rsrc2
带符号的
带符号的
+
+
17
0
两个全精度
17位带符号和
带符号的
17
0
带符号的
剪切到[2^{15}-1 . . . -2^{15}]
剪切到[2^{15}-1 . . . -2^{15}]
31
15
0
rdest
带符号的
带符号的

图 8

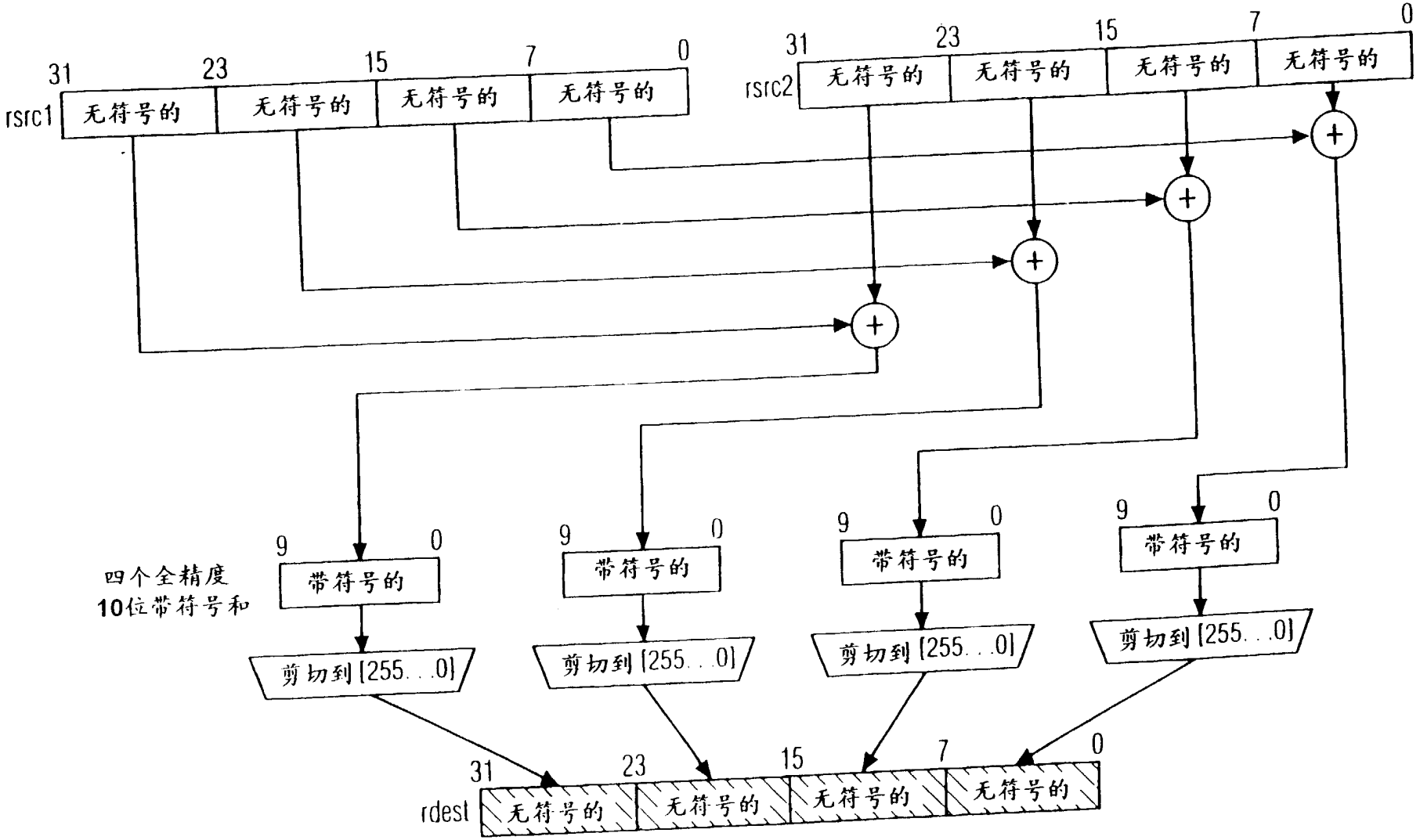

31
23
15
7
0
rsrc1
无符号的
无符号的
无符号的
无符号的
rsrc2
无符号的
无符号的
无符号的
无符号的
四个全精度
10位带符号和
9
0
带符号的
剪切到[255...0]
rdest

图 9

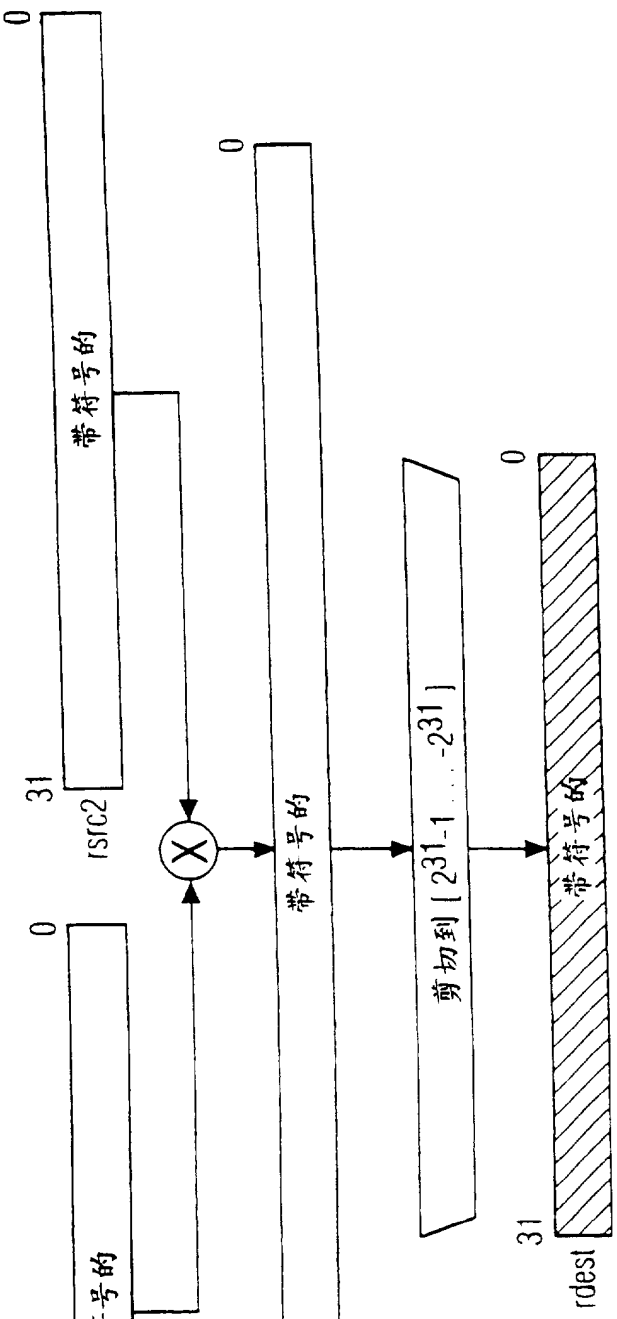

0
31
带符号的
rsrc2
0
31
带符号的
rsrc1
X
0
63
带符号的
全精度
64位结果
剪切到 [2^{31}-1 ... -2^{31}]
0
31
带符号的
rdest

图 10

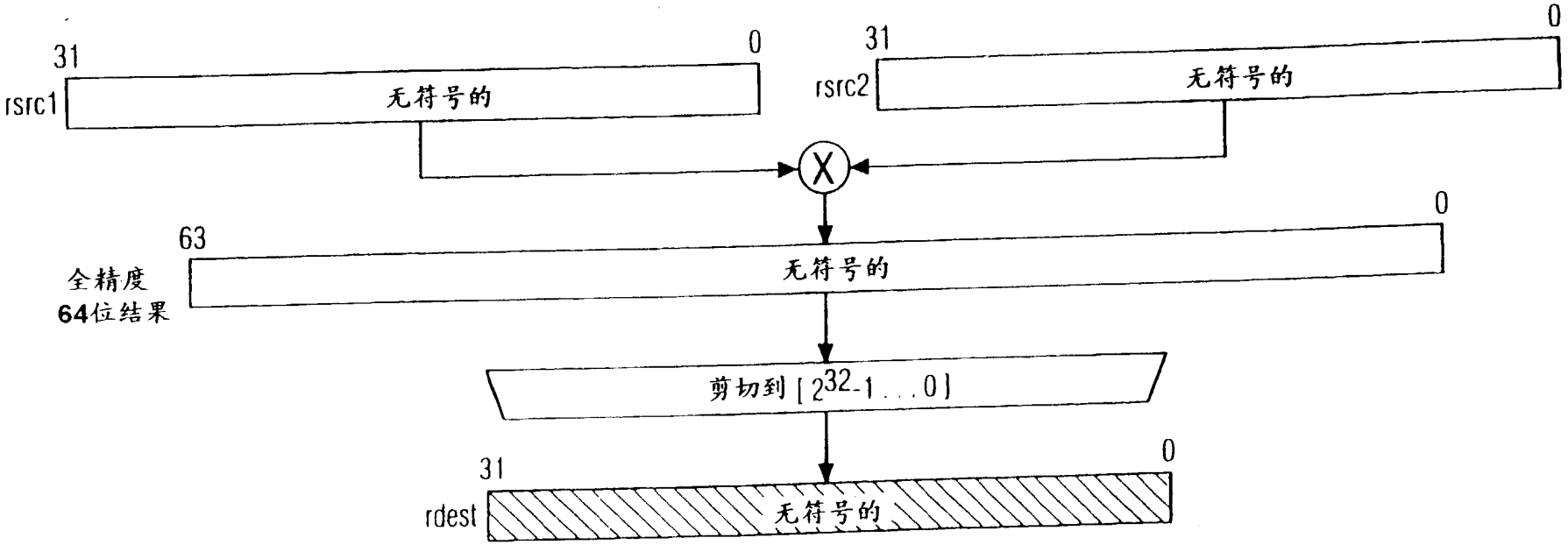

31
0
rsrc1
无符号的
31
0
rsrc2
无符号的
X
63
0
全精度
64位结果
无符号的
剪切到 [2^32-1 . . . 0]
31
0
rdest
无符号的

图 11

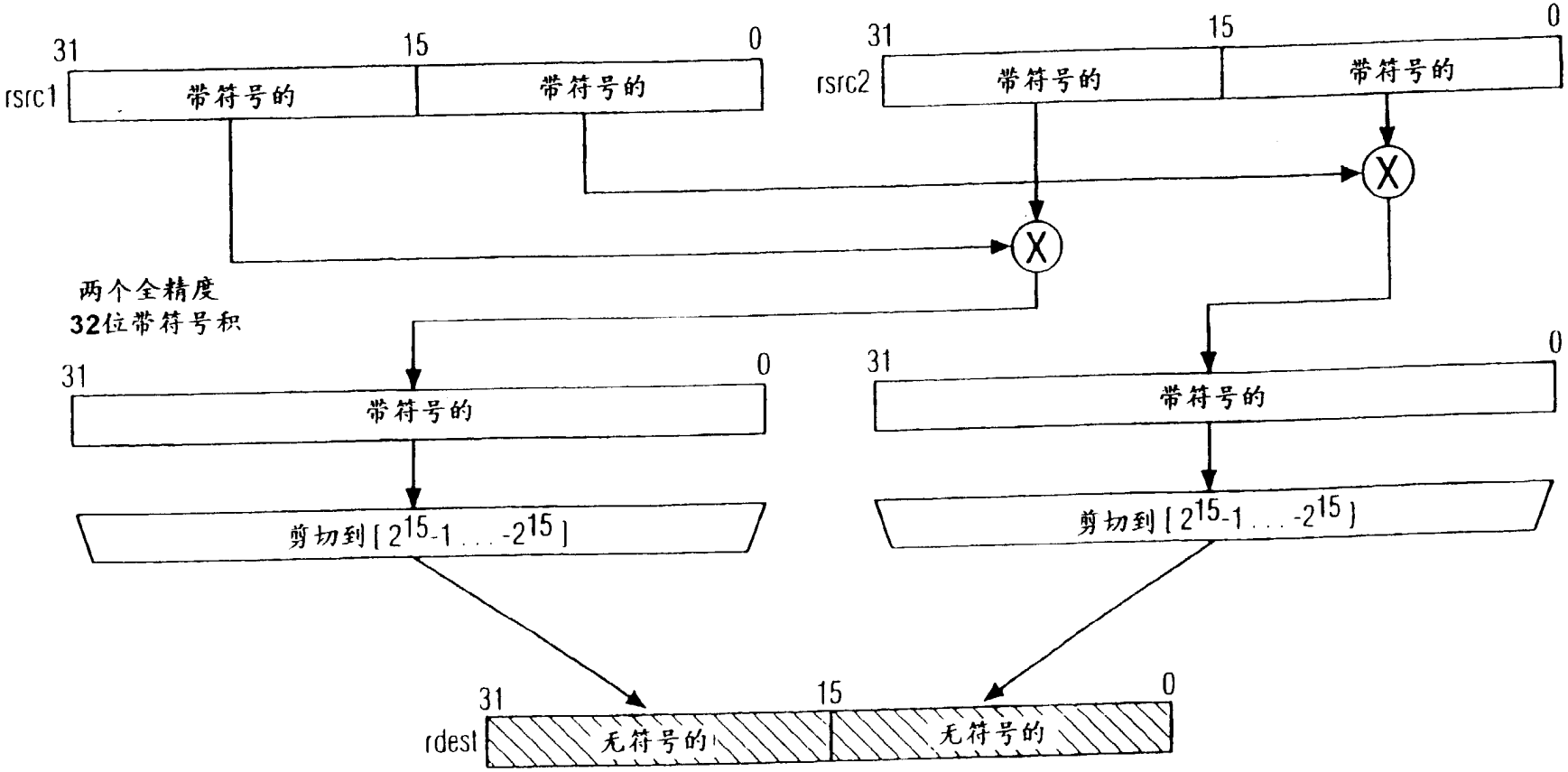

31
15
0
rsrc1
带符号的
带符号的
31
15
0
rsrc2
带符号的
带符号的
X
X
两个全精度
32位带符号积
31
0
带符号的
31
0
带符号的
剪切到 [2^15-1 . . . -2^15]
剪切到 [2^15-1 . . . -2^15]
31
15
0
rdest
无符号的
无符号的

图 12

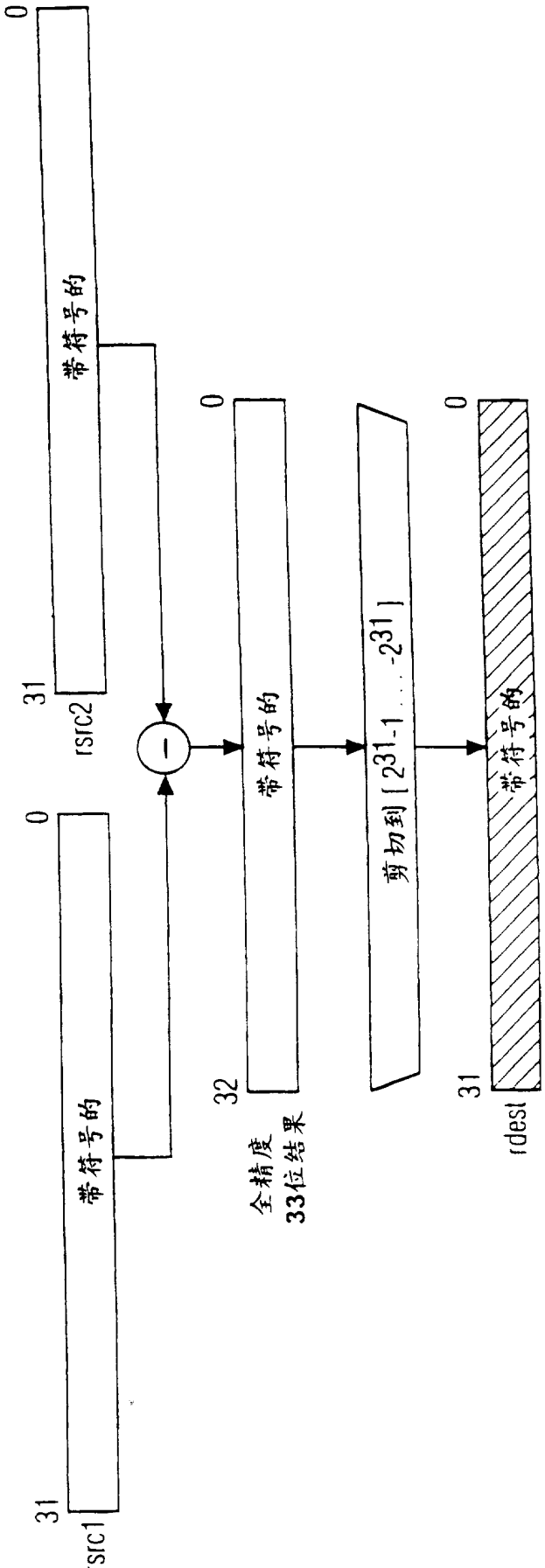

0
带符号的
31
rsrc2
0
带符号的
31
rsrc1
−
0
带符号的
32
全精度
33位结果
剪切到 [$2^{31}-1$ . . . -2^{31}]
0
带符号的
31
rdest

图 13

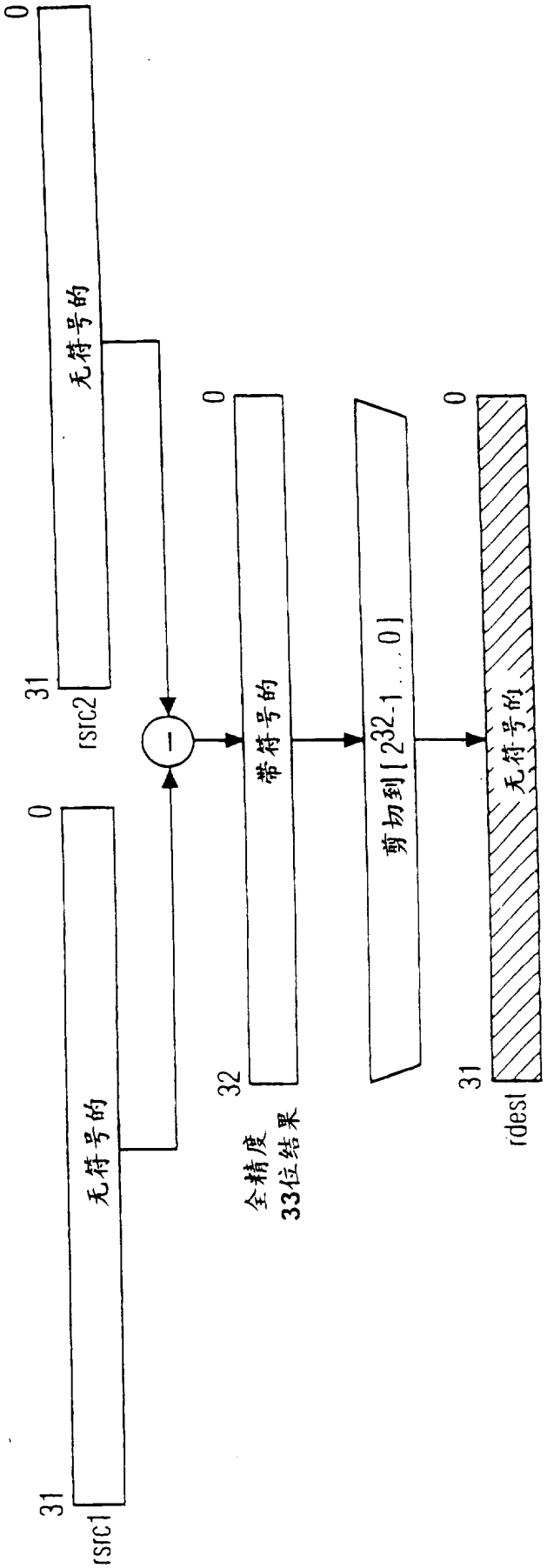

0
31
rsrc2
无符号的
0
31
rsrc1
无符号的
−
0
32
全精度
33位结果
带符号的
剪切到[2^{32-1}...0]
0
31
rdest
无符号的

图 14

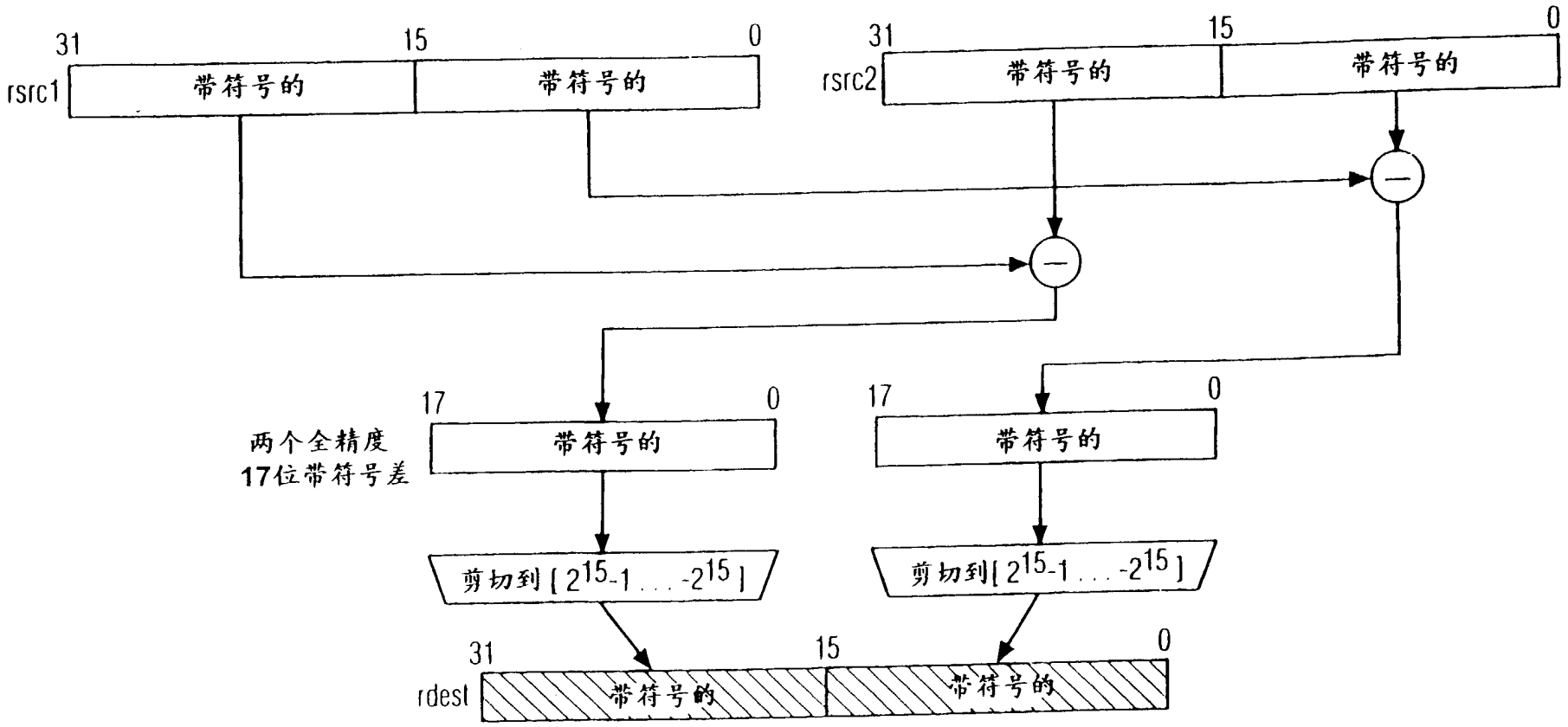

31
15
0
rsrc1
带符号的
带符号的
31
15
0
rsrc2
带符号的
带符号的
17
0
两个全精度
17位带符号差
带符号的
17
0
带符号的
剪切到[2^{15}-1 . . . -2^{15}]
剪切到[2^{15}-1 . . . -2^{15}]
31
15
0
rdest
带符号的
带符号的

图 15

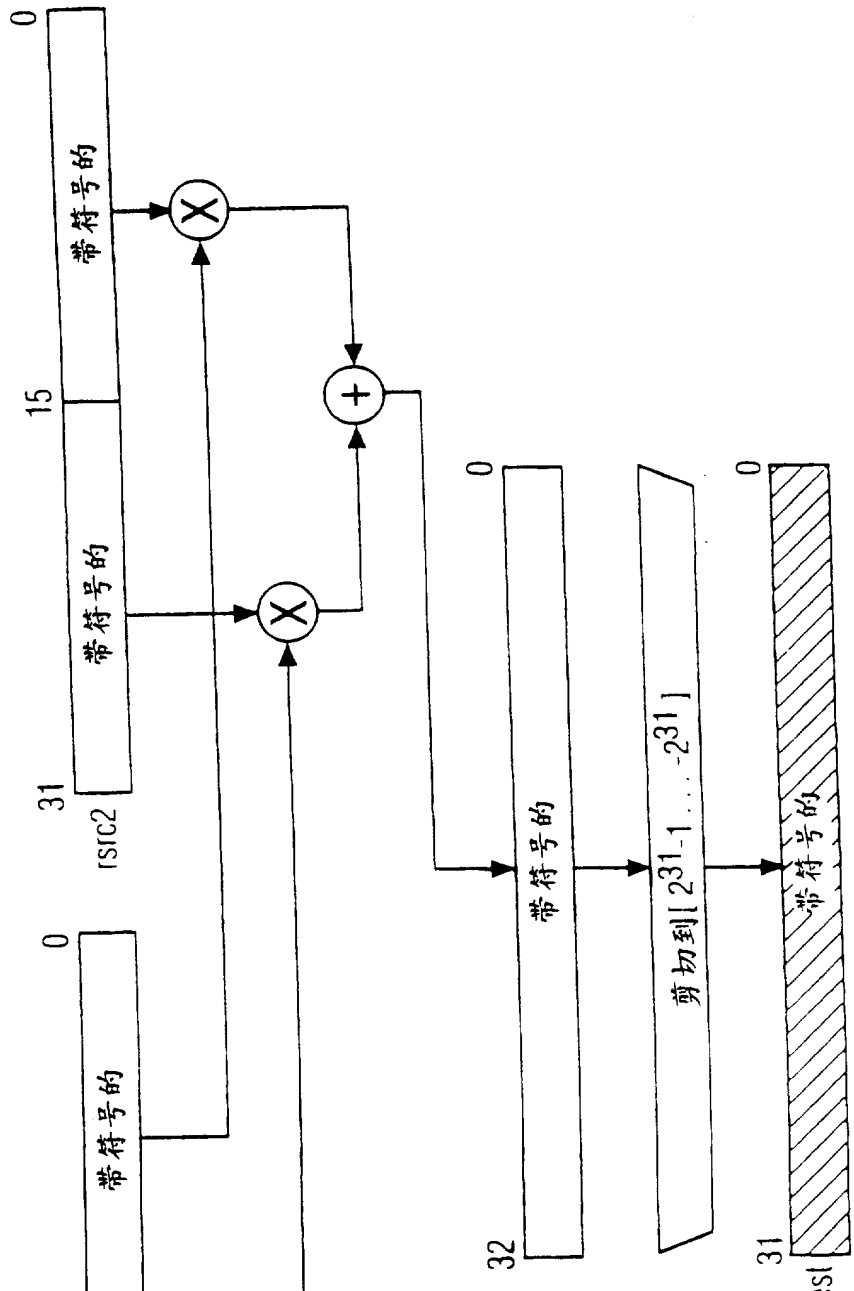

0
15
带符号的
31
rSIC2
0
15
带符号的
31
rSIC1
0
32
带符号的
全精度
33位结果
剪切到[$2^{31}-1$ . . . -2^{31}]
0
31
带符号的
rdest

图16

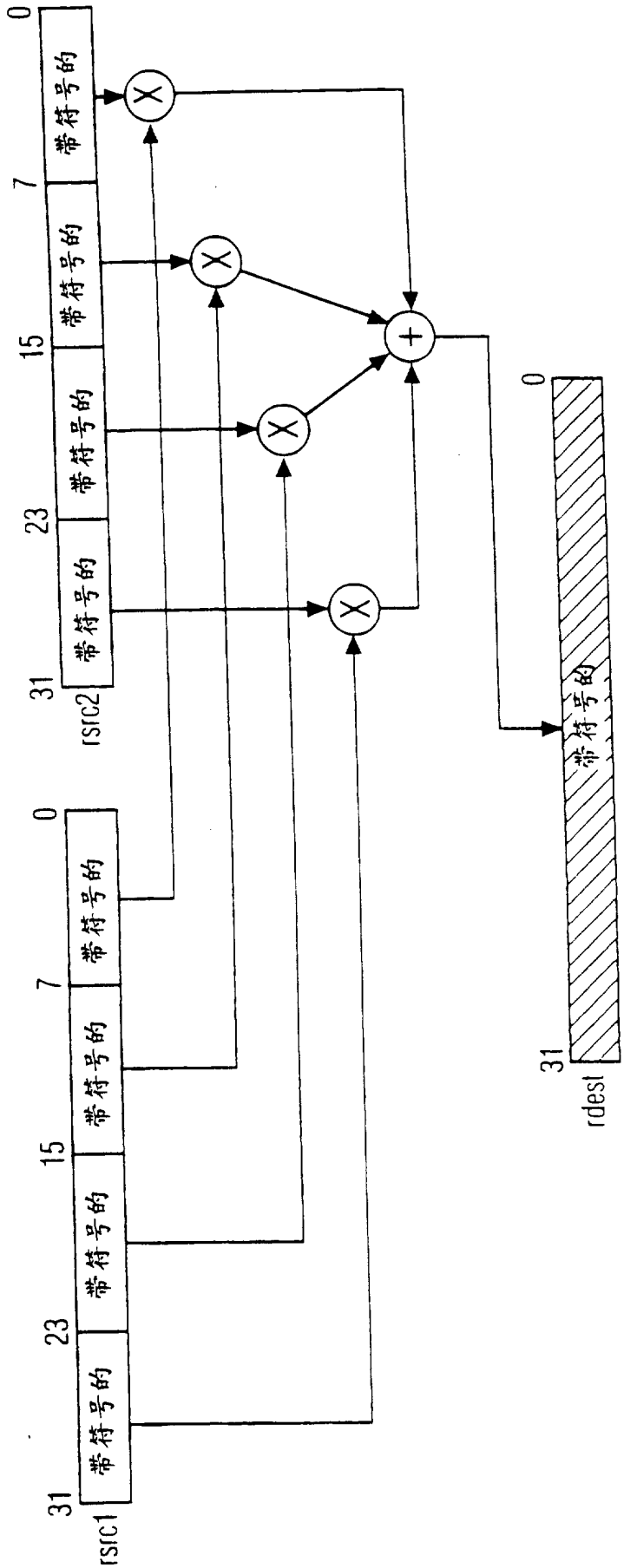


图 17

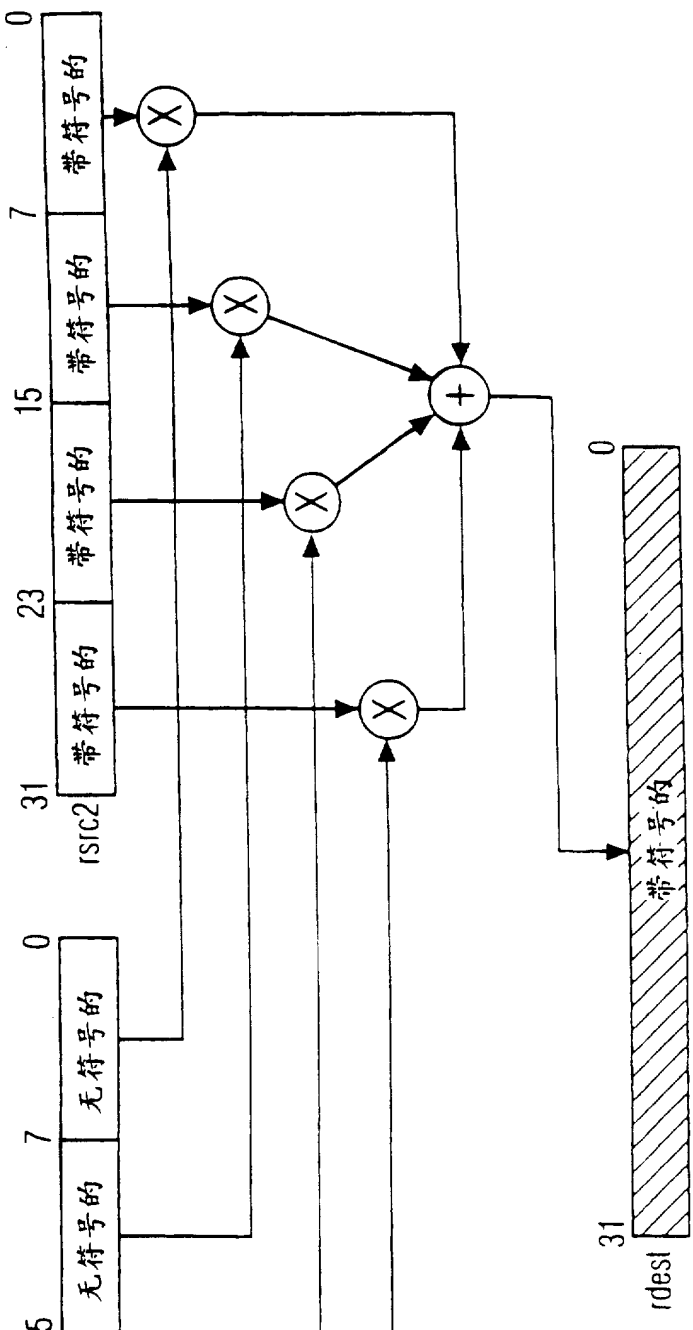


图 18

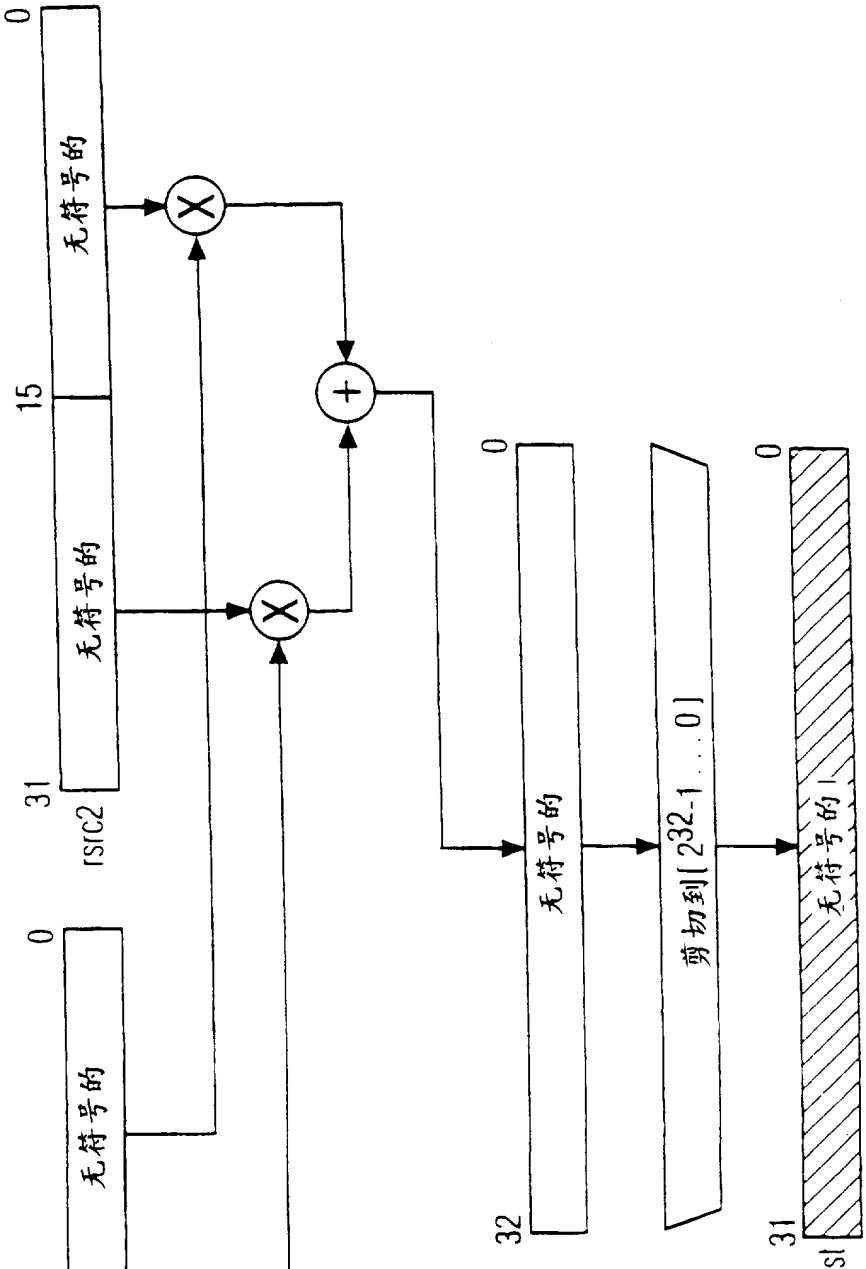

0
15
无符号的
无符号的
31
rSrc2
0
15
无符号的
无符号的
31
rSrc1
0
32
无符号的
全精度
32位结果
剪切到[232-1...0]
0
31
无符号的
rDest

图 19

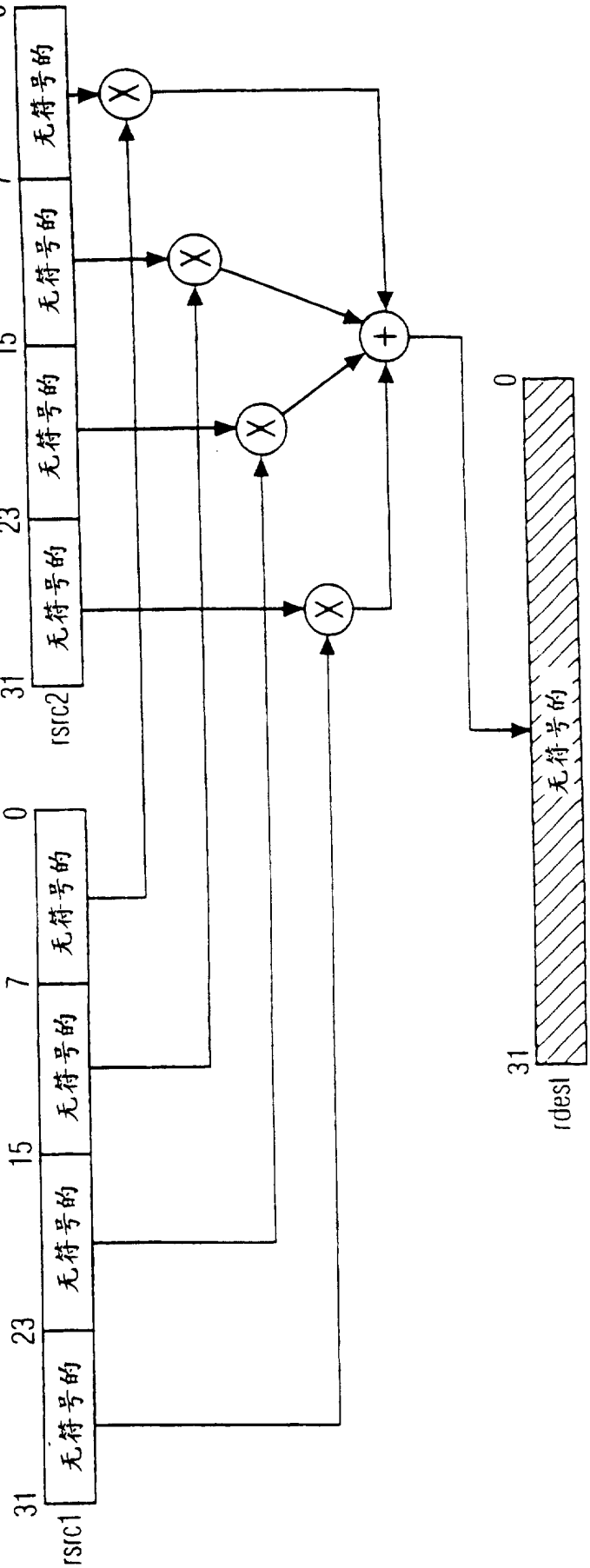


图 20

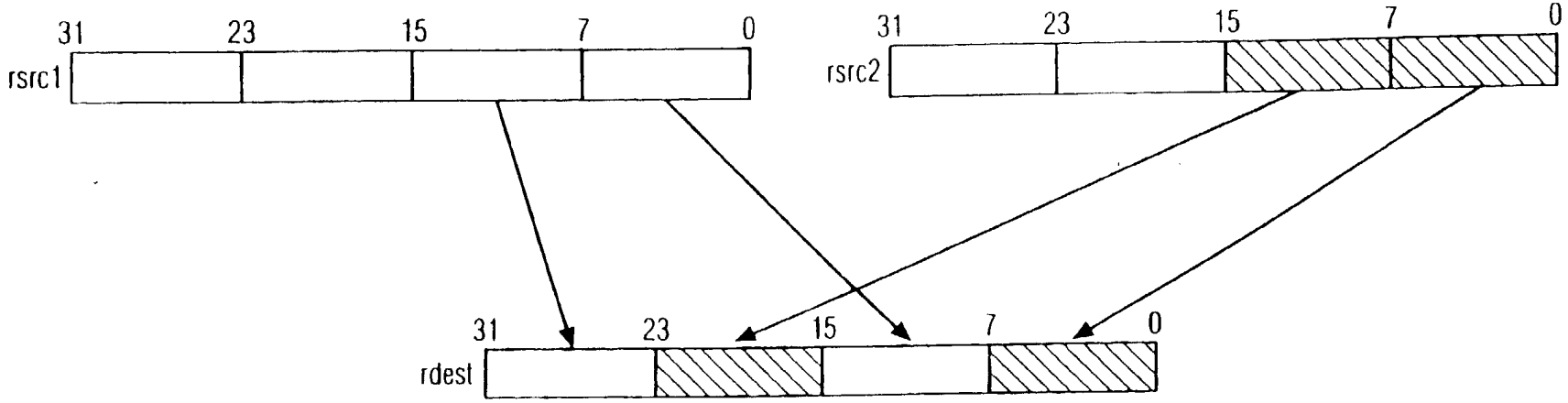


图21

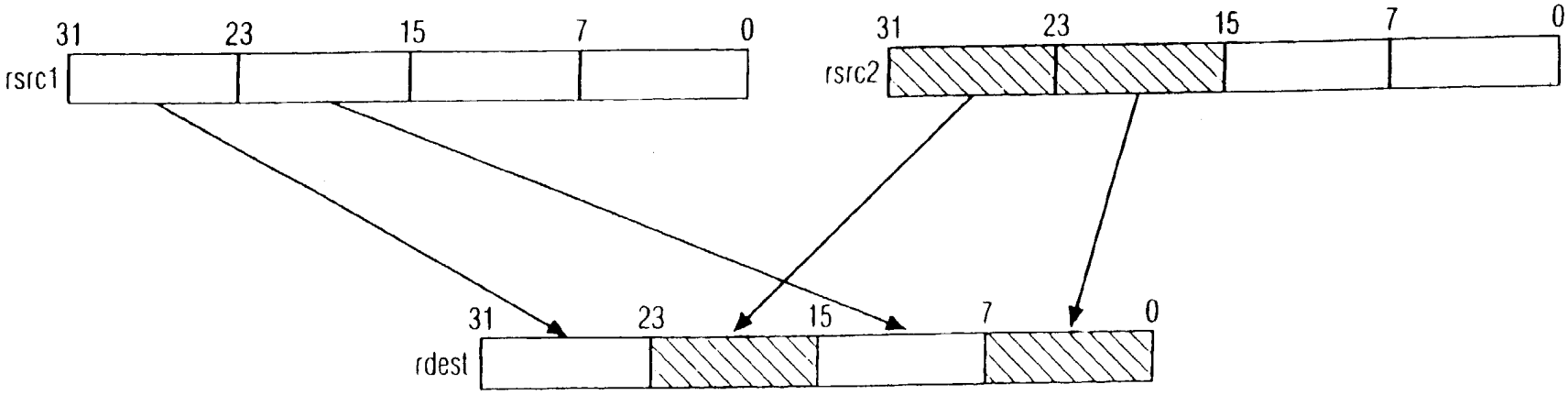


图 22

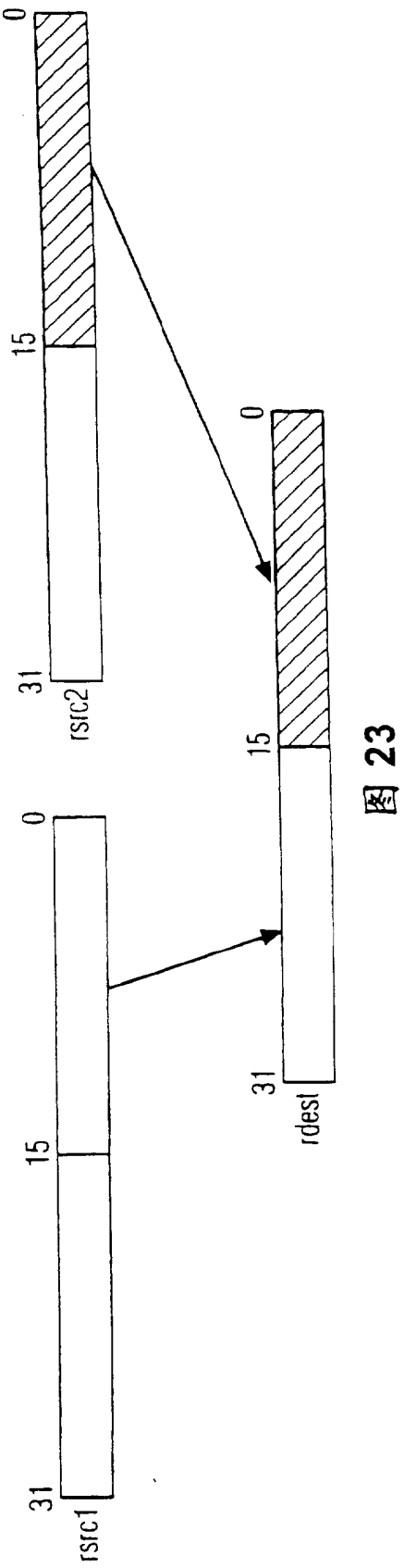

0
15
31
rsrc2
0
15
31
rdest
0
15
31
rsrc1

图 23

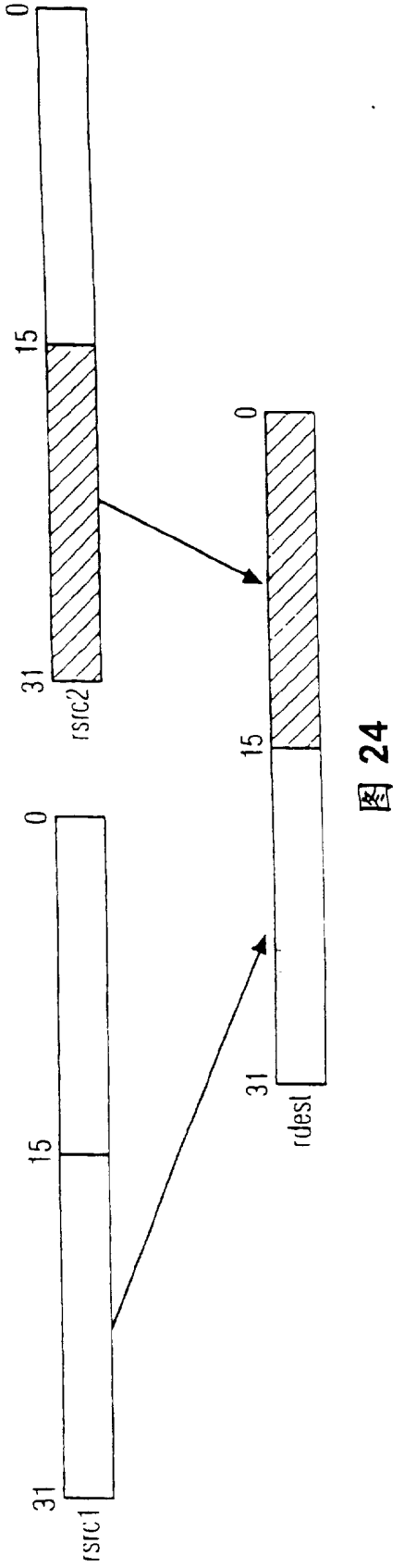

0
15
31
rsrc2
0
15
31
rdest
0
15
31
rsrc1

图 24

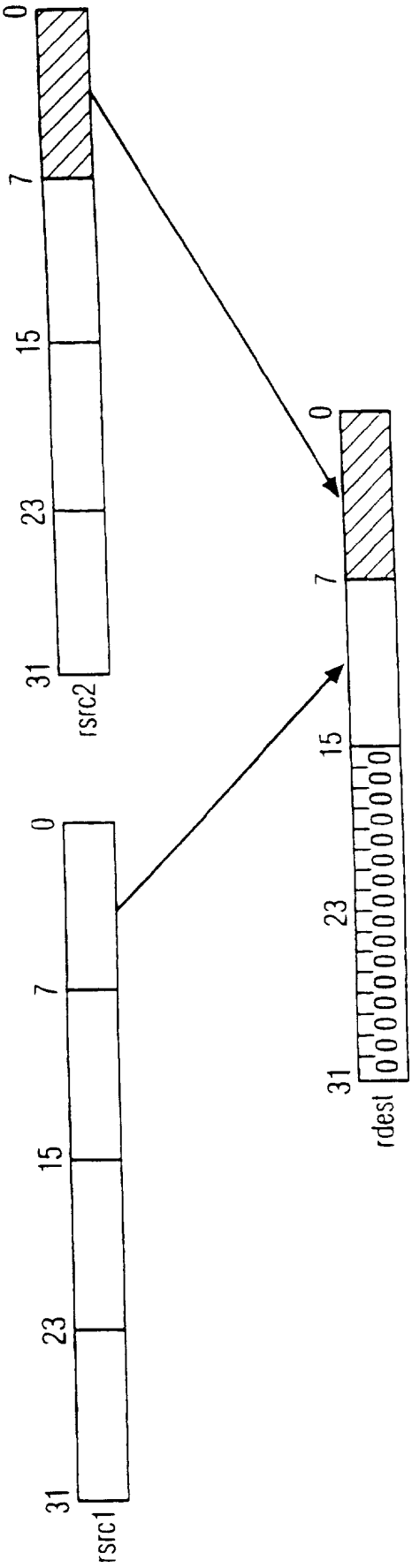


图 25

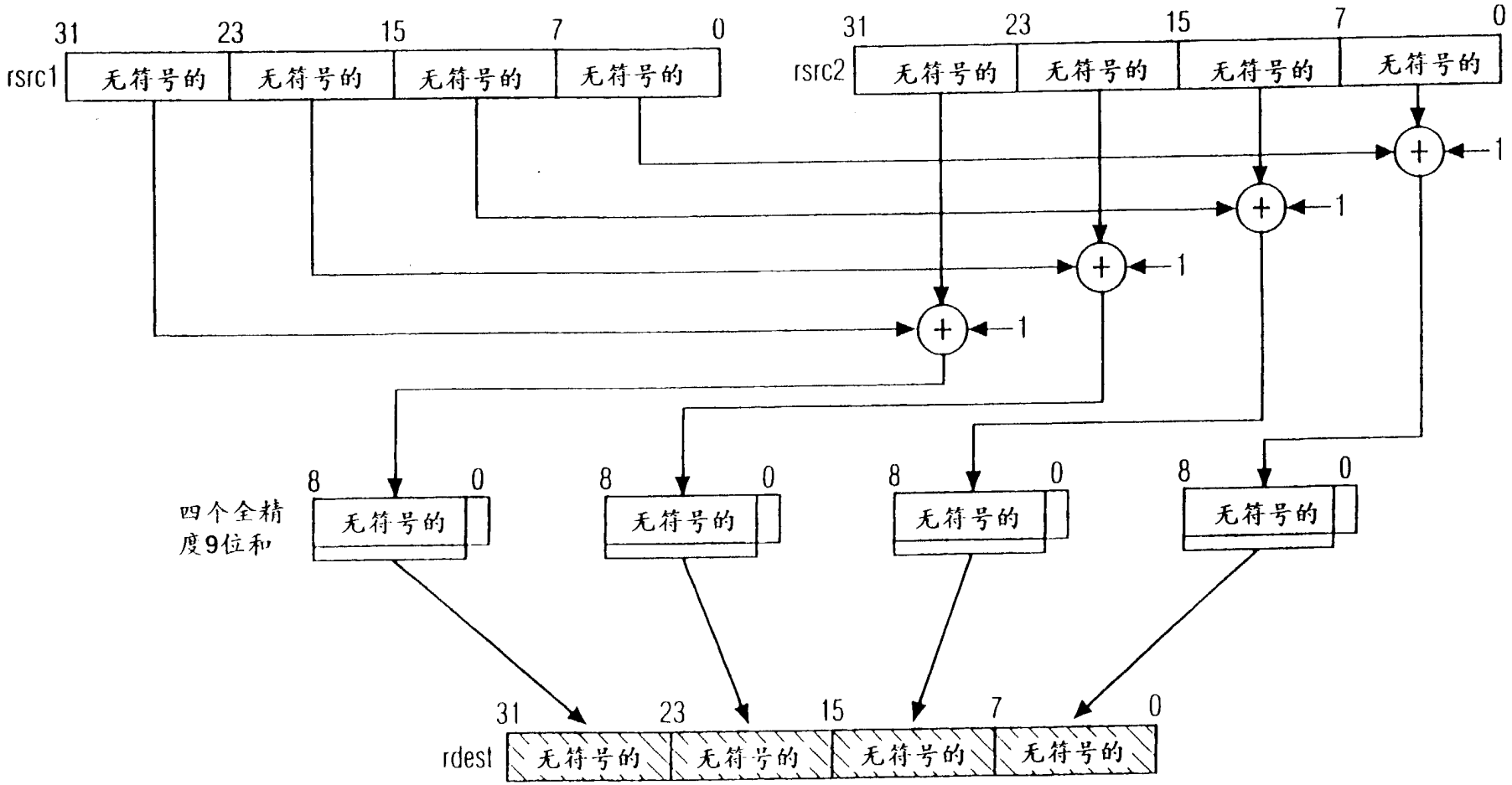

31
23
15
7
0
rsrc1
无符号的
无符号的
无符号的
无符号的
rsrc2
无符号的
无符号的
无符号的
无符号的
1
1
1
1
8
0
四个全精度9位和
无符号的
无符号的
无符号的
无符号的
rdest
无符号的
无符号的
无符号的
无符号的

图 26

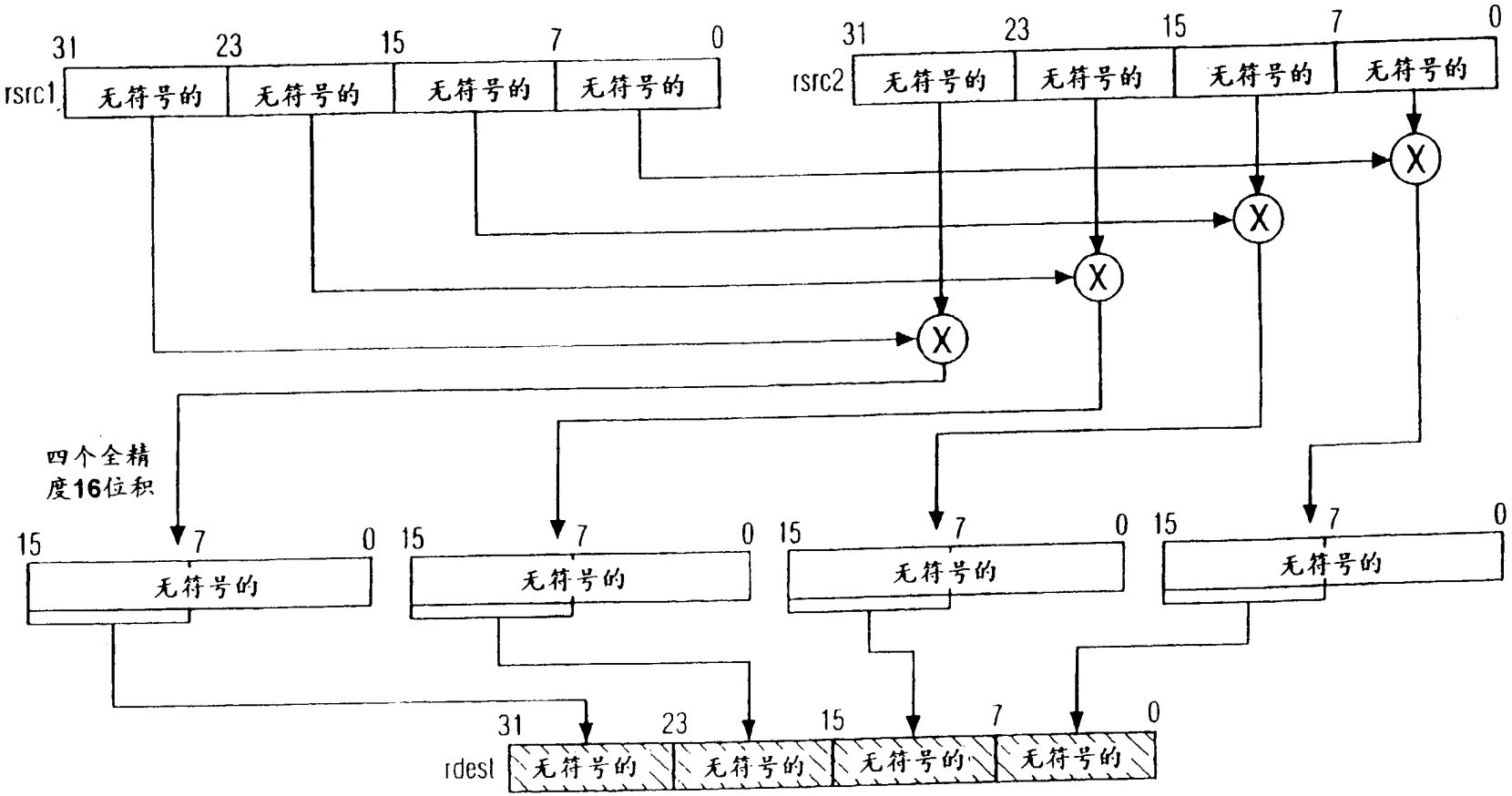

31
23
15
7
0
rsrc1
无符号的
无符号的
无符号的
无符号的
rsrc2
无符号的
无符号的
无符号的
无符号的
X
X
X
X
四个全精
度16位积
15
7
0
无符号的
无符号的
无符号的
无符号的
rdest
无符号的
无符号的
无符号的
无符号的

图 27

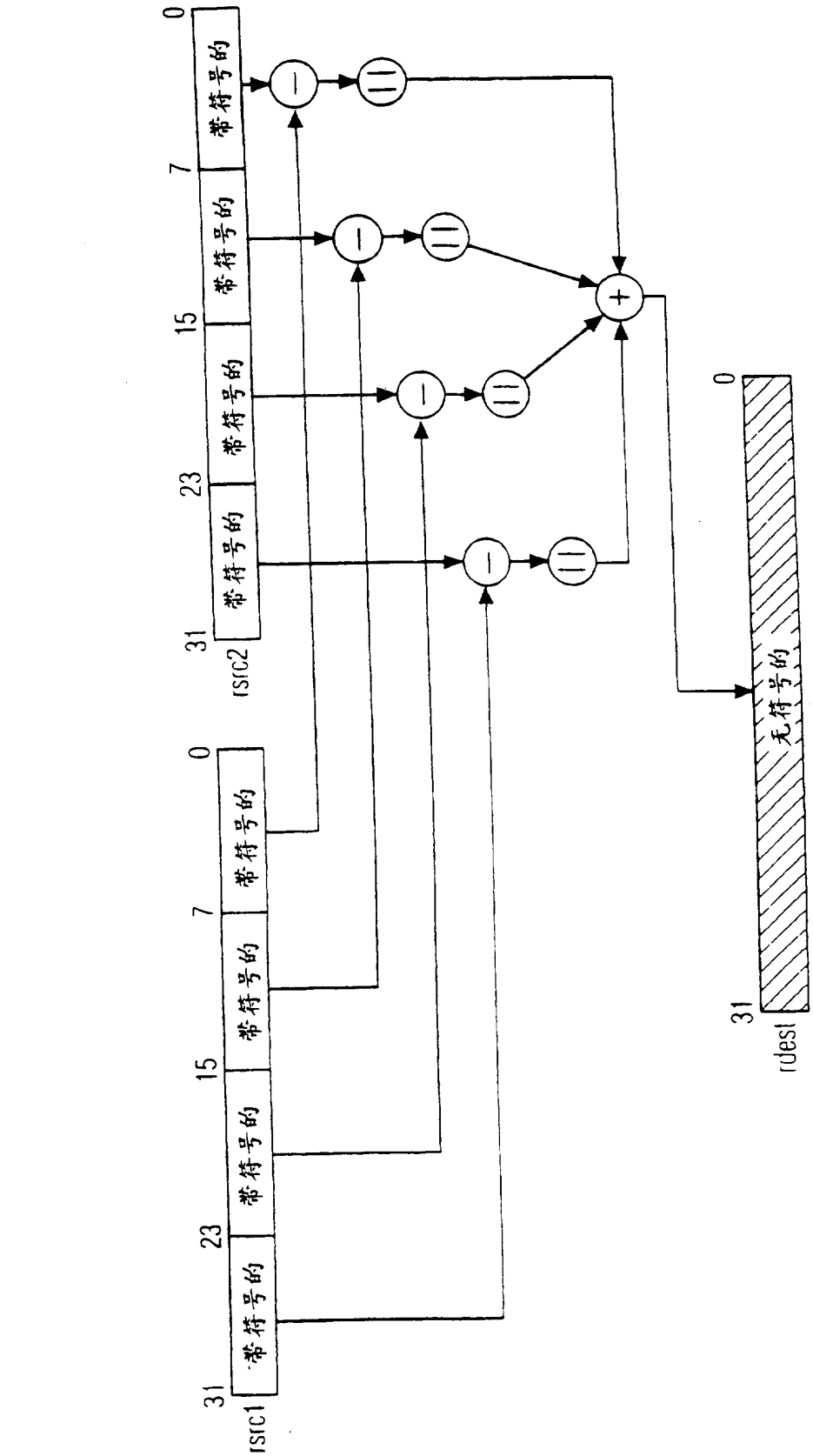


图 28

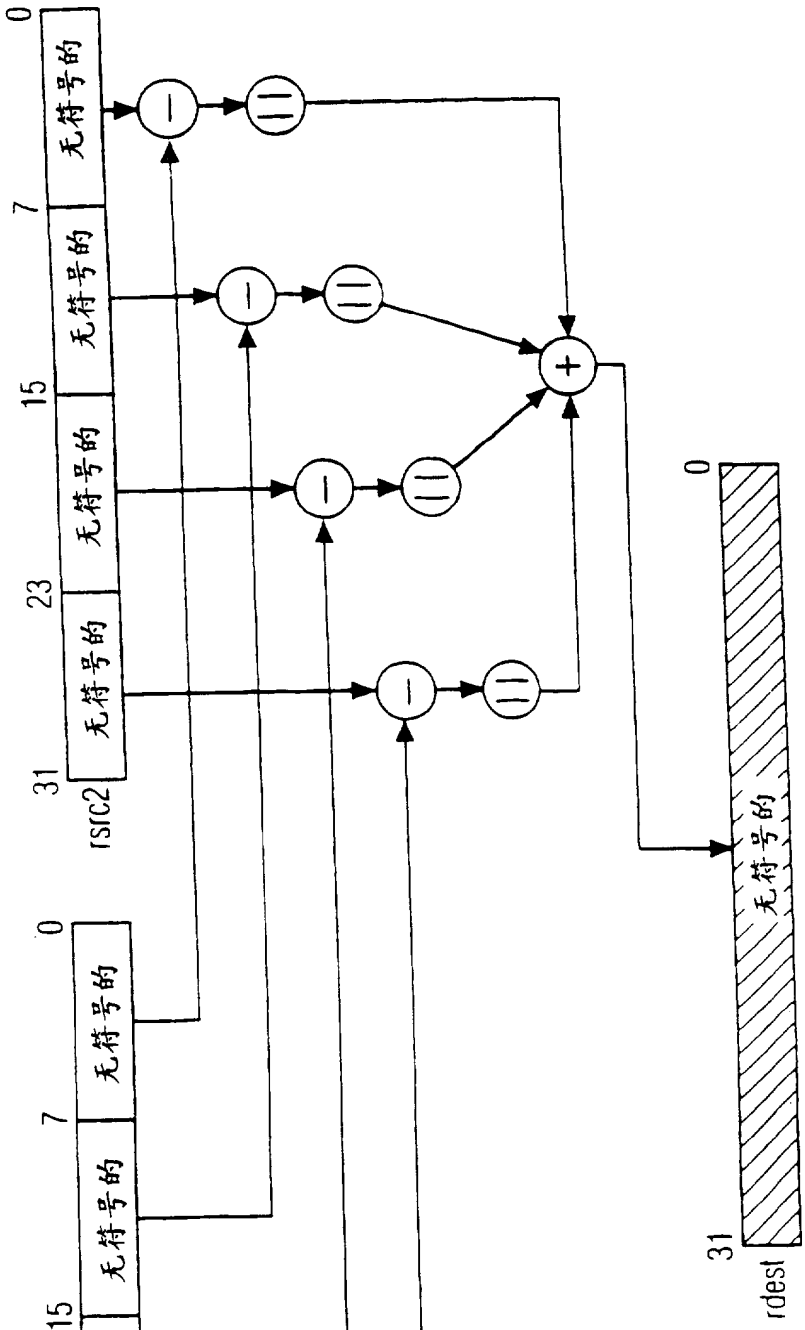


图 29

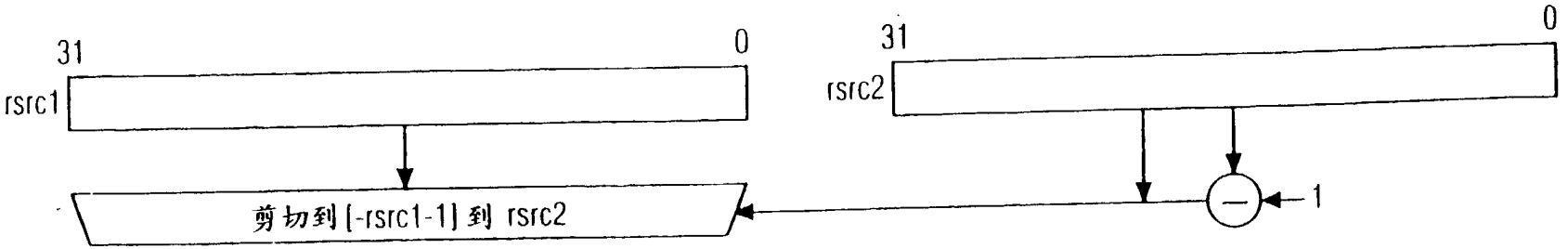

31
0
rsrc1
31
0
rsrc2
剪切到[-rsrc1-1]到 rsrc2
1

图 30

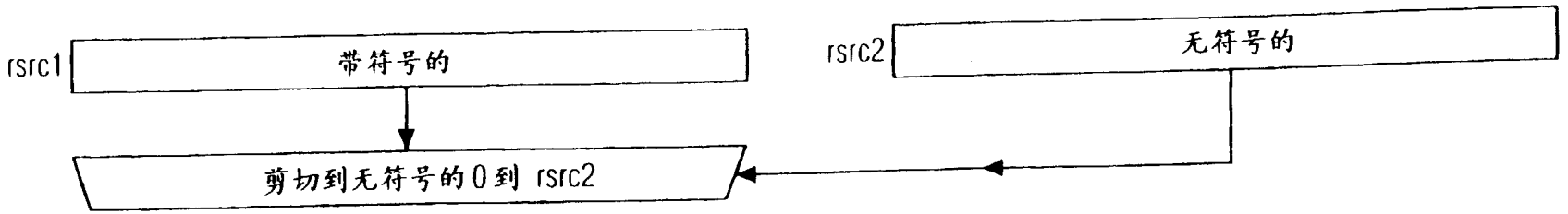

rsrc1
带符号的
rsrc2
无符号的
剪切到无符号的0到 rsrc2

图 31

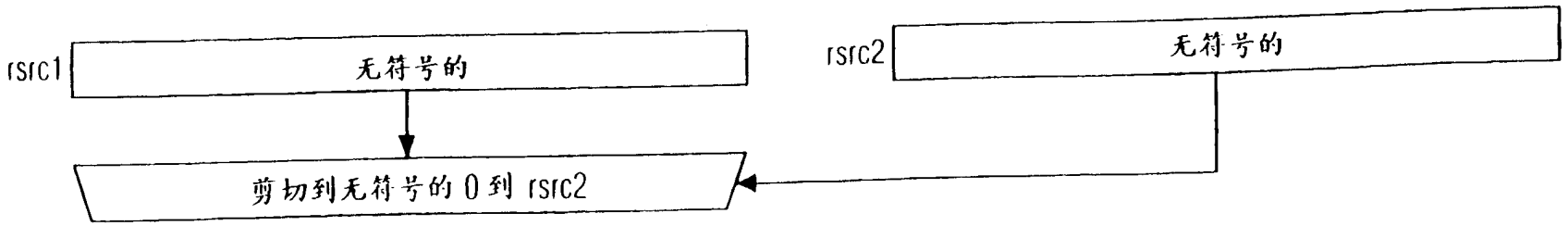

rsrc1
无符号的
rsrc2
无符号的
剪切到无符号的0到 rsrc2

图 32