

[19] 中华人民共和国国家知识产权局

[51] Int. Cl⁷

G06F 9/00

G06F 9/30 G06F 12/04

G06F 13/40



[12] 发明专利申请公开说明书

[21] 申请号 03153030.3

[43] 公开日 2004 年 8 月 25 日

[11] 公开号 CN 1523491A

[22] 申请日 1999.3.4 [21] 申请号 03153030.3

分案原申请号 99806366.5

[30] 优先权

[32] 1998. 3. 18 [33] US [31] 09/044,087

[32] 1998. 3. 18 [33] US [31] 09/044,088

[32] 1998. 3. 18 [33] US [31] 09/044,089

[32] 1998. 3. 18 [33] US [31] 09/044,104

[32] 1998. 3. 18 [33] US [31] 09/044,086

[32] 1998. 3. 18 [33] US [31] 09/044,108

[71] 申请人 高通股份有限公司

地址 美国加州圣地埃哥

[72] 发明人 G·C·西赫 周群真

S·K·杰哈 I·坎 林 健

Q·莫蒂瓦拉 K·约翰 张 利

张海涛 李维斯 C·E·萨卡玛基

P·A·坎塔克

[74] 专利代理机构 上海专利商标事务所

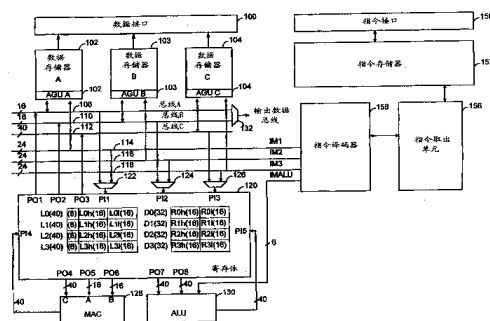
代理人 沈昭坤

权利要求书 4 页 说明书 38 页 附图 9 页

[54] 发明名称 数字信号处理器

[57] 摘要

数字信号处理电路要求使用可变长度指令组。示例的 DSP 包括一组可在其上与寄存器库(120)交换数据的三条数据总线(108、110、112)和三个数据存储器(102、103、104)。寄存器的寄存器可被至少两个处理单元(128、130)访问。使用的取指令单元(156)接收存储在指令存储器(152)里的可变长度指令,该指令存储器可用三个数据存储器(102、102、104)分开。



1. 一种用于控制数字信号处理器的系统，包括：
一存储器，其地址空间能以地址字寻址，所述地址字具有前沿字边界，
一组存储在所述存储器中的可变长度指令，所述可变长度指令各具有前沿指令边界，其中
所述前沿指令边界的第一部分对应于所述前沿字边界，而所述前沿指令边界的第二部分不同于所述前沿字边界。
2. 如权利要求 1 的系统，其特征在于所述可变长度指令包括数量可变的指令段。
3. 如权利要求 1 的系统，其特征在于所述可变长度指令包含表示各可变长度指令有多长的首部。
4. 如权利要求 1 的系统，其特征在于所述存储器位于所述数字信号处理器的指令存储器内。
5. 一种用于控制数字信号处理器的系统，包括：
一存储器，其地址空间能以地址字寻址，所述地址字具有后沿字边界，
一组存储在所述存储器中的可变长度指令，所述可变长度指令各具有后沿指令边界，其中，
所述后沿指令边界的第一部分相对于所述后沿字边界，而所述后沿指令边界的第二部分不同于所述后沿字边界。
6. 如权利要求 5 的系统，其特征在于所述可变长度指令包含数量可变的指令段。
7. 如权利要求 5 的系统，其特征在于所述可变长度指令包含表示各可变长度指令有多长的首部。
8. 如权利要求 5 的系统，其特征在于所述存储器位于所述数字信号处理器的指令存储器内。
9. 一种用于控制微处理器的系统，包括步骤：
存储指令数据的存储器，所述存储器具有地址空间；
多个请求操作的可变长度指令，所述多个指令的大部分置于所述地址空间基本上连续的位置。
10. 如权利要求 9 的系统，其特征在于 10 条以上指令置于所述存储器内

基本上连续的位置。

11. 如权利要求 9 的系统, 其特征在于 90% 以上的所述指令置于所述存储空间内基本上连续的位置。

12. 如权利要求 9 的系统, 其特征在于 25% 以上的所述指令置于所述存储空间内基本上连续的位置。

13. 如权利要求 9 的系统, 其特征在于所述可变长度指令根据可变长度指令中有多少指令数据而变化。

14. 一种控制集成电路的方法, 包括步骤:

- a) 产生一组对应于一组要执行的操作的指令段;
- b) 将能够同时执行的所述指令段编成指令段组;
- c) 对每个指令段组加一首部, 表明要得出可变长度全指令该可变长度指令有多长;
- d) 将所述可变长度全指令在地址空间中的连续地址写到所述集成电路, 其中所述地址空间具有字边界。

15. 如权利要求 14 所述的方法, 其中所述可变长度指令包括 16 位、32 位和 48 位指令。

16. 如权利要求 14 所述的方法, 其中每个首部位于所述可变长度全指令内最高位置上, 且包括五位。

17. 如权利要求 14 所述的方法, 其中所述指令段的子组可以包含一个、两个或三个指令段。

18. 一种用于控制数字信号处理器的方法, 包括步骤:

- a) 当并行执行数量减少的操作时, 产生短指令; 及
- b) 当并行执行数量较大的操作时, 产生长指令。

19. 如权利要求 18 所述的方法, 其中将所述短指令与长指令存储在存储器中具有地址字边界的连续位置内。

20. 如权利要求 18 所述的方法, 其中步骤(a)包括步骤:

识别无法与任何其它操作同时执行的操作;
用指令段和首部构成所述短指令, 首部表示该短指令有多长。

21. 如权利要求 18 所述的方法, 其中步骤(b)包括步骤:

识别一组可同时执行的操作;
用一组执行该组操作的指令段和表示该长指令有多长的首部构成所述长

指令。

22. 如权利要求 20 所述的方法，其中所述首部为 5 位。

23. 如权利要求 21 所述的方法，其中所述首部为 5 位。

24. 一种用于处理包含数量可变的指令段的可变长度指令的方法，包括：

(a) 读第一组指令数据，它大得足以包含具有最大可能长度的可变长度指令；

(b) 处理存储在所述第一组指令数据内的第一可变长度指令；

(c) 当未处理的指令数据的量小于所述最大可能长度时，装载附加指令数据。

25. 如权利要求 24 所述的方法，其中步骤(a)包括步骤：

读第一数据字；和

读第二数据字。

26. 如权利要求 25 所述的方法，其中步骤(c)包括步骤：

当未处理的指令数据的量在所述最大可能长度的一个指令字以内时，读一个指令字；及

当未处理的指令数据量比小于所述最大可能长度的一个指令字更多时，读两个指令字。

27. 一存储器，包括：

在存储器内以基本上连续的存储空间隔开的多个可变长度指令，所述多个指令中的每个指令具有：

表示该指令长度的首部；及

规定要执行的操作的主体。

28. 一种用于在数字信号处理器中处理指令数据的控制系统，包括：

具有第一与第二存储库的指令存储器，所述第一与第二存储库能以字寻址，所述指令存储器存储未处理的指令数据，所述指令数据具有最大指令长度；

从所述指令存储器读的指令寄存器；及

控制系统，当所述指令寄存器中未处理的指令数据的量在所述最大指令长度的一个字以内时，将一个字的指令数据从所述指令存储器装入所述指令寄存器，而当所述最大指令长度与所述指令寄存器中所述未处理的数据之间的不足不止一个字时，则将两个字的指令数据从所述指令存储器装入所述指令寄存器。

29. 如权利要求 28 的控制系统还包括:

以一组旋转配置之一输出存储在所述指令寄存器中的数据的旋转器, 其中所述控制系统还利用所述旋转器使来自所述指令寄存器的未处理的数据处于最高位置。

30. 如权利要求 28 的控制系统, 其中所述可变长度指令的长度小于或等于所述字。

31. 如权利要求 28 的控制系统, 其中所述可变长度指令具有前沿和后沿指令边界, 并将所述可变长度指令存储在所述指令存储器中的字边界之间, 指令存储器具有在存储器地址空间中相接的两个连续指令的前沿与后沿边界。

32. 如权利要求 28 的控制系统, 其中所述可变长度指令包含数量可变的指令段, 每个指令段执行一特定操作。

33. 一种对 32 位数 A 和 16 位数 B 作双倍精度乘法操作的系统, 所述 32 位数 A 具有低 16 位 A1 与高 16 位 Ah, 包括:

a) 在第一时钟周期内以 A1 乘 B, 得出第一中间值 I1;

b) 将所述中间值 I 右移 16 位, 得出移位的中间值 SI1;

c) 将 B 乘 Ah, 得出第二中间值 I2;

d) 将 I2 与所述移位的中间值 SI1 相加, 其中步骤 b)-d)在第二时钟周期内执行。

34. 一种数字信号处理器, 其中存储器可经由第二多个可选总线与第一多个处理装置连接, 如此选择所述第二多个可选总线, 从而优化存储器与处理装置之间可变长度数据的传递。

数字信号处理器

本申请是 1999 年 3 月 4 日提交的 99806366.6 (PCT/US99/04887) 专利申请的分案申请。

I. 发明背景

本发明涉及数字信号处理器。本发明特别适用于(但不专用于)应用高度平行、高度流水线处理技术的数字信号处理。

II. 相关技术的说明

数字信号处理器(DSP)通常用于实时处理数字信号。数字信号一般是一系列数字或数字值,用于代表相应的模拟信号。DSP 已广泛应用于各种场合,包括密致盘播放机等音频放蜂窝电话等无线通信系统。

DSP 往往被认为是微处理器的一种特殊形式。像微处理器一样,一般将 DSP 构制在硅基半导体集成电路上。另外,像微处理器那样,通过应用缩简指令组(RISC)运算技术,DSP 的运算能力增强了,RISC 运算技术包括使用较少数量的同尺寸指令来控制 DSP 的操作,每条指令的执行时间相同。RISC 运算技术的应用,提高了执行指令的速率或时钟速率,附大了 DSP 内指令流水线操作量,由此增强了 DSP 的整体运算能力。

用 RISC 运算技术编制 DSP 也产生了不希望有的特性。特别是,基于 RISC 的 DSP 要执行大量指令来完成给定的任务。执行附加的指令附大了 DSP 的功耗,尽管由于基于 RISC 的 DSP 的时钟速度提高而缩短了执行这些指令的时间。另外,应用数量更大的指令还增大了 DSP 内片上指令存储器的尺寸。存储器在 DSP 内要求相当大的电路面积(常常超过总面积的 50%),增大了 DSP 的尺寸与成本。因此,对于数字蜂窝网电话或其它类型电池操作的无线通信系统等低成本、小功率应用而言,采用基于 RISC 的 DSP 并非理想。

图 1 是用原有技术构制的 DSP 的简化框图。算术逻辑单元(ALU)16 耦合至 AIU 寄存器 17,乘积电路(MAC)36 耦合至 MAC 寄存器 27.数据总线 20 将 MAC 寄存器 27、ALU 寄存器 17 和(片上)数据存储器 10 耦接起来。指令总线 22 将 MAC

寄存器 27、(片上)指令存储器 12、MAC 寄存器 27 与 ALU 寄存器 17 耦接起来。指令译码器 18 耦合到 MAC26 与 ALU16,而在某些原有技术系统中,将译码器 18 直接耦合到指令存储器 12。数据存储器 10 还耦合至数据接口 11,指令存储器 12 还耦合至指令接口 13。数据接口 12 与指令接口 13 与片外存储器 6 交换数据与指令。

操作时,指令存储器 12 中的指令由译码器 18 译码。相应地,指令译码器 18 产生供给 ALU16 与 MAC26 的内部控制信号。该控制信号通常令 ALU16 获取在 ALU 寄存器 17 与数据存储器 10 或指令存储器 12 之间变换的数据,还令 MAC26 获取在 MAC 寄存器 27 与指令存储器 12 或数据存储器 10 之间交换的指令数据,还要令 ALU16 与 MAC26 根据分别存储在 ALU 寄存器 17 与 MAC 寄存器 27 中的数据作各种操作。

在示例性操作中,指令存储器 12 可以存有供 ALU16 与 MAC 使用的系数数据,而数据存储器 10 可以包含被处理的数据(信号数据)。在一般实践中,系数数据可用 DSP 构成频率滤波器。滤波时,将来自数据存储器 10 的信号数据和来自指令存储器 12 的系数数据读入 MAC 寄存器 27。指令存储器 12 内的附加指令数据也通过指令数据总线 11 或通过直接连接供给指令译码器 18,它规定了准备由 MAC27 执行的操作。通常把 MAC26 产生的结果读回到数据存储器 10。

这种原有处理技术导致许多处理上的低效性,例如包括总线对指令存储器 12 的访问争用,而总线必须将指令数据提供给 MAC 寄存器 16 和指令译码器 18,还包括总线对数据存储器 10 的访问争用,而总线又必须读出信号数据并写入输出数据。另外,在许多场合中,对输出数据的附加处理必须由 ALU16 来完成,更加增加了对数据存储器 10 的访问,造成对数据总线 20 的争用,因为输出数据必须从 MAC 寄存器 27 写入数据存储器 10,然后读出给 ALU 寄存器 17。这种读写操作都在总线 20 上进行,要耗用附加的总线周期。这类低效性降低了 DSP 的处理性能。

发明概述

本发明通过解决上述诸问题与低效时,并提供本说明描述的其它一些特性与改进,以期改善 DSP 的性能与有效时。

本发明为 DSP 提供一种新的改进的方法与电路。

根据本发明的一个方面提供的 DSP,存储器可通过第二组可选总线与第一

组处理装置连接，而把可选总线选成在存储器与处理装置之间传输的可变长度数据得以优化。

本发明各种其它方面均由所附的权利要求书限定。

本发明可用可变长度指令组实现。一部分可变长度指令与结束出现在存储器的字边界上。利用包含可变数量的指令同的指令可实现本发明附加的诸方面。每个指令段可执行特定的操作，以在每个时钟周期内作多次操作，从而减少了执行任务所需的时钟周期总数。

一示例性 DSP 包括一组三根数据总线，数据可在其上与寄存库和三只数据存储器作交换。应用两条以上尤其是三条数据总线，实现了本发明明显减少总线争用的另一方面。本发明的一个实施例要求数据总线包括一条宽总线和两条窄总线，前面耦合至宽数据存储器，后者耦合至两个窄数据存储器。

本发明一实施例的一个方面是应用于寄存库，其寄存器可至少由两个处理单元访问，从而可用多个处理单元对特定数据组作多次操作，无须对存储器读写数据。本发明该示例性实施例中的处理单元，包括一个 ALU 和一个 MAC 单元。当与多总线结构、高度平行指令组合应用时，就实现了本发明的另一方面，即执行高度流水线的多操作处理。

通过包含一个指令取出单元接收存储在指令存储器里的可变长度指令，可实现本发明的其它方面。运用与三只数据存储器组分开的指令存储器，可实现本发明的还有一个方面。指令译码器对来自指令存储器的指令译码并产生控制信号，让数据在各种寄存器、数据存储器与功能单元之间交换，允许在每个时钟周期内作多次操作。将本发明的各个方面协同组合起来，可产生意料不到的理想结果。例如，应用连续存在存储器内的可变长度指令，减少了 DSP 必需的电路面积，便于对 DSP 增设多条数据总线，并增设可被多个处理单元访问的寄存器，提高 DSP 的总体性能。组合本发明各个方面提供的其它优点是明显的，下面将作详述。

根据参照附图对本发明示例性实施例的详述，特别是所附权项所规定的本发明的上述特征与其它特征及其优点就更清楚了。

附图简述

根据下面结合附图对本发明诸示例性实施例的详述，本发明的特征、目的和优点将变得更加清楚，图中同样的字符表示相应的元件，其中：

图 1 是按原有技术构制的 DSP 的方框图；

图 2 是实施本发明的 DSP 的方框图；

图 3 是寄存库输入与寄存器之间连接的方框图；

图 4 是寄存库的寄存器与输出之间连接的方框图；

图 5 是存储在本发明一实施例的存储空间里一组可变长度指令的示图；

图 6 是表示指令取出单元操作的流程图；

图 7 是按本发明一实施例构制的指令取出单元的方框图；

图 8 是按本发明一实施例构制的 MAV 单元的方框图；

图 9 是本发明一实施例使用的指令体系的方框图。

较佳实施例的详细描述

本发明是新的改进的数据信号处理方法与电路。在整个说明书中，对信号、指令与数据采取了各种基准。这类信号、指令与数据较佳地用电压、电流表示，电流库包括带电、光学或磁性粒子，或它们的某种组合，其用法是众所周知的。用各种化学与生物化合物来表示这类信号、指令与数据，一般也符合本发明的使用，但是不作优先考虑，因为难以使用、管理和操纵这类项目。

另外，参照了本发明的各个方面、有利性、特征或优点(当不特定参照时，在此统称为方面)。本发明的有些实施例中，可以单独实现这些不同的方面而无须是提出本发明的其它方面。然而，在本发明的其它实施例中，可以将本发明的两个或多个方面一起实现，由此产生的协同和意料之外的优点，比只以本发明两个或多个组合方面中的一个方面实现的实施例更多。

I. DSP 操作原理与指令存储

图 2 是一部分 DSP 电路的方框图，该电路按本发明一示例性实施例构制。数据存储单元 102-104 经地址发生单元 (AGU) 105-107 分别耦合到数据总线 A、B 和 C，并耦合至数据接口 100。数据总线 A、B 和 C 经多路复用器 122-126 分别耦合至寄存库 120 的输出 P01、P02 与 P03，并耦合至寄存库 120 的输入 PI1、PI2 与 PI3。较佳地，数据总线 A、B、C 对数据存储单元 102-104 与寄存库 120 内寄存器之间的数据作读写。

使用三条数据总线与三只数据存储单元，可让更多数据在寄存库与数据存储单元之间交换而不产生总线争用。例如，使用三条数据总线 A、B 与 C，可同时从三只存储单元 102-104 里作三次取出操作。同样地，由于能任意组合三次取写操作，故能同时作三次写操作。

添加第四条数据总线可作更多次操作，这与本发明的某些实施例相一致。

然而，只使用三条数据总线特别有利，因为三第总线有利于 DSP 通常执行的滤波等许多任务。因此，添加第四条数据总线并不像添加第三条数据总线那样提供同样增加的性能提高，且要求同样的额外电路面积量。因此，添加第四条数据总线的效果比不上添加第三条总线。故在本发明的许多实施例中，较佳地只用三条数据总线。

输出口 P04、P05 和 P06 或寄存器 120 耦合至乘积(MAC)单元 128，后者的输出又耦合至寄存器 120 的输入口 PI4。寄存器 120 的输出口 PI4。寄存器 120 的输出口 P007 与 P08 耦合至 ALU130，而后者的输出耦合至寄存器 120 的输入口 PI5。

指令存储器 152 耦合至指令取出单元 156 与指令接口 150。指令译码器 158 耦合至指令取出单元 156，并耦合至即时总线 Im1、Im2 与 Im3 以及 ImALU。即时总线 Im1、Im2 与 Im3 耦合至多路复用器 122、124 与 126。即时总线 ImALU 耦合至 ALU130。除了上述的数据耦合外，译码器 158 还耦合至由控制接线(为便于给画而未示出)表示的各种子系统。

寄存器 120 含有 8 只标为 L0-L3 和 D0-D3 的寄存器。寄存器 L0-L3 是 40 位宽寄存器，还能经高字寄存器 L0h-L3h 与低与寄存器 L0I-L3I 以 16 位段访问。寄存器 D0-D3 是 32 位宽，能经子寄存器 R0-R7 以 16 位段访问。一般而言，寄存器与子寄存器均指简单的“寄存器”，寄存器的特定特征由提供的特定寄存器号指明。

通过拥有对其耦合并由多个输入与输出口访问的某些寄存器，可实现本发明的一个方面。在一实施例中，这种多连接性用耦合至每个寄存器的输入和每个输出口的多路复用器提供。其它多连接性方法是显然的，且与应用本发明某些方面相符，如包括使用数据总线与可导址存储器。然而，在有些实施例中使用多路复用器为佳，因为它们能迅速可控地访问各个寄存器与端口。

在本发明使用即时数据总线(这里提出的示例性实施例)的实施例中，实现了本发明的其它方面。例如，可将指令数据中包含的数据读入寄存器 120，无须与存储器 102-105 接口。这样，指令处理系统无须与数据存储器接口就能提供附加数据，进一步减少了总线争用。

图 3 是一框图，表示寄存器 120 内一组寄存器与一组输入口 PI1-PI5 之间的连接。寄存器定为 L0h-L3h、L0I-L3I、R0-R7。寄存器 L0 包括寄存器 L0h 与 L0I。在图 3 和 4 中，寄存器 L0h-L3h 为 24 位宽，L0h-L3h、L0I-L3I 和 R0-R7

为 16 位宽，使寄存器 L0-L3 为 40 位宽。同样地，输入口 PI3-PI5 包括 24 位的输入口 PI3h-PI5h 和 16 位的 PI3I-PI5I，总共 40 位。输入口 PI1 与 PI2 仅 16 位，用于写入寄存器 L0h-L3h 时，只写 24 位中的最低 16 位。

如图 3 所示，有些寄存器从所有输入口接收数据，而另一些寄存器只从有些或部分输入口接收数据。特别是，所有的寄存器 L0-L3 从多路复用器 500-514 接收所有输入口 PI1-PI5 的数据，16 位输入口能对寄存器 L0-L3 内的高和低寄存器进行写。这样，寄存器 L0-L3 从任何总线 A-C(对应于输入口 PI0-PI3)和从 MAC 单元 128 与 ALU130(对应于输入口 PI4 与 PI5)接收输入。寄存器 R0-R7 经多路复用器 516-530 从总线 A-C 接收输入数据。然而，它们都不从 MAC 单元 128(输入口 PI4)接收输入数据。另外，寄存器 R0-R3 经多路复用器 516/518/524 与 526 从 ALU 单元 130 接收输入数据。PI3h-PI5h 和 16 位的 PI3I-PI5I，总共 40 位。口 PI1 与 PI2 仅 16 位，用于写入寄存器 L0h-L3h 时，只写 24 位中的最低 16 位。

如图 3 所示，有些寄存器从所有输入口接收数据，而另一些寄存器只从有些或部分输入口接收数据。特别是，所有的寄存器 L0-L3 从多路复用器 500-514 接收所有输入口 PI1-PI5 的数据，16 位输入口能对寄存器 L0-L3 内的高和低寄存器进行写。这样，寄存器 L0-L3 从任何总线 A-C(对应于输入口 PI0-PI3)和从 MAC 单元 128 与 ALU130(对应于输入口 PI4-PI5)接收输入。寄存器 R0-R7 经多路复用器 516-530 从总线 A-C 接收输入数据。然而，它们都不从 MAC 单元 128(输入口 PI4)接收输入数据。另外，寄存器 R0-R3 经多路复用器 516、518、524 与 526 从 ALU 单元 130 接收输入数据。

图 3 实施例有诸多优点。特别是，它在输入口与寄存器之间提供了足够的连接性，有利于最常见的操作，但总的连接性保持为最小，以减少电路构成制所需的总电路面积。例如，MAC 单元 128 的输出只接至长寄存器 L0-L3，这是有利的，因为乘和累加操作的结果通常超出 32 位，因而将 MAC 单元 128 的输出接至 D0-D3 提供了最小的便利。在另一例中，ALU 单元 130 能对寄存器 L0-L3 和 R0-R3 输出，这增强了灵活性，因为数据可从 ALU 单元 130 写到各种寄存器，这很有用，因为 ALU 单元 13 执行了更多次操作，使它能有效地将数据输出到更多个寄存器。然而，ALU 单元 130 不接至所有寄存器，故避免了不必要和过多的连接。

图 3 是一框图，表示当按本发明一实施例执行时，将寄存器 120 的输出口

接至寄存器的情况。如图所示，输出至总线 A 的输出口 P01 经多路复用器 540 接至寄存器 L0h-L3h、L01-L31 与 R0-R7，当被作为小子寄存器访问时，它包括所有有效的寄存器。同样地，输出至总线 13 的输出口 P02 经多路复用器 542 接至寄存器 L0h-L3H、L01-L31 与 R0-R7。输出至 40 位宽总线 C 的输出口 P03 由多路复用器 530 接至寄存器 L0-L3 与 D0-D3，当被作为全寄存器访问时，它包括全部有效的寄存器。

接至 MAC 单元 128 的 40 位输入的输出口 P04，经多路复用器 532 接至寄存器 L0-L3。鉴于乘和累加操作的特征，MAC128 累加的值很大，因而将输出口 P04 只接至 40 位“长”寄存器 L0-L3 可提供最佳耦合配置，因为附加接至寄存器 D0-D3 的利用率很小，给出的各种乘法操作的累加一般超过 32 位。

接至 MAC 单元 128 的一个 16 位输入的输出口 P05，经多路复用器 534 接至寄存器 L0h-L3h、R0、R4 与 R6。接至 MAC 单元时的第二 16 位输入口的输出口 P06，接至寄存器 L0h-L3h、L01-L31 与 R0-R7。将 MAC 单元 128 的一个 16 位输入接至全部有效寄存器，并将第二个 17 位输入口接至有效寄存器小组，可实现有用的折衷。特别在寄存器空间普得有限时，至少可将待处理的一段数据放在任一有效寄存器里。然而，限制接至其它输入端的寄存器数量，减少了连接电路的总量，有利于提供其它功能与特点，如在其它寄存器、输入口与输出口之间有更高的连接性。

接至 ALU130 的输入端的输出口 P07，经多路复用器 546 接至寄存器 L0-L3、L0h-L3h 与 R0-R3。其中 Z0h-L03h 与 R0-R3 同一组逻辑零一起输出。即，寄存器 L0h=L03h 与 R0-R3 输出至 P07 的位 31-16(位号为 0-39)，位 0-15 置逻辑零，位 39-32 用位 31 扩展符号。接至 ALU130 的另一输入端的输出口 P08，也经多路复用器 548 连同一组逻辑零接至寄存器 L0-L3 和 R0-R7。以此方式连接 ALU130 的输入端，可对全部有效的长寄存器 L0-L3 作逻辑操作，因而可对大量寄存器作逻辑操作，这对归一化和定标等多类信号处理操作是有用的。另外，可在寄存器 R0-R7 与 L0h-L3h 以及 R0-R7 之间作算术运算，以可使用的寄存器组来衡量，灵活性很大，同时还限制了必需的连接数量，因而限制了要求的电路面积。应理解，有效的算术逻辑操作并一限于上述情况。

应用能被多数据总线与多处理单元访问的寄存器有各种好处。例如，寄存器在数据总线与处理单元间提供了接口，减少了每条数据总线通往每个处理单元的布线要求，从而节省了电路面积，降低了芯片成本。

再者，将最少量(一组)寄存器接至多个处理单元，可用多个处理单元对同一数据多次操作，一必定要通过数据总线向存储器读写数据，这样节省了总线周期，减少总线争用。采用指令处理流水线操作法也是有利的，因为第一处理单元在第一指令周期中处理的数据，然后可用第二处理单元在第二处理周期中在同一个寄存器内再作处理。

然而，并非所有数据都要求用多处理单元处理的，所以其它寄存器(另一组)只被一个处理单元或少于处理单元总数(有两个以上处理单元的场所)的处理单元访问。使用这另一组寄存器就减少了连接数，因而减少了电路面积，可在寄存器连接性与电路面积之间(因而在性能与效率之间)实现最佳平衡。

还有，在本发明一实施例中，利用寄存器 128 里的二相时钟寄存器可进一步增强流水线操作能力。在同一个全时钟(“处理”)周期内，二相时钟寄存器在时钟第一相上读，在第二相上写。这样在特定处理周期内，可在第一时钟相内读出已被 MAC128 等第一处理单元处理过的数据，并在该处理周期的剩余部分内由 ALU130 等第二处理单元作进一步处理。

另外，在处理周期第二相内，可刚由 MAC128 处理的新数据写入同一寄存器，从而在一个处理周期内，可在两个处理单元之间作整个流水线处理。而且，这些操作一必使数据通过任一条内部总线，可进一步避免总线争用。

再参照图 2，操作期间，指令取出单元 156 从指令存储器 152 中检索二进制指令，若查一出，则从外部在存储器中检索。外部存储器有多种形式，如动态与静态随机存取存储器(DRAM 与 SRAM)及其衍生产品，磁或光硬盘存储器，或某些其它众所周知的数据存储媒体。在本发明该示例性实施例中，指令为可变长度指令，指令取出单元(下称取指令单元)确定指令长度，并确定在每次处理或时钟周期内要限出多少附加指令。另外，指令存储在内外存储器内连续的存储位置。下面详述取指令单元 156 的操作和内外存储器内指令数据的存储情况。

指令译码器 158 接收由取指令单元 156 检索的指令，并将指令译为控制信号供给一个或多个组成 DSP 的子系统，子系统包括数据存储器、寄存器、MAC 与 ALU。另外，指令译码器 158 可以将接收指令内所含的即时数据经即时总线 Im1、Im2、Im3 或 ImALU 送往有关系统。即时数据通常是存储在指令数据内的数值，而指令数据用于对存储在数据存储器 102-106II 中的数据作操作，或可以限定或修正某个地址。

由图 2 中 DSP 执行的操作包括将数据从数据存储器经数据总线之一载入某一寄存器位置。数据也可从寄存器写入数据存储器。而且, MAC128 或 ALU130 可对寄存器 120 内一个或多个寄存器里的数据作操作, 一般将结果写回到寄存器 120 内的寄存器。

上述 DSP 结构有多种优点。例如使用三条数据总线有利于不间断地对数据作流水线处理。在 DSP 执行的示例性滤波期间, 将待滤波的数据(信号数据)存储在一只数据存储器中, 而将应用于该数据存储在另一数据存储器中。由于操作结果通常要求比操作数更多的位数, 因此将信号数据与系数存储在两只较窄的存储器里较佳。然后将系数与信号数据读入寄存器 120, 再由 MAC 单元 128 相乘累加。这些操作的结果可以存储在寄存器 120 内的第二寄存器里, 或者写入先前已存储了输入数据的寄存器。接着在第三总线(总线 C)上将任何结果从该寄存器写入更宽的数据存储器(存储器 C)。

由于输出数据经第三总线写入第三存储器, 而输入数据级经第一与第二数据总线从第一与第二数据存储器中读出, 所以很少或一会出现存储器访问冲突或总线争用。这样, 数据可以不中断地处理, 减小了以高于内部总线或其它子系统的速率对任何存储器子系统或数据总线的计时要求, 从而既保持或提高了处理速度, 又降低了功耗。

另外, 在若干不同数据值在 DSP 内不同级以任何给定时间作不同处理の場合, 数据不中断地通过 DSP 有利于数据的流水线处理。而且, 在组合应用的并行指令时, 结合这种高效的流水线处理, 可实现明显的处理灵活性, 提供高度通用、有效的有力的 DSP 系统。

应当理解, 应用多条总线提高了以各种减少总线冲突的附加方式使数据围绕 DSP 运行的能力。例如, 要作除法的数据可作为输入数据从存储器 C 经数据总线 C 提供, 除数由另一存储器与总线提供, 如存储器 A 与数据总线 A 提供, 然后利用剩余总线(数据总线 B)将结果存入剩余的存储器(存储器 B)里。

在得益于多条数据总线与存储器的另一示例性操作中, 由 MAC 单元 130 累加的数据经第一存储器与第一总线(如存储器 A 与数据总线 A)提供。在完成一组累加后, 把所得的数据经数据总线 C 写到存储器 C。同时, 在数据总线 C 不携带来自 MAC 单元 128 的结果数据的处理周期内, 大部分时间这类结果数据仅在完成一组累加操作后可得到, 准备逻辑移位的数据从存储器 C 经数据总线 C 提供给 ALU 单元 130。逻辑移位的数据经数据总线 B 同时写到存储器 B。这样,

应用多条数据总线与存储器，一般特别是结合多个处理单元，通过在 DSP 内移动数据提供更多的可能性而有利于多次操作。

如上所述，应用能被多个处理单元(如 MAC 单元 128 与 ALU 单元 130)访问的寄存器，可实现本发明的另一方法。由多个处理单元访问的寄存器，让必须由这些处理单元处理的数据不必移动通过任何内部数据总线而予以访问。例如，数据可由第一处理单元写到某个寄存器，然后再由同样可以访问该寄存器的第二处理单元处理。这样进一步减轻了总线争角与阻塞，因而保持了高的数据吞吐量。

另外，当结合使用并行操作指令和下面要详述的并行处理的能力时，可进一步增强高度流水线、多重操作地处理数据的能力。相反地，典型的流水线操作要涉及交错排列一组操作的不同阶段(即取出、译码、处理)，所以缩短了各指令开始之间的处理时间。多重操作流水线法所附加好处在于，数据通过一组不同的操作，这种操作对不同的数据组是同时执行的。这种多重操作流水线法使每个处理周期执行的指令数提高到超过了普通的指令流水线法。

下述示例性处理进一步示出了上述结构的最佳协同作用。在一示例性滤波操作中(也如上所述)，随着树其作累加的乘积数据的增大，MAC26 产生的结果在大小上也增大(用于代表该值的绝对值与位数两者)。最后，结果必须作标定或“归一化”，通常要求用 ALU 单元 130 作逻辑位移操作。

在描述的系统，标定操作可同滤波的乘法与累加操作同时执行。在同时处理的处理周期内，将未处理的信号数据与滤波系数从数据存储器 102 与 103 读入寄存器 120 内的寄存器(如 L0h 与 L01)。同时，MAC 单元 128 读取先前存储在这些寄存器(L0h 与 L01)里的值并作乘法与累加操作，其中将输出写到第二寄存器(如 L1)。同时，ALU 单元 130 读取先前存储在第二寄存器(L1)中的数据，并执行标定操作，将标定值写到第三寄存器(如 L2)。而且在同一处理周期内，用总线 C112 将存储在第三寄存器(D0)中的先前的值写入数据存储器 104。显然，特定的操作可以按执行的特定任务改变。而且很清楚，使用允许作多重操作的高度并行的指令，进一步有利于高度流水线的多指令操作。高度度行的指令允许在每个周期内对不同操作的规程作流水性操作。

应用两相读写操作，如上述例子，这种处理可在单个处理周期内全部完成，其中在第一时钟相内以每个寄存器里读出数据，并用处理单元处理，而在第二时钟相内的老的数据上将结果写入寄存器。应当理解，在单个时钟周期内，同

一值不经受该处理的所有步骤，而是一组值通过 DSP 作流水线操作，每个值随着处理的执行移至下一步骤。

这里描述的本发明的各个方面都有利于许多其它的操作。例如，声音编码就是话音数据的编码处理，它要求执行许多不同类型的操作，有些操作可独立执行，因而可同时执行。应用多余数据总线和多个处理单元，有助于这些操作。

应用独立的指令存储器与指令译码具有进一步的优点。例如，与上述数据处理的同时，取指令 156 从指令存储器 152 读出指令，接着令指令译码器 158 产生控制信号，以便控制 DSP 内各其它子系统的操作(为便于画图和清楚，未画出接线)。同样，数据总线也不携带指令数据，因而可以不中断指令数据而移动处理信号数据。这样，通过将指令处理与数据处理分开，消除指令数据移动所耗用的数据总线周期，进一步提高了性能。

图 5 是一张表，说明将一组可变长度指令装在根据本发明一实施例的图 2 指令存储器 152 的一部分可寻址存储空间内。在本发明有些实施例中，也可像图 2 那样把可变长度指令存储在外存系统里以实现附加的存储器效率。举例地址示于左列，指向 32 位数据安排每个地址示于存储器 275 的中间和右边诸列。中间列代表各数据字的 16 位高次子字，最右列代表 6 位低次子字。在本发明的该较佳实施例中，高低次子字都不能单独寻址，以便减少必需的地址逻辑量。

在存储器 275 内，可变长度指令 A-L 以图示的装填结构存储。指令 A 是 48 位指令，前两上双字节 A(1)与 A(2)存储在地址字 0x0000，第三双字节 A(3)存储在地址 0x0001 的高次子字里。指令 A 后面的指令 B 是 32 位指令，第一双字节 B(1)存储在地址 0x0001 的低次字里，第二双字节 B(2)存储在地址 0x0002 的高次子字里。指令 C 是 16 位指令，其第一和唯一的双字节(C1)存储在地址 0x0002 的低次子字里。

从指令 A-C 的存储位置可见，本发明将部分不同的指令存储在同一地址字内，减少了存储器 275 存储一组指令所必需的尺寸或量。例如，将指令 A 的第三双字节 A(3)与指令 B 的第一双字节 B(1)一起存储。

将变第指令跨过字边界或特别在存储器地址空间内的连续位置中存储起来，本发明减少了指令存储器存储给定指令数据必需的容量。减少指令存储器的容量就减少了对 DSP 给以规定指令贮藏容量所必需的芯模的尺寸与成本。如图 3 所示，将变第指令 D-L 置于存储器 275 内，进一步示明了指令的装填状况。

应该理解，在本发明有些实施例中，不必像上述那样把所有指令都装在连

续位置中。例如，本发明不同的实施例仅将指令的主要部分(90%或以上)装在存储空间的连续位置中。在本发明另一实施例中，只将指令的主要部分(如25-50%)有利地装在连续的存储空间中。其它实施例可使用其它百分比的装填指令。

而且，不一定要使用连续的位置。指令必须简单地置于比指令数据总量大得不多的总存储空间内。这可以将指令置于存储空间内相邻的位置而较佳地实现，然而，指令会在整个存储空间里搞混，但是只要指令能按原定的执行次序读出就成了。本领域的技术人员将明白，这类预定的混乱作为存储空间的再映射，而且这种再映射一般不影响本发明的操作，否则要加大不希望有的复杂性。

同样地，最好具有能应用于很大一组指令的装填方案。例如，在本发明某些实施例中，最好对至少 10 条指令应用该装填方案。

另外，用于本发明该示例性实施例的特定装填方案，在其它实施例中不一定使用。例如，本发明的其它一些实施例可能在连续存储位置中没有指令，而指令可用某种小容量的存储空间分开，其中包括使用指令分离码。较佳地，小存储空间小于存储空间在存储字边界上保持指令边界所必需的容量。然而，上述装填法由于其简单、完整和高效，在许多场合中是较佳的方法。总之，在本发明的不同实施例中，在装填完整性与装填方案复杂性之间的折衷选择可以是不同的。

而且如上面暗示的那样，本发明的有些实施例只对部分而不是整组有效指令使用该装填方案。例如，只对用于执行某特定任务或子程序的几组指令执行指令装填。

应该指出，将高度装填的指令存储与可变长度指令组合起来使用，进一步减少了对 DSP 的存储器要求，因为可变长度指令只耗用请求所需操作所需的数据量，而且高度装填的指令存储将总存储量保持等于该组变称指令的耗用时，因而是最小量，减小的存储器尺寸也减少了芯片尺寸与成本。

除了减小 DSP 尺寸并由此减少成本等好处以外，在结合上述结构的其它特点时，应用高度装填指令与可变长度指令还具有意料不到的优点。例如，减小指令存储器的尺寸，附加的电路面积可供 DSP，还能具备将可变长度指令存储在存储空间连续位置内的能力。图 6 是取指令单元 156 从本发明一实施例的指令存储器 152 取指令时的操作的流程图。过程在步骤 200 开始，在步骤 202，从指令存储器 152 中读出第一组指令。在本发明的该示例性实施例中，在步骤

202 检索指令数据的两个 32 位字，即 64 位。

在步骤 204，指令译码器 158 处理包含在这 64 位指令数据中的第一指令。在本发明的该示例性实施例中，该指令可以是 16、32 或 48 位长。指令长度由包含在每条指令里指明指令长度的一组首部位确定，下面再作详述。显然还有各种其它规定指令长度的方法，包括用代码区分两条指令，或用超首部指令规定几组后接指令的长度。在某些场合中最好使用首部位，因为指令长度信息通常十分接近该指令，由此降低了存储或保持有关指令处理的状态信息的要求。

在处理了包含在 64 位检索指令数据内的第一指令后，在步骤 206，确定 48 位或更多未处理的指令数据是否保持在 64 位检索指令数据里。若保持着，则在步骤 204 处理包含在剩余 48 位未处理数据中的下一个指令。

在步骤 206，若不到 48 位未处理指令数据保持在检索指令数据中，就从指令存储器 152 装载附加指令数据。各种装载附加指令的方法都可试用。在本发明一实施例中，从指令存储器装载足够多的附加指令数据，使存储在取指令单元里的未处理数据量增至 48 位。确保 48 位未处理数据存储于取指令单元内，以确保对指令译码器 158 至少有一条完整的指令。

在本发明一较佳实施例中，当保持的未处理数据不足 48 位时，根据已经处理的特定数据量，取指令单元检索可变的数据量。特别在处理的数据量等于或超过一个数据字(32 位)时，就检索新指令数据的附加数据字(32 位)。若先前处理的数据量等于或超过两个数据字(64 位)，则取指令单元就检索两个新的数据字。

根据处理的数据字数检索数据量较佳，因为它对指令译码器 158 保持着足够量的未处理数据，同时允许对构成指令存储器的存储库作更有效的字长访问。一旦在步骤 206 检索了附加的未处理指令数据，就在现有的未处理指令数据总量内处理下一个指令。

图 7 是按本发明一实施例构制的取指令单元 156 与指令存储器 152 的框图。指令存储器 152 包括偶数个存储库 302(RAM0)和奇数个存储库 300(RAM1)，每一个都可读写 32 位数据字。把存储库标为奇偶，因为二者都在同一地址空间内寻址，但是偶地址偶数存储库 302，奇地址针对奇数存储体 300。

可读写其它字尺寸(包括 8、16、24、48、64 位字)的存储库可应用于本发明的其它实施例。可以使用其它不同数量的存储库，包括 108 个存储库。然而，最好使用两个带 32 位字的存储库，因为能降低整体复杂性，而且允许指令数

据以写管理的块来寻址。

控制逻辑 304 让数据字从存储库 300 与 302 读到指令寄存器 106 与 107。特定的存储位置读由地址线 310 与 314 规定，而读指令由使能线 332、315、316 和 318 控制。指令寄存器 306 与 307 的位输出以 16 位部分加到旋转器 308 的输入端 A、B、C 和 D，旋转器 308 则输出 48 位指令数据 324，而数据 324 包括四选三(3: 4)输入端 A、B、C 和 D，每个输入组包含 16 位，下面再作语详述。

操作期间，控制逻辑 304 按参照图 5 描述的方法从指令存储库 300 与 302 装载指令数据。特别是，控制逻辑 304 通过从偶数存储库 302 和奇数存储库 300 读 32 位数据字，第一次将总计 64 位的未处理指令数据装入指令寄存器 306 和 307。如果处理了 16 位的指令，不用装载新数据，因为寄存器 306 和 307 还包含着 48 位未处理的指令数据。若接着处理了 32 位的指令，则指令寄存器 306 要装上 32 位字的附加指令数据，因为保持的未处理指令数据不足 48 位。再次装载 32 位指令字，将 48 位未处理指令数据置入寄存器 306 和 307，16 个未处理位置入寄存器 307，其余 32 位置入寄存器 306。如果再处理 48 位指令，就没有未处理指令数据可保持，所以两寄存器 306 和 307 均装有指令数据的 32 位字，该数据是 64 位的未处理指令数据，要大于必需的 48 位指令数据。虽然装载 64 位指令数据并无十分必要，但是这样做是有用的，因为这样可以使用两个 32 位字的指令存储器与寄存器，在行业中是常见的。使用其它方法保持足量的未处理指令数据，与应用本发明的某些方面相符。

一旦将 64 位新指令数据装入指令寄存器 306 和 307，控制逻辑 304 就再配置旋转器 308，用控制信号 320 根据下列条件输出在输入端 A、B、C 和 D 上接收的下一个 48 位的指令数据：1) 指令数据在指令地址空间内的位置，2) 已处理的该组指令数据，及 3) 已处理的前一指令的长度。特别是将旋转器 308 配置成输出一组准备用最高有效或最左边位置中指令数据的下一在线位作下一在线处理的 48 位指令指数。

例如，一旦将前两个字即 64 位指令数据装入寄存器 306 和 307，如果偶数指令寄存器 307 中的指令数据处于下一在线，则旋转器 308 的输出就包括在输入端 A、B 和 C(ABC)上依次接收的指令数据。如果奇数指令寄存器 306 中的指令数据处于下一在线，则旋转器 308 配置成输出在输入端 C、D 和 A(CDA)上依次的指令数据。

由于指令已处理，就像上述那样将新的指令数据装入数据寄存器 306 和

307, 为旋转器 308 根据先前处理的指令的尺寸, 配置成在输出端 324 上不断输出下一在线指令数据。先前处理的指令的尺寸由首部数据 322(输出端 324 上前五位的副本)传送给控制逻辑 304。上述已指出, 任何对控制逻辑 304 规定指令长度的预定方法均符合本发明的实践, 尽管使用前五位是较佳的, 因为这样可直接按指令数据确定指令长度。

在本发明一示例性实施例中, 前一指令的尺寸按表 1 编码成两位状态 I1 和 I0。

I1 I0	指令的尺寸
0 0	分支转移/停止/复位
0 1	16 位
1 0	32 位
1 1	48 位

表 1. 全指令模式

另外, 旋转器 308 的配置由两个构成控制 320 的选择位 S1 和 S0 控制, 其编码如表 2 所列。

S1 S0	旋转器输出
00	A B C
01	B C D
10	C D A
11	D A B

表 2. 旋转器选择控制位与输出

很明显, 随着 A1 与 S0 状态的递增, 旋转器 308 的输出就左转或筒状偏移。左转就是每个输入组(A、B、C 和 D)在输出端上向左移。在输出最左边位置的输入组被取消。先前未在输出端认定的输入组以后在最右边位置上输出。

S1 与 S0 状态和旋转器 308 的配置被修正或转动的量, 根据各种长度的指令而变化。具体而言, 将代表正被处理指令长度的值(I1, I0)加到控制位 S1 和 S0, 去除任何执行值。即

$$S1(t+1), S0(t+1) = S1(t), S0(t) + I0, I1 \quad (1)$$

对于转移或复位状态, S1 和 S0 的值根据要处理转移或复位的特定指令而复位, 所以不用公式(1)。在本领域中, 处理转移、复位与停止指令的各种方

法是众所周知的，而且由于这种处理与本发明无多大关系，下不再作描述。

在一示例性处理中，旋转器 308 开始时为 ABC 输出，选择位 S1 和 S0 为 00。如果接收 6 位指令，将 I1 和 I0 为 01 的相应指令长度位加到 S1 和 S0，得到为 01 的 S1 和 S0，对应于旋转器 308 为 BCD 的输出 324。在处理了前 1 位指令数据(输入 A)后，BCD 输出就是下一在线的指令数据组。

如果下一指令是个 32 位指令，则把指令长度 I1 和 I0(10)加给当前的 S1 和 S0 状态 01，得到 11，得出的输出是 DAB，对应于还未处理的下一个 48 位指令数据，用位于最高有效或最左边位置的输入端 D 上接收的下一在线指令数据配置。先前来自输入端 B 和 C 的指令数据已了处理。应该指出，在先前的 32 位指令处理期间，按上述的指令数据装载处理法将新数据装入指令寄存器 307。

如果接下来处理 48 位指令，就对选择位 S1 和 S0 的状态增加指令长度 I1 和 I0(11)，得到 S1 和 S0 为 10，去除执行，将旋转器 308 配置成输出 CDA。输出 CDA 对应于下一个待处理的 48 位指令数据，其下一在线位配置在了高有效位置上。先前在输入端 D、A 和 B 上的指令数据已作过处理。再次指出，在前一指令处理期间，已将新的指令数据读入寄存器 306。作为举例，已提供了上述执行相关指令数据处理的逻辑，而使用其它逻辑来执行上述的指令数据处理是显然的，与本发明的应用相符合。

这样，在该示例性实施例中，控制逻辑将旋转器 308 配置成输出从指令寄存器 306 和 307 接收的下一个 48 位指令数据，而把要处理的下一个指令位置于最左边位置。本领域的技术人员将明白，以任何预定的配置将指令提交给指令处理单元 158，从而可识别下一组要处理的指令数据，这同样符合本发明的使用。

如上述指出的，除了输出下一个 48 位未处理的指令数据以外，旋转器 308 还向控制逻辑 304 指明下一个要处理的指令的尺寸。具体而言，旋转器 308 向控制逻辑 304 输出附加的下一个要处理的 5 位指令数据的副本。在本发明该较佳实施例中，由该指令的前五位规定该指令的长度。

图 8 是按本发明一实施例配置的 MAC 单元 128 的框图。右移 900 接收要累加的 40 位输入，并将该值移动 0 位或 16 位，输出加到多路复用器 901 的一个输入端。多路复用器 901 的另一输入端接收值 0×8000 。乘法器 902 接收两个要相乘的 16 位值，并一起从指令译码器 158 接收符号位，每个输入总计 17 位。

乘法器 902 的输出由左移 904 接收，左移 904 按指令译码器 158 的规定，

将输出移动 0、1、2 或 3 位。加法器 906 接收多路复用器 901 和左移 904 的输出，并按指令译码器 158 的命令对两个输入值作加或减，

并将结果输出给本发明该示例性实施例中的寄存器输入口 PI4。

在本发明有些实施例中，在 MAC 单元 128 里应用右移 16 单元 900 比其它类型的 MAC 单元具有附加的应用性。更具体地遇，应用右移 16 单元 900 便于在减少时钟周期数时作双倍精度操作。例如，为了执行将 32 位数(A)乘以 16 位数(B)的双倍精度操作，在第一时钟周期内，先将 32 位数的低 16 位(A1)乘以 16 位数 B，得出一个存储在寄存器 120 里有中间值 I。

在第二时钟周期内，将中间值 I 输入右移 16 单元 900 右移 16 位。另外，将 16 位数 B 与 32 位数 A 的高 16 位(Ah)相乘，将结果与右移 16 单元 900 里的各移的中间值 I 相加。这样，双倍精度乘法是用两个时钟周期而不是三个时钟周期执行的。一般而言，许多双倍精度操作要求一个或多个变量相对于其它变量而移位，这样可像乘法或累加操作之一一样在同一个时钟周期内作移位，减少了双倍精度操作必需的周期数。

通常要求三个时钟周期，因为第一次乘法操作、移位操作和第二次乘法操作通常各要求一个时钟周期。因此，使用移位电路减少了双倍精度乘法所需的时钟周期数。利用右移单元 900 还有利于涉及不同大小操作数据的其它双倍精度操作。

II. 指令配置

A. 概述

图 9 是一框图，表示本发明该示例性实施例使用的指令体系。方框 402 表示可变长度全指令，包括控制 DSP 操作的 16、32 或 48 位。可变长度指令又由包括一般指令段的指令段和存储器移动与程序流(MMOF)指令组成，如方框 403 所示。本发明该示例性实施例中使用的指令段包括 MAC8、MAC16、ALU8、ALU16、DMOV16、DMOV24 与 DL40 指令段。MMPF 指令段包括 OneMem11、TwoMem19、TwoMov19 和 ThreeMem24 指令段。MMPF 指令段由方框 406 所示的 MMPF 指令子段组成。MMPF 指令子段包括 LD(A)、LD(B)、ST(A)、ST(B)、LS(C)、DMOVA、DMOVB 和 PF8。下面详细描述各种全指令、指令段和指令子段。

B. 全指令

在本发明该较佳实施例中，用长度为 16、32 和 48 位的指令段而形成。全

指令被制成可在指令存储器 152 内连续存储并由 DSP 处理。下面描述全指令格式与结构，接着说明指令段的格式与结构。在操作期间，DSP 每个时钟周期处理一条全指令。这样，在处理每条全指令期间，可作多次操作，特定的操作由选择的特定组的指令段决定。

用于本发明该示例性实施例的三条全指令的格式如表 3 所示。

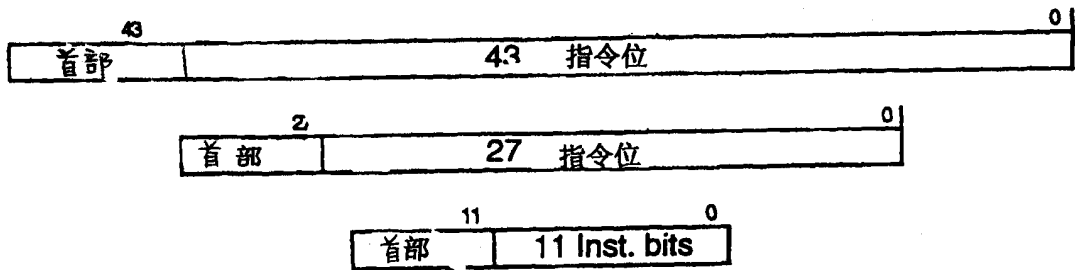


表 3. 全指令格式

用于每条全指令的五位首部，指明该一指令的长度和某些有关全指令内容的附加信息。表 4 提供了用于本发明该示例性实施例的首部的格式。

5-bit 首部					指令长度	
0	0	0	0	X	16-bit 指令	(2 types)
0	0	0	1	X	32-bit 指令	(2 types)
0	0	1	X	X	48-bit 指令	(4 types)
0	1	X	X	X	32-bit 指令	(8 types)
1	X	X	X	X	48-bit 指令	(16 types)

表 4. 全指令首部格式

每条全指令(16、32 和 48 位长)包含一个或多个指令段。表 5 列出本发明该示例性实施例中的现有指令段。在讨论了全指令后，再详述指令段的格式与操作。

指令段	描述	宽度
MAC8*	8-bit MAC 工作	8
ALU8*	8-bit ALU 工作	8
OneMem11*	One Memory 工作	11
MAC16*	16-bit MAC 工作	16
ALU16*	16-bit ALU 工作	16
DMOV16*	Conditional Reg Move / Inport / Outport	16
TwoMem19	两个存储工作	19
TwoMov19*	两个存储/数据移动工作	19
DMOV24	装载/直接存储/装载地址/跳跃	24
ThreeMem27	三个存储工作	27
DL40	双装载	40

表 5. 指令段

表 6-8 列出指令段的各种组合，可应用于本发明该示例性实施例中的 48、

31 和 16 位全指令里。虽然指令段的其它组合符合本发明的使用和操作，但是这里揭示的组合的某些特点较佳，如下面要详述的那样。另外，全部或部分全指令被示为“保留”，在描述的未规定或使用特定的指令组合，但是可以设想出将来使用这些全指令组合。

表 6 列出 16 位全指令在按本发明该示例性实施例执行时的格式化。全指令由五位首部后接 11 个指令位组成。

11					0		
部首					11 Inst. Bits /		
0	0	0	0	0	MAC8		0 0 0
0	0	0	0	0	ALU8		0 0 1
0	0	0	0	0	保留		0 1 0
0	0	0	0	0	保留		0 1 1
0	0	0	0	0	保留		1 0 0
0	0	0	0	0	保留		1 0 1
0	0	0	0	0	保留		1 1 0
0	0	0	0	0	保留		1 1 1
0	0	0	0	1	OneMem11		

表 6. 16 位全指令格式

首部位指明该指令的长度及有关指令类型的一些信息。对于首部 00000 而言，最低有效三个尾位用来进一步规定执行的操作。具体而言，尾位 000 指明剩余的 8 位包含一个 MAC8 指令段。尾位 001 指明剩余的 8 位包含一个 ALU8 指令段。对于其它尾位组合，不规定指令。

对于首部 00001，剩余 11 位包含一个 OneMem11 指令段。通过提供允许 ALU、MAC 或存储器移动待执行操作的 16 位全指令，可用最短的全指令作最常见的操作。由于最短的指令要求存储器存储的量最少，所以应用所述的 16 位全指令减少了执行特定一组操作所需的指令存储器容量。这样，DSP 的整体尺寸减小了。也因此减少了成本和功耗。

16 位指令通常在只能作一次或少数几次操作的条件下使用。一般而言，可以缩小只限定一次操作所需的指令的尺寸，于是对一次操作就使用半字即 16 位指令。另外，16 位指令可用于 MAC、ALD/存储器移动或程序流程操作，这类操作包罗了几乎所有人们想执行的操作。

在按本发明一实施例配置时，表 7 列出了指令段组合和相关的 32 位全指令格式化。

27					19										11										0									
部首					27 指令位																													
0 0 0 1 0					ThreeMem27																													
0 0 0 1 1					ALU8										TwoMem19																			
0 1 0 0 0					ALU8										TwoMov19																			
0 1 0 0 1					MAC8										TwoMem19																			
0 1 0 1 0					MAC8										TwoMov19																			
0 1 0 1 1					MAC8										ALU8										OneMem11									
0 1 1 0 0					MAC16																				OneMem11									
0 1 1 0 1					ALU16																				OneMem11									
0 1 1 1 0					DMOV16																				OneMem11									
0 1 1 1 1					DMOV24																									0 0 0				
0 1 1 1 1					ALU16															MAC8										1 0 0				
0 1 1 1 1					MAC16															ALU8										0 1 0				
0 1 1 1 1					保留																									1 1 0				
0 1 1 1 1					保留																									1				

表 7. 32 位全指令格式

如上所述，五个首部位指出了全指令长度及特定的指令段组合。例如，首部 00010 表示剩余 27 个指令位含一个 ThreeMem27 指令段，而首部 00011 表示剩余 27 个指令位含一 ALU8 指令段后接一 TwoMwm19 指令段。

对于首部 01111，最低尾位还指出指令段的组合。例如，对于最低尾位 0，下两个最低位表明剩余 24 位饮食 4、ALU16 后接 MAC8 还是 MAC16 后接 ALU8 指令段。其它尾位状态，如最低尾位 1，限定着保留的组合。

32 位指令可同时执行许多最常见的操作，既有利于流水线操作，也缩小了指令尺寸。例如，像滤波这样的应用，通常要作两次限出操作和一次乘/累加操作。32 位指令能以流水线方式作这种成组操作，且不要求填满 48 位指令空间。

此外，32 位指令可同时作 MAC 与 ALU 操作以及程序转移与调用操作，也不用最大的指令尺寸。

当按本发明一实施例执行时，表 8 列出了 48 位全指令的格式和指令段组合。

43					27					19					11					0														
部首					43 指令位																													
0	0	1	0	0	DMOV24										MAC8					OneMem11														
0	0	1	0	1											DMOV24										ALU8					OneMem11				
0	0	1	1	0																					DMOV24									
0	0	1	1	1											DMOV24																			
1	0	0	0	0	MAC16																				ThreeMem27 ThreeMem27 ThreeMem27									
1	0	0	0	1	MAC8					ALU8																								
1	0	0	1	0	ALU16										MAC8					TwoMem19														
1	0	0	1	1											ALU16										MAC8					TwoMov19				
1	0	1	0	0																					ALU16									
1	0	1	0	1											ALU16																			
1	0	1	1	0	DMOV16																				MAC16					OneMem11				
1	0	1	1	1																					DMOV16									
1	1	0	0	0	DMOV16																													
1	1	0	0	1											MAC8					TwoMem19														
1	1	0	1	0											MAC8					TwoMov19														
1	1	0	1	1											ALU8					TwoMem19														
1	1	1	0	0	DMOV16										ALU8					TwoMov19														
1	1	1	0	1											MAC16										ALU8					TwoMem19				
1	1	1	1	0																					MAC16									
1	1	1	1	1											MAC16																			
1	1	1	1	0	MAC8																				DMOV24					0 0 1				
1	1	1	1	1																					ALU16									
1	1	1	1	1	DMOV16																													
1	1	1	1	1											MAC16					ALU8					1 0 0									
1	1	1	1	1											DMOV16					ALU16					MAC8					1 0 1				
1	1	1	1	1											DL40																			
1	1	1	1	1	保留																													

表 8.48 位全指令格式

五个首部位规定了该指令的长度及具体的指令段组合。例如,首部位 00100 表明,43 个剩余指令位包括 DMOV24、MAC8 和 OneMem11 指令段。首部位 10011 表明,43 个剩余位包括 ALU16、MAC8 和 TwoMwm19 指令段。

对于首部位 11111，三个最低尾位还指出了饮包含于剩余指令位置的指令段。例如，尾位 000 表示剩余 40 个指令位包含 MAC16 和 DMOV24 指令段。尾位 001 表示剩余 40 个指令位包含 MAC8、ALU8 和 DMOV24 指令段。尾位 110 表示剩余 400 个指令位包含 DL40 指令段。

48 位人指令提供的指令段组合可同时作重操作，故比串行操作快得多。例如，若干 48 位全指令允许同时作 ALU 操作、AMC 操作和存储器操作。存储器操作包括装载、存储和数据移动操作，且常常允许一次访问多个存储位置。

48位指令允许乘法操作结合ALU操作和取数据与程序流程操作一起以流水

线方式进行。这对滤波操作在结合标定操作时是有用的，因为此时通常要作 MAC 操作，接着作 ALU(如移位)操作。应用 MAC 与 ALU 操作的其它应用场合包括组合三种或多种数据流。48 位指令，尤其在组合应用三总线结构方面，有利于流水线操作。

这样可有效地将能以单条 48 位全指令执行的操作种数增加为五种 (MAC、ALU、FETCH1、FETCH2 和 STORE)。通常，利用 DSP 的多条内部总线在 DSP 内耦接各种处理系统，进一步增强了在 DSP 中同时执行多条指令的能力。利用不同的总线，可以同时移动和访问不同的数据组。

根据能执行的操作种数来改变指令长度，进一步提高了使用指令存储器的效率。任何具体的任务都具有可同时作多重操作的周期，而其它周期只能作少数或只有一次操作。按能同时进行的操作种数调整指令长度，可减少指令存储器容量。

当结合使用紧密指令装填法时，可进一步减少上述示例性方法所需的指令存储器。使用可变长度指令或紧密装填指令，或使用这两者，通过用更多的电路面积实施特性，有助于应用多总线结构和多重访问寄存库。这样，将本发明这些方面的组合协同地组合起来，可同时提高性能和效率。

C. 指令段

如上所述，全指令包括一组以预定方式集群的一个或多个指令段。在本发明该示例性实施例中，这组现有的指令段如表 5 所列。利用本发明该示例性实施例所提供的全指令而得出的指令段和组合，设计成让该组操作所需的指令存储器容量。下面讨论本发明该示例性实施例所应用的各种指令段的格式与操作状态。

C1. 指令段术语

在下对指令段与子段的整个讨论中，表 9 和 10 列出了用来参照诸寄存器的缩写。另外，表的左边示出了本发明该示例性实施例使用的具体位码。

Dreg		R0-R7	Lh/Ll	A0-A7	Lreg/Dreg	AS		AL	
0000	R0	000	R0	L0h	A0	L0	0	AS0	AL0
0001	R1	001	R1	L1h	A1	L1	1	AS1	AL1
0010	R2	010	R2	L2h	A2	L2			
0011	R3	011	R3	L3h	A3	L3			
0100	R4	100	R4	L0l	A4	D0			
0101	R5	101	R5	L1l	A5	D1			
0110	R6	110	R6	L2l	A6	D2			
0111	R7	111	R7	L3l	A7	D3			
1000	L0h								
1001	L1h								
1010	L2h								
1011	L3h								
1100	L0l								
1101	L1l								
1110	L2l								
1111	L3l								

	R0-R3	L0-L3	D0-D3	C0-C3	Cmod
0 0	R0	L0	D0	C0	++
0 1	R1	L1	D1	C1	--
1 0	R2	L2	D2	C2	++CM0
1 1	R3	L3	D3	C3	++CM1

cond										
00000	LT		01000	L0LT		10000	L1LT		11000	L2LT
00001	LE		01001	L0LE		10001	L1LE		11001	L2LE
00010	EQ		01010	L0EQ		10010	L1EQ		11010	L2EQ
00011	NE		01011	L0NE		10011	L1NE		11011	L2NE
00100	GE		01100	L0GE		10100	L1GE		11100	L2GE
00101	GT		01101	L0GT		10101	L1GT		11101	L2GT
00110	OV		01110	L0OV		10110	L1OV		11110	L2OV
00111	Uncond		01111	Rsvd		10111	Rsvd		11111	Rsvd

表 9. 指令段术语与代码

	RegA	regB
00000	R0	R0
00001	R1	R1
00010	R2	R2
00011	R3	R3
00100	R4	R4
00101	R5	R5
00110	R6	R6
00111	R7	R7
01000	L0h	L0h
01001	L1h	L1h
01010	L2h	L2h
01011	L3h	L3h
01100	L0l	L0l
01101	L1l	L1l
01110	L2l	L2l
01111	L3l	L3l
10000	A0	B0
10001	A1	B1
10010	A2	B2
10011	A3	B3
10100	A4	B4
10101	A5	B5
10110	A6	B6
10111	A7	B7
11000	AS0	BS0
11001	AS1	BS1
11010	AL0	BL0
11011	AL1	BL1
11100	AM0	BM0
11101	AM1	BM1
11110	保留	保留
11111	保留	保留

	regC
0000	L0
0001	L1
0010	L2
0011	L3
0100	D0
0101	D1
0110	D2
0111	D3
1000	C0
1001	C1
1010	C2
1011	C3
1100	CM0
1101	CM1
1110	保留
1111	保留

表 10. 指令段术语和代码

C2. 指令段说明

指令段组包括两类 MAC 指令段：MAC8 与 MAC16。MAC8 指令段支持带符号-不带符号和带符号-带符号的乘法类型，结果存在累加器 L0 或 L1 中。MAC8 指令段允许用 16 位全指令作 MAC 操作而节省了指令 RAM，而且对于许多并行指令组合，要求 MAC 操作编码为 32 位指令而不是 48 位指令。一般而言，MAC8 指令执行的处理符合下述公式：

$$\begin{Bmatrix} L0 \\ L1 \end{Bmatrix} = \begin{bmatrix} L0 \pm \\ L1 \pm \end{bmatrix} \begin{Bmatrix} R0 \\ R2 \\ R4 \\ R6 \end{Bmatrix} * \begin{Bmatrix} R0 \\ R1 \\ R3 \\ R5 \end{Bmatrix} \begin{bmatrix} (SU) \\ (SS) \end{bmatrix} \quad (2)$$

如公式(2)所示，MAC8 指令段让寄存器 L0 或 L1 的内容与寄存器 R0、R2、R4 与 R6 的乘积相加，并将 R0、R1、R3 和 R5 直接设置到寄存器的乘积里。另

外，还可规定带符号或不带符号的乘法。通过限制可用 MAC8 指令作 MAC 操作的寄存器数量，能将指令长度保持为 8 位，允许用更短的 8 位指令段作 MAC 操作。

MAC8 指令执行的特定操作由构成表 11 所列指令的 8 位值限定。

7	6	5	4	3	2	1	0
MAC 工作				mac8Op1		mac8Op2	

表 11. MAC8 指令段格式

SU/SS 规定带符号或不带符号的乘法。表 12 列出的代码应用于规定 MAC8 指令段内的各种操作。

MAC 工作		mac8Op1		mac8Op2	
000	L0=	0 0	R0	0 0	R0
001	L1=	0 1	R2	0 1	R1
010	L0=L0+	1 0	R4	1 0	R3
011	L1=L1+	1 1	R6	1 1	R5
100	L0=L0-	SU/SS			
101	L1=L1-				
110	L0=L1+				
111	L0=L1-				
		0	SU		
		1	SS		

表 12. MAC8 指令段代码

这样，MAC8 指令 0x99 将寄存器 L0 的内容与寄存器 R0 和 R3 不带符号的乘积之和放入寄存器 L0。

NAC16 指令段通过让附加寄存器应用于乘法-累加而提供的灵活性。公式 (3) 建立了能用 MAC16 指令段执行的操作。

$$\begin{Bmatrix} L0 \\ L1 \\ L2 \\ L3 \end{Bmatrix} = \begin{Bmatrix} L0 >> 16 \\ L1 >> 16 \\ L2 >> 16 \\ L3 >> 16 \end{Bmatrix} \pm macOp1 * macOp2 \quad [(mtype)] \quad [<< mshift] \quad [:CPS]; \quad (3)$$

例如，虽然在乘法-累加指令中不是所有组合都可离开累加器，但是每个累加器 (L0-L3) 都可用作目的地。CPS 区段发出信号表示协同处理器应该执行并行的特定操作。该特定操作由 MAC16 指令执行，由构成表 13 中指令的 16 位值规定。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MAC 工作				macOp1			macOp2				mtype		mshift		CP S

表 13. MCA16 指令段格式

表 14 列出的代码用于规定 MAC16 指令段内的各种操作。

	MAC 工作	macOp2	macOp1	mtype
0000	L0=	R0	0 0 0 R0	0 0 (SU)
0001	L1=	R1	0 0 1 R2	0 1 (UU)
0010	L0=L0 [>>16] +	R2	0 1 0 R4	1 0 (SS)
0011	L1=L1 [>>16] +	R3	0 1 1 R6	1 1 See note
0100	L0=L0 [>>16] -	R4	1 0 0 L0h	
0101	L1=L1 [>>16] -	R5	1 0 1 L1h	
0110	L0=L1 [>>16] +	R6	1 1 0 L2h	
0111	L0=L1 [>>16] -	R7	1 1 1 L3h	
1000	L2=	L0h		
1001	L3=	L1h		
1010	L2=L2 [>>16] +	L2h		
1011	L3=L3 [>>16] +	L3h		
1100	L2=L2 [>>16] -	L0l		
1101	L3=L3 [>>16] -	L1l		
1110	L2=L3 [>>16] +	L2l		
1111	L2=L3 [>>16] -	L3l		

mshift	
0 0	<< 0
0 1	<< 1
1 0	<< 2
1 1	<< 3

CPS	
0	OFF
1	ON

表 14. MAC16 指令段代码

注：将指令 $L0=R0 \times R0 (SU) \ll 0$ 译成 NOP。

Mtype11 用作直接相乘的 RND，对于

带符号-带符号乘法/累加指令，将

累加器右移 16。

带 mtypeSU 与 macOp1=macOp2 的 MAC 指令均性非法。

MAC16 指令段可左移达到 3，在单单相乘(无累加)时能作舍入运算，舍入在移位后出现。在累加时，要加的累加器与带符号-带符号相乘同时下移 16。CPS 位是协同处理器选通位，表示应将 MAC 操作中使用的数据送给协同处理器。

应当指出，MAC8 指令段执行的操作子小组采用 MAC16 执行。对 MAC8 指令段选择的特定组指令，是可用 MAC16 指令段执行的该组操作以外最经常执行的指令组。通过用 MAC8 指令段执行大多数 MAC 操作，就节省了程序存储器。

8 位 ALU8 指令段包括的 ALU 操作，是最经常与 MAC 操作(MAC8 与 MAC16)并行执行的操作，它不含中间数据。所有的 ALU8 移位操作都是算术移位，利用一只内部移位寄存器(SR)节省指令编码位。表 15 列出用 ALU8 指令段执行的操作。

$NOP;$	NOP(需要平行组合)
$LD = DETNORM(LS);$	决定块归一化因数
$LD = SET(LS);$	拷贝累加器(不饱和)
$LD = LS \ll SR;$	移位累加器
$LD = RND(LS \ll SR);$	移位和巡回累加器
$LD = LD \pm (LS \ll SR);$	累加移位累加器
$LD = LS \pm LT;$	加减累加器
$LS \pm LT;$	加/减 ACCS 结果自由(设置标志)

表 15. ALU8 指令段操作

LS 是装载源(L0-L3)，L0 是装载目的地(L0-L3)。ALU8 指令段执行的特定操作，由组成表 16 中指令段的 8 位值规定。

7	6	5	4	3	2	1	0
0	ALUOp			LS		LD	
0	1	1	Sign	LS		LT	
1	LD			Sign		LT	

表 16. ALU 指令段格式

表 17 列出的特定代码用于限定利用 ALU8 指令段执行的操作。

ALUOp		LD / LS / LT	
0 0 0	$LD=DETNORM(LS)$	0 0	L0
0 0 1	$LD=SET(LS)$	0 1	L1
0 1 0	$LD=LS \ll SR$	1 0	L2
0 1 1	$LD=RND(LS \ll SR)$	1 1	L3
1 0 0	$LD=LD + (LS \ll SR)$	Sign	
1 0 1	$LD=LD - (LS \ll SR)$		
1 1 0	$LS+LT$		
1 1 1	$LS-LT$		
		0	[LD=] [LS+LT]
		1	[LD=] LS-LT

表 17. ALU 指令段代码

注：全零指令 $L0=DETNORM(L0)$ 译码为 NOP。

其中 $LD=DETNORM(LD)$ 的指令为非法。

ALU8 清除累加器指令为 $LD=LD=LD$ 。

ALU16 指令段可作算术与逻辑两种移位，表 18 列出了 ALU16 指令执行的具体操作。

$\ll$ 表示算术移位， $\lll$ 表示逻辑移位。

(a) IF cond NOP;	有条件 NOP(对于平行组合)
(b) IF cond LD = SET(LS);	有条件拷贝累加器
(c) IF cond LD = LS $\pm$ LT;	有条件加/减 accs
(d) LD = NORM(LS, SR);	归一化累加器
(e) LD = ABS(LS);	累加器绝对值
(f) LD = -LS;	否定累加器
(g) LD = $\sim$ LS;	交错累加器(1's 补码)
(h) LD = BIT(immediate5);	产生位屏蔽(LD=0 $\times$ 1 $\ll$ imm5) 1 $\ll$ imm5).
(i) LD = $\sim$ BIT(immediate5);	产生反位屏蔽(LD= $\sim$ (0 $\times$ 1 $\ll$ imm5)) LD = (0 $\times$ 1 $\ll$ imm5)).
(j) LD = LS $\left\{ \begin{array}{l} \& \\ \\ \wedge \end{array} \right\} \left\{ \begin{array}{l} LT \\ \text{BIT(immediate5)} \\ \sim\text{BIT(immediate5)} \end{array} \right\};$	逐位 AND, OR, XOR
(k) LD = [RND] $\left(\text{LS} \left\{ \begin{array}{l} \ll \\ \ll\ll \end{array} \right\} \left\{ \begin{array}{l} R0 - R3 \\ \text{immediate6} \end{array} \right\} \right);$	移位[巡回]acc
(l) LD = LS $\pm$ (LT $\left\{ \begin{array}{l} \ll \\ \ll\ll \end{array} \right\}$ immediate6);	移位和加法 accs
(m) $\left\{ \begin{array}{l} L0h - L3h \\ R0 - R3 \end{array} \right\} = \left\{ \begin{array}{l} L0h - L3h \\ R0 - R3 \end{array} \right\} \pm \{R0 - R7\};$	加法 regs
(n) $\left\{ \begin{array}{l} L0h - L3h \\ R0 - R3 \end{array} \right\} = \left\{ \begin{array}{l} L0h - L3h \\ R0 - R3 \end{array} \right\} + \text{immediate6}$	直接加
(o) $\left\{ \begin{array}{l} L0h - L3h \\ R0 - R3 \end{array} \right\} = \text{SET(immediate6)};$	直接负载
(p) $\left\{ \begin{array}{l} L0h - L3h \\ R0 - R3 \end{array} \right\} = \left\{ \begin{array}{l} L0h - L3h \\ R0 - R3 \end{array} \right\} \left\{ \begin{array}{l} \& \\ \\ \wedge \end{array} \right\} \{R0 - R7\}$	16-bit 逻辑
(q) $\{R0 - R3\} = SR \pm \{R0 - R7\};$	将记录加到 SR
(r) $\{R0 - R3\} = SR + \text{immediate6};$	直接加至 SR
(s) $SR = \{R0 - R3\} + \text{immediate6};$	使 SR 负载直接的总数
(t) $SR = \text{SET(immediate6)};$	直接使 SR 负载

表 18. ALU16 指令段操作

指令 L0=SET(L0) 译码为 NOP。

表 19 列出 ALU16 指令段的格式。

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(l)	0	0	+/-	AL	LT		LS		LD		immediate6					
(c)	0	1	0	0	LT		LS		LD		Cop1	cond				
(k)	0	1	0	1	Sop ₂	AL	LS		LD		immediate6					
(j)	0	1	1	BitOp		Inv	LS		LD		immediate6					
(p)	0	1	1	1	1	BitOp		dregh(dst)			dregh(src)		R0-R7 (src)			
(o)	0	1	1	1	1	1	1				immediate6					
(n)	1	0	0	0	dregh(src)			dregh(dst)			immediate6					
(m)	1	0	0	1	0	0	+/-	dregh (dst)			dregh(src)		R0-R7 (src)			
	1	0	0	1	0	1	保留(所有零)									
	1	0	0	1	1	保留(所有零)										
	1	0	1	保留(所有零)												
(b)	1	1	0	0	0	0	LS		LD		Cop2	cond				
(h-i)	1	1	0	0	0	1	0	Inv	LD		immediate6					
(r)	1	1	0	0	0	1	1	0	R0-R3 (d)		immediate6					
(s)	1	1	0	0	0	1	1	1	R0-R3 (s)		immediate6					
(k)	1	1	0	0	1	0	0	0	R0-R3 (s)		Sop1	AL	LS		LD	
(j)	1	1	0	0	1	0	0	1	BitOp		LT		LS		LD	
(d-g)	1	1	0	0	1	0	0	1	1	1	AccOp		LS		LD	
(t)	1	1	0	0	1	0	1	0	0	0	immediate6					
(q)	1	1	0	0	1	0	1	0	0	1	+/-	R0-R3 (d)		R0-R7 (src)		
	1	1	0	0	1	0	1	0	1	保留(所有零)						
	1	1	0	0	1	0	1	1	保留(所有零)							
	1	1	0	0	1	1	保留(所有零)									
	1	1	0	1	保留(所有零)											
	1	1	1	保留(所有零)												

表 19. 16 位 ALU 指令段格式与代码

ALU16 指令段执行的具体操作，由组成表 20 的指令段的位值规定。

DregH		Sop1		Cop1	
000	R0	0	LD=LS<<R0-R3	0	LD=LS+LT
001	R1	1	LD=RND(LS<<R0-R3)	1	LD=LS-LT
010	R2	Sop2		Cop2	
011	R3	0	LD=LS<<imm6	0	LD=SET(LS)
100	L0h	1	LD=RND(LS<<imm6)	1	保留
101	L1h	AL		+/-	
110	L2h	0	算术移位	0	+
111	L3h	1	逻辑移位	1	-
BitOp		AccOp		Inv	
00	AND	00	LD=NORM(LS,SR)	0	正常位屏蔽
01	OR	01	LD=ABS(LS)	1	反相位屏蔽
10	XOR	10	LD=-LS		
		11	LD=~LS		

表 20. ALU16 指令段代码

注：全零指令 L0=SET(L0)；译码为 NOP。

对 BIT 指令(h, i, j)，通过附加带即时 5 的零符号位，汇编程序对即时 6 编码(可简化译码)。

DMOV16 指令段是一种 16 位指令段，可执行不同的数据移动、数据入口和数据出口操作，如表 21 所示。

(a)	<i>NOP</i> ;	平行指令的 NOP
(b)	<i>LC = immediate9</i> ;	负载回路计数器
(c)	$\begin{cases} AM0 - AM1 \\ BM0 - BM1 \\ CM0 - CM1 \end{cases} = immediate10$;	负载寄存器
(d)	$\begin{cases} AL0 - AL1 \\ BL0 - BL1 \end{cases} = immediate11$;	负载圆长度寄存器
(e)	$\begin{cases} L0 - L3 \\ D0 - D3 \\ L0h - L3h \end{cases} = INPORT(port_addr)$;	入口工作
(f)	$OUTPORT(port_addr) = \begin{cases} R0 - R7 \\ L0h - L3h \\ L0l - L3l \end{cases}$	出口工作
(g)	<i>OUTPORTA(port_addr)</i> ;	总线 A 上的出口值
(h)	<i>OUTPORTB(port_addr)</i> ;	总线 B 上的出口值

表 21. DMOV16 指令段操作

表 22 列出的格式与代码，可用 DMOV16 指令段执行现有的操作。

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	0	保留(所有零)												
(d)	0	0	1	0	AL	immediate11 (im1)										
(d)	0	0	1	1	BL	immediate11 (im2)										
(c)	0	1	0	0	0	AM	immediate10 (im1)									
(c)	0	1	0	0	1	BM	immediate10 (im2)									
(c)	0	1	0	1	0	CM	immediate10 (im3)									
	0	1	0	1	1	保留(所有零)										
(b)	0	1	1	0	0	0	0	immediate9 (im1)								
	0	1	1	0	0	0	1	保留(所有零)								
	0	1	1001-1111				保留(所有零)									
(e)	1	0	0	0	0	0	L0-L3		入口地址 (PI3)							
(e)	1	0	0	0	0	1	D0-D3		入口地址 (PI3)							
(e)	1	0	0	0	1	0	L0h-L3h		入口地址 (PI3)							
	1	0	0011-1111				保留(所有零)									
(f)	1	1	0	0	dreg				出口地址 (A总线, PO1)							
(f)	1	1	0	1	dreg				出口地址 (B总线, PO2)							
(g)	1	1	1	0	0	0	0	0	出口地址 (读A总线)							
(h)	1	1	1	0	0	0	0	1	出口地址 (读B总线)							
	1	1	10001-11111					保留(所有零)								

表 22. DMOV16 指令段格式

注：指令 L0=0 译码为 NOP。

即时 10 带符号，即时 9 与 11 不带符号。

指令 OUTPORTA(port-addr)读 A 总线上的值，并将它输出到指定端口。通过从存储器 A 同时读一值，该指令可将值直接从存储器 A 送到该端口。OUTPORTB(port-addr)的工作与此相同。

DMOV24 指令段是个 24 位指令段，执行不同的装载/存储寄存器直接或装载寄存器即时操作，如表 23 所示。

(a)	$\{regA\} = memA(address14);$	
(b)	$memA(address14) = \{regA\};$	L/S 直接存储器 A
(c)	$\{regB\} = memB(address14);$	
(d)	$memB(address14) = \{regB\};$	L/S 直接存储器 B
(e)	$\{regC\} = memC(address14);$	
(f)	$memC(address14) = \{regC\};$	L/S 直接存储器 C
(g)	$\begin{Bmatrix} A0-A7 \\ B0-B7 \\ C0-C3 \end{Bmatrix} = address14;$	立即装载地址寄存器
(h)	$\begin{Bmatrix} AS0-AS1 \\ BS0-BS1 \end{Bmatrix} = address14;$	立即装载环形起动寄存器
(i)	$\begin{Bmatrix} R0-R7 \\ L0h-L3h \\ L0l-L3l \end{Bmatrix} = immediate16;$	立即装载数据寄存器
(j)	LOOP UNTIL address17;	直至结束地址回路
(k)	CALL address17;	功能呼叫
(l)	$[IF\ cond] \begin{Bmatrix} JUMP \\ JUMPD \end{Bmatrix} address17;$	[条件][延迟]跳跃

表 23. DMOV24 指令段操作

起动寄存器位于 AGU 单元里。

表 24 列出的格式和某些代码，可用本发明该示例性实施例中的 DMOV24 指令段执行各种现有的操作。

	23	22	21	20	19	18	17	16	15	14	13-0
(l)	0	0	cond					地址 17 (JUMP)			
(l)	0	1	cond					地址 17 (JUMPD)			
(j)	1	0	0	0	0	0	0	地址 17 (LOOP)			
(k)	1	0	0	0	0	0	1	地址 17 (CALL)			
(g)	1	0	0	0	0	1	0	A0-A7	地址 4 (im1)		
(g)	1	0	0	0	0	1	1	B0-B7	地址 4 (im2)		
(e)	1	0	0	0	1	0		regC (dst)	地址 14 (im3, Cbus+PI3)		
(f)	1	0	0	0	1	1		regC (src)	地址 14 (im3, Cbus+PO3)		
(a)	1	0	0	1	0			regA (dst)	地址 14 (im1, Abus+PI1)		
(b)	1	0	0	1	1			regA (src)	地址 14 (im1, Abus+PO1)		
(c)	1	0	1	0	0			regB (dst)	地址 14 (im2, Bbus+PI2)		
(d)	1	0	1	0	1			regB (src)	地址 14 (im2, Bbus+PO2)		
(i)	1	0	1	1				dreg	immediate16 (im1+PI1)		
(i)	1	1	0	0				dreg	immediate16 (im2+PI2)		
(g)	1	1	0	1	0	0	0	C0-C3	地址 14 (im3)		
(h)	1	1	0	1	0	0	0	1	0	AS	地址 14 (im1)
(h)	1	1	0	1	0	0	0	1	1	BS	地址 14 (im2)
	1	1	0		1001-1111			保留(所有零)			
	1	1	1					保留(所有零)			

表 24. DMOV24 指令段格式与代码

注：地址 14 与 17 不带符号，即时 16 带符号。

应该指出，对于 DMOV24 和其它指令段，有些操作可编码两次。例如，行 (i) 与 (j) 里规定的格式对同一操作编码，一个格式规定使用即时总线 Im1，另一格式规定使用即时总线 Im2。编码两次可让该指令段与众多其它也能要求使用即时总 1 或 2 的指令段组合起来。

40 位双载指令段(DL40)是一种 40 位的指令段，可作即时装载地址操作。表 25 列出了本发明该示例性实施例中执行的具体操作。

(a)	$\{A0 - A3\} = address14, \{B0 - B3\} = address14;$
(b)	$\{A0 - A3\} = address14, \{C0 - C3\} = address14;$
(c)	$\{B0 - B3\} = address14, \{C0 - C3\} = address14;$
(d)	$\begin{matrix} \{A0 - A3\} \\ \{B0 - B3\} \\ \{C0 - C3\} \end{matrix} = address14, \begin{matrix} \{R0 - R7\} \\ \{L0h - L3h\} \\ \{L0l - L3l\} \end{matrix} = immediate16;$
(e)	$\begin{matrix} \{L0 - L3\} \\ \{D0 - D3\} \end{matrix} = immediate32;$
(f)	$\{A0 - A3\} = address14, regB = memB(address14);$
(g)	$\{B0 - B3\} = address14, regA = memA(address14);$
(h)	$\{A0 - A3\} = address14, regC = memC(address14);$
(i)	$\{C0 - C3\} = address14, regA = memA(address14);$
(j)	$\{B0 - B3\} = address14, regC = memC(address14);$
(k)	$\{C0 - C3\} = address14, regB = memB(address14);$
(l)	$regA = memA(address14), regB = memB(address14);$
(m)	$regA = memA(address14), regC = memC(address14);$
(n)	$regB = memB(address14), regC = memC(address14);$

表 25. DL40 指令段格式

表 26 列出了 DL40 指令段对每次操作的格式。

	3	2	1	0	3	2	1	0	3	2	1	0	29-16	15-14	13-0
	0	0	0	0	0	Reserved									
(e)	0	0	0	0	1	Lreg/Dreg		immediate32							
(d)	0	0	0	1	A0-A3		dreg		address14				immediate16		
(d)	0	0	1	0	B0-B3		dreg		address14				immediate16		
(d)	0	0	1	1	C0-C3		dreg		address14				immediate16		
(b)	0	1	0	0	0	0	0	0	C0-C3		Caddress14		A0-A3	Aaddress14	
(c)	0	1	0	0	0	0	0	1	C0-C3		Caddress14		B0-B3	Baddress14	
(a)	0	1	0	0	0	0	1	0	A0-A3		Aaddress14		B0-B3	Baddress14	
	0	1	0	0	0	0	1	1	Reserved (all zeros)						
	0	1	0	0	0	1	Reserved (all zeros)								
(h)	0	1	0	0	1	0	regC		Caddress14				A0-A3	Aaddress14	
(j)	0	1	0	0	1	1	regC		Caddress14				B0-B3	Baddress14	
(g)	0	1	0	1	0	regA		Aaddress14				B0-B3	Baddress14		
(i)	0	1	0	1	1	regA		Aaddress14				C0-C3	Caddress14		
(f)	0	1	1	0	0	regB		Baddress14				A0-A3	Aaddress14		
(k)	0	1	1	0	1	regB		Baddress14				C0-C3	Caddress14		
	0	1	1	1	Reserved										
(m)	1	0	0	regC		re...		Caddress14				..gA	Aaddress14		
(n)	1	0	1	regC		re...		Caddress14				..gB	Baddress14		
(l)	1	1	regA		re...		Aaddress14				..gB	Baddress14			

表 26. DL40 指令段格式与代码

注：地址 14 不带符号，即时 16 与 32 带符号。

同样如表 5 所示，在本发明该示例性实施例中提供了四类存储器移动和程序流指令段，它们列于表 27。

表 27. 存储器移动和程序流指令段

OneMem 11
TwoMen 19
TwoMev 19
ThreeMem 27

每个存储器移动和程序流指令 (MMPF) 段由表 28 列出的一组 MMPF 子段组成。

表 28. 组合数据移动和程序流指令子段

	指令子段	
LD(A)	装存储器 A 间接	8
ST(A)	装存储器 A 间接	8
LD(B)	装存储器 B 间接	8
ST(B)	装存储器 B 间接	8
LS(C) *	装/存储存储器 C 间接	8
DMOVA *	总线 A 寄存器移动	8
DMOVB *	总线 B 寄存器移动	8
PF8	8 位程序流	8

首先讨论 MMPF 指令段的格式与操作，然后详细讨论 MMPF 子段的格式与操作。

OneMem11MMPF 指令段用来执行单个存储器装载与存储操作、数据移动操作和程序流操作。在这里提供的示例性实施例中，用 OneMem11MMPF 指令段执行八种不同的操作，由 11 位段中前三位表示的特定操作如表 29 所示，该表列出了可用 OneMem11 数据移动指令段执行的操作。

10	9	8	7	6	5	4	3	2	1	0
0	0	0	LD(A)							
0	0	1	ST(A)							
0	1	0	LD(B)							
0	1	1	ST(B)							
1	0	0	LS(C)							
1	0	1	DMOVA							
1	1	0	DMOVB							
1	1	1	PF8							

表 29. OneMem11 指令段格式

TwoMwm19MMPF 指令段是一种 19 位指令段, 可执行表 30 列出的存储器装载与存储操作的八种不同的组合。

18	17	16	15-8	7-0
0	0	0	LD(A)	LD(B)
0	0	1	LD(A)	ST(B)
0	1	0	LD(A)	LS(C)
0	1	1	ST(A)	LD(B)
1	0	0	ST(A)	ST(B)
1	0	1	ST(A)	LS(C)
1	1	0	LS(C)	LD(B)
1	1	1	LS(C)	ST(B)

表 30. TwoMem19 指令段格式

TwoMov19MMPF 指令段是一种 19 位指令段, 能与数据移动操作一起作八种不同组合的存储器装载与存储操作, 如表 31 所示。

18	17	16	15-8	7-0
0	0	0	LD(A)	DMOVB
0	0	1	ST(A)	DMOVB
0	1	0	DMOVA	LD(B)
0	1	1	DMOVA	ST(B)
1	0	0	DMOVA	LS(C)
1	0	1	LS(C)	DMOVB
1	1	0	DMOVA	DMOVB
1	1	1	保留	

表 31. TwoMov 19 指令段格式

ThreeMem27MMPF 指令段是一种 27 位指令段, 能执行表 32 所示的八种不同组合的存储器装载、存储器存储和数据操作。

26	25	24	23-16	15-8	7-0
0	0	0	LS(C)	LD(A)	LD(B)
0	0	1	LS(C)	LD(A)	ST(B)
0	1	0	LS(C)	ST(A)	LD(B)
0	1	1	LS(C)	ST(A)	ST(B)
1	0	0	LS(C)	DMOVA	LD(B)
1	0	1	LS(C)	DMOVA	ST(B)
1	1	0	LS(C)	LD(A)	DMOVB
1	1	1	LS(C)	ST(A)	DMOVB

表 32. ThreeMem 27 指令段格式

公式(4)给出由 LD(A)指令子段执行的操作。

$$\begin{Bmatrix} R0-R7 \\ L0h-L3h \\ L0l-L3l \end{Bmatrix} = *AX \begin{bmatrix} ++ \\ -- \\ ++AM0 \\ ++AM1 \end{bmatrix}; \quad (4)$$

表 33 列出本发明该示例性实施例提供的 LD(A)指令子段。

7	6	5	4	3	2	1	0
dreg				A0-A3		Amod	

表 33. LD(A)指令子段格式

公式(5)是 LD(B)指令子段执行的操作。

$$\begin{Bmatrix} R0-R7 \\ L0h-L3h \\ L0l-L3l \end{Bmatrix} = *BX \begin{bmatrix} ++ \\ -- \\ ++BM0 \\ ++BM1 \end{bmatrix}; \quad (5)$$

表 34 列出本发明该示例性实施例提供的 LD(B)指令子段。

7	6	5	4	3	2	1	0
dreg				B0-B3		Bmod	

表 34. LD(B)指令子段格式

公式(6)是 ST(A)指令子段执行的操作。

$$*AX \begin{bmatrix} ++ \\ -- \\ ++AM0 \\ ++AM1 \end{bmatrix} = \begin{Bmatrix} R0-R7 \\ L0h-L3h \\ L0l-L3l \end{Bmatrix}; \quad (6)$$

表 35 列出本发明该示例性实施例提供的 ST(A)指令子段。

7	6	5	4	3	2	1	0
dreg				A0-A3		Amod	

表 35. ST(A)指令子段格式

公式(7)是 ST(B)指令子段执行的操作。

$$*BX \begin{bmatrix} ++ \\ -- \\ ++BM0 \\ ++BM1 \end{bmatrix} = \begin{Bmatrix} R0-R7 \\ L0h-L3h \\ L0l-L3l \end{Bmatrix}; \quad (7)$$

表 36 列出本发明该示例性实施例提供的 ST(B)指令子段。

7	6	5	4	3	2	1	0
dreg				B0-B3		Bmod	

表 36. ST(A) 指令子段格式

表 37 列出 DMOVA 指令子段执行的操作。

(a)	<i>NOP</i> ;	平行指令的 NOP
(b)	<i>TLOOP</i> ;	紧密环路(单指令环路)
(c)	$\{R0-R7\} = \begin{Bmatrix} R0-R7 \\ L0h-L3h \\ L0l-L3l \end{Bmatrix};$	转移数据记录器
(d)	$\{R0-R7\} = \{A0-A3\};$	将地址记录器转移至数据记录器
(e)	$\begin{Bmatrix} A0-A3 \\ AM0-AM1 \end{Bmatrix} = \{R0-R3\};$	将数据记录器转移到 AGU 记录器
(f)	$\begin{Bmatrix} A0-A3 \\ B0-B3 \\ C0-C3 \end{Bmatrix} = \begin{Bmatrix} A0-A3 \\ B0-B3 \\ C0-C3 \end{Bmatrix};$	转移地址记录器
(g)	<i>IF cond</i>	平行指令的条件 <i>al Instructions.</i>

表 37. DMOVA 指令子段操作

表 38 是本发明该示例性实施例提供的 DMOVA 指令子段格式。

	7	6	5	4	3	2	1	0
(a,b,c)	0	R0-R7 (dst)			dreg (src)			
(d)	1	0	0	0	A0-A3 (src)		R0-R3 (dst)	
(e)	1	0	0	1	A0-A3 (dst)		R0-R3 (src)	
(f)	1	0	1	0	A0-A3 (dst)		A0-A3 (src)	
(f)	1	0	1	1	B0-B3 (dst)		B0-B3 (src)	
(f)	1	1	0	0	C0-C3 (dst)		C0-C3 (src)	
(d)	1	1	0	1	A0-A3 (src)		R4-R7 (dst)	
(g)	1	1	1		cond			

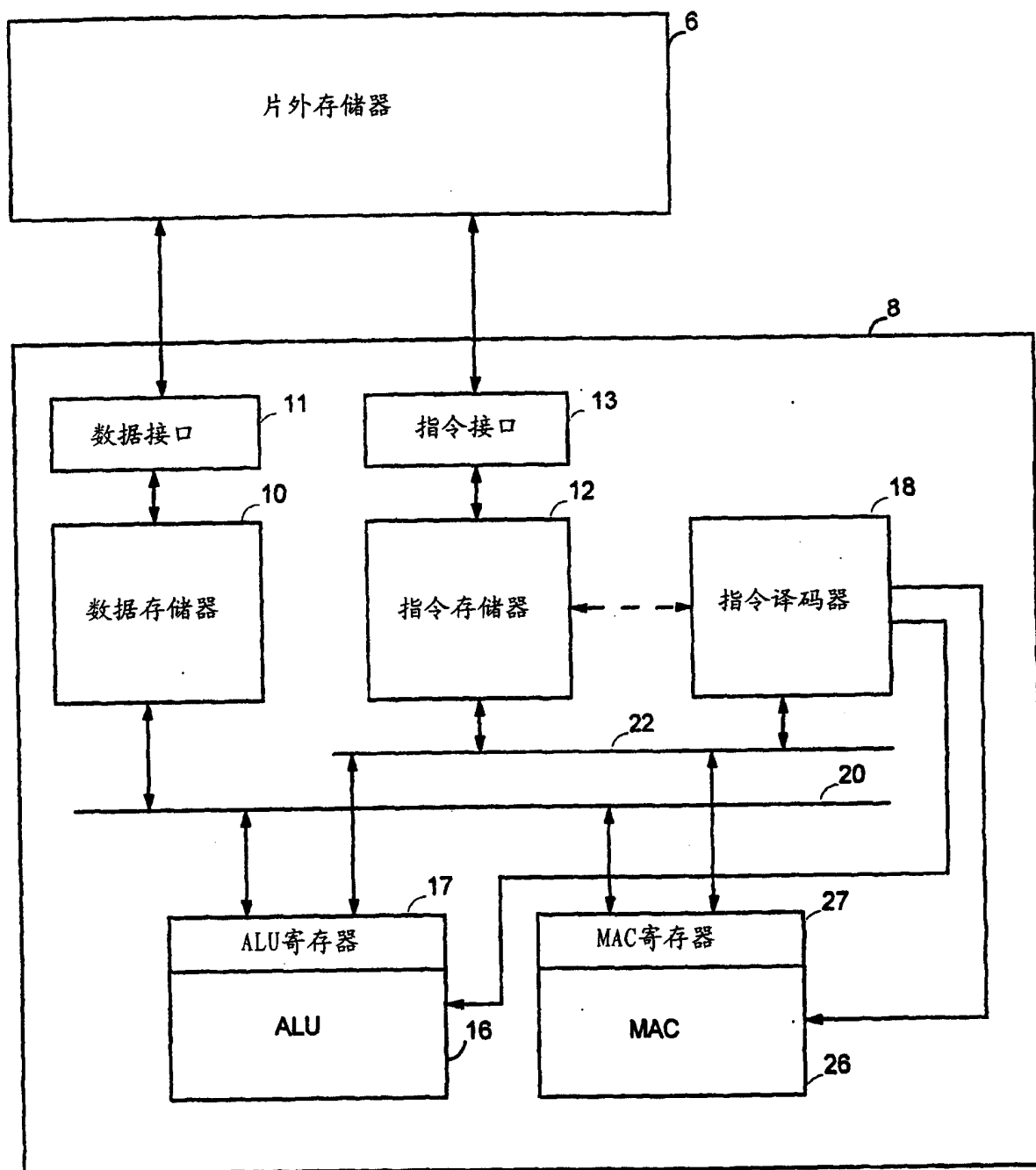
表 38. DMOVA 指令子段格式

这样, 提供可包含一个或多个指令子段的 MMPF 指令段, 进一步增加了可用全指令执行的操作种数。例如, 全指令可让算术与 MAC 操作同一组三种存储器移动与程序流操作一起执行。用单条指令执行这么许多操作的能力, 进一步减少了作某种给定操作所需的指令总数, 从而减少了对 DSP 要求的整个指令存储器。减少指令存储器可减小芯模尺寸, 从而减少了 DSP 的成本与功耗, 这对于范围广泛的包括移动无线电话的应用而言, 使 DSP 更为合适。

这样已经描述了应用并行可变长度指令组控制 DSP 的系统与方法。以上描述的诸较佳实施例, 能让本领域的任何技术人员使用本发明。对他们而言, 完全明白这些实施例的种种改型, 而且不须运用发明能力就可将这里定义的基本

原理应用于其它实施例。例如，虽然描述的系统与方法同 DSP 相关，但其各个方面都适用于一般的计算系统与装置。

通过参照诸较佳实施例面对本发明所作的描述，可以充分理解到，有关实施例仅用作示例，对于具备相关知识与技能的人们，各种改型与变化并不背离由所附的权项及其等效的规定的发明精神与范围。



(已有技术)

图 1

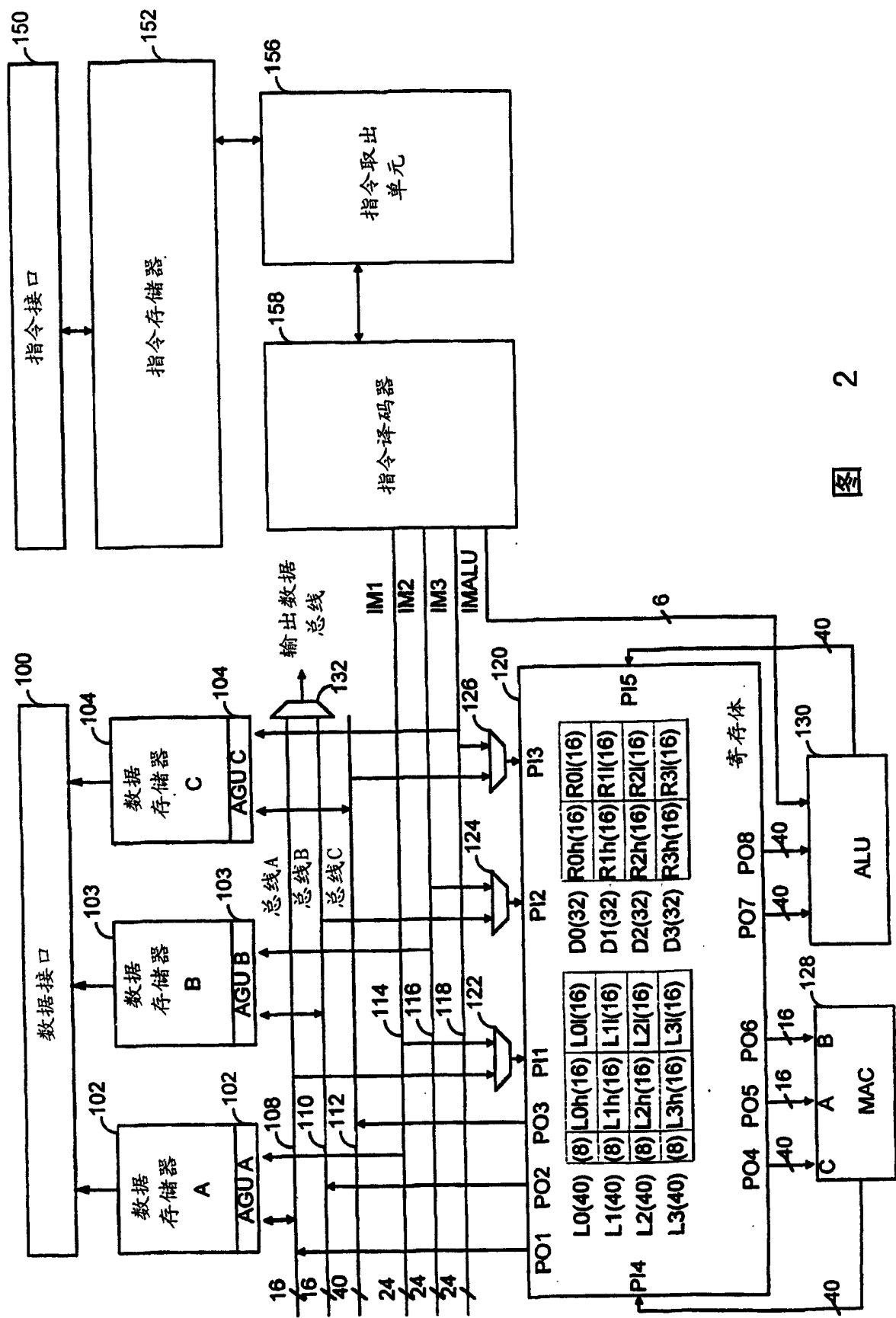
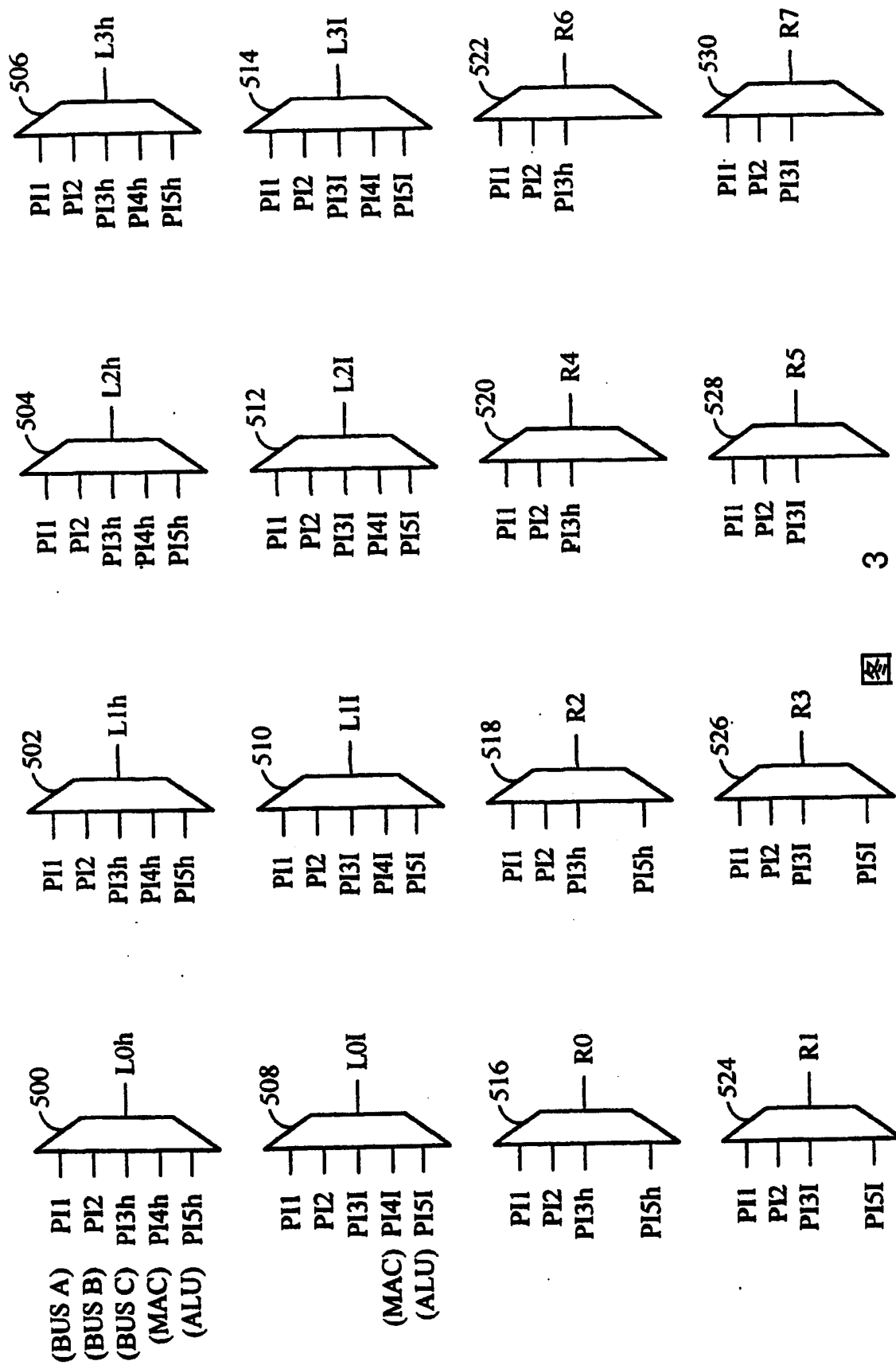
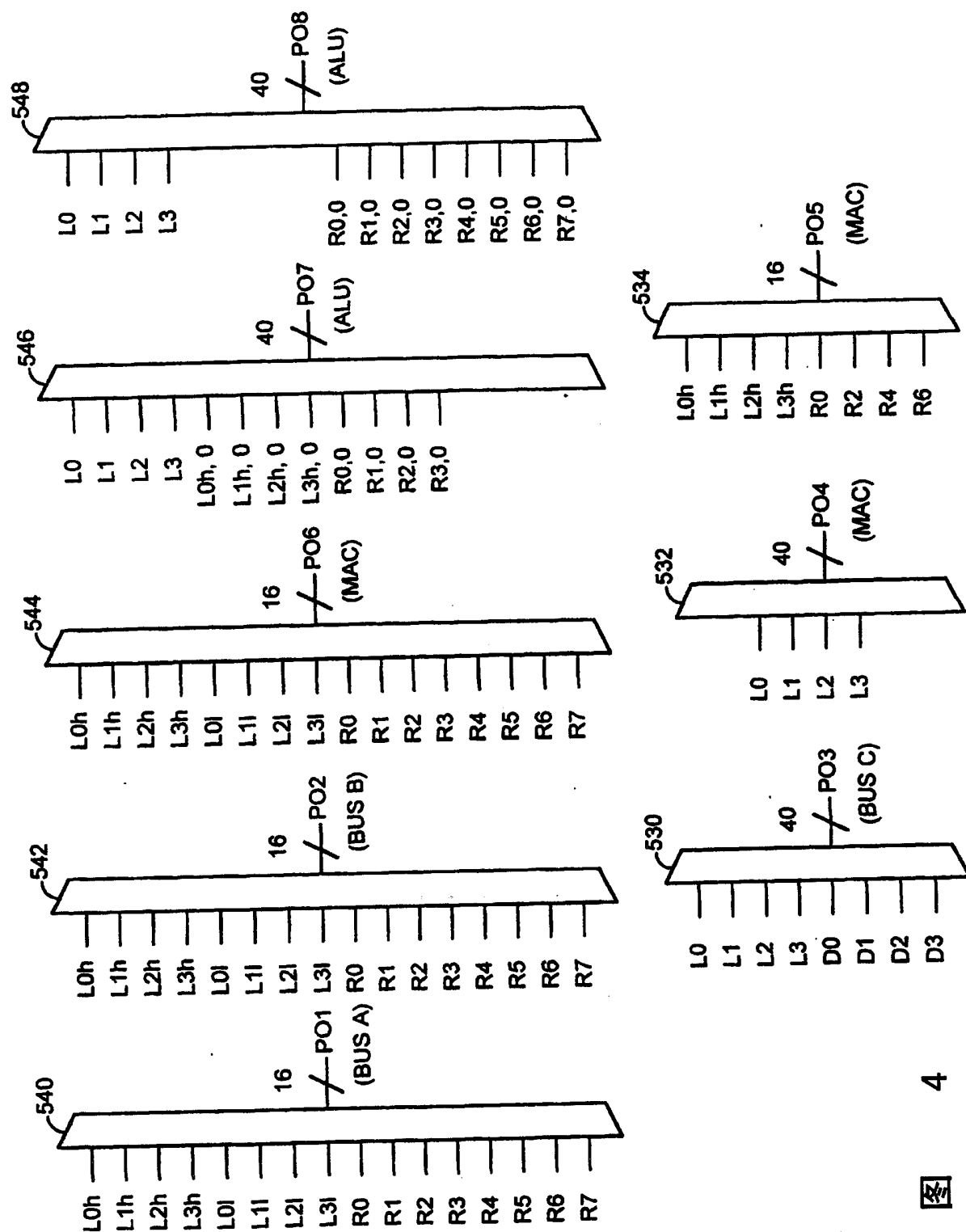


图 2



3

图



4

图

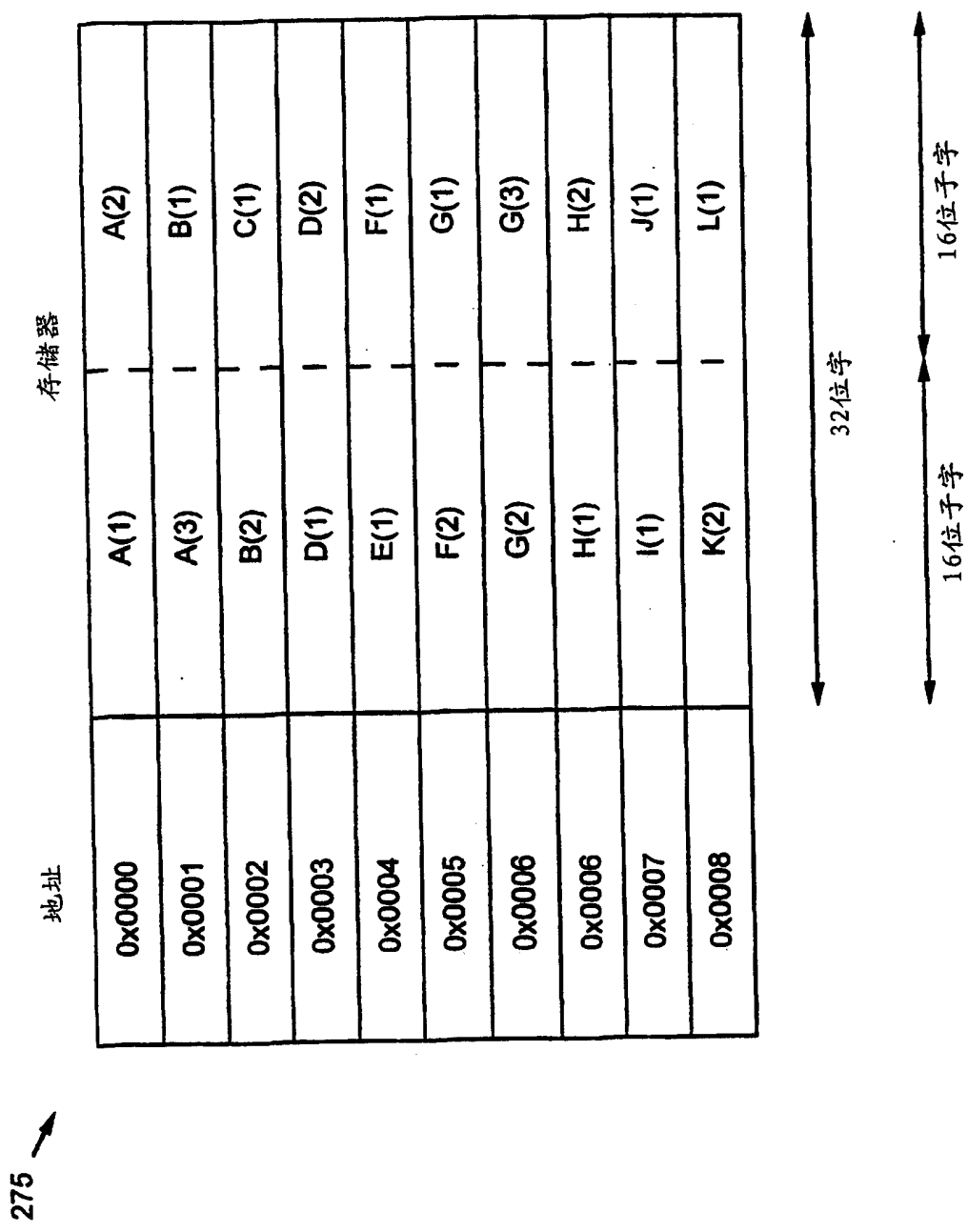


图 5

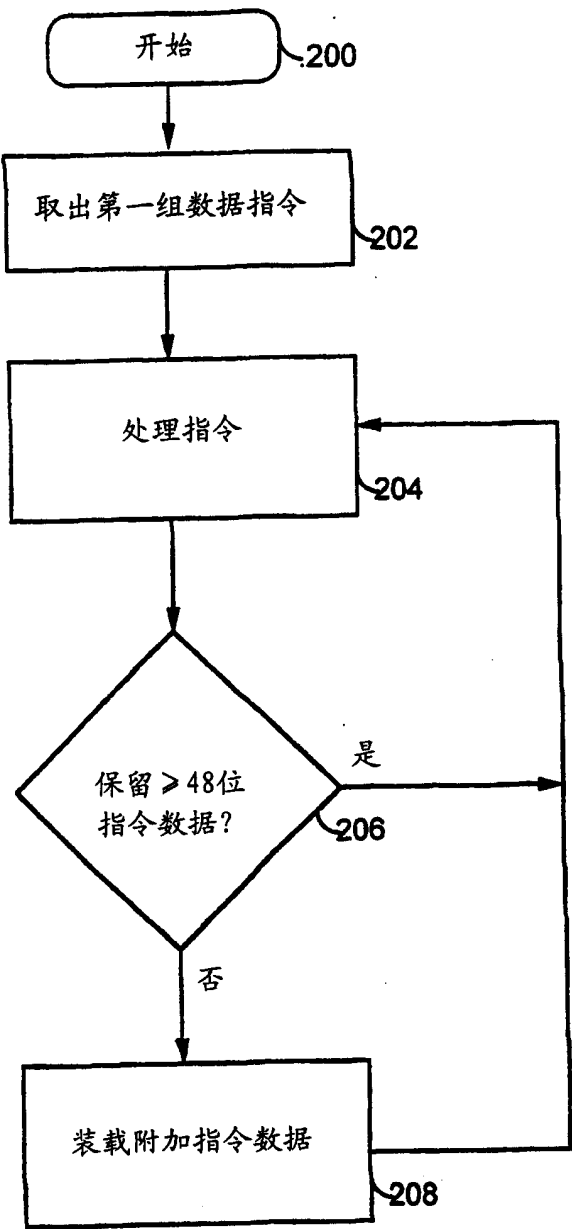


图 6

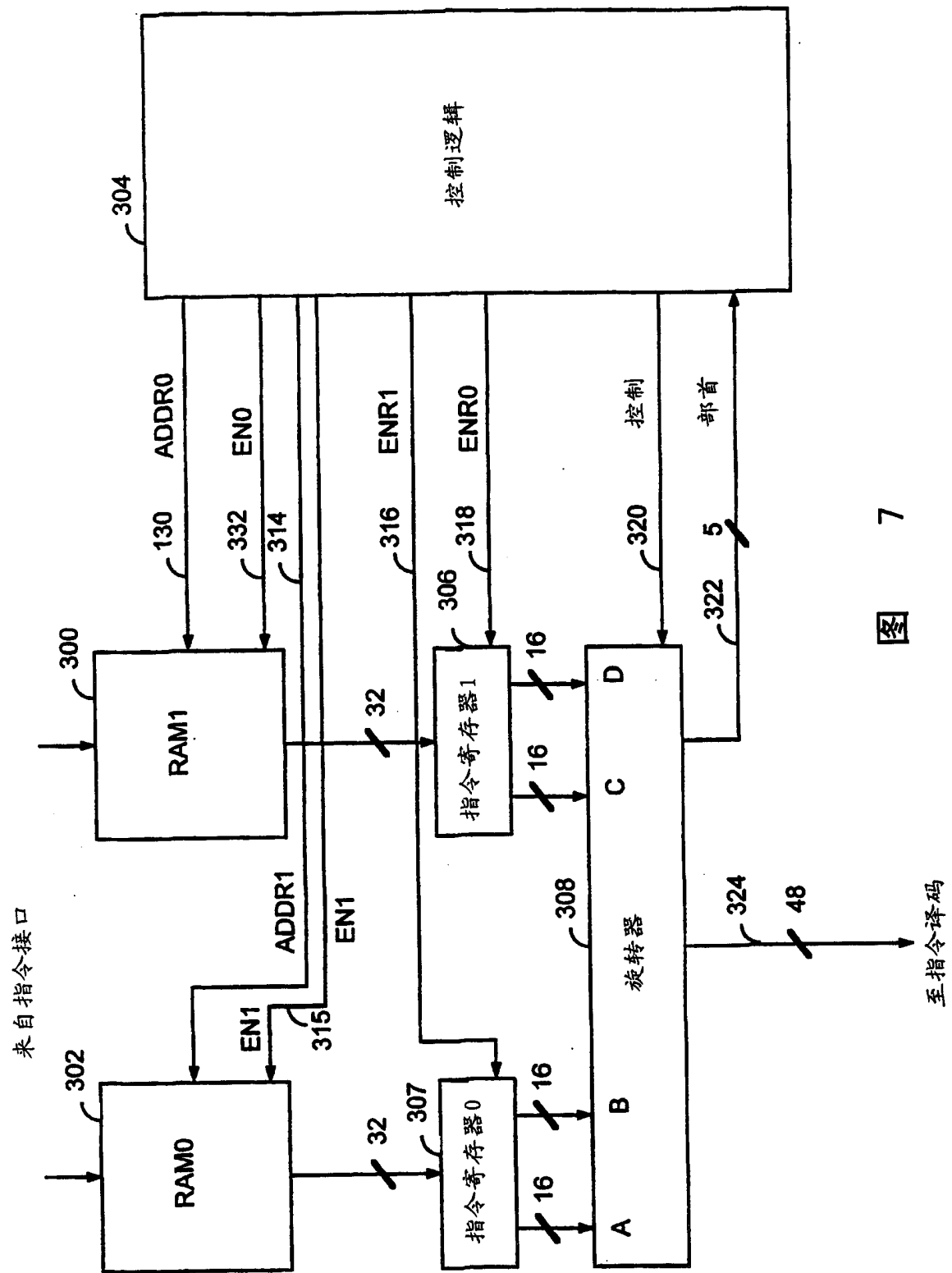


图 7

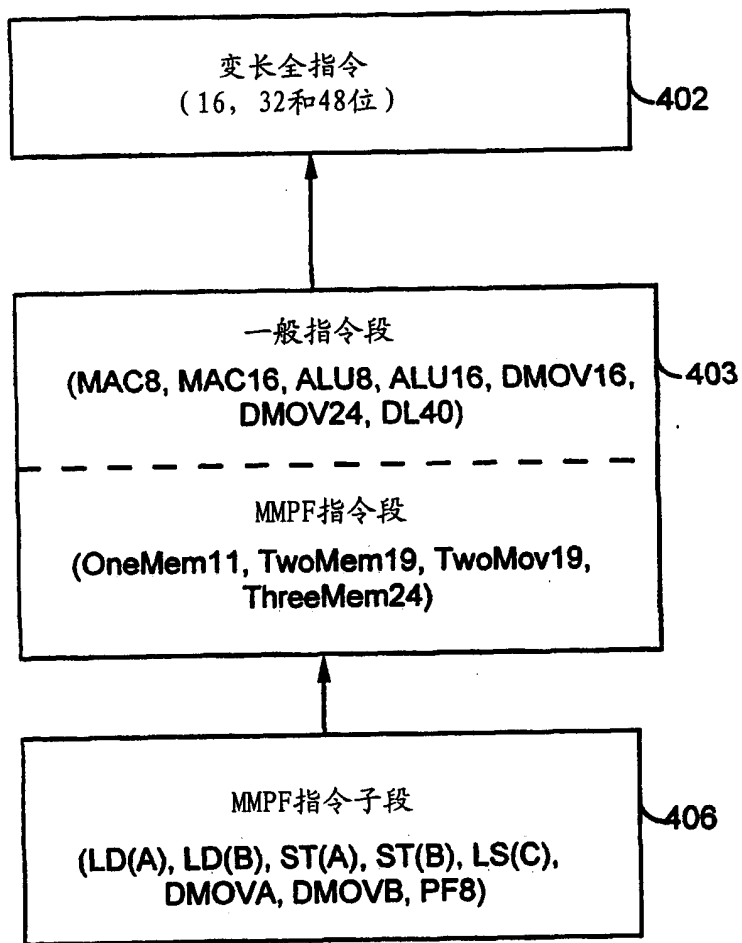


图 9