(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 101187861 B
(45)授权公告日 2012.02.29

(21)申请号 200710180647.7

(22)申请日 2007.09.20

(30)优先权数据
11/524852 2006.09.20 US

(73)专利权人 英特尔公司
地址 美国加利福尼亚州

(72)发明人 R·佐哈 M·塞科尼
R·帕塔萨拉蒂 S·钦努帕蒂
M·布克斯顿 C·德西尔瓦

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

(51)Int.Cl.
G06F 9/38(2006.01)

(56)对比文件
郑纬民、汤志忠. 计算机系统结构 第二版. 清华大学出版社, 2002, 253-391, 451-495.

审查员 郭从征

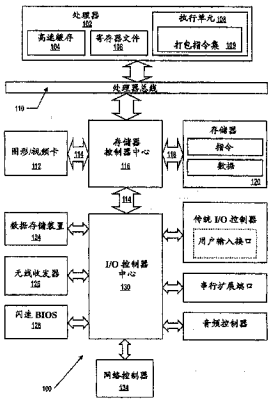
权利要求书 3 页 说明书 16 页 附图 15 页

(54)发明名称

用于执行点积运算的指令和逻辑

(57)摘要

本发明用于执行点积运算的指令和逻辑。本发明提供了用于执行点积操作的方法、装置和程序部件。在一个实施例中,装置包括执行第一指令的执行资源。响应第一指令,所述执行资源将等于至少两个操作数的点积的结果值存储到某个存储单元中。



CN 101187861 B

1. 一种用于执行点积操作的设备：

确定各具有第一数据类型的多个打包值的至少两个操作数的点积结果的部件；

存储所述点积结果的部件。

2. 如权利要求 1 所述的设备，其特征在于，所述第一数据类型为整数数据类型。

3. 如权利要求 1 所述的设备，其特征在于，所述第一数据类型为浮点数据类型。

4. 如权利要求 1 所述的设备，其特征在于，所述至少两个操作数各仅具有两个打包值。

5. 如权利要求 1 所述的设备，其特征在于，所述至少两个操作数各仅具有四个打包值。

6. 如权利要求 1 所述的设备，其特征在于，所述多个打包值的每一个为单精度值，并且由 32 位来表示。

7. 如权利要求 1 所述的设备，其特征在于，所述多个打包值的每一个为双精度值，并且由 64 位来表示。

8. 如权利要求 1 所述的设备，其特征在于，所述至少两个操作数和所述点积结果将存储在至少两个存储多达 128 位数据的寄存器中。

9. 一种用于执行点积操作的装置，包括：

第一逻辑，对第一数据类型的至少两个打包操作数执行单指令多数据点积指令。

10. 如权利要求 9 所述的装置，其特征在于，所述单指令多数据点积指令包含源操作数指示符、目标操作数指示符以及至少一个立即值指示符。

11. 如权利要求 10 所述的装置，其特征在于，所述源操作数指示符包括具有存储多个打包值的多个单元的源寄存器的地址。

12. 如权利要求 11 所述的装置，其特征在于，所述目标操作数指示符包括具有存储多个打包值的多个单元的目标寄存器的地址。

13. 如权利要求 12 所述的装置，其特征在于，所述立即值指示符包括多个控制位。

14. 如权利要求 9 所述的装置，其特征在于，所述至少两个打包操作数各为双精度整数。

15. 如权利要求 9 所述的装置，其特征在于，所述至少两个打包操作数各为双精度浮点值。

16. 如权利要求 9 所述的装置，其特征在于，所述至少两个打包操作数各为单精度整数。

17. 如权利要求 9 所述的装置，其特征在于，所述至少两个打包操作数各为单精度浮点值。

18. 一种用于执行点积操作的系统，包括：

第一存储器，存储单指令多数据点积指令；

处理器，耦合到所述第一存储器以执行所述单指令多数据点积指令。

19. 如权利要求 18 所述的系统，其特征在于，所述单指令多数据点积指令包含源操作数指示符、目标操作数指示符以及至少一个立即值指示符。

20. 如权利要求 19 所述的系统，其特征在于，所述源操作数指示符包括具有存储多个打包值的多个单元的源寄存器的地址。

21. 如权利要求 20 所述的系统，其特征在于，所述目标操作数指示符包括具有存储多个打包值的多个单元的目标寄存器的地址。

22. 如权利要求 21 所述的系统，其特征在于，所述立即值指示符包括多个控制位。

23. 如权利要求 18 所述的系统，其特征在于，所述至少两个打包操作数各为双精度整数。

24. 如权利要求 18 所述的系统，其特征在于，所述至少两个打包操作数各为双精度浮点值。

25. 如权利要求 18 所述的系统，其特征在于，所述至少两个打包操作数各为单精度整数。

26. 如权利要求 18 所述的系统，其特征在于，所述至少两个打包操作数各为单精度浮点值。

27. 一种用于执行点积操作的方法，包括：

将第一打包操作数的第一数据元素与第二打包操作数的第一数据元素相乘，以产生第一乘积；

将所述第一打包操作数的第二数据元素与所述第二打包操作数的第二数据元素相乘，以产生第二乘积；

将所述第一乘积与所述第二乘积相加，以产生点积结果。

28. 如权利要求 27 所述的方法，其特征在于，还包括将所述第一打包操作数的第三数据元素与所述第二打包操作数的第三数据元素相乘，以产生第三乘积。

29. 如权利要求 28 所述的方法，其特征在于，还包括将所述第一打包操作数的第四数据元素与所述第二打包操作数的第四数据元素相乘，以产生第四乘积。

30. 一种用于执行点积操作的处理器，包括：

源寄存器，存储包括第一数据值和第二数据值的第一打包操作数；

目标寄存器，存储包括第三数据值和第四数据值的第二打包操作数；

根据所述点积指令所指示的控制值来执行单指令多数据点积指令的逻辑，所述逻辑包括将所述第一数据值和第三数据值相乘以产生第一乘积的第一乘法器、将所述第二数据值和第四数据值相乘以产生第二乘积的第二乘法器，所述逻辑还包括将所述第一乘积和第二乘积相加以产生至少一个和数的至少一个加法器。

31. 如权利要求 30 所述的处理器，其特征在于，所述逻辑还包括根据所述控制值的第一位在所述第一乘积与空值之间进行选择的第一复用器。

32. 如权利要求 31 所述的处理器，其特征在于，所述逻辑还包括根据所述控制值的第二位在所述第二乘积与空值之间进行选择的第二复用器。

33. 如权利要求 32 所述的处理器，其特征在于，所述逻辑还包括在将被存储在所述目标寄存器的第一单元中的所述和数与空值之间进行选择的第三复用器。

34. 如权利要求 33 所述的处理器，其特征在于，所述逻辑还包括在将被存储在所述目标寄存器的第二单元中的所述和数与空值之间进行选择的第四复用器。

35. 如权利要求 30 所述的处理器，其特征在于，所述第一数据值、第二数据值、第三数据值和第四数据值为 64 位整数值。

36. 如权利要求 30 所述的处理器，其特征在于，所述第一数据值、第二数据值、第三数据值和第四数据值为 64 位浮点值。

37. 如权利要求 30 所述的处理器，其特征在于，所述第一数据值、第二数据值、第三数

据值和第四数据值为 32 位整数值。

38. 如权利要求 30 所述的处理器，其特征在于，所述第一数据值、第二数据值、第三数据值和第四数据值为 32 位浮点值。

39. 如权利要求 30 所述的处理器，其特征在于，所述源寄存器和目标寄存器将存储至少 128 位数据。

用于执行点积运算的指令和逻辑

技术领域

[0001] 本发明涉及执行数学运算的处理装置及相关软件和软件序列的领域。

背景技术

[0002] 计算机系统已经越来越深入我们的社会。计算机的处理能力已经提高了各种职业的工人的效率和生产力。由于购买和拥有计算机的费用持续下降，所以越来越多的消费者能够利用更新、更快的机器。此外，许多人由于使用自由而乐于使用笔记本电脑。移动计算机使用户可在离开办公室或旅行时轻松地传输数据以及进行工作。这种情况在营销人员、公司管理人员甚至学生中是常见的。

[0003] 随着处理器技术的进步，还产生了更新的软件代码来在具有这些处理器的机器上运行。用户一般预期并要求他们的计算机的更高性能，而不管所使用的软件类型。从处理器内实际执行的指令和操作的种类中可能产生一个这样的问题。根据操作的复杂度和 / 或所需电路的类型，某些类型的操作需要更多时间来完成。这提供了优化在处理器内部执行某些复杂操作的方式的机会。

[0004] 十多年来，媒体应用推动了微处理器的发展。实际上，媒体应用推动了近年来的大多数计算升级。这些升级主要在消费者方面发生，但是，对于娱乐性增强的教育和通信目的，在企业方面也看到显著的进步。然而，未来的媒体应用需要更高的计算要求。因此，将来的个人计算体验在视听效果方面更为丰富，并且更容易使用，更重要的是，计算将与通信融合。

[0005] 因此，图像的显示以及共同称作内容的音频和视频数据的回放已经逐渐成为当前计算装置的流行应用。滤波和卷积操作是对内容数据、如图像音频和视频数据所执行的最常见操作的一部分。这类操作是计算密集的，但是提供可通过采用各种数据存储装置（如单指令多数据 (SIMD) 寄存器）的有效实现来利用的高级数据并行性。许多当前的体系结构还需要多个操作、指令或子指令（通常称作“微操作”或“μop”）来对多个操作数执行各种数学运算，由此减小吞吐量并增加执行数学运算所需的时钟周期数量。

[0006] 例如，可能需要由多个指令组成的指令序列来执行产生点积所必需的一个或多个运算，包括将由处理装置、系统或计算机程序中的各种数据类型所表示的两个或两个以上数值之积相加。但是，这类现有技术可能需要许多处理周期，并且可能使处理器或系统消耗不必要的功率以产生点积。此外，一些现有技术可能在可进行操作的操作数的数据类型方面受到限制。

发明内容

[0007] 根据本发明的一个方面，提供了一种已在其中存储了指令的机器可读介质，所述指令在由机器执行时，使所述机器执行包括以下步骤的方法：确定各具有第一数据类型的多个打包值的至少两个操作数的点积结果；存储所述点积结果。

[0008] 根据本发明的另一方面，提供了一种装置，包括：第一逻辑，对第一数据类型的至

少两个打包操作数执行单指令多数据点积指令。

[0009] 根据本发明的又一方面，提供了一种系统，包括：第一存储器，存储单指令多数据点积指令；处理器，耦合到所述第一存储器以执行所述单指令多数据点积指令。

[0010] 根据本发明的再一方面，提供了一种方法，包括：将第一打包操作数的第一数据元素与第二打包操作数的第一数据元素相乘，以产生第一乘积；将所述第一打包操作数的第二数据元素与所述第二打包操作数的第二数据元素相乘，以产生第二乘积；将所述第一乘积与所述第二乘积相加，以产生点积结果。

[0011] 此外，本发明还提供了一种处理器，包括：源寄存器，存储包括第一数据值和第二数据值的第一打包操作数；目标寄存器，存储包括第三数据值和第四数据值的第二打包操作数；根据所述点积指令所指示的控制值来执行单指令多数据点积指令的逻辑，所述逻辑包括将所述第一数据值和第三数据值相乘以产生第一乘积的第一乘法器、将所述第二数据值和第四数据值相乘以产生第二乘积的第二乘法器，所述逻辑还包括将所述第一乘积和第二乘积相加以产生至少一个和数的至少一个加法器。

附图说明

[0012] 通过附图、作为实例而非限制地来说明本发明：

[0013] 图 1A 是采用处理器组成的计算机系统的框图，所述处理器包括根据本发明的一个实施例执行点积操作的指令的执行单元；

[0014] 图 1B 是根据本发明的一个备选实施例的另一个示范性计算机系统的框图；

[0015] 图 1C 是根据本发明的另一个备选实施例的再一个示范性计算机系统的框图；

[0016] 图 2 是一个实施例的处理器的微体系结构的框图，所述处理器包括根据本发明执行点积操作的逻辑电路；

[0017] 图 3A 示出根据本发明的一个实施例的多媒体寄存器中的各种打包 (packed) 数据类型表示；

[0018] 图 3B 示出根据一个备选实施例的打包数据类型；

[0019] 图 3C 示出根据本发明的一个实施例的多媒体寄存器中的各种有符号和无符号打包数据类型表示；

[0020] 图 3D 示出一种操作编码（操作码）格式的一个实施例；

[0021] 图 3E 示出一种备选操作编码（操作码）格式；

[0022] 图 3F 示出又一种备选操作编码格式；

[0023] 图 4 是根据本发明对打包数据操作数执行点积操作的逻辑 (logic) 的一个实施例的框图；

[0024] 图 5A 是根据本发明的一个实施例对单精度打包数据操作数执行点积操作的逻辑的框图；

[0025] 图 5B 是根据本发明的一个实施例对双精度打包数据操作数执行点积操作的逻辑的框图；

[0026] 图 6A 是根据本发明的一个实施例用于执行点积操作的电路的框图；

[0027] 图 6B 是根据本发明的另一个实施例用于执行点积操作的电路的框图；

[0028] 图 7 是根据一个实施例对数据进行打包符号操作的示意图。

[0029] 图 7A 是根据一个实施例可通过执行 DPPS 指令来执行的操作的伪码表示；

[0030] 图 7B 是根据一个实施例可通过执行 DPPD 指令来执行的操作的伪码表示。

具体实施方式

[0031] 以下说明描述在处理装置、计算机系统或软件程序中执行点积操作的一种技术的实施例。在以下描述中，阐述诸如处理器类型、微体系结构条件、事件、启用机制等的大量具体细节，以提供对本发明的充分理解。然而，本领域的技术人员会理解，没有这类具体细节，也可实施本发明。另外，没有详细说明一些公知的结构、电路等，以免不必要地影响对本发明的理解。

[0032] 虽然参照处理器来描述以下实施例，但是，其它实施例适用于其它类型的集成电路和逻辑装置。本发明的相同技术和理论可容易地应用到可获益于较高流水线吞吐量和改进的性能的其它类型的电路或半导体器件。本发明的理论适用于执行数据操作的任何处理器或机器。但是，本发明不限于执行 256 位、128 位、64 位、32 位或 16 位数据操作的处理器或机器，而是可适用于其中需要操纵打包数据的任何处理器和机器。

[0033] 为便于说明，以下描述中阐述了大量具体细节，以便提供对本发明的充分理解。但是，本领域的技术人员会理解，这些具体细节不是实施本发明所必需的。在其它情况下，没有对公知的电气结构和电路进行具体的详细阐述，以免不必要地影响对本发明的理解。另外，为了说明的目的，以下描述提供实例，以及附图示出各种实例。但是，这些实例不应当以限制的意义来理解，因为它们旨在提供本发明的实例，而不是提供本发明的所有可能实现的穷尽列表。

[0034] 虽然以下实例在执行单元和逻辑电路的上下文中来描述指令处理和分配，但是，本发明的其它实施例可通过软件来实现。在一个实施例中，本发明的方法以机器可执行指令来体现。这些指令可用于使采用指令编程的通用或专用处理器执行本发明的步骤。本发明可作为计算机程序产品或软件来提供，它可包括其中已存储指令的机器或计算机可读介质，这些指令可用于对计算机（或其它电子设备）编程以执行根据本发明的过程。作为备选的方案，本发明的步骤可由包含用于执行所述步骤的硬连线逻辑的特定硬件部件来执行，或者由已编程计算机部件和定制硬件部件的任何组合来执行。这种软件可存储在系统的存储器中。类似地，代码可经由网络或者通过其它计算机可读媒体来分配。

[0035] 因此，机器可读介质可包括用于存储或传输机器（例如计算机）可读形式的信息的任何机构，包括但不限于软盘、光盘、光盘只读存储器 (CD-ROM) 以及磁光盘、只读存储器 (ROM)、随机存取存储器 (RAM)、可擦除可编程只读存储器 (EPROM)、电可擦除可编程只读存储器 (EEPROM)、磁或光卡、闪存 (flash memory)、通过因特网的传输、电、光、声或其它形式的传播信号（例如载波、红外信号、数字信号等）等。相应地，计算机可读介质包括适于存储或传输机器（如计算机）可读形式的电子指令或信息的任何类型的媒体 / 机器可读介质。此外，本发明还可作为计算机程序产品来下载。因此，程序可从远程计算机（例如服务器）传送到请求计算机（例如客户机）。程序的传送可通过电气、光学、声音或者在载波或其它传播介质中包含的其它形式的数据信号经由通信链路（例如调制解调器、网络连接等）来进行。

[0036] 设计可能经过从创建到仿真 (simulation) 直到制造的各种阶段。表示设计的数

据可通过多种方式来表示设计。首先，如在仿真中可用的那样，硬件可采用硬件描述语言或者另一种功能描述语言来表示。另外，采用逻辑和 / 或晶体管门电路的电路级模型可在设计过程的某些阶段产生。此外，在某个阶段，大部分设计达到表示硬件模型中的各种装置的物理设置的数据级。在采用常规半导体制造技术的情况下，表示硬件模型的数据可以是指定用于生产集成电路的掩模的不同掩模层上的各种特征是否存在的数据。在该设计的任何表示中，数据可存储在任何形式的机器可读介质中。经调制或者以其它方式产生来传输这种信息的光或电波、存储器或者磁或光存储装置（如光盘）可以是机器可读介质。这些介质的任一种可“携带”或“指示”设计或软件信息。在传输指示或携带代码或设计的电载波以执行电信号的复制、缓冲或重传时，制作新的副本。因此，通信提供商或网络提供商可制作体现本发明的技术的产品（载波）的复制品。

[0037] 在现代处理器中，多个不同的执行单元用来处理和执行各种代码及指令。并非所有指令都同等地创建，因为一些指令会较快地完成，而另一些指令则耗用大量时钟周期。指令的吞吐量越大，处理器的整体性能就越好。因此，让许多指令尽可能快地执行是有利的。但是，存在具有较高复杂度并且在执行时间和处理器资源方面要求更多的某些指令。例如存在浮点指令、加载 / 存储操作、数据移动等。

[0038] 随着越来越多的计算机系统用于互联网和多媒体应用，随时间引入了附加处理器支持。例如，单指令多数据 (SIMD) 整数 / 浮点指令和流式 (streaming)SIMD 扩展 (SSE) 是减少执行特定程序任务所需的指令的总体数量的指令，它转而可降低功耗。通过并行地对多个数据元素进行操作，这些指令可加速软件执行。因此，可在包括视频、语音和图像 / 照片处理的大量应用中实现性能增益。微处理器以及相似类型的逻辑电路中的 SIMD 指令的实现通常涉及多个问题。此外，SIMD 操作的复杂度往往导致需要附加电路，以正确地处理和操纵数据。

[0039] 当前，SIMD 点积指令不可用。在不存在 SIMD 点积指令的情况下，在诸如音频 / 视频压缩、处理和操纵之类的应用中可能需要大量指令和数据寄存器来实现同样的结果。因此，根据本发明的实施例的至少一个点积指令可减少代码开销和资源要求。本发明的实施例提供一种实现作为利用 SIMD 相关硬件的算法的点积操作的方式。当前，对 SIMD 寄存器中的数据执行点积操作有些困难且冗长。一些算法需要比执行那些操作的指令的实际数量更多的指令来安排用于算术运算的数据。通过实现根据本发明的实施例的点积操作，实现点积处理所需的指令数量可显著减少。

[0040] 本发明的实施例包括用于实现点积操作的指令。点积操作一般包括将至少两个值相乘并将该乘积加到至少两个其它值的乘积上。可对通用点积算法进行其它变更，包括将各个点积操作的结果相加以产生另一个点积。例如，根据一个实施例，应用于数据元素的点积操作可一般表示为：

[0041] DEST1 ← SRC1*SRC2 ;

[0042] DEST2 ← SRC3*SRC4 ;

[0043] DEST3 ← DEST1+DEST2 ;

[0044] 对于打包 SIMD 数据操作数，该流程可应用于各个操作数的各个数据元素。

[0045] 在以上流程中，“DEST”和“SRC”是表示相应数据或操作的源和目标 (destination) 的一般术语。在一些实施例中，它们可通过具有不同于所述的名称或功能的寄存器、存储

器或其它存储区来实现。例如，在一个实施例中，DEST1 和 DEST2 可以是第一和第二暂时存储区（例如“TEMP1 和“TEMP2”寄存器），SRC1 和 SRC3 可以是第一和第二目标存储区（例如“DEST1”和“DEST2”寄存器）等。在另一些实施例中，SRC 和 DEST 存储区的两个或两个以上可对应于相同存储区（例如 SIMD 寄存器）中的不同数据存储单元 (data storage element)。此外，在一个实施例中，点积操作可产生通过上述一般流程所产生的点积之和。

[0046] 图 1A 是采用处理器组成的示范性计算机系统的框图，所述处理器包括根据本发明的一个实施例执行点积操作的指令的执行单元。根据本发明，例如在本文所描述的实施例中，系统 100 包括采用包含执行处理数据的算法的逻辑的执行单元的部件，例如处理器 102。系统 100 表示基于可向 Intel Corporation(Snata Clara, California) 购买的 **PENTIUM®** III、**PENTIUM®** 4、Xeon™、**Itanium®**、XScale™ 和 / 或 StrongARM™ 微处理器的处理系统，但是也可采用其它系统（包括具有其它微处理器的个人计算机 (PC)、工程工作站、机顶盒等）。在一个实施例中，示例系统 100 可运行可向 Microsoft Corporation(Redmond, Washington) 购买的一种版本的 WINDOWS™ 操作系统，但也可采用其它操作系统（例如 UNIT 和 Linux)、嵌入式软件和 / 或图形用户界面。因此，本发明的实施例不限于硬件电路和软件的任何特定结合。

[0047] 实施例不限于计算机系统。本发明的备选实施例可用于其它装置（如手持装置）和嵌入式应用。手持装置的一些实例包括蜂窝电话、因特网协议装置、数字照相机、个人数字助理 (PDA) 和手持 PC。嵌入式应用可包括微控制器、数字信号处理器 (DSP)、片上系统、网络计算机 (NetPC)、机顶盒、网络集线器、广域网 (WAN) 交换机或者对操作数执行点积操作的其它任何系统。此外，已经实现一些体系结构以使指令能够同时对若干数据进行操作，从而提高多媒体应用的效率。随着数据的类型和容量增加，必须增强计算机及其处理器以通过更有效的方法来操纵数据。

[0048] 图 1A 是根据本发明的一个实施例采用处理器 102 组成的计算机系统 100 的框图，所述处理器包括一个或多个执行单元 108 来执行计算一个或多个操作数中的数据元素的点积的算法。一个实施例可在单处理器台式或服务器系统的上下文中来描述，但是备选实施例可包含在微处理器系统中。系统 100 是中心体系结构的一个实例。计算机系统 100 包括处理数据信号的处理器 102。处理器 102 可以是复杂指令集计算机 (CISC) 微处理器、简化指令集计算 (RISC) 微处理器、超长指令字 (VLIW) 微处理器、实现指令集的组合的处理器或者例如数字信号处理器之类的其它任何处理器装置。处理器 102 耦合到可在处理器 102 与系统 100 中的其它部件之间传输数据信号的处理器总线 110。系统 100 的元件执行本领域的技术人员公知的常规功能。

[0049] 在一个实施例中，处理器 102 包括第一级 (L1) 内部高速缓冲存储器 104。根据该体系结构，处理器 102 可具有单个内部高速缓存或多级内部高速缓存。作为备选的方案，在另一个实施例中，高速缓冲存储器可位于处理器 102 的外部。根据具体实现和需要，另一些实施例也可包括内部和外部两种高速缓存的组合。寄存器文件 (registerfile)106 可在包括整数寄存器、浮点寄存器、状态寄存器和指令指针寄存器的各种寄存器中存储不同类型的数据。

[0050] 包含执行整数和浮点运算的逻辑的执行单元 108 也位于处理器 102 中。处理器 102 还包括存储某些宏指令的微码的微码 (μ code)ROM。对于该实施例，执行单元 108 包括

处理打包指令集 109 的逻辑。在一个实施例中，打包指令集 109 包括用于计算多个操作数的点积的打包点积指令。通过在通用处理器 102 的指令集中包含打包指令集 109，结合执行指令的相关电路，许多多媒体应用使用的操作可采用通用处理器 102 中的打包数据来执行。因此，通过采用处理器的数据总线的全宽度 (full width) 对打包数据执行操作，可加速并且更有效地执行许多多媒体应用。这可消除通过处理器的数据总线传送较小的数据单元以一次对一个数据元素执行一个或多个操作的需要。

[0051] 执行单元 108 的备选实施例也可用于微控制器、嵌入式处理器、图形装置、DSP 和其它类型的逻辑电路。系统 100 包括存储器 120。存储器 120 可以是动态随机存取存储器 (DRAM) 装置、静态随机存取存储器 (SRAM) 装置、闪存装置或者其它存储装置。存储器 120 可存储通过可由处理器 102 执行的数据信号所表示的指令和 / 或数据。

[0052] 系统逻辑芯片 116 耦合到处理器总线 110 和存储器 120。所述实施例中的系统逻辑芯片 116 是存储器控制器中心 (memory controllerhub)(MCH)。处理器 102 可经由处理器总线 110 与 MCH 116 通信。MCH 116 为指令和数据存储以及为图形命令、数据和文本的存储提供到存储器 120 的高带宽存储器通路 118。MCH 116 引导处理器 102、存储器 120 和系统 100 中的其它部件之间的数据信号，并且作为处理器总线 110、存储器 120 和系统 I/O 122 之间的数据信号的桥梁。在一些实施例，系统逻辑芯片 116 可提供用于耦合到图形控制器 112 的图形端口。MCH 116 通过存储器接口 118 耦合到存储器 120。图形卡 112 通过加速图形端口 (AGP) 互连 114 耦合到 MCH 116。

[0053] 系统 100 采用专有中心 (hub) 接口总线 122 将 MCH 116 耦合到 I/O 控制器中心 (ICH)130。ICH 130 通过本地 I/O 总线提供到一些 I/O 装置的直接连接。本地 I/O 总线是用于将外部设备连接到存储器 120、芯片组和处理器 102 的高速 I/O 总线。一些实例是音频控制器、固件中心 (闪速 BIOS)128、无线收发器 126、数据存储装置 124、包含用户输入和键盘接口的传统 I/O 控制器、诸如通用串行总线 (USB) 之类的串行扩展端口和网络控制器 134。数据存储装置 124 可包括硬盘驱动器、软盘驱动器、CD-ROM 装置、闪存装置或者其它海量存储装置。

[0054] 对于系统的另一个实施例，执行具有点积指令的算法的执行单元可与片上系统配合使用。片上系统的一个实施例包括处理器和存储器。一种这样的系统的存储器是闪存。闪存可与处理器和其它系统部件位于相同的晶片上。另外，诸如存储控制器或图形控制器等其它逻辑块也可设置在片上系统中。

[0055] 图 1B 示出实现本发明的一个实施例的原理的数据处理系统 140。本领域的技术人员容易理解，本文所述的实施例可与备选处理系统配合使用，而不会背离本发明的范围。

[0056] 计算机系统 140 包括能够执行包括点积操作的 SIMD 操作的处理核心 159。对于一个实施例，处理核心 159 表示任何类型的体系结构的处理单元，包括但不限于 CISC、RISC 或 VLIW 类型的体系结构。处理核心 159 还可适于以一种或多种加工技术制造，并且通过在机器可读媒体上充分详细地表示，可适合于促进所述制造。

[0057] 处理核心 159 包括执行单元 142、寄存器文件集合 145 和解码器 144。处理核心 159 还包括对本发明的理解不是必要的附加电路 (图中未示出)。执行单元 142 用于执行处理核心 159 所接收的指令。除了识别典型的处理器指令之外，执行单元 142 还可识别用于对打包数据格式执行操作的打包指令集 143 中的指令。打包指令集 143 包括用于支持点积

操作的指令，并且还可包括其它打包指令。执行单元 142 通过内部总线耦合到寄存器文件 145。寄存器文件 145 表示处理核心 159 上用于存储包括数据在内的信息的存储区。如前所述，会理解到，用于存储打包数据的存储区不是关键的。执行单元 142 耦合到解码器 144。解码器 144 用于将处理核心 159 所接收的指令解码为控制信号和 / 或微码入口点 (entry point)。响应这些控制信号和 / 或微码入口点，执行单元 142 执行适当的操作。

[0058] 处理核心 159 与总线 141 耦合，用于与各种其它系统装置进行通信，它们例如可包括但不限于同步动态随机存取存储器 (SDRAM) 控制装置 (control)146、静态随机存取存储器 (SDRAM) 控制装置 147、突发 (burst) 闪存接口 148、个人计算机存储卡国际联盟 (PCMCIA)/ 压缩闪存 (compact flash)(CF) 卡控制装置、液晶显示器 (LCD) 控制装置 150、直接存储器存取 (DMA) 控制器 151 以及备选总线主接口 152。在一个实施例中，数据处理系统 140 还可包括 I/O 桥接器 154，用于经由 I/O 总线 153 与各种 I/O 装置进行通信。这类 I/O 装置例如可包括但不限于通用异步接收器 / 发射器 (UART)155、通用串行总线 (USB)156、蓝牙无线 UART 157 和 I/O 扩展接口 158。

[0059] 数据处理系统 140 的一个实施例提供移动、网络和 / 或无线通信以及能够执行包括点积操作在内的 SIMD 操作的处理核心 159。处理核心 159 可采用各种音频、视频、成像和通信算法来编程，所述算法包括诸如沃尔什 - 哈达玛变换、快速傅立叶变换 (FFT)、离散余弦变换 (DCT) 及其各自的逆变换之类的离散变换，诸如色彩空间变换、视频编码运动估计或视频解码运动补偿之类的压缩 / 解压缩技术，以及诸如脉冲编码调制 (PCM) 之类的调制 / 解调 (MODEM) 功能。本发明的一些实施例还可适用于图形应用，例如三维 (“3D”) 建模、呈现、对象冲突检测、3D 对象变换和照明等。

[0060] 图 1C 说明能够执行 SIMD 点积操作的数据处理系统的又一备选实施例。根据一个备选实施例，数据处理系统 160 可包括主处理器 166、SIMD 协处理器 161、高速缓冲存储器 167 和输入 / 输出系统 168。输入 / 输出系统 168 可任选地耦合到无线接口 169。SIMD 协处理器 161 能够执行包括点积操作在内的 SIMD 操作。处理核心 170 可适合于以一种或多种加工技术制造，并且通过在机器可读媒体上充分详细地表示，可适合于促进包括处理核心 170 在内的数据处理系统 160 的全部或部分的制造。

[0061] 对于一个实施例，SIMD 协处理器 161 包括执行单元 162 和寄存器文件集合 164。主处理器 165 的一个实施例包括解码器 165，以识别包括供执行单元 162 执行的 SIMD 点积计算指令在内的指令集 163 的指令。对于备选实施例，SIMD 协处理器 161 还包括解码器 165B 的至少一部分，以对指令集 163 的指令进行解码。处理核心 170 还包括对本发明的实施例的理解不是必要的附加电路 (图中未示出)。

[0062] 在操作中，主处理器 166 执行数据处理指令流，所述指令控制包括与高速缓冲存储器 167 和输入 / 输出系统 168 进行交互在内的一般类型的数据处理操作。嵌入数据处理指令流中的是 SIMD 协处理器指令。主处理器 166 的解码器 165 将这些 SIMD 协处理器指令识别为属于应当由附属的 SIMD 协处理器 161 来执行的类型。因此，主处理器 166 在协处理器总线 166 上发出这些 SIMD 协处理器指信令 (或者表示 SIMD 协处理器指令的控制信号)，由此，它们由任何附属的 SIMD 协处理器来接收。在这种情况下，SIMD 协处理器 161 将接收并执行发送给它的任何所接收的 SIMD 协处理器指令。

[0063] 数据可经由无线接口 169 来接收，以供 SIMD 协处理器指令进行处理。对于一个实

例，可采取数字信号的形式来接收语音通信，它可通过 SIMD 协处理器指令进行处理，以再生 (regenerate) 表示语音通信的数字音频样本。对于另一个实例，可采取数字比特流的形式来接收压缩音频和 / 或视频，它可通过 SIMD 协处理器指令进行处理，以再生数字音频样本和 / 或运动视频帧。对于处理核心 170 的一个实施例，主处理器 166 和 SIMD 协处理器 161 集成到包括执行单元 162、寄存器文件集合 164 和解码器 165 的单个处理核心 170 中，以识别包括 SIMD 点积指令在内的指令集 163 的指令。

[0064] 图 2 是根据本发明的一个实施例的处理器 200 的微体系结构的框图，所述处理器包括执行点积指令的逻辑电路。对于点积指令的一个实施例，该指令可将第一数据元素与第二数据元素相乘，并且将该乘积与第三和第四数据元素之积相加。在一些实施例中，点积指令可实现成对于具有字节、字、双字、四字等大小以及诸如单和双精度整数及浮点数据类型之类的数据类型的数据元素进行操作。在一个实施例中，有序前端 201 是处理器 200 的组成部分，它取出待执行的宏指令，并对它们进行准备以供之后在处理器流水线中使用。前端 201 可包括若干单元。在一个实施例中，指令预取器 226 从存储器中取出宏指令，并将其馈送到指令解码器 228，指令解码器 228 转而将这些宏指令解码为称作微指令或微操作 (又称作 micro-op 或 μop) 的机器可执行的原语。在一个实施例中，追踪高速缓存 (trace cache)230 取出解码后的 μop，并将其组装为 μop 队列 234 中的程序排序序列或路线 (trace) 供执行。当追踪高速缓存 230 遇到复杂宏指令时，微码 ROM 232 提供完成该操作所需的 μop。

[0065] 许多宏指令被转换为单个微操作，而其它的则需要若干微操作来完成整个操作。在一个实施例中，若需要四个以上微操作来完成宏指令，则解码器 228 访问微码 ROM 232 来执行宏指令。对于一个实施例，可将打包点积指令解码为少量微操作以在指令解码器 228 上进行处理。在另一个实施例中，若需要多个微操作来完成该操作，则打包点积算法的指令可存储在微码 ROM 232 中。追踪高速缓存 230 参照入口点可编程逻辑阵列 (PLA) 来确定用于读取微码 ROM 232 中的点积算法的微码序列的正确微指令指针。在微码 ROM 232 完成当前宏指令的定序 (sequencing) 微操作之后，机器的前端 201 继续从追踪高速缓存 230 中取出微操作。

[0066] 某种 SIMD 和其它多媒体类型的指令被看作复杂指令。大多数浮点相关的指令也是复杂指令。因此，当指令解码器 228 遇到复杂宏指令时，在适当位置上对微码 ROM 232 进行访问，以检索那个宏指令的微码序列。将执行那个宏指令所需的各个微操作传送给元序执行引擎 203，以在适当的整数和浮点执行单元上执行。

[0067] 无序执行引擎 203 是在其中准备微指令供执行的单元。无序执行逻辑具有多个缓冲器以在微指令沿流水线传输并被安排执行时对所述微指令的流程进行平滑处理及重新排序来优化性能。分配器逻辑分配给各 μop 执行所需的机器缓冲器和资源。寄存器重命名逻辑将逻辑寄存器重命名到寄存器文件的条目上。在以下指令调度器之前，分配器还分配两个 μop 队列之一中的各 μop 的条目，所述两个队列中的一个用于存储器操作，一个用于非存储器操作 :存储器调度器，快速调度器 202，慢速 / 通用浮点调度器 204，以及简单浮点调度器 206。μop 调度器 202、204、206 根据它们的相关输入寄存器操作数源的预备状态以及 μop 完成其操作所需的执行资源的可用性来确定何时 μop 预备执行。该实施例的快速调度器 202 可在主时钟周期的每一半上进行调度，而其它调度器在每个主处理器时钟周

期只可调度一次。调度器对分配端口进行仲裁，以调度用于执行的 μop。

[0068] 寄存器文件 208、210 位于调度器 202、204、206 与执行块 211 的执行单元 212、214、216、218、220、222、224 之间。存在分别用于整数和浮点操作的独立寄存器文件 208、210。该实施例的各寄存器文件 208、210 还包括旁路网络 (bypass network)，它可向新的相关 μop 分流 (bypass) 或转发还未写入寄存器文件的刚完成的结果。整数寄存器文件 208 和浮点寄存器文件 210 还能互相传送数据。对于一个实施例，整数寄存器文件 208 被分为两个独立寄存器文件，一个寄存器文件用于数据的低阶 32 住，而第二寄存器文件用于数据的高阶 32 位。一个实施例的浮点寄存器文件 210 具有 128 位宽的条目，因为浮点指令通常具有从 64 到 128 位宽的操作数。

[0069] 执行块 211 包含执行单元 212、214、216、218、220、222、224，指令实际上在这些执行单元中执行。该部分包括寄存器文件 208、210，它们存储微指令需要执行的整数和浮点数据操作数值。该实施例的处理器 200 包括多个执行单元 :地址生成单元 (AGU)212，AGU 214，快速 ALU 216，快速 ALU 218，慢速 ALU 220，浮点 ALU 222，浮点移动单元 224。对于该实施例，浮点执行块 222、224 执行浮点、MMX、SIMD 和 SSE 操作。该实施例的浮点 ALU 222 包括 64 位乘 64 位浮点除法器，以执行除法、平方根及余数 (remainder) 微操作。对于本发明的实施例，涉及浮点值的任何动作采用浮点硬件进行。例如，整数格式与浮点格式之间的转换涉及浮点寄存器文件。类似地，浮点除法操作在浮点除法器上进行。另一方面，非浮点数值和整数类型采用整数硬件资源来处理。非常频繁的简单 ALU 运算转到高速 ALU 执行单元 216、218。该实施例的快速 ALU 216、218 可采用半个时钟周期的有效等待时间来执行快速运算。对于一个实施例，大多数复杂整数操作转到慢速 ALU 220，因为慢速 ALU 220 包括用于长等待时间类型的操作的整数执行硬件，例如乘法器、移位、标志 (flag) 逻辑和分支处理。存储器加载 / 存储操作由 AGU 212、214 执行。对于该实施例，在对 64 位数据操作数执行整数操作的上下文中描述整数 ALU 216、218、220。在备选实施例中，可实现 ALU 216、218、220 来支持包括 16、32、128、256 等的各种数据位。类似地，可实现浮点单元 222、224 来支持具有各种宽度的位的一系列操作数。对于一个实施例，结合 SIMD 和多媒体指令，浮点单元 222、224 可对 128 位宽的打包数据操作数进行操作。

[0070] 在该实施例中，μop 调度器 202、204、206 在父负荷已经完成执行之前分发 (dispatch) 相关操作。由于 μop 在处理器 200 中推测地调度和执行，所以处理器 200 还包括处理存储器未命中的逻辑。若数据负荷不在数据高速缓存中，则在流水线中可能存在使调度器具有暂时不正确数据的执行中 (in flight) 相关操作。重放 (replay) 机构跟踪并重新执行采用不正确数据的指令。只有相关操作才需要被重放，并允许不相关操作完成。处理器的一个实施例的调度器和重放机构还设计成捕捉点积操作的指令序列。

[0071] 术语“寄存器”在本文中用来表示用作标识操作数的宏指令的一部分的板载 (on-board) 处理器存储单元。换言之，本文提到的寄存器是从处理器外部 (从程序员的角度) 可见的。但是，实施例的寄存器的含义不应当限于特定的电路类型。相反，实施例的寄存器只需要能够存储和提供数据以及执行本文所述的功能。本文所述的寄存器可通过处理器中的电路采用任何数量的不同技术来实现，例如专用物理寄存器、采用寄存器重命名的动态分配物理寄存器、专用和动态分配物理寄存器的组合等。在一个实施例中，整数寄存器存储 32 位整数数据。一个实施例的寄存器文件还包含用于打包数据的 16 个 XMM 和通

用寄存器、8 个多媒体（例如“EM64T”加法）多媒体 SIMD 寄存器。对于以下论述，寄存器被理解为设计成保存打包数据的数据寄存器，例如采用 Intel Corporation(Santa Clara, California) 开发的 MMX 技术实现的微处理器中的 64 位宽 MMX™ 寄存器（在某些情况下又称作“mm”寄存器）。可用于整数和浮点这两种形式的这些 MMX 寄存器可与伴随 SIMD 和 SSE 指令的打包数据元素配合操作。类似地，与 SSE2、SSE3、SSE4 或者以上（一般称作“SSEx”）的技术相关的 128 位宽 XMM 寄存器也可用于保存这类打包数据操作数。在该实施例中，在存储打包数据和整数数据时，寄存器无需区分这两种数据类型。

[0072] 在以下附图的实例中，描述多个数据操作数。图 3A 示出根据本发明的一个实施例的多媒体寄存器中的各种打包数据类型表示。图 3A 示出 128 位宽操作数的打包字节 310、打包字 320 和打包双字 (dword)330 的数据类型。该实例的打包字节格式 310 是 128 位长的，并包含 16 个打包字节数据元素。字节在这里定义为 8 位数据。各字节数据元素的信息是这样存储的：字节 0 存储在 0 至 7 位，字节 1 存储在 8 至 15 位，字节 2 存储在 16 至 23 位，以及最后，字节 15 存储在 120 至 127 位。这样，寄存器中的所有可用的位都被使用。这种存储方案增加了处理器的存储效率。另外，通过访问 16 个数据元素，现在可并行地对 16 个数据元素执行一个操作。

[0073] 一般来说，数据元素是与相同长度的其它数据元素一起存储在单个寄存器或存储单元中的一段单独的数据。在与 SSEx 技术相关的打包数据序列中，XMM 寄存器中存储的数据元素的数量是 128 位除以单独的数据元素的位的长度。类似地，在与 MMX 和 SSE 技术相关的打包数据序列中，MMX 寄存器中存储的数据元素的数量是 64 位除以单独的数据元素的位的长度。虽然图 3A 所示的数据类型为 128 位长，但是，本发明的实施例还可与 64 位宽或者其它大小的操作数配合操作。该实例的打包字格式 320 是 128 位长的，并且包含 8 个打包字数据元素。各打包字包含 16 位的信息。图 3A 的打包双字格式 330 是 128 位长，并且包含四个打包双字数据元素。各打包双字数据元素包含 32 位的信息。打包四字是 128 位长，并包含两个打包四字数据元素。

[0074] 图 3B 示出备选寄存器中数据存储格式。各打包数据可包括一个以上的独立数据元素。示出三个打包数据格式，即打包半字 341、打包单字 342 和打包双字 343。打包半字 341、打包单字 342 和打包双字 343 的一个实施例包含定点数据元素。对于一备选实施例，打包半字 341、打包单字 342 和打包双字 343 这三者中的一个或多个可包含浮点数据元素。打包半字 341 的一个备选实施例是包含八个 16 位数据元素的 128 位长的。打包单字 342 的一个实施例为 128 位长，并且包含四个 32 位数据元素。打包双字 343 的一个实施例为 128 位长，并且包含两个 64 位数据元素。大家会理解，这类打包数据格式还可扩展为其它寄存器长度，例如扩展为 96 位、160 位、192 位、224 位、256 位或者更大的长度。

[0075] 图 3C 示出根据本发明的一个实施例的多媒体寄存器中的各种有符号和无符号打包数据类型表示。无符号打包字节表示 344 示出在 SIMD 寄存器中的无符号打包字节的存储。各字节数据元素的信息是这样存储的：字节零存储在零至七位，字节一存储在八至十五位，字节二存储在十六至二十三位，以及最后，字节十五存储在一百二十至一百二十七位。这样，寄存器中的所有可用的位都被使用。这种存储方案可增加处理器的存储效率。另外，通过访问十六个数据元素，现在可通过并行方式对十六个数据元素执行一个操作。有符号打包字节表示 345 示出有符号打包字节的存储。注意，每一个字节数据元素的第八位是符

号指示符。无符号打包字表示 346 示出如何在 SIMD 寄存器中存储字七至字零。有符号打包字表示 347 与无符号打包字寄存器内 (in-register) 表示 346 相似。注意，各字数据元素的第十六位是符号指示符。无符号打包双字表示 348 示出如何存储双字数据元素。有符号打包双字表示 349 与无符号打包双字寄存器内表示 348 相似。注意，必要的符号位是各双字数据元素的第三十二位。

[0076] 图 3D 是对操作编码（操作码）格式 360 的一个实施例的描述，其中具有三十二或者更多位，以及寄存器 / 存储器操作数寻址模式符合在以下文献中描述的一种类型的操作码格式 :“IA-32Intel 体系结构软件开发人员手册第 2 卷 :指令集参考”，可在万维网 (www) 的 intel.com/design/litcentr 上从 Intel Corporation(Santa Clara，CAA) 获得。在一个实施例中，点积操作可通过字段 361 和 362 这两者中的一个或多个来编码。可识别每个指令总共两个操作数位置，包括总共两个源操作数标识符 364 和 365。对于点积指令的一个实施例，目标操作数标识符 366 与源操作数标识符 364 相同，而在其它实施例中，它们是不同的。对于一个备选实施例，目标操作数标识符 366 与源操作数标识符 365 相同，而在其它实施例中，它们是不同的。在点积指令的一个实施例中，通过源操作数标识符 364 和 365 标识的源操作数之一被点积操作的结果改写，而在其它实施例中，标识符 364 对应于源寄存器元件，而标识符 365 对应于目标寄存器元件。对于点积指令的一个实施例，操作数标识符 364 和 365 可用来标识 32 位或 64 位源和目标操作数。

[0077] 图 3E 是对具有四十或更多位的另一种备选操作编码（操作码）格式 370 的描述。操作码格式 370 符合操作码格式 360，并包括任选的前置字节 378。点积操作的类型可通过字段 378、371 和 372 这三者中的一个或多个来编码。可通过源操作数标识符 374 和 375 以及通过前置字节 378 来标识每个指令总共两个操作数位置。对于点积指令的一个实施例，前置字节 378 可用来标识 32 位或 64 位源和目标操作数。对于点积指令的一个实施例，目标操作数标识符 376 与源操作数标识符 374 相同，而在其它实施例中，它们是不同的。对于一个备选实施例，目标操作数标识符 376 与源操作数标识符 375 相同，而在其它实施例中，它们是不同的。在一个实施例中，点积操作将操作数标识符 374 和 375 所标识的操作数之一与操作数标识符 374 和 375 所标识的另一个操作数相乘，该点积操作的结果会重写寄存器中的数据元素，而在其它实施例中，标识符 374 和 375 所标识的操作数的点积被写入另一个寄存器中的另一个数据元素。操作码格式 360 和 370 允许部分由 MOD 字段 363 和 373 以及由任选的 scale-index-base 和移位字节所指定的寄存器到寄存器 (register to register)、存储器到寄存器 (memoryto register)、寄存器通过存储器 (register by memory)、寄存器通过寄存器 (register by register)、寄存器通过立即寻址 (register by immediate)、寄存器到存储器 (register to memory) 的寻址。

[0078] 接下来看图 3F，在一些备选实施例中，64 位单指令多数据 (SIMD) 算术运算可通过协处理器数据处理 (CDP) 指令来执行。操作编码（操作码）格式 380 示出具有 CDP 操作码字段 382 和 389 的一种这样的 CDP 指令。对于点积操作的备选实施例，CDP 指令的类型可通过字段 383、384、387 和 388 这四者中的一个或多个来编码。可标识每个指令总共三个操作数位置，包括总共两个源操作数标识符 385、390 和一个目标操作数标识符 386。协处理器的一个实施例可对 8、16、32 和 64 位的值进行操作。对于一个实施例，对整数数据元素执行点积操作。在一些实施例中，可采用选择字段 381 来有条件地执行点积指令。对于一些点

积指令，源数据大小可通过字段 383 来编码。在点积指令的一些实施例中，可在 SIMD 字段上进行零 (Z)、负值 (N)、进位 (C) 和溢出 (V) 检测。对于一些指令，饱和的类型可通过字段 384 来编码。

[0079] 图 4 是根据本发明对打包数据操作数执行点积操作的逻辑的一个实施例的框图。本发明的实施例可实现为与诸如以上所述之类的各种类型的操作数配合工作。对于一种实现，根据本发明的点积操作实现为对指定数据类型进行操作的指令集。例如，提供点积打包单精度 (DPPS) 指令以确定包括整数和浮点在内的 32 位数据类型的点积。类似地，提供点积打包双精度 (DPPD) 指令以确定包括整数和浮点在内的 64 位数据类型的点积。虽然这些指令具有不同名称，但它们执行的一般点积操作是相似的。为了简洁起见，以下讨论和实例在处理数据元素的点积指令的上下文中进行。

[0080] 在一个实施例中，点积指令识别各种信息，包括 :第一数据操作数 DATA A 410 的标识符和第二数据操作数 DATA B 420 的标识符，以及点积操作的所得结果 RESULTANT440 的标识符 (在一个实施例中，它可能与第一数据操作数标识符之一相同)。对于以下论述，DATAA、DATA B 和 RESULTANT 一般称作操作数或数据块，但不限于此，并且还包括寄存器、寄存器文件和存储单元。在一个实施例中，将各点积指令 (DPPS、DPPD) 解码为一个微操作。在一个备选实施例中，可将各指令解码为各种数量的微操作，以对数据操作数执行点积操作。对于该实例，操作数 410、420 是在具有字宽数据元素的源寄存器 / 存储器中存储的 128 位宽的信息段。在一个实施例中，操作数 410、420 保存在 128 位长的 SIMD 寄存器 (如 128 位 SSEx XMM 寄存器) 中。对于一个实施例，RESULTANT 440 也是 XMM 数据寄存器。此外，RESULTANT 440 也可能是与源操作数之一相同的寄存器或存储单元。根据具体实现，操作数和寄存器可能是诸如 32、64 和 256 位等的其它长度，并且具有字节、双字或四字大小的数据元素。虽然该实例的数据元素为字大小，但是，同样的概念可扩展到字节和双字大小的元素。在其中的数据操作数为 64 位宽的一个实施例中，MMX 寄存器用来代替 XMM 寄存器。

[0081] 该实例中的第一操作数 410 包括八个数据元素的集合 :A3、A2、A1 和 A0。各个单独的数据元素对应于所得结果 440 中的数据元素位置。第二操作数 420 包括八个数据段的另一个集合 :B3、B2、B1 和 B0。在这里，数据段具有相等长度，并且各包括数据的单字 (32 位)。但是，数据元素和数据元素位置可具有与字不同的粒度 (granularity)。若各数据元素为字节 (8 位)、双字 (32 位) 或四字 (64 位)，则 128 位操作数分别具有十六字节宽、四个双字宽或者两个四字宽的数据元素。本发明的实施例不限于特定长度的数据操作数或数据段，而是可能对于各实现来适当地确定大小。

[0082] 操作数 410、420 可驻留在寄存器或存储单元或寄存器文件或者它们的组合中。数据操作数 410、420 与点积指令一起被发送到处理器中的执行单元的点积计算逻辑 430。在一个实施例中，当点积指令到达执行单元时，先前应当已经在处理器流水线中对指令进行了解码。因此，点积指令可能采取微操作 (μop) 或者其它某种已解码格式的形式。对于一个实施例，在点积计算逻辑 430 上接收两个数据操作数 410、420。点积计算逻辑 430 产生第一操作数 410 的两个数据元素的第一乘积，其中的两个数据元素的第二乘积处于第二操作数 420 的对应数据元素位置中，以及将第一和第二乘积之和存储在所得结果 440 中的适当位置上，该位置可能对应于与第一或第二操作数相同的存储单元。在一个实施例中，第一和第二操作数中的数据元素为单精度 (例如 32 位)，而在其它实施例中，第一和第二操作数中

的数据元素为双精度（例如 64 位）。

[0083] 对于一个实施例，并行处理所有数据位置的数据元素。在另一个实施例中，一次可共同处理某个部分的数据元素位置。在一个实施例中，根据是执行 DPPD 还是 DPPS，所得结果 440 分别包括两个或四个可能的点积结果位置：$DOT\text{-}PRODUCT_{A310\text{-}0}$、$DOT\text{-}PRODUCT_{A63\text{-}32}$、$DOT\text{-}PRODUCT_{A95\text{-}64}$、$DOT\text{-}PRODUCT_{A127\text{-}96}$（对于 DPPS 指令的结果），以及 $DOT\text{-}PRODUCT_{A63\text{-}0}$、$DOT\text{-}PRODUCT_{A127\text{-}64}$（对于 DPPD 指令的结果）。

[0084] 在一个实施例中，所得结果 440 中的点积结果的位置取决于与点积指令相关联的选择字段。例如，对于 DPPS 指令，所得结果 440 中的点积结果的位置在选择字段等于第一值时为 $DOT\text{-}PRODUCT_{A31\text{-}0}$，在选择字段等于第二值时为 $DOT\text{-}PRODUCT_{A63\text{-}32}$，在选择字段等于第三值时为 $DOT\text{-}PRODUCT_{A95\text{-}64}$，以及在选择字段等于第四值时为 $DOT\text{-}PRODUCT_{A127\text{-}64}$。在 DPPD 指令的情况下，所得结果 440 中的点积结果的位置在选择字段为第一值时是 $DOT\text{-}RPODUCT_{A63\text{-}0}$，在选择字段为第二值时是 $DOT\text{-}PRODUCT_{A127\text{-}64}$。

[0085] 图 5A 示出根据本发明的一个实施例的点积指令的操作。具体来说，图 5A 示出根据一个实施例的 DPPS 指令的操作。在一个实施例中，图 5A 所示的实例的点积操作实质上可由图 4 的点积计算逻辑 430 来执行。在另一些实施例中，图 5A 的点积操作可由包括硬件、软件或者它们的某种结合在内的其它逻辑来执行。

[0086] 在另一些实施例中，图 4、图 5A 和图 5B 所示的操作可按照任何组合或顺序来执行，以产生点积结果。在一个实施例中，图 5A 示出包括总共存储各为 32 位的四个单精度浮点或整数值 A0-A3 的存储单元的 128 位源寄存器 501a。类似地，图 5A 中所示的是包括总共存储各为 32 位的四个单精度浮点或整数值 B0-B3 的存储单元的 128 位目标寄存器 505a。在一个实施例中，源寄存器中存储的每个值 A0-A3 与目标寄存器的对应位置中存储的对应值 B0-B3 相乘，以及各所得值 A0*B0、A1*B1、A2*B2、A3*B3（本文中称作“乘积”）存储在包括总共存储各为 32 位的四个单精度浮点或整数值的存储单元的第一 128 位临时寄存器（“TEMP1”）510a 的对应存储单元中。

[0087] 在一个实施例中，将乘积对相加在一起，以及各个和数（本文中称作“中间和数”）存储到第二 128 位临时寄存器（“TEMP2”）515a 和第三 128 位临时寄存器（“TEMP3”）520a 的存储单元中。在一个实施例中，乘积存储到第一和第二临时寄存器的最低有效 32 位元素存储单元中。在另一些实施例中，它们可存储在第一和第二临时寄存器的其它元素存储单元中。此外，在一些实施例中，乘积可存储在相同寄存器（如第一或第二临时寄存器）中。

[0088] 在一个实施例中，中间和数相加在一起（本文中称作“最终和数”），并存储到第四 128 位临时寄存器（“TEMP4”）525a 的存储单元中。在一个实施例中，最终和数存储到 TEMP4 的最低有效 32 位存储单元中，而在另一些实施例中，最终和数存储到 TEMP4 的其它存储单元中。最终和数然后存储到目标寄存器 505a 的存储单元中。最终和数将要存储到其中的准确的存储单元可取决于点积指令中可配置的变量。在一个实施例中，包含多个位存储单元的立即字段（“IMMy[x]”）可用来确定最终和数将要存储到其中的目标寄存器存储单元。例如，在一个实施例中，若 IMM8[0] 字段包含第一值（例如“1”），则最终和数存储到目标寄存器的存储单元 B0 中，若 IMM8[1] 字段包含第一值（例如“1”），则最终和数存储到存储单元 B1 中，若 IMM8[2] 字段包含第一值（例如“1”），则最终和数存储到目标寄存器的存储单元 B2 中，以及若 IMM8[3] 字段包含第一值（例如“1”），则最终和数存储到目标寄存器的存

储单元 B3 中。在另一些实施例中，其它立即字段可用来确定最终和数将要存储到其中的目标寄存器的存储单元。

[0089] 在一个实施例中，立即字段可用来控制各乘法和加法运算是否在图 5A 所示的操作中执行。例如，IMM8[4] 可用来表明（例如通过设置为“0”或“1”）A0 是否将与 B0 相乘且结果被存储到 TEMP1 中。类似地，IMM8[5] 可用来表明（例如通过设置为“0”或“1”）A1 是否将与 B1 相乘且结果被存储到 TEMP1 中。同样，IMM8[6] 可用来表明（例如通过设置为“0”或“1”）A2 是否将与 B2 相乘且结果被存储到 TEMP1 中。最后，IMM8[7] 可用来表明（例如通过设置为“0”或“1”）A3 是否将与 B3 相乘且结果被存储到 TEMP1 中。

[0090] 图 5B 示出根据一个实施例的 DPPD 指令的操作。DPPS 与 DPPD 指令之间的一个差别在于，DPPD 对双精度浮点和整数值（例如 64 位值）而不是单精度值进行操作。相应地，在一个实施例中，执行 DPPD 指令与执行 DPPS 指令相比，存在更少要管理的数据元素，因此涉及更少的中间操作和存储装置（例如寄存器）。

[0091] 在一个实施例中，图 5B 示出包括总共存储各为 64 位的两个双精度浮点或整数值 A0-A1 的存储单元的 128 位源寄存器 501b。类似地，图 5B 中所示的是包括总共存储各为 64 位的两个双精度浮点或整数值 B0-B1 的存储单元的 128 位目标寄存器 505b。在一个实施例中，源寄存器中存储的各个值 A0-A1 与目标寄存器的对应位置中存储的对应值 B0-B1 相乘，以及各所得值 A0*B0、A1*B1（本文中称作“乘积”）存储在包括总共存储各为 64 位的两个双精度浮点或整数值的存储单元的第一 128 位临时寄存器（“TEMP1”）510b 的对应存储单元中。

[0092] 在一个实施例中，乘积对相加在一起，以及各个和数（本文中称作“最终和数”）存储到第二 128 位临时寄存器（“TEMP2”）515b 的存储单元中。在一个实施例中，乘积和最终和数分别存储到第一和第二临时寄存器的最低有效 64 位元素存储单元中。在另一些实施例中，它们可存储在第一和第二临时寄存器的其它元素存储单元中。

[0093] 在一个实施例中，最终和数存储到目标寄存器 505b 的存储单元中。最终和数将要存储到其中的准确的存储单元可取决于点积指令中可配置的变量。在一个实施例中，包含多个位存储单元的立即字段（“IMMy[x]”）可用来确定最终和数将要存储到其中的目标寄存器存储单元。例如，在一个实施例中，若 IMM8[0] 字段包含第一值（例如“1”），则最终和数存储到目标寄存器的存储单元 B0 中，若 IMM8[1] 字段包含第一值（例如“1”），则最终和数存储到存储单元 B1 中。在另一些实施例中，其它立即字段可用来确定最终和数将要存储到其中的目标寄存器的存储单元。

[0094] 在一个实施例中，立即字段可用来控制各乘法运算是否在图 5B 所示的点积操作中执行。例如，IMM8[4] 可用来表明（例如通过设置为“0”或“1”）A0 是否将与 B0 相乘且结果被存储到 TEMP1 中。类似地，IMM8[5] 可用来表明（例如通过设置为“0”或“1”）A1 是否将与 B1 相乘且结果被存储到 TEMP1 中。在另一些实施例中，可采用用于确定是否执行点积的乘法运算的其它控制技术。

[0095] 图 6A 是根据一个实施例对单精度整数或浮点值执行点积操作的电路 600a 的框图。该实施例的电路 600a 通过乘法器 610a-613a 将两个寄存器 601a 和 605a 的对应单精度元素相乘，其结果可采用立即字段 IMM8[7:4] 由复用器 615a-618a 进行选择。作为备选的方案，复用器 615a-618a 可选择零值采代替各元素的乘法运算的对应乘积。复用器 615a-618a

选择的结果然后由加法器 620a 相加在一起，且相加的结果被存储在结果寄存器 630a 的单元的任一个中，根据立即字段 IMM8[3:0] 的值，采用复用器 625a-628a 来选择来自加法器 620a 的对应和数结果。在一个实施例中，若和数结果没有被选择成存储在结果单元中，则复用器 625a-628a 可选择零值来填充结果寄存器 630a 的单元。在另一些实施例中，更多加法器可用来产生各个乘积之和。此外，在一些实施例中，中间存储单元可用来存储乘积或和数结果，直到对它们进行进一步操作为止。

[0096] 图 6B 是根据一个实施例对单精度整数或浮点值执行点积操作的电路 600b 的框图。该实施例的电路 600b 通过乘法器 610b、612b 将两个寄存器 601b 和 605b 的对应单精度元素相乘，其结果可采用立即字段 IMM8[7:4] 由复用器 615b、617b 进行选择。作为备选的方案，复用器 615b、618b 可选择零值来代替各元素的乘法运算的对应乘积。复用器 615b、618b 选择的结果然后由加法器 620b 相加在一起，且相加的结果被存储在结果寄存器 630b 的单元的任一个中，根据立即字段 IMM8[3:0] 的值，采用复用器 625b、627b 来选择来自加法器 620b 的对应和数结果。在一个实施例中，若和数结果没有被选择成存储在结果单元中，则复用器 625b-627b 可选择零值来填充结果寄存器 630b 的单元。在另一些实施例中，更多加法器可用来产生各个乘积之和。此外，在一些实施例中，中间存储单元可用来存储乘积或和数结果，直到对它们进行进一步操作为止。

[0097] 图 7A 是根据一个实施例执行 DPPS 指令的操作的伪码表示。图 7A 所示的伪码表明，源寄存器（“SRC”）中在 0-31 位上存储的单精度浮点或整数值将与目标寄存器（“DEST”）中在 0-31 位上存储的单精度浮点或整数值相乘，且仅当立即字段（“IMM8[4]”）中存储的立即值等于“1”时，才将结果存储在临时寄存器（“TEMP1”）的 0-31 位中。否则，位存储单元 31-0 可包含空值，如全零。

[0098] 图 7A 中还示出了伪码来表明，SRC 寄存器中在 63-32 位上存储的单精度浮点或整数值将与 DEST 寄存器中在 63-32 位上存储的单精度浮点或整数值相乘，且仅当立即字段（“IMM8[5]”）中存储的立即值等于“1”时，才将结果存储在 TEMP1 寄存器的 63-32 位中。否则，位存储单元 63-32 可包含空值，如全零。

[0099] 类似地，图 7A 中还示出了伪码来表明，SRC 寄存器中在 95-64 位上存储的单精度浮点或整数值将与 DEST 寄存器中在 95-64 位上存储的单精度浮点或整数值相乘，且仅当立即字段（“IMM8[6]”）中存储的立即值等于“1”时，才将结果存储在 TEMP1 寄存器的 95-64 位中。否则，位存储单元 95-64 可包含空值，如全零。

[0100] 最后，图 7A 中还示出了伪码来表明，SRC 寄存器中在 127-96 位上存储的单精度浮点或整数值将与 DEST 寄存器中在 127-96 位上存储的单精度浮点或整数值相乘，且仅当立即字段（“IMM8[7]”）中存储的立即值等于“1”时，才将结果存储在 TEMP1 寄存器的 127-96 位中。否则，位存储单元 127-96 可包含空值，如全零。

[0101] 接下来，图 7A 示出 31-0 位被加到 TEMP1 的 63-32 位，且结果被存储到第二临时寄存器（“TEMP2”）的位存储单元 31-0 中。类似地，95-64 位被加到 TEMP1 的 127-96 位，且结果被存储到第三临时寄存器（“TEMP3”）的位存储单元 31-0 中。最后，TEMP2 的 31-0 位被加到 TEMP3 的 31-0 位，且结果被存储到第四临时寄存器（“TEMP4”）的位存储单元 31-0 中。

[0102] 在一个实施例中，临时寄存器中存储的数据然后被存储到 DEST 寄存器。要存储数

据的DEST寄存器中的具体位置可取决于DPPS指令中的其它字段，如IMM8[x]中的字段。具体来说，图7A说明，在一个实施例中，TEMP4的31-0位在IMM8[0]等于“1”时存储到DEST位存储单元31-0中，在IMM8[1]等于“1”时存储到DEST位存储单元63-32中，在IMM8[2]等于“1”时存储到DEST位存储单元95-64中，或者在IMM8[3]等于“1”时存储到DEST位存储单元127-96中。否则，对应的DEST位存储单元将包含空值，如全零。

[0103] 图7B是根据一个实施例执行DPPD指令的操作的伪码表示。图7B所示的伪码表明，源寄存器(“SRC”)中在63-0位上存储的单精度浮点或整数值将与目标寄存器(“DEST”)中在63-0位上存储的单精度浮点或整数值相乘，且仅当立即字段(“IMM8[4]”)中存储的立即值等于“1”时，才将结果存储在临时寄存器(“TEMP1”)的位63-0中。否则，位存储单元63-0可包含空值，如全零。

[0104] 图7B中还示出了伪码来表明，SRC寄存器中在127-64位上存储的单精度浮点或整数值将与DEST寄存器中在127-64位上存储的单精度浮点或整数值相乘，且仅当立即字段(“IMM8[5]”)中存储的立即值等于“1”时，才将结果存储在TEMP1寄存器的位127-64中。否则，位存储单元127-64可包含空值，如全零。

[0105] 接下来，图7B示出，63-0位被加到TEMP1的127-64位，且结果被存储到第二临时寄存器(“TEMP2”)的位存储单元63-0中。在一个实施例中，临时寄存器中存储的数据然后可存储到DEST寄存器中。要存储数据的DEST寄存器中的具体位置可取决于DPPS指令中的其它字段，如IMM8[x]中的字段。具体地说，图7A示出，在一个实施例中，若IMM8[0]等于“1”，则TEMP2的63-0位存储到DEST位存储单元63-0中，或者若IMM8[1]等于“1”，则TEMP2的63-0位存储在DEST位存储单元127-64中。否则，对应的DEST位存储单元将包含空值，如全零。

[0106] 图7A和图7B中公开的操作只是可用于本发明的一个或多个实施例的操作的一种表示。具体地说，图7A和图7B所示的伪码对应于按照具有128位寄存器的一个或多个处理器体系结构所执行的操作。其它实施例可在具有任何大小的寄存器或者其它类型的存储区的处理器体系结构中执行。此外，其它实施例可能不采用与如图7A和图7B所示的寄存器完全相同的寄存器。例如，在一些实施例中，不同数量的临时寄存器或者根本没有任何寄存器可用来存储操作数。最后，本发明的实施例可采用任何数量的寄存器或数据类型在众多处理器或处理核心之间来执行。

[0107] 这样，公开了用于执行点积操作的技术。虽然在附图中描述和说明了某些示范性实施例，但是要理解，这些实施例只是对宽泛的发明的说明而不是限制，并且本发明不限于所示及所述的具体构造和配置，因为本领域的技术人员在研究本公开之后可能会想到其它各种修改。在例如增长迅速并且不易预见进一步发展的这样的技术的领域中，通过实现技术发展的推动，可在不背离本公开的原理或所附权利要求的范围的条件下，容易地对所公开的实施例进行配置和细节方面的修改。

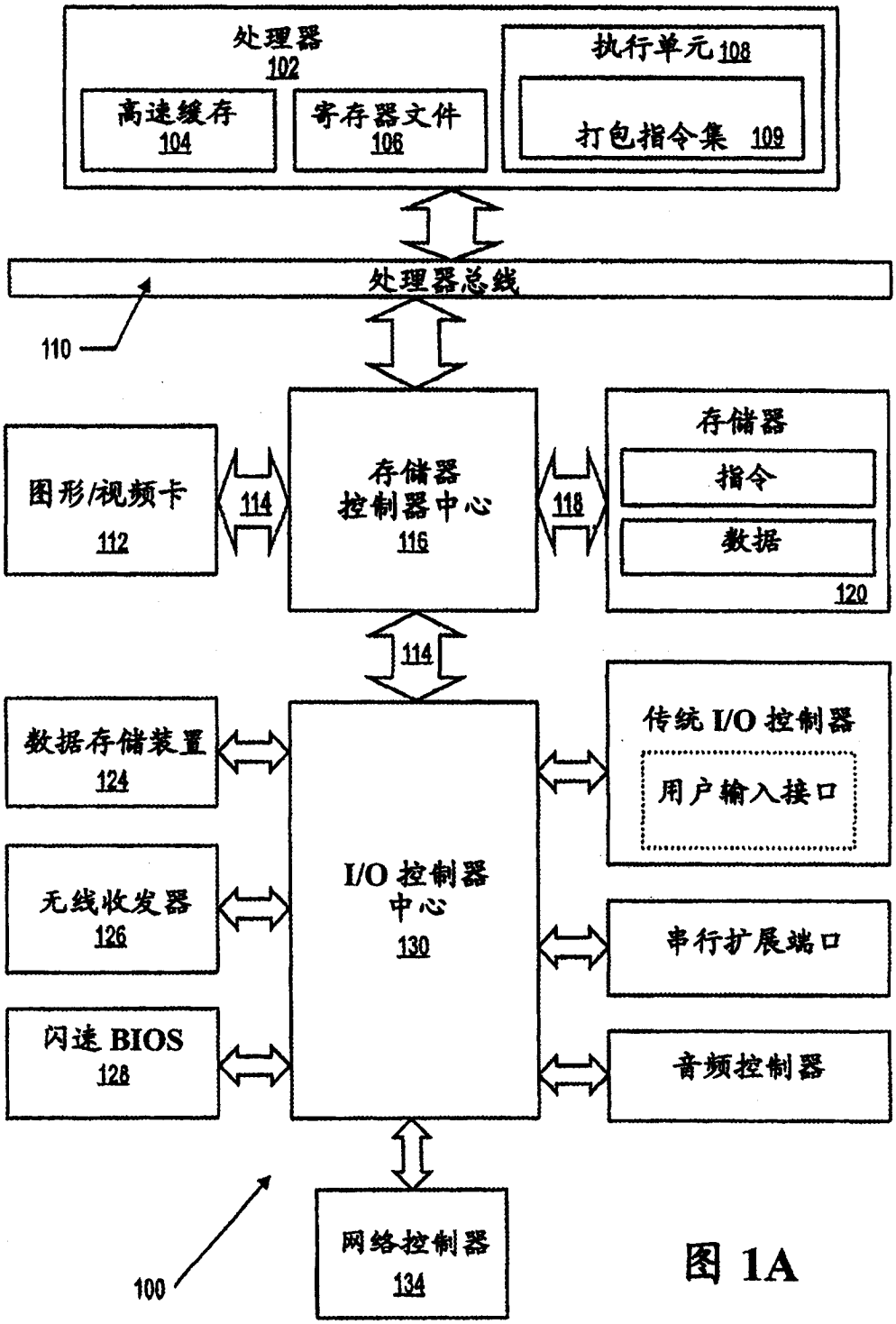

处理器
102
高速缓存
104
寄存器文件
106
执行单元 108
打包指令集 109
处理器总线
110
图形/视频卡
112
114
存储器
控制器中心
116
118
存储器
指令
数据
120
114
数据存储装置
124
无线收发器
126
闪速 BIOS
128
I/O 控制器
中心
130
传统 I/O 控制器
用户输入接口
串行扩展端口
音频控制器
网络控制器
134
100

图 1A

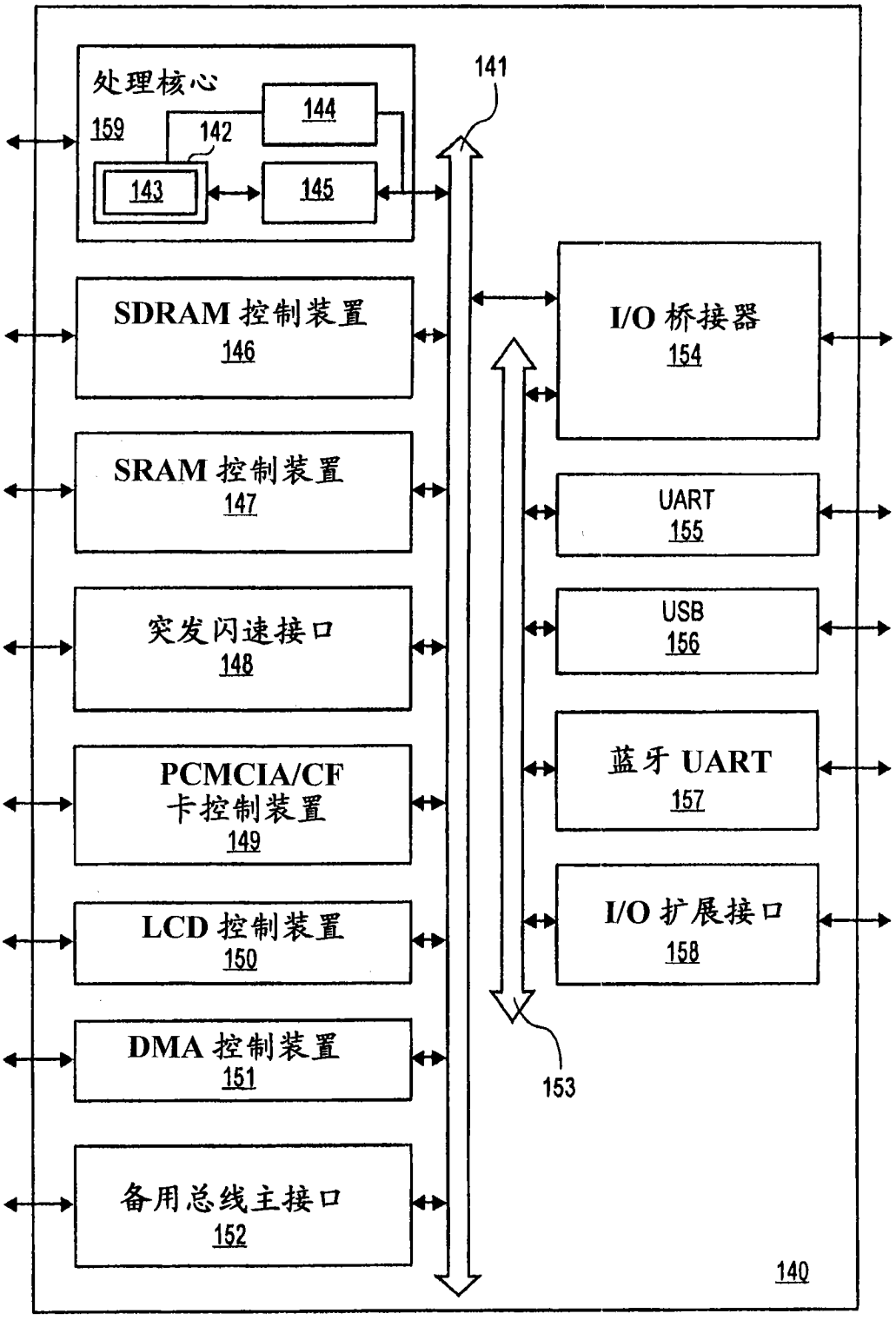

处理核心
159
144
142
143
145
141
SDRAM 控制装置
146
SRAM 控制装置
147
突发闪速接口
148
PCMCIA/CF
卡控制装置
149
LCD 控制装置
150
DMA 控制装置
151
备用总线主接口
152
I/O 桥接器
154
UART
155
USB
156
蓝牙 UART
157
I/O 扩展接口
158
153
140

图 1B

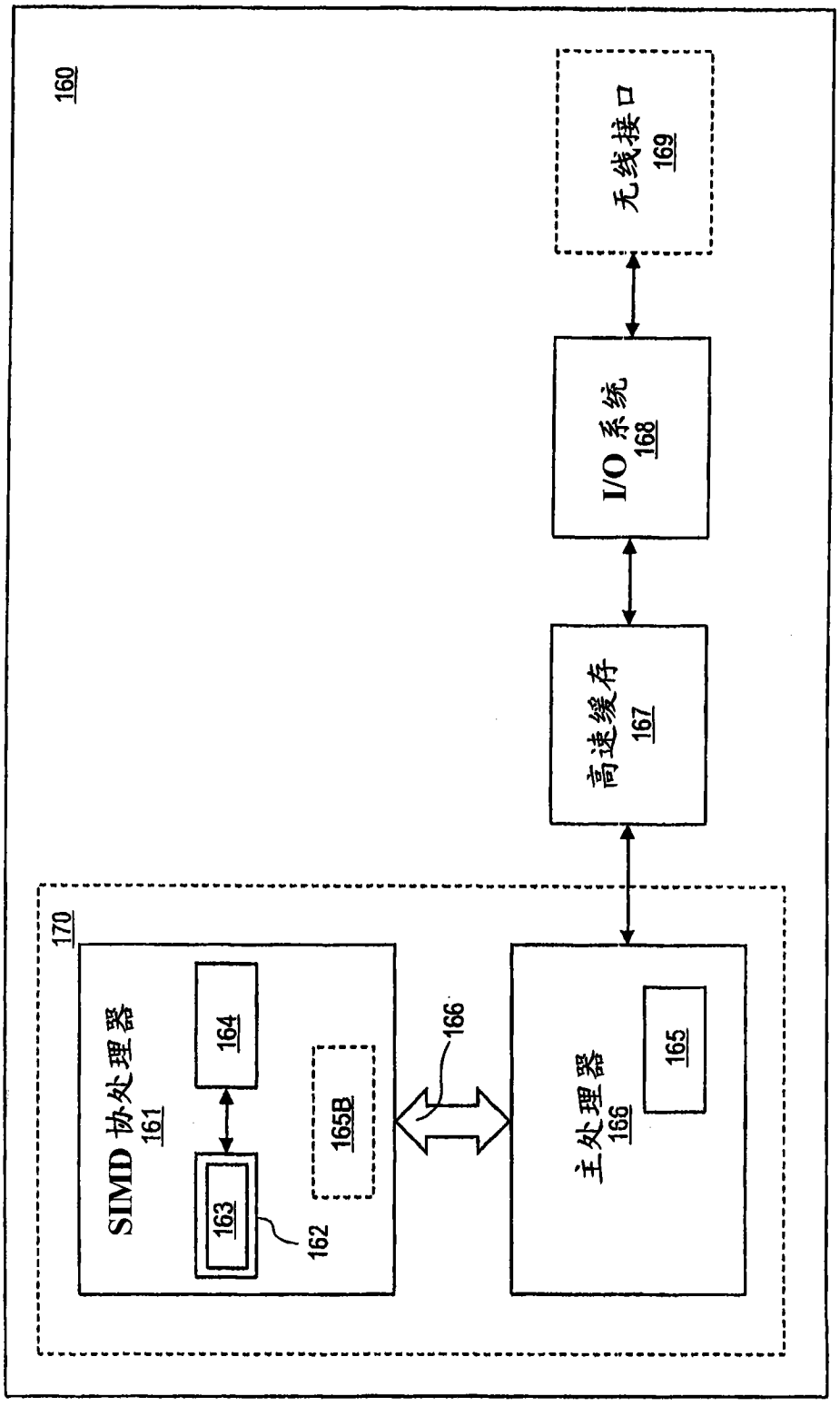
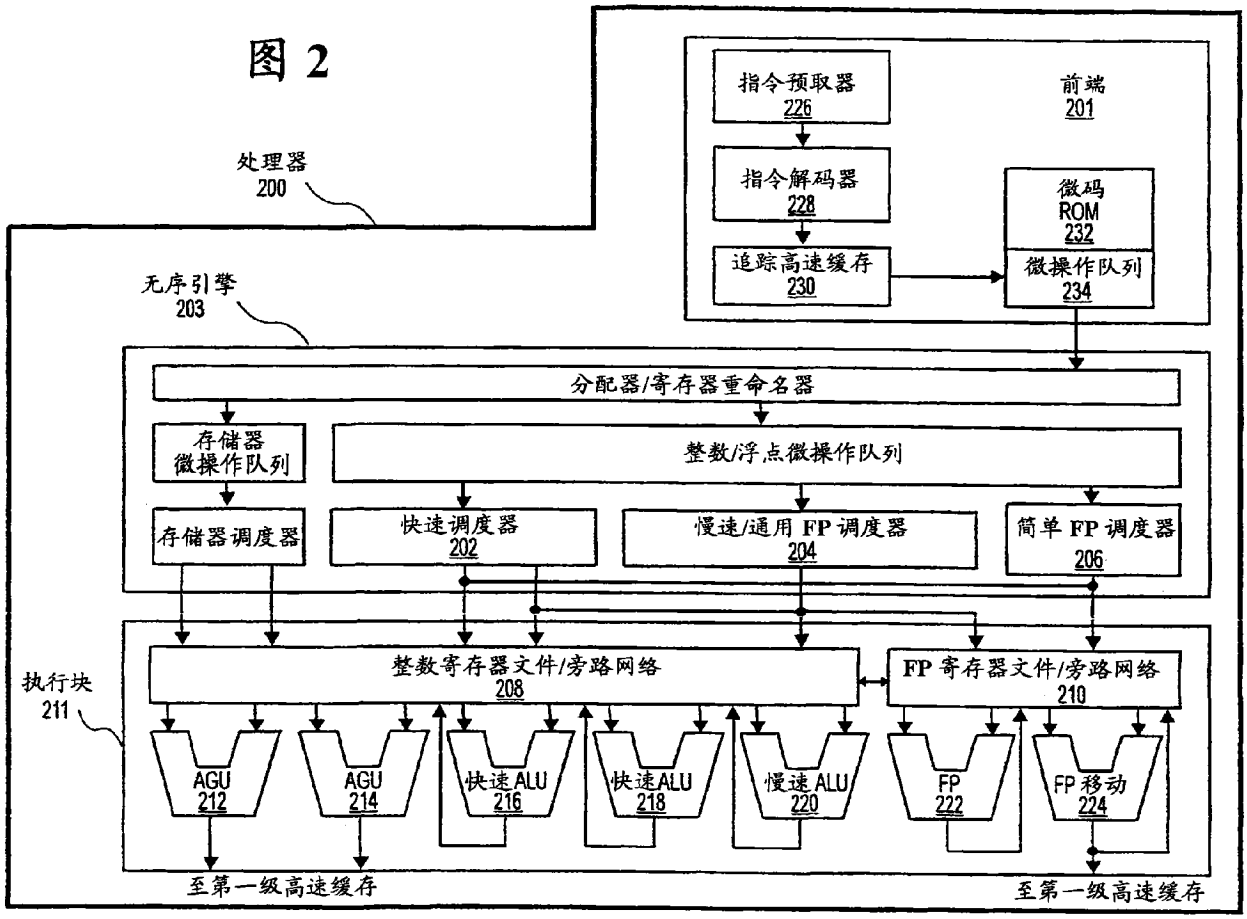

160
无线接口
169
I/O 系统
168
高速缓存
167
170
SIMD 协处理器
161
164
163
162
165B
166
主处理器
166
165

图 1C

图 2


处理器
200
前端
201
指令预取器
226
指令解码器
228
追踪高速缓存
230
微码
ROM
232
微操作队列
234
无序引擎
203
分配器/寄存器重命名器
存储器
微操作队列
整数/浮点微操作队列
存储器调度器
快速调度器
202
慢速/通用 FP 调度器
204
简单 FP 调度器
206
执行块
211
整数寄存器文件/旁路网络
208
FP 寄存器文件/旁路网络
210
AGU
212
AGU
214
快速ALU
216
快速ALU
218
慢速 ALU
220
FP
222
FP 移动
224
至第一级高速缓存
至第一级高速缓存

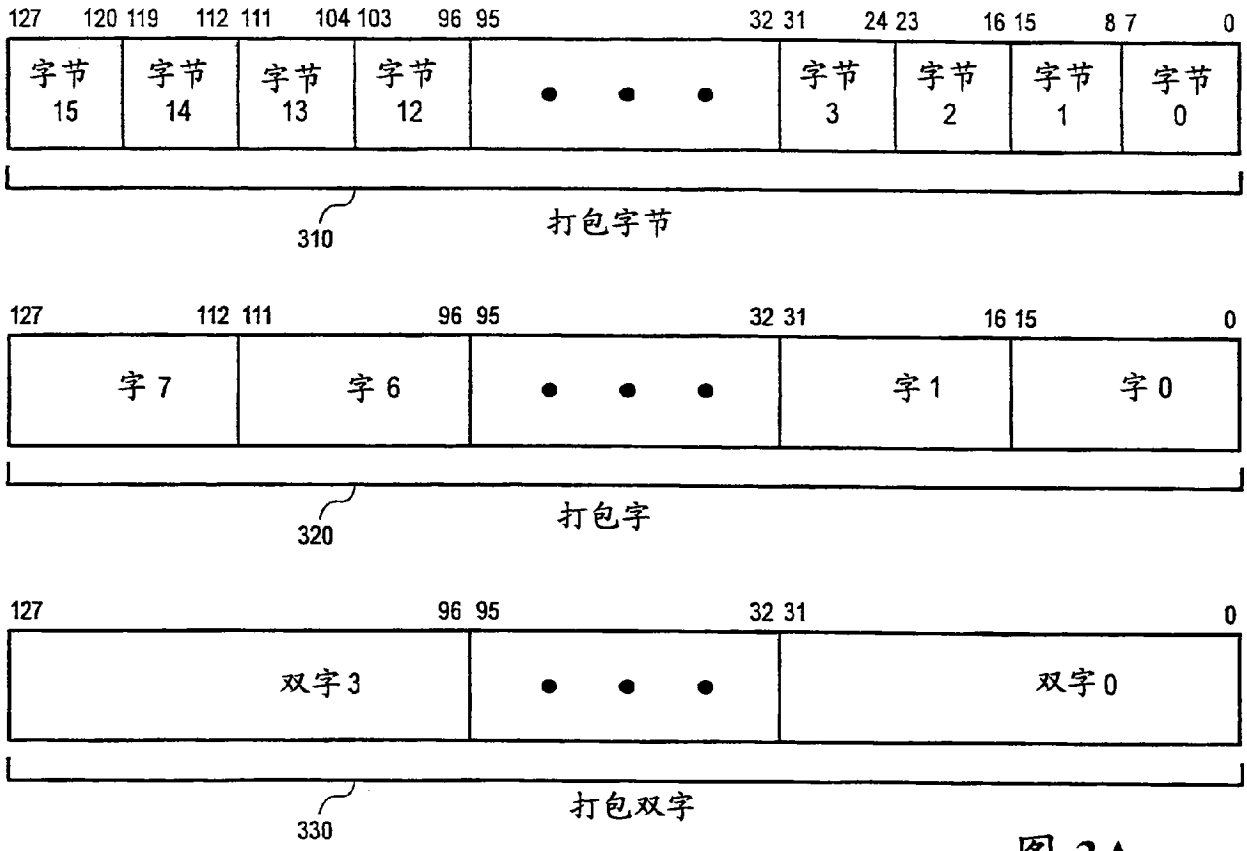

127 120 119 112 111 104 103 96 95 32 31 24 23 16 15 8 7 0
字节 15
字节 14
字节 13
字节 12
字节 3
字节 2
字节 1
字节 0
310
打包字节
127 112 111 96 95 32 31 16 15 0
字 7
字 6
字 1
字 0
320
打包字
127 96 95 32 31 0
双字 3
双字 0
330
打包双字

图 3A

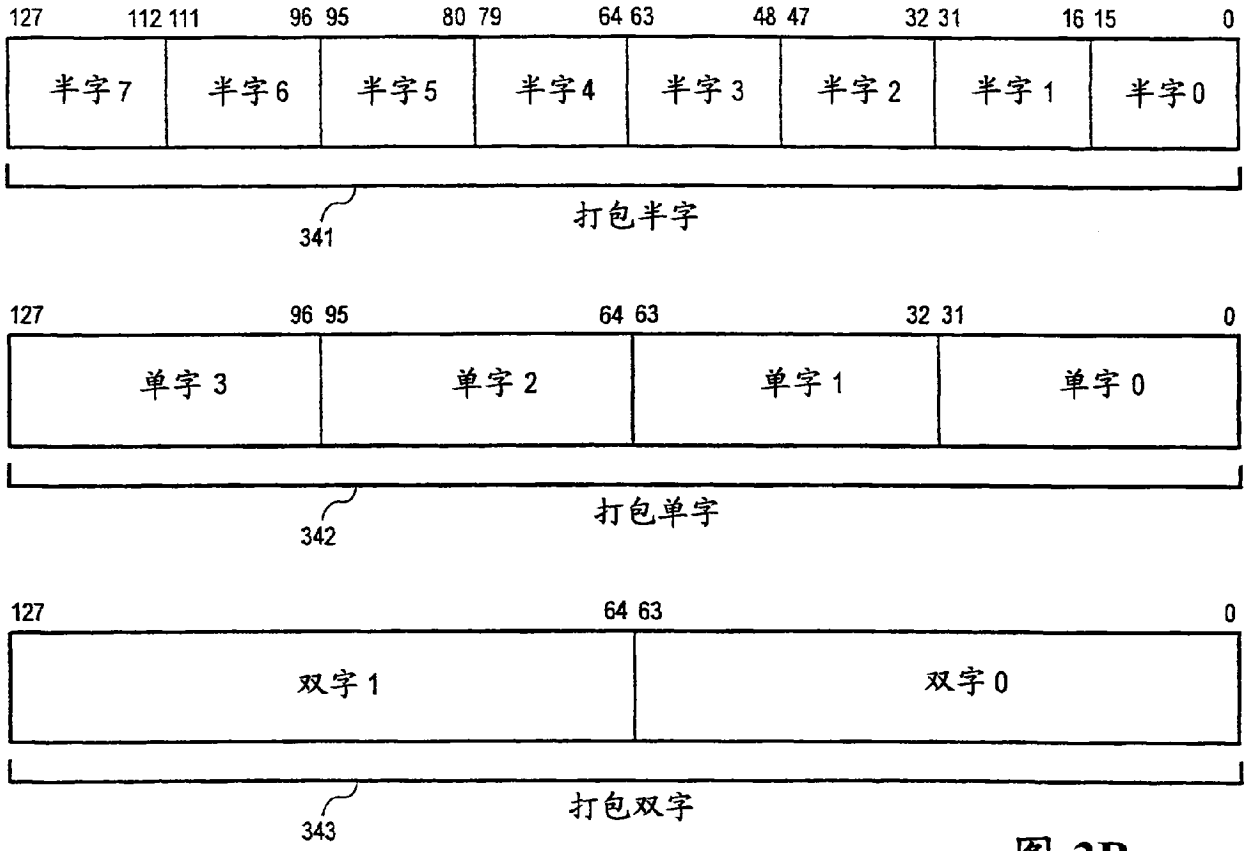
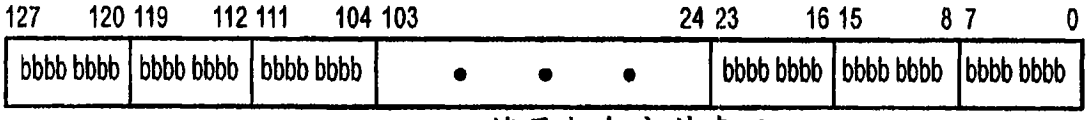

127
112 111
96 95
80 79
64 63
48 47
32 31
16 15
0
半字 7
半字 6
半字 5
半字 4
半字 3
半字 2
半字 1
半字 0
341
打包半字
127
96 95
64 63
32 31
0
单字 3
单字 2
单字 1
单字 0
342
打包单字
127
64 63
0
双字 1
双字 0
343
打包双字

图 3B

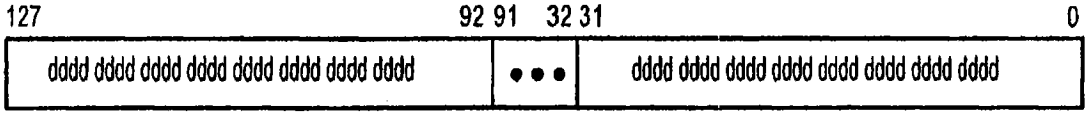


无符号打包字节表示 344

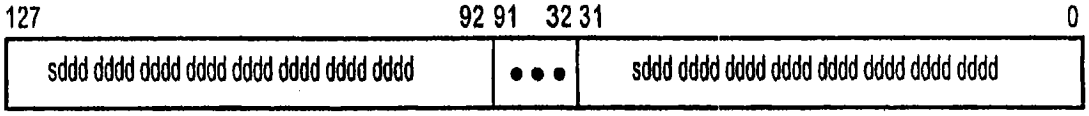
有符号打包字节表示 345

无符号打包字表示 346

有符号打包字表示 347



无符号打包双字表示 348



有符号打包双字表示 349

图 3C

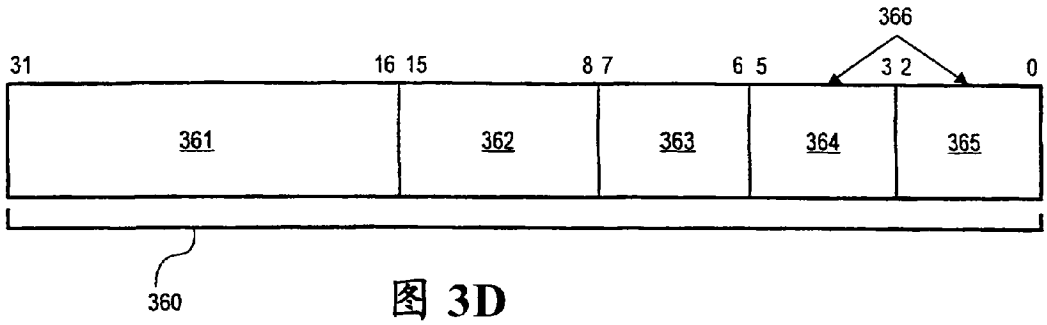

366
31
16 15
8 7
6 5
3 2
0
361
362
363
364
365
360

图 3D

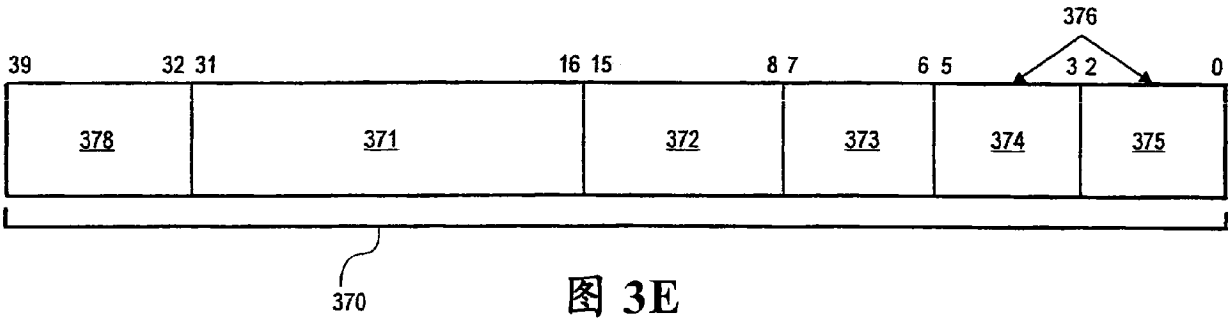

376
39
32 31
16 15
8 7
6 5
3 2
0
378
371
372
373
374
375
370

图 3E

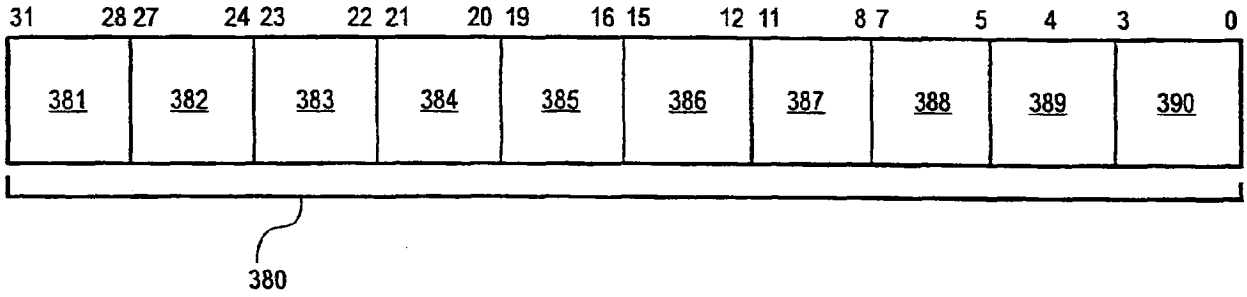

31
28 27
24 23
22 21
20 19
16 15
12 11
8 7
5
4
3
0
381
382
383
384
385
386
387
388
389
390
380

图 3F

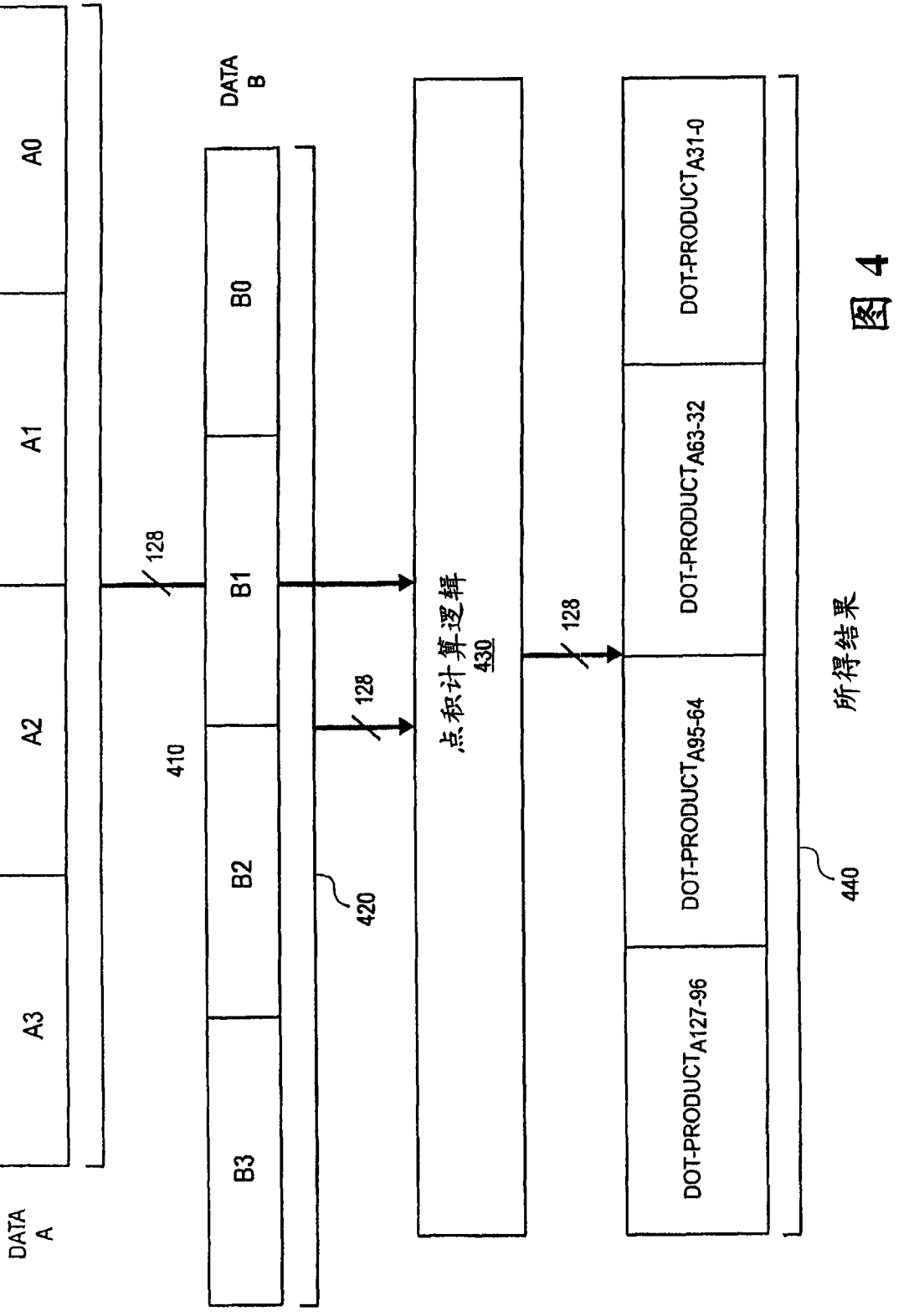

A0
A1
A2
A3
DATA A
128
410
DATA B
B0
B1
B2
B3
128
420
点积计算逻辑
430
128
DOT-PRODUCT A31-0
DOT-PRODUCT A63-32
DOT-PRODUCT A95-64
DOT-PRODUCT A127-96
440
所得结果

图 4

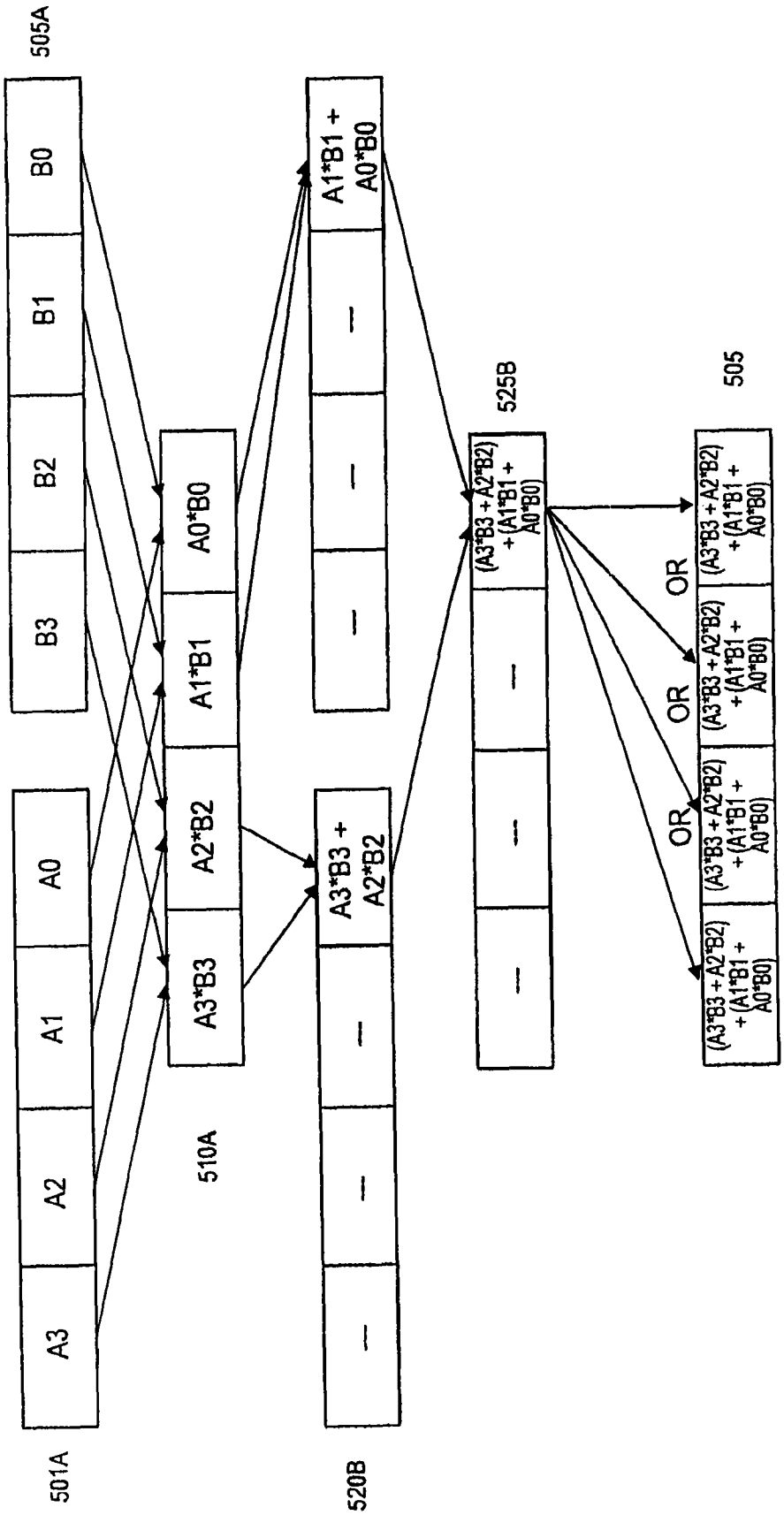

505A
B0
B1
B2
B3
A0
A1
A2
A3
501A
A0*B0
A1*B1
A2*B2
A3*B3
510A
A1*B1 + A0*B0
A3*B3 + A2*B2
520B
525B
(A3*B3 + A2*B2) + (A1*B1 + A0*B0)
OR
OR
OR
505

图 5A

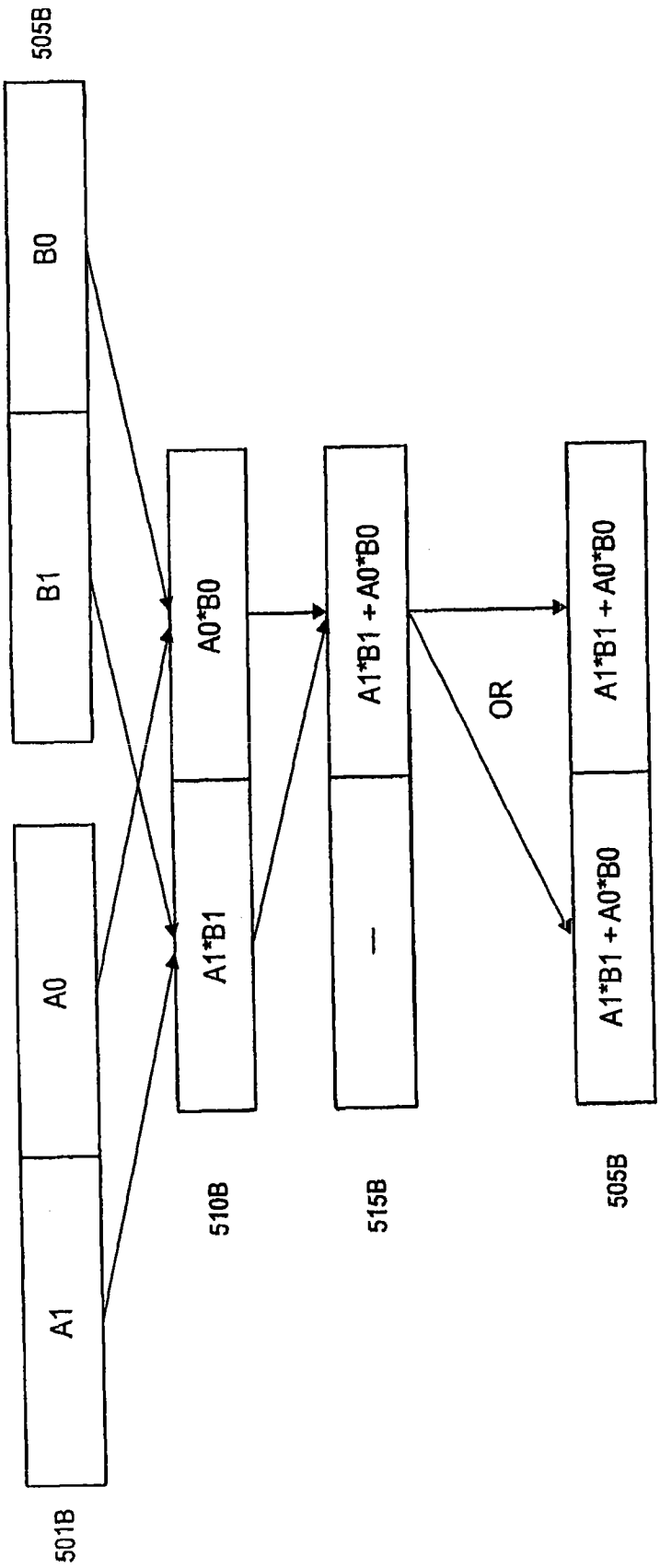

505B
B0
B1
A0
A1
501B
A0*B0
A1*B1
510B
A1*B1 + A0*B0
—
515B
OR
A1*B1 + A0*B0
A1*B1 + A0*B0
505B

图 5B

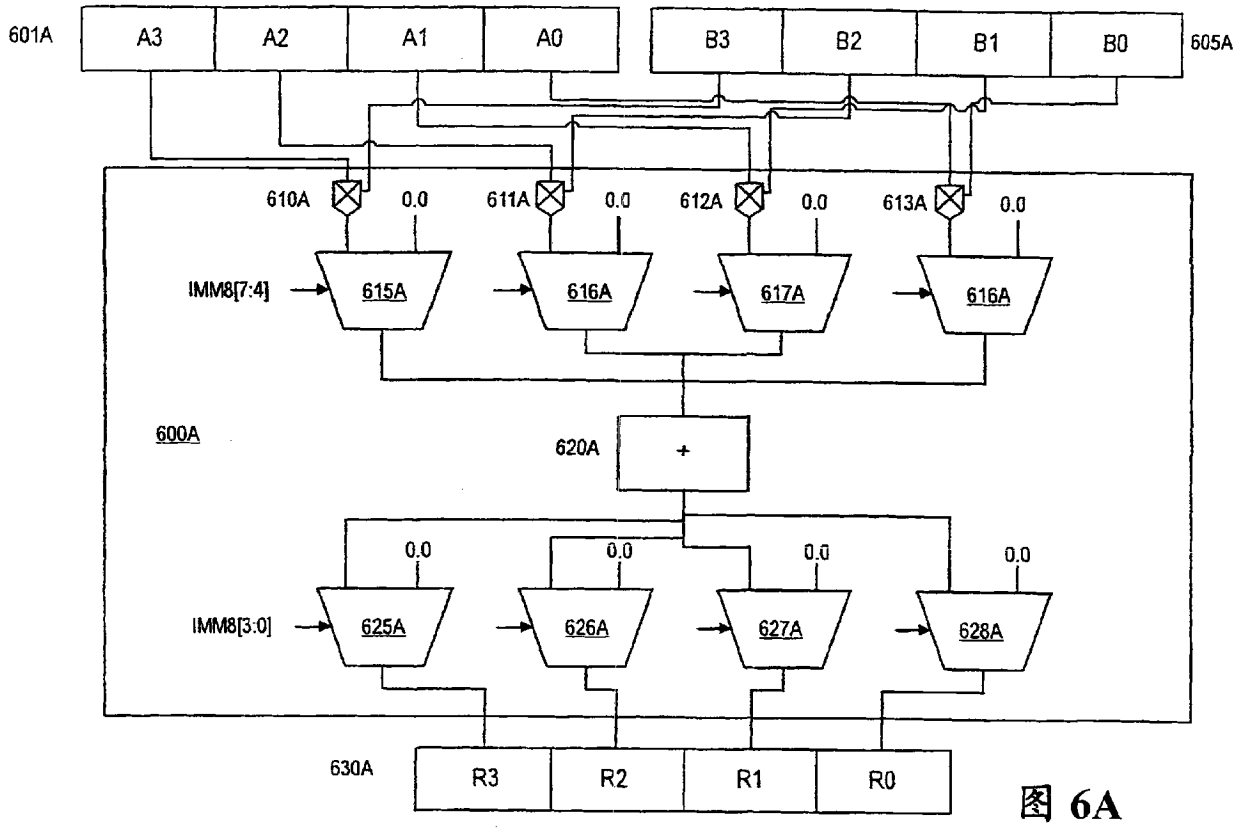

601A
A3
A2
A1
A0
B3
B2
B1
B0
605A
610A
0.0
611A
0.0
612A
0.0
613A
0.0
IMM8[7:4]
615A
616A
617A
616A
600A
620A
+
0.0
0.0
0.0
0.0
IMM8[3:0]
625A
626A
627A
628A
630A
R3
R2
R1
R0

图 6A

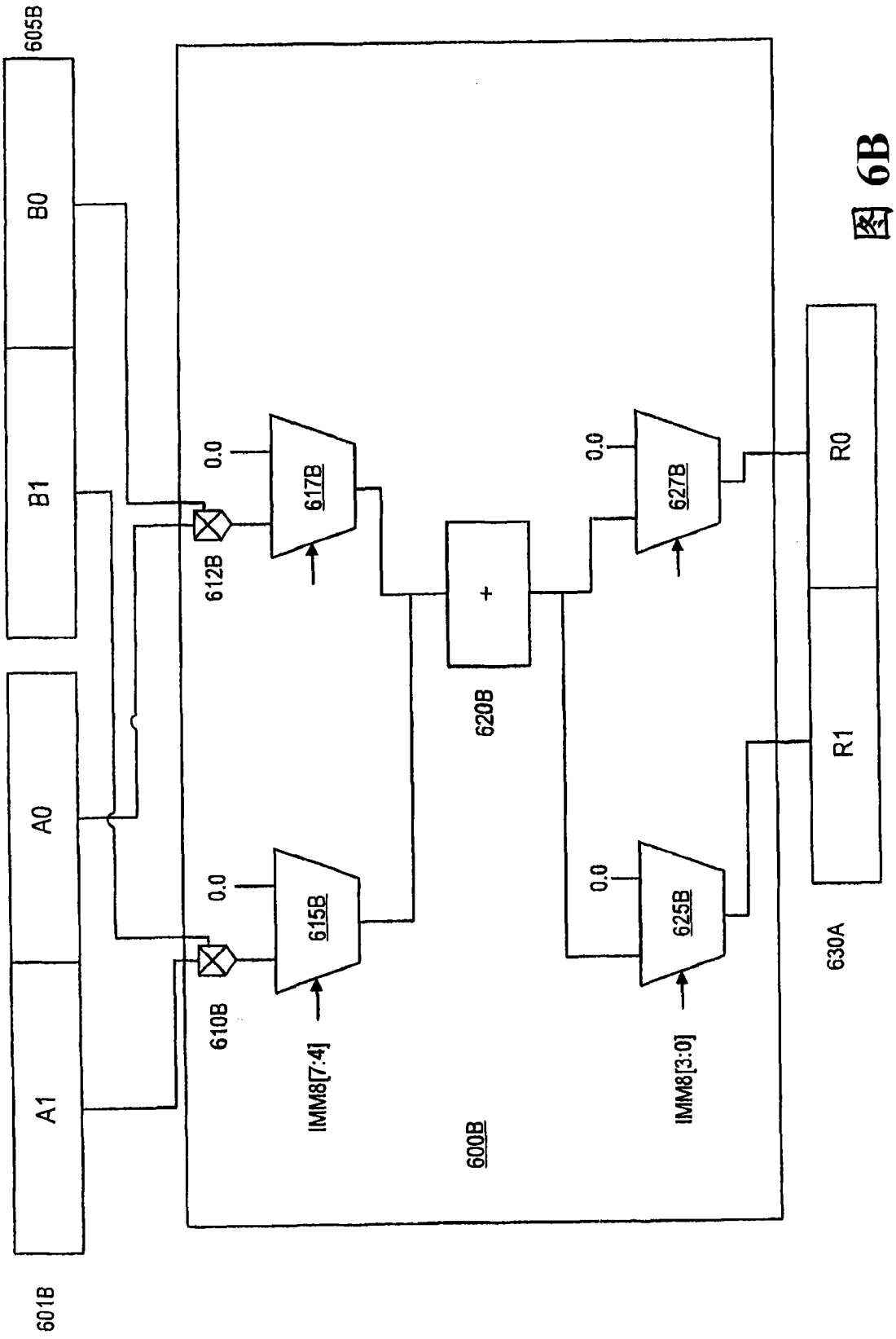

605B
B0
B1
A0
A1
601B
612B
610B
0.0
617B
615B
IMM8[7:4]
620B
+
627B
625B
IMM8[3:0]
R0
R1
630A
600B

图 6B

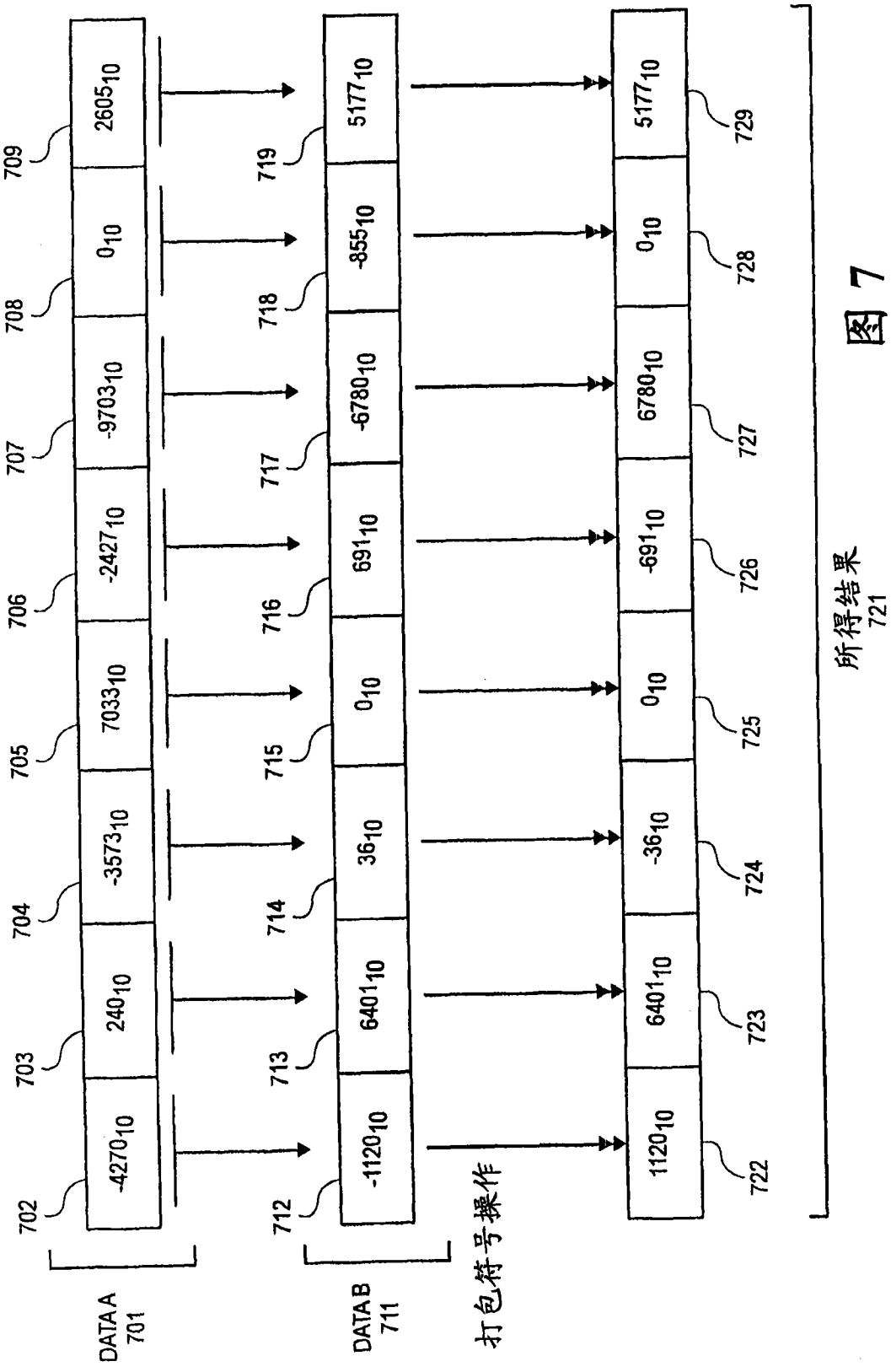

DATA A
701
DATA B
711
打包符号操作
所得结果
721
702
703
704
705
706
707
708
709
712
713
714
715
716
717
718
719
722
723
724
725
726
727
728
729
-427010
24010
-35731 0
703310
-242710
-970310
010
260510
-112010
640110
3610
010
6911 0
-678010
-85510
517710
112010
640110
-3610
010
-6911 0
678010
010
517710

图 7

```
IF (imm8 [4] == 1) THEN Temp1 [31:0] <- DEST [31:0] * SRC [31:0];
ELSE Temp1 [31:0] <- 0.0;
IF (imm8 [5] == 1) THEN Temp1 [63:32] <- DEST [63:32] * SRC [63:32];
ELSE Temp1 [63:32] <- 0.0;
IF (imm8 [6] == 1) THEN Temp1 [95:64] <- DEST [95:64] * SRC [95:64];
ELSE Temp1 [95:64] <- 0.0;
IF (imm8 [7] == 1) THEN Temp1 [127:96] <- DEST [127:96] * SRC [127:96];
ELSE Temp1 [127:96] <- 0.0;
Temp2 [31:0] <- Temp1 [31:0] + Temp1 [63:32];
Temp3 [31:0] <- Temp1 [95:64] + Temp1 [127:96];
Temp4 [31:0] <- Temp2 [31:0] + Temp3 [31:0];
IF (imm8 [0] == 1) THEN DEST [31:0] <- Temp4 [31:0];
ELSE DEST [31:0] <- 0.0;
IF (imm8 [1] == 1) THEN DEST [63:32] <- Temp4 [31:0];
ELSE DEST [63:32] <- 0.0;
IF (imm8 [2] == 1) THEN DEST [95:64] <- Temp4 [31:0];
ELSE DEST [95:64] <- 0.0;
IF (imm8 [3] == 1) THEN DEST [127:96] <- Temp4 [31:0];
ELSE DEST [127:96] <- 0.0p;
```

图 7A

```
IF (imm8 [4] == 1) THEN Temp1 [63:0] <- DEST[63:0] * SRC[63:0];
ELSE Temp1[63:0] <- 0.0;
IF (imm8[5] == 1) THEN Temp1[127:64] <- DEST[127:64] * SRC[127:64];
ELSE Temp1[127:64] <- 0.0;
Temp2[63:0] == 1) <- Temp1[63:0] + Temp1[127:64];
IF (imm8[0] == 1) THEN DEST[63:0] <- Temp2 [63:0];
ELSE DEST[63:0] <- 0.0;
IF (imm8[1] == 1) THEN DEST[127:64] <- Temp2[63:0];
ELSE DEST[127:64] <- 0.0;
```

图 7B