(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 106354477 A
(43)申请公布日 2017.01.25

(21)申请号 201610556654.1

(22)申请日 2016.07.14

(30)优先权数据
2015-142265 2015.07.16 JP

(71)申请人 瑞萨电子株式会社
地址 日本东京

(72)发明人 木村优之

(74)专利代理机构 中原信达知识产权代理有限责任公司 11219
代理人 穆森 戚传江

(51)Int.Cl.
G06F 9/30(2006.01)
G06F 9/38(2006.01)

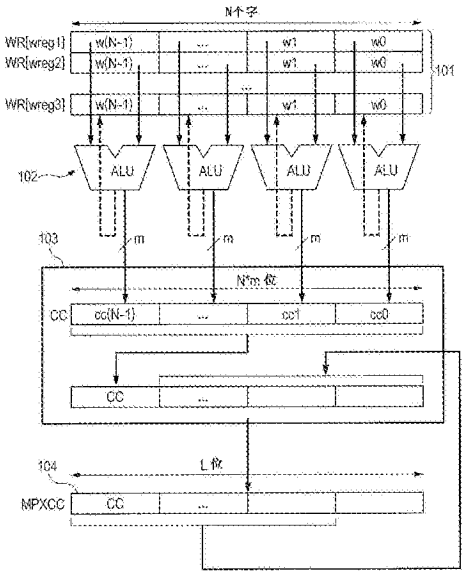
权利要求书3页 说明书15页 附图14页

(54)发明名称

半导体器件

(57)摘要

本发明涉及一种半导体器件。该半导体器件包括能够执行向量指令的中央处理单元。向量指令是用于对每个元素计算向量寄存器的指令，基于对每个元素所计算的结果来对附加信息进行组合，使与向量寄存器不同的寄存器的内容向右或向左移位，将组合的附加信息插入到由于移位所引起的空的部分中，并且使附加信息积聚在寄存器中。



CN 106354477 A

1.一种半导体器件,所述半导体器件包括能够执行向量指令的数据处理器,

其中,所述数据处理器包括第一向量寄存器和第二向量寄存器,以及通用寄存器或专用寄存器,

其中,所述向量指令是下述指令:所述指令用于对每个元素计算所述第一向量寄存器的内容以及所述第二向量寄存器的内容,基于对每个元素的计算结果来对附加信息进行组合,使所述通用寄存器或所述专用寄存器的内容向右或向左移位,将组合的附加信息插入到由于移位所引起的空的部分中,并且使所述附加信息积聚在所述通用寄存器或所述专用寄存器中。

2.根据权利要求1所述的器件,

其中,所述第一向量寄存器和所述第二向量寄存器的每一个能够存储N段元素,并且

其中,所述数据处理器能够并行地对所述N段元素执行运算并且产生N段附加信息。

3.根据权利要求2所述的器件,

其中,所述向量指令是用于将所述第一向量寄存器的内容与所述第二向量寄存器的内容进行相互比较的指令,并且

其中,所述附加信息是基于比较结果的标志;在与比较条件一致的情况下为1或0,而在与所述比较条件不一致的情况下为0或1。

4.根据权利要求3所述的器件,

其中,所述向量指令能够明确地指定向右或向左移位、所述比较条件、以及并行计算的元素的数目,并且隐含地指定所述通用寄存器或所述专用寄存器。

5.根据权利要求4所述的器件,进一步包括:

第三向量寄存器,

其中,所述向量指令将所述计算结果存储在所述第三寄存器中。

6.根据权利要求5所述的器件,

其中,N是从1至4并且一个元素具有32位的宽度。

7.根据权利要求6所述的器件,

其中,所述第一向量寄存器、所述第二向量寄存器、以及所述第三向量寄存器的每一个具有128位的宽度,所述通用寄存器具有32位的宽度,并且所述专用寄存器具有32位的宽度。

8.根据权利要求2所述的器件,进一步包括:

第一组合电路,所述第一组合电路对所述附加信息进行组合;

移位电路,所述移位电路使所述通用寄存器或所述专用寄存器的内容向右或向左移位;以及

第二组合电路,所述第二组合电路对所述第一组合电路的输出与所述移位电路的输出进行组合。

9.根据权利要求8所述的器件,

其中,所述专用寄存器能够并行地进行数据读取和写入。

10.根据权利要求9所述的器件,

其中,所述数据处理器能够执行标量指令,并且

所述标量指令包括用于将所述专用寄存器的内容传输到所述通用寄存器的指令以及

用于从所述通用寄存器的低位或高位对包括1或0的第一位置进行检测的指令。

11.一种半导体器件,包括:

中央处理单元,所述中央处理单元能够执行向量指令和标量指令;以及

存储单元,所述存储单元能够存储所述向量指令和所述标量指令,

其中,所述中央处理单元包括:

第一向量寄存器、第二向量寄存器、以及第三向量寄存器,

通用寄存器,以及

专用寄存器,

其中,所述向量指令是下述指令:所述指令用于对每个元素将所述第一向量寄存器的内容与所述第二向量寄存器的内容进行相互比较,将比较结果存储在所述第三寄存器中,基于对每个元素的所述比较结果对附加信息进行组合,使所述通用寄存器或所述专用寄存器的内容向右或向左移位,将组合的附加信息插入到由于移位所引起的空的部分中,并且使所述附加信息积聚在所述通用寄存器或所述专用寄存器中。

12.根据权利要求11所述的器件,

其中,所述第一向量寄存器、所述第二向量寄存器、以及所述第三向量寄存器的每一个能够存储N段元素,并且

其中,所述中央处理单元能够并行地对所述N段元素执行比较并且产生N段附加信息。

13.根据权利要求11所述的器件,

其中,N是从1至4并且一个元素具有32位的宽度。

14.根据权利要求13所述的器件,

其中,所述第一向量寄存器、所述第二向量寄存器、以及所述第三向量寄存器的每一个具有128位的宽度,所述通用寄存器具有32位的宽度,并且所述专用寄存器具有32位的宽度。

15.根据权利要求12所述的器件,

其中,所述附加信息是基于所述比较结果的标志;在与比较条件一致的情况下为1或0,而在与所述比较条件不一致的情况下为0或1。

16.根据权利要求15所述的器件,

其中,所述向量指令能够明确地指定向右或向左移位、所述比较条件、以及并行计算的元素的数目,并且隐含地指定所述通用寄存器或所述专用寄存器。

17.根据权利要求16所述的器件,进一步包括:

第一组合电路,所述第一组合电路对所述附加信息进行组合;

移位电路,所述移位电路使所述通用寄存器或所述专用寄存器的内容向右或向左移位;以及

第二组合电路,所述第二组合电路对所述第一组合电路的输出与所述移位电路的输出进行组合。

18.根据权利要求17所述的器件,

其中,所述专用寄存器能够并行地进行数据读取和写入。

19.根据权利要求11所述的器件,

其中,所述标量指令包括用于将所述专用寄存器的内容传输到所述通用寄存器的指令

以及用于从所述通用寄存器的低位或高位对包括1或0的第一位置进行检测的指令。

20.根据权利要求19所述的器件，

其中，所述中央处理单元包括：

向量运算单元，所述向量运算单元执行所述向量指令；以及

标量运算单元，所述标量运算单元执行所述标量指令。

半导体器件

[0001] 相关申请的交叉引用

[0002] 通过参考将于2015年7月16日提交的包括说明书、附图、以及摘要的日本专利申请No.2015-142265的全部公开内容引入本文。

技术领域

[0003] 本公开涉及一种半导体器件，并且例如，它可应用于包含用于执行向量指令的CPU的半导体器件。

背景技术

[0004] 存在用于对两个打包的运算对象的数据元素进行比较以对字符串进行处理的单指令多数据(SIMD)指令(例如美国专利公开No.2008/0077773)。SIMD指令还被称为向量指令并且在本公开中，被称为向量指令。

发明内容

[0005] 当数组的大小超过了一个向量指令所处理的元素的数目时，在数组中的数据搜索中，必须将标量指令插入向量指令之间，这使得难以有效地使用向量指令。

[0006] 从对本说明书和附图的描述可显而易见地得知其它问题和新颖特征。

[0007] 对于本公开，如下简要地描述了典型一个的概要。简单地说，向量指令将创建与运算结果不同的附加信息并且使该附加信息积聚在与附加信息不同的寄存器中。

[0008] 根据本发明，可有效地使用向量指令。

附图说明

[0009] 图1是用于描述根据一个实施例的向量指令的方框图。

[0010] 图2是用于描述根据第一示例的半导体器件的方框图。

[0011] 图3是用于描述根据第一示例的向量指令的方框图。

[0012] 图4是用于描述插入操作的方框图。

[0013] 图5是用于描述插入操作的方框图。

[0014] 图6是用于描述图3中的专用电路的操作的方框图。

[0015] 图7是用于描述根据比较示例的向量指令的方框图。

[0016] 图8是用于描述通过使用根据比较示例的向量指令在连续数组中的比较操作的视图。

[0017] 图9是用于描述通过使用根据第一示例的向量指令在连续数组中的比较操作的视图。

[0018] 图10是用于描述根据第二示例的向量指令的方框图。

[0019] 图11是用于描述图10中的专用寄存器的方框图。

[0020] 图12是用于描述在使用根据比较示例的向量指令的情况下用于执行算法的指令

的结构的方框图。

[0021] 图13是用于描述在使用根据比较示例的向量指令来执行算法的情况下的执行处理的方框图。

[0022] 图14是用于描述在使用根据第二示例的向量指令的情况下用于执行算法的指令的结构的方框图。

[0023] 图15是用于描述在使用根据第二示例的向量指令来执行算法的情况下的执行处理的方框图。

具体实施方式

[0024] 在下文中，利用附图对实施例或示例进行描述。在下面的描述中，相同代码被附于相同部件并且可以省略对其的重复描述。

[0025] <实施例>

[0026] 图1是用于描述根据一个实施例的向量指令的视图。根据该实施例的向量指令是用于执行向量寄存器中的运算以同时计算N段数据的指令。就此，向量指令产生N段运算结果并且还取决于该运算结果产生用于支持运算结果的信息(诸如运算结果和比较结果的标志这样的附加信息)。

[0027] 根据该实施例的向量指令是这样的指令，该指令用于计算第一向量寄存器(WR[wreg1])的内容和第二向量寄存器(WR[wreg2])的内容、将运算结果存储到第三向量寄存器(WR[wreg3])中、产生与运算结果相独立的附加信息(CC)、并且使上述信息积聚在与向量寄存器(WR)101不同的用于存储附加信息的寄存器(MPXCC)104中。并不总是必须将运算结果存储在向量寄存器(WR)101中。此外，可以不将运算结果存储在第三向量寄存器(WR[wreg3])中，但可以存储在第一向量寄存器(WR[wreg1])或第二向量寄存器(WR[wreg2])中。每个向量寄存器(WR)101存储N段元素(w0,w1,...,w(N-1))。

[0028] 根据实施例的执行向量指令的数据处理器包括向量寄存器(WR)101、用于计算向量寄存器(WR)101的内容的N段算术单元(ALU)102、专用电路103、以及寄存器(MPXCC)104。相应N个算术单元(ALU)102产生相应附加信息元素(cc0,cc1,...,cc(N-1))。可通过专用电路103将附加信息元素(cc0,cc1,...,cc(N-1))组合为附加信息(CC)。组合是指将一些位或位串组合在一起作为一个位串。当附加信息元素(cc0,cc1,...,cc(N-1))的每一个具有m位时，附加信息(CC)变为N*m位。专用电路103使寄存器(MPXCC)104的现有内容向左或向右移位并且此后，将附加信息(CC)插入到空位区域。换句话说，存储在寄存器(MPXCC)中的附加信息(CC)不覆写寄存器(MPXCC)104的整个内容。当将寄存器(MPXCC)104的宽度定义为L位时，寄存器(MPXCC)104可存储L/(N*m)段附加信息(CC)。根据实施例的向量指令即使当超过了一个指令可运算的数据数目时也可仅通过连续地执行向量指令而使附加信息积聚到寄存器中。

[0029] 在下文中，用于存储附加信息(CC)的寄存器被称为附加信息存储寄存器(MPXCC)，并且就MPXCC而言，可以使用在正常算术运算中使用的通用寄存器或者可以使用专用寄存器。运算结果的数据是各种的；例如，8位至64位，这取决于向量指令的类型。在每N段运算中所产生的m位的附加信息在标志的情况下通常是2至3位并且在比较运算的结果的情况下通常是1位。

[0030] [第一示例]

[0031] 图2是示出了根据第一示例的半导体器件的结构的方框图。根据第一示例的半导体器件100包括在一个半导体衬底上的作为数据处理器的中央处理单元(CPU)1以及存储设备(存储器)2。CPU 1拥有能够执行向量运算(SIMD运算)的单元。指令取出单元12从存储器2取出指令,指令发送单元13将所取出的指令传递到向量运算单元11,并且向量运算单元11执行该指令。除了向量运算单元11之外,CPU 1包括用于执行标准指令的标量运算单元14以及可以访问存储器2的存储器访问单元15。向量运算单元11与标量运算单元14和存储器访问单元15相耦合以请求它们进行数据传送和接收以及对存储器访问的代理。存储器2存储向量运算单元11所执行的向量指令以及标量运算单元14所执行的标量指令。使用向量寄存器111的指令被称为向量指令并且使用通用寄存器16的指令被称为标量指令。在此,通用寄存器16包括例如每一个具有32位宽度(GR[0]至GR[31])的32个寄存器单元。

[0032] 除了用于存储在运算过程中的结果的通用寄存器16之外,CPU 1还包括用于对CPU 1的控制信息以及访问信息进行管理的系统寄存器17。向量运算单元11还具有通常用于保持向量运算的设置信息以及标志的内容的系统寄存器17。通用指令可以访问通用寄存器16,但不能访问系统寄存器17。系统寄存器访问指令可用于将通用寄存器16的内容传输到系统寄存器17并且将系统寄存器17的值传输到通用寄存器16。存储器2是由诸如高速缓冲存储器这样的易失性存储器或者诸如闪速存储器这样的电可重写的非易失性存储器构成的。

[0033] 图3是用于描述根据第一示例的向量指令的功能的方框图。向量运算单元11包括向量寄存器(WR)111、算术单元(ALU)112、以及电路113。向量寄存器(WR)111的每一个存储四个元素(w0,w1,w2,w3)。因此,向量运算单元11具有用于计算向量寄存器(WR)111的内容的四个算术单元(ALU)112。这四个算术单元(ALU)112分别产生附加信息元素(cc0,cc1,cc2,cc3)。通过专用电路113将附加信息元素(cc0,cc1,cc2,cc3)组合为附加信息(CC)。附加信息(CC)具有4位。专用电路113使为MPXCC的通用寄存器(GR[1])114的现有内容向右或向左移位,并且此后将附加信息(CC)插入到空位区域。换句话说,存储在通用寄存器(GR[1])114中的附加信息(CC)不是全部覆写通用寄存器(GR[1])114的内容。当将通用寄存器(GR[1])114的宽度定义为32位时,可将32/4=8段的附加信息(CC)存储在通用寄存器(GR[1])114中。在该示例中,虽然通用寄存器的GR[1]用作MPXCC,但是它并不限于此,而是可以使用任何寄存器,只要它是通用寄存器。

[0034] 根据第一示例的向量指令是这样的指令,该指令用于使用两个向量寄存器来执行运算、将运算结果写入到向量寄存器之中、并且输出取决于运算结果支持运算结果的附加信息;例如,指令如下:

[0035] cmp1.N order,cond,wreg1,wreg2,wreg3。

[0036] 根据该示例的向量指令对向量寄存器(wreg1)的内容与向量寄存器(wreg2)的内容进行比较,将结果存储到向量寄存器(wreg3)中,并且同时将附加信息存储到隐含指定的通用寄存器(GR[1])114中。根据该示例的向量指令在比较结果为不一致的情况下将0存储在wreg3中并且在比较结果为一致的情况下存储1。wreg1、wreg2、wreg3是128位长度并且被分成N(=1,2,4)段数据。在N=1的情况下,使用向量寄存器中的最低有效字w0;在N=2的情况下,使用向量寄存器中的两个低字w1和w0;并且在N=4的情况下,使用向量寄存器中的整

体w3,w2,w1,w0。一个字具有32位并且w3,w2,w1,w0的每一个具有32位。作为比较结果,根据该示例的向量指令产生N位的附加信息(CC)并且将相同信息插入到通用寄存器(GR[1])114之中。将N位的附加信息(CC)插入到由于通用寄存器(GR[1])的值向右或向左移位N位而产生的空的部分之中。在这点上,取决于"阶序(order)",确定是从通用寄存器(GR[1])中的高阶序(向右移位)还是低阶序(向左移位)插入附加信息(CC)。这使得能够从高阶序地址进行搜索以及从低阶序地址进行搜索。图3示出了向右移位的情况。"cond"指定附加信息的设置条件(=,>,<,≥,≤,≠)。

[0037] 图4和5是用于描述插入操作的视图。图4是从寄存器的低阶序插入数据的情况并且图5是从寄存器的高阶序插入数据的情况。在将n位数据插入到L位的寄存器(sysreg(GR[1]))的情况下,如下以Verilog-HDL语言描述具体操作。

[0038] 在从寄存器的低阶序插入数据的情况下(图4):

[0039] sysreg[L-1:0]<={sysreg[L-n:0],FLAG[n-1:0]}

[0040] 在从寄存器的高阶序插入数据的情况下(图5):

[0041] sysreg[L-1:0]<={FLAG[0:n-1],sysreg[L-n:0]}

[0042] 如图4所示,在从寄存器的低阶序插入数据的情况下,L位的寄存器(sysreg)的内容向左移位n位并且将n位的信息(FLAG:标志)存储在sysreg的低阶序中。将sysreg中的低阶序(L-n)位与n位的FLAG进行组合并且抛弃sysreg中的高阶序n位。如图5所示,在从寄存器的高阶序插入数据的情况下,L位的寄存器的内容向右移位n位并且将n位的FLAG存储在寄存器的高阶序中。将n位的FLAG与sysreg中的高阶序(L-n)位进行组合并且抛弃sysreg中的低阶序n位。

[0043] 将描述作为用于将附加信息存储到通用寄存器(GR[1])114之中的电路的专用电路113。图6是用于描述图3中的专用电路的操作的方框图。根据该示例的向量指令对所产生的附加信息元素(cc[3:0])进行组合以作为组合电路1131中的运算结果,产生附加信息(CC),并且将上述信息存储在通用寄存器(GR[1])114中。为了将附加信息(CC)存储在通用寄存器(GR[1])114中,通过数据路径115一次从存储目的地的通用寄存器(GR[1])114读取寄存器值,通过移位器1132在那里执行移位处理,通过组合电路1133插入附加信息(CC),并且通过数据路径116将该结果的值重写到通用寄存器(GR[1])114。移位器1132使数据向通过"阶序"所指定的方向(左或右)移位由"N"所指定的固定值(例如4位)。

[0044] <比较示例>

[0045] 将描述在本公开之前由发明人等所研究的技术(在下文中称为比较示例)。图7是用于描述根据比较示例的向量指令的方框图。根据比较示例的向量指令是这样的指令,该指令用于利用两个向量寄存器来执行运算,将运算结果写入到向量寄存器,并且输出取决于运算结果支持运算结果的信息(运算结果的标志以及通过对比较结果的附加信息进行处理所获得的索引(index));例如,指令如下:

[0046] cmp3.N order,cond,wreg1,wreg2,wreg3

[0047] 根据比较示例的向量指令是这样的指令,该指令用于将wreg1与wreg2之间的每个元素与被认为是字符串的向量寄存器(wreg1)和向量寄存器(wreg2)的内容进行比较,将结果存储在向量寄存器(wreg3)中,同时计算与比较结果(附加信息)中的条件相匹配的最低和最高有效位的位置,并且将上述存储在通用寄存器中(隐含地指定的寄存器,例如GR

[1])。简单地说，根据比较示例的向量指令将与比较条件(结果的位置信息)相匹配的第一位的位置存储在通用寄存器中。

[0048] 为了从一些数组搜索出与某个条件相匹配的数据，将比较结果的附加信息移动到通用寄存器，由此被转换成按序处理，这在搜索中需要大量时间。更具体地说，假定利用根据比较示例的向量指令实现下述第一算法，所述第一算法用于从如下按照降序或升序所排列的数组搜索超过一些边界值的位置。在该公开中，伪码用于描述算法。伪码基于C语言。从头为“//”开始的句子是注释。

[0049]
```
for（i= 0;i < M;i++）{
    //array[]是要搜索的数组， border 是搜索中的边界
    if（border > array[i]）return i;
}
```

[0050] 如图7所示，在向量寄存器311的内容被认为是字符串的情况下，根据比较示例的向量指令利用向量算术单元312进行比较并且通过专用电路313的组合电路3131来收集结果。此后，专用电路313的索引产生电路3132计算比较结果的附加信息的位串中具有1位的最低有效位的位置，以产生索引。将该结果存储在通用寄存器(GR[1])314中。当不存在与比较条件相匹配的向量元素时，将特定数值写入到通用寄存器(GR[1])314中。按照下述方式来检查在向量寄存器311中是否存在作为比较目标的字：在执行了根据比较示例的向量指令并且此后读取通用寄存器(GR[1])314之后，检查通用寄存器(GR[1])314是否包括用于表示不存在匹配的向量元素的特定数值。基于该结果，确定是否读取下一字符串并且在向量寄存器311中进行比较。该处理是通过利用标量指令来执行的。

[0051] 在利用根据比较示例的向量指令的这种情况下，因为从比较结果的附加信息所产生的信息是索引信息，因此必须在每个比较中参考通用寄存器来确认该搜索是否成功。换句话说，当根据比较示例的向量指令可同时执行四个比较(在N＝4的情况)时，它作为算法来每4个一次地检查相关值是否存在于四个数组中。因为索引存储在通用寄存器中，因此根据比较示例的向量指令需要诸如比较指令和分支指令这样的标量指令并且以混合方式包括向量指令和标量指令，这干扰了流水线的有效利用。当连续地执行根据比较示例的向量指令而无需检查通用寄存器的内容时，通用寄存器的内容被覆写并且过去的向量指令中的比较结果的附加信息不被继承。

[0052] 具体地说，在使用根据比较示例的向量指令的情况下，必须经历以下步骤。

[0053] 步骤1：ANS＝0。ANS是用于表示搜索字的索引的一些通用寄存器。

[0054] 步骤2：执行根据比较示例的向量指令。

[0055] 步骤3：检查GR[1]＝4。当GR[1]＝4时，在执行了ANS＝ANS+GR[1]之后，该操作移动到步骤4。当GR[1]≠4时，该操作移动到步骤5。GR[1]＝4是用于表示不存在匹配元素的特定数值。

[0056] 步骤4：将下一字符串加载到向量寄存器中并且移动到步骤2。

[0057] 步骤5：结束。ANS＝ANS+GR[1]变为搜索字的索引。

[0058] 如上所述，根据比较示例的向量指令需要除向量指令之外的大量标量指令。需要这么多指令以搜索索引的理由是根据比较示例的向量指令不继承向量指令中的先前比较

结果的附加信息并且每当执行根据比较示例的向量指令中的比较时标量指令必须检查比较结果。

[0059] 在根据比较示例的向量指令中，将索引的存储目的地定义为通用寄存器；因此，为了读取并检查向量指令的结果，在通过向量指令将索引的附加信息写入到通用寄存器中之后，必须从通用寄存器读取并通过标量指令来计算附加信息，并且其结果是，出现排队(流水线安装)以便解决写后读(RAW)危险。根据此，根据比较示例的向量指令可使比较本身加速；然而，当它应用于实际算法时，不能有效地使用CPU流水线。

[0060] 在根据第一示例的指令中，可以每一个指令地对于向量算术单元的数目(N位，如果可同时执行N段计算的话)将结果插入到寄存器之中。当四个向量算术单元并行地执行向量指令的比较时，产生包括每个向量单元1位的总共4位的比较结果，作为附加信息。在此，通用寄存器(GR[1])114的宽度是32位。根据此，可连续地执行通过向量指令的比较，直到装满整个通用寄存器114(GR[1])114(结束对32个元素的比较)。换句话说，当将算术单元112的数目定义为4并且通用寄存器的位数目是32位时，即使当32/4＝8次的执行向量指令时，通用寄存器(GR[1])也不会使结果溢出。同时，根据比较示例的向量指令必须插入用于恰好在执行一个指令之后检查运算结果的标量指令。根据第一示例的向量指令比根据比较示例的向量指令可更有效地搜索数组，因为它可连续地执行向量运算指令。

[0061] 作为示例，将描述根据比较示例的向量指令以及根据该示例的向量指令对数组A＝[0,4,5,10,12,8,16,27,9,1,5,8,1,0,1,1]与数组B＝[1,3,7,9,15,9,20,13,11,0,3,1,9,0,0,0]进行比较的情况。当将向量指令的并行度定义为4时，每个数组是通过每四个元素加载的以进行比较。在此，作为附加信息存储寄存器的通用寄存器(GR[1])具有初始值0，并且当A[i]<B[i]时，将标志(附加信息元素)定义为1；否则，将标志定义为0。

[0062] 图8是用于描述通过使用根据比较示例的向量指令在连续数组中的比较操作的视图。在使用根据比较示例的向量指令的比较中，数组A和B的每四个元素被加载并且返回首先与比较条件相匹配的索引。在下文中，将详细地描述。

[0063] (1)初始的四个元素，A＝[0,4,5,10]和B＝[1,3,7,9]被加载到向量寄存器以进行比较。将第一元素存储在向量寄存器的最低有效字中并且将第四元素存储在向量寄存器的最高有效字中。因此，wreg1＝[10,5,4,0]和wreg2＝[9,7,3,1]并且最低有效字与比较条件相匹配；作为比较结果，附加信息(index：索引)＝0。

[0064] (2)将比较结果存储在向量寄存器中，如wreg3＝[0x0000_0000,0xffff_ffff,0x0000_0000,0xffff_ffff]。在此，“0x”表示十六进制数。

[0065] (3)将附加信息(索引)0存储在通用寄存器(GR[1])中。在此，GR[1]＝0000_0000_0000_0000。

[0066] (4)对数组A和B的接下来的元素重复上述(1)至(3)。

[0067] 因为第二的四个元素是A＝[12,8,16,27]和B＝[15,9,20,13]，因此wreg1＝[27,16,8,12]和wreg2＝[13,20,9,15]并且最低有效字与比较条件相匹配；其结果是，索引＝0并且GR[1]＝0x0000。

[0068] 因为第三的四个元素是A＝[9,1,5,8]和B＝[11,0,3,1]，因此wreg1＝[8,5,1,9]和wreg2＝[1,3,0,13]并且最低有效字与比较条件相匹配；其结果是，索引＝0并且GR[1]＝0x0000。

[0069] 因为第四的四个元素是A＝[11,0,1,1]和B＝[9,0,0,0],因此wreg1＝[1,1,0,11]和wreg2＝[0,0,0,9],并且每个字与比较条件不匹配;其结果是,索引＝4并且GR[1]＝0x0004。

[0070] 如上所述,总是更新附加信息存储寄存器(GR[1])的值并且先前比较结果的附加信息不保留。因此,就在通过向量运算的比较之后,必须检查附加信息存储寄存器(GR[1])的值。在根据比较示例的向量指令中,返回与比较条件相匹配的第一元素的索引并且比与比较条件相匹配的上述元素要晚的元素的比较结果不反映在附加信息存储寄存器(GR[1])中。

[0071] 图9是用于描述使用根据第一示例的向量指令在连续数组中的比较操作的视图。在根据第一示例的向量指令中,将作为比较结果的附加信息表示为位串并且将该结果向下或向上地推入到作为附加信息存储寄存器的通用寄存器(GR[1])之中。在下文中,将详细地描述。

[0072] (1)初始的四个元素,A＝[0,4,5,10]和B＝[1,3,7,9]被加载到向量寄存器以进行比较。将第一元素存储在向量寄存器的最低有效字中并且将第四元素存储在向量寄存器的最高有效字中。因此,wreg1＝[10,5,4,0]和wreg2＝[9,7,3,1];作为比较结果,附加信息(标志)＝[0,1,0,1]。

[0073] (2)将比较结果存储在向量寄存器中,如wreg3＝[0x0000_0000,0xffff_ffff,0x0000_0000,0xffff_ffff]。在此,“0x”表示十六进制数。

[0074] (3)使附加信息存储寄存器(GR[1])的内容向右移位并且将4位标志[0,1,0,1]插入到GR[1]中。从GR[1]的高位插入附加信息,如GR[1]＝0101_0000_0000_0000。

[0075] (4)对数组A和B的接下来的元素重复上述(1)至(3)。

[0076] 因为第二的四个元素是A＝[12,8,16,27]和B＝[15,9,20,13],因此wreg1＝[27,16,8,12]和wreg2＝[13,20,9,15]并且标志＝[0,1,1,1];其结果是,GR[1]＝0111_0101_0000_0000。

[0077] 因为第三的四个元素是A＝[9,1,5,8]和B＝[11,0,3,1],因此wreg1＝[8,5,1,9]和wreg2＝[1,3,0,13]并且标志＝[0,0,0,1];其结果是,GR[1]＝0001_0111_0101_0000。

[0078] 因为第四的四个元素是A＝[11,0,1,1]和B＝[9,0,0,0],因此wreg1＝[1,1,0,11]和wreg2＝[0,0,0,9]并且标志＝[0,0,0,0];其结果是,GR[1]＝0000_0001_0111_0101。

[0079] 根据上述操作,存储在附加信息存储寄存器(GR[1])中的值是十六进制数的0x1175,其表示相应比较结果的附加信息的值。

[0080] 如上所述,在根据第一示例的向量指令中,将向量指令中的先前比较结果的附加信息保持在附加信息存储寄存器中,直至它由于寄存器宽度的限制而被推出。因此,即使连续地执行向量指令,比较结果的附加信息也可保持在其容量范围之内的附加信息存储寄存器中。根据比较示例的向量指令不会接管向量指令中的先前比较结果的附加信息,但是根据第一示例的向量指令可使附加信息积聚在附加信息存储寄存器(GR[1])中并且接管向量指令中的先前结果,除非附加信息存储寄存器(GR[1])溢出。

[0081] 根据第一示例的向量指令产生与向量指令的运算结果相独立的附加信息并且将上述信息插入到与向量寄存器不同的寄存器中;因此,即使当向量指令超过了可同时执行的并行数据的数目时,可仅通过连续执行向量指令而使结果积聚在寄存器中。与比较示例

不同，不是必须在每次执行一个向量指令时通过标量指令来确认标志的结果等等，而是直到附加信息存储寄存器满了之前可执行向量指令，并且最终，仅检查附加信息存储寄存器就足够了。

[0082] [第二示例]

[0083] 根据第一示例的向量指令需要对通用寄存器(GR[1])读写以便实现通过通用寄存器(GR[1])将附加信息(CC)插入到寄存器之中并且需要通用寄存器排队。换句话说，当根据第一示例的向量指令继续时，排队发生以便解决RAW危险。因此，根据第二示例的向量指令具有用于存储附加信息的专用寄存器和专用电路。

[0084] 图10是用于描述根据第二示例的向量指令的方框图。图11是用于描述图10中的专用寄存器的方框图。除向量运算单元的结构之外，执行根据第二示例的向量指令的半导体器件与根据第一示例的半导体器件相同。除了向量运算单元11A的专用电路113与专用电路213相耦合并且专用电路213与通用寄存器16相耦合之外，根据第二示例的向量运算单元11A与根据第一示例的向量运算单元11相同。专用电路213可以提供于向量运算单元11A之外。专用电路213包括专用寄存器(SR)214和选择器217。

[0085] 根据第二示例的向量指令是这样的指令，该指令用于利用两个向量寄存器来执行运算、将运算结果写入到向量寄存器之中、并且输出取决于运算结果支持运算结果的附加信息；例如，指令如下：

[0086] cmp2.N order,cond,wreg1,wreg2,wreg3

[0087] 根据第二示例的向量指令将向量寄存器(wreg1)的内容与向量寄存器(wreg2)的内容进行比较，将结果存储到向量寄存器(wreg3)之中，并且同时地将附加信息存储到隐含地指定的专用寄存器(SR)之中。除了附加信息的存储目的地之外，根据第二示例的向量指令与根据第一示例的向量指令相同。

[0088] 根据第二示例的向量指令通过组合电路1131对所产生的作为运算结果的附加信息元素(cc[3:0])进行组合以产生附加信息(CC)并且将其存储在专用寄存器(SR)214中。为了将附加信息(CC)存储在专用寄存器(SR)214中，通过数据路径215一次从存储目的地的专用寄存器(SR)214读取寄存器值，通过移位器1132执行移位处理，通过组合电路1133插入附加信息(CC)，并且通过数据路径216将该结果的值重写到专用寄存器(SR)214中。移位器1132使数据向通过“阶序”所指定的方向(右或左)移位由“N”所指定的固定值(例如4位)。

[0089] 与系统寄存器17相似，根据读写专用寄存器的指令(从专用寄存器移动到通用寄存器的指令或者从通用寄存器移动到专用寄存器的指令)来执行专用寄存器(SR)214的读写。专用寄存器(SR)214包括在同一周期中对具有32位宽度的数据进行读写的电路。因此，专用寄存器(SR)214可并行地执行通过数据路径214的数据读取和通过数据路径218的数据写入；因此，当向量指令继续时可更新寄存器而没有任何RAW危险。

[0090] 为了取出并检查附加信息，就在根据第二示例的向量指令之后，执行从专用寄存器移动到通用寄存器的指令。专用寄存器(SR)214可并行地执行通过数据路径220的数据读取和通过数据路径218的数据写入；因此，通用寄存器16可读取数据而不会产生RAM危险。根据从通用寄存器移动到专用寄存器的指令，通过数据路径219、选择器217、以及数据路径218将数据写入到专用寄存器(SR)214中。

[0091] 接下来，将考虑用于从按照降序或升序排列的数组搜索超过了一些边界的位置

(索引)的第一算法。

[0092] 为了实现第一算法,非向量指令用于一个接一个地对数组的元素进行比较,或者向量指令用于同时对多个元素进行比较。用于一个接一个地对数组的元素进行比较的方法是用于利用非向量指令(基本上利用通用寄存器而无需参考向量寄存器的指令,还被称为标量指令)来比较值的方法。另一方面,当利用向量指令时,对于每若干值可同时对存储在数组[]中的值与边界进行比较。如下所示可将第一算法变为第二算法。为了简单起见,假定数组的元素M是向量指令中的并行数N的倍数。

```
//当通过能够对 N 个字同时运算的向量运算指令来对 N 个字执行同时比较时
//按照向量寄存器 vborder 的所有方式存储边界（border）的值。
vborder= {border，border，…，border，border};
for（i= 0；i < M / N；i++）{
```

[0093]

```
//从数组取出值并且将它存储在向量寄存器中
varray= {array[i*N+（N-1）]，array[i*N+（N-2）]，…，array[i*N+1]，array[i*N+0]};
//比较
vresult= v_compare（vborder，array）；
}
```

[0094] 在上述第二算法中,通过利用向量指令可对每N个字相互比较值;然而,需要大量指令以便搜索作为比较结果的附加信息变化的位置(数组的值变为比边界更大的位置)。通常,采用如下所示的第三算法。

```
//当通过能够对 N 个字同时运算的向量运算指令来对 N 个字执行同时比较时
//按照向量寄存器 vborder 的所有方式存储边界的值。
vborder= {border，border，…，border，border};
    index= 0;
for（i= 0;i < M / N;i++）{
    //从数组取出值并且将它存储在向量寄存器中
    varray= {array[i*N+（N-1）]，array[i*N+（N-2）]，…，array[i*N+1]，array[i*N+0]};
    //比较
     //对 vborder 与 varray 之间的每个元素进行比较并且将结果存储在 vresult 中
      //将每个向量元素的标志存储在 flag 中（N 位）
[0095]
    vresult= v_compare（vborder，varray，flag）;
     //在通过向量比较指令对 N 个字执行了比较之后
    if（比较结果与所有运算结果中的条件不匹配（参考标志））{
        //当不包括作为比较结果的相应向量元素时，退出
        break;
      } else {
        index= index + N;     //在比较的向量串中未命中
      }
}
//当包括作为比较结果的相应向量元素时，一个接一个地检查哪个向量元素与条件相匹配。
    for（i= 0;i < N;i++）{
```

```
        if（flag[i]== 1）{
            break;
        } else {
[0096]          index= index + 1;
        }
    }
```

[0097] 作为示例，将描述在使用根据比较示例的向量指令的情况下用于从按照升序的数组A＝[0,1,2,4,5,7,8,10,12,15,16,20,22,25,30,31]搜索超过了值15的数组的索引的第三算法。

[0098] 图12是示出了在使用根据比较示例d向量指令的情况下用于执行算法的指令的结构的视图。图13是示出了在使用根据比较示例的向量指令来执行算法的情况下执行处理的视图。在根据比较示例的向量指令中，将与作为比较结果的附加信息相对应的索引存储在通用寄存器(GR[1])314中。当相应结果未出现在比较结果中时，根据比较示例的向量指令将4作为索引存储在通用寄存器(GR[1])314中。在下文中，将参考图13对使用根据比较示例的向量指令的情况的过程进行描述。将向量指令的并行度定义为4。当A[i]>B[i]时将比较指令定义为1；否则定义为0。

[0099] 步骤1：

```
if (GR[1]! = 4){
    found a value exceeding border（发现超过边界的值）
}else{
[0100]      ANS= ANS +4
}
```

[0101] (1)将边界的值15存储在向量寄存器(wreg2)中；其结果是，wreg2＝[15,15,15,15]。

[0102] (2)将数组A[3-0]的每个值存储在向量寄存器(wreg1)中；其结果是，wreg1＝[4,2,1,0]。

[0103] (3)将wreg1与wreg2进行比较；其结果是，索引＝4并且GR＝0000_0000_0000_0100。

[0104] 步骤2：

```
if (GR[1]!=4) {
    found a value exceeding border（发现超过边界的值）
}else{
[0105]      ANS= ANS +4
}
```

[0106] (1)将数组A[7-4]的每个值存储在向量寄存器(wreg1)中；其结果是，wreg1＝[10,

8,7,5]。

[0107] (2)将wreg1与wreg2进行比较;其结果是,索引=4并且GR=0000_0000_0000_0100。

[0108] 步骤3:

```
if (GR[1]!=4) {
    found a value exceeding border（发现超过边界的值）
    ANS= ANS + GR[1]⇒ loop end
}
    ANS= ANS + GR[1];
}
```

[0109]

[0110] (1)将数组A[11-8]的每个值存储在向量寄存器(wreg1)中;其结果是,wreg1=[20,16,15,12]。

[0111] (2)将wreg1与wreg2进行比较;其结果是,索引=2并且GR=0000_0000_0000_0100。

[0112] 对于数组A[12-15](步骤4),不执行根据比较示例的向量指令。

[0113] 根据比较示例的向量指令覆写附加信息存储寄存器(GR[1])314,而无需存储先前结果;因此,必须插入下述标量指令,所述标量指令用于检查在根据比较示例的向量指令的完全执行中是否发现超过边界的值。该检查是通过利用标量运算单元的算术单元141来执行的。此外,通用寄存器16可替代地由向量指令和标量指令访问。必须执行向量指令和标量指令(检查值是否超过4),从而劣化了性能。

[0114] 如上所述,虽然根据比较示例的向量指令可同时对多个值进行比较,但是此后,必须从移动到通用寄存器的索引搜索值与比较条件相匹配的位置。为了执行第三算法,需要用于对通用寄存器的内容进行比较的指令(比较指令)以及用于基于比较指令的结果来分支的分支指令,这不意味着向量指令的有效使用可被有效地使用。

[0115] 另一方面,当使用根据第二示例的向量指令时,在能够对N个字同时运算的指令的情况下,针对正向无穷取整次数(M/N)执行根据第二示例的向量指令;其结果是,将M位信息排列在附加信息存储寄存器中,如二进制表示的11...10...000。通过使用用于对附加信息存储寄存器中的从最高有效阶序或最低有效阶序至位置0/1的数目进行计数的指令,可计算边界值的索引。具体地说,变为如下所示的第四算法。这是使用能够存储K位的向量运算结果的附加信息的专用寄存器(SR)214作为附加信息存储寄存器的情况。

```
vborder={border, border, ..., border, border};
for (i=0;i < M/K;i++) {
    head_idx=i * K;
    for (j=0;j < K/N;j++) {
        //从数组取出值并且将它存储在向量寄存器中 。
        varray={array[head_idx+(N-1)],array[head_idx+(N-2)],...
```

[0116]

```
        array[head_idx+0]};
                    //比较
                    vresult=v_compare（vborder，array）;
                    head_idx=head_idx + N;
                }
                if（exclusive register !=0x00）{
                    goto finish;
[0117]              }
            }
            finish：
                //search_1_from_right 是从 LSB 连续地搜索具有 1 的位的位置
                //在许多 CPU 中该功能是作为指令而存在的
                one_index=search_1_from_right（专用寄存器）;
                return head_idx + one_index;
```

[0118] 作为示例，将描述在使用根据第二示例的向量指令的情况下用于从按照升序的数组A＝[0,1,2,4,5,7,8,10,12,15,16,20,22,25,30,31]搜索超过值15的数组的索引的第四算法。在此，假定M＝16，K＝16，并且N＝4。虽然已对32位宽度的专用寄存器(SR)214进行了描述，但是为了使得附图和其描述容易起见使用16位宽度(K＝16)的寄存器。

[0119] 图14是示出了在使用根据第二示例的向量指令的情况下用于执行算法的指令的结构的视图。图15是示出了在使用根据第二示例的向量指令来执行算法的情况下的执行处理的视图。在该算法的最内循环中，除了向量指令之外什么都不存在。这与图14的虚线所包围的向量指令相对应。向量指令可连续地重复执行运算(K(＝16)位)，直至填满用于存储比较结果的附加信息的专用寄存器(SR)214。在此，执行K/N(＝16/4＝4)次。根据第二示例的向量指令不必在最内循环中将向量指令的结果移动到通用寄存器16，但是可连续地执行比较，直至专用寄存器(SR)214填满K(＝16)的位。

[0120] 一旦对K(＝16)位的比较完成，估计专用寄存器(SR)214；当存储非0时，这意味着存在超过边界的值。当将0存储在专用寄存器(SR)214中时，这意味着在已比较的K(＝16)数组中不存在超过边界的值，因而从下一数组(最外循环)的位置重新开始比较。这对应于图14的虚线所包围的标量指令。

[0121] 在下文中，参考图15对使用根据第二示例的向量指令的情况的过程进行描述。将向量指令的并行度定义为4。当A[i]>B[i]时将比较指令定义为1；否则定义为0。专用寄存器SR＝0。

[0122] 步骤1：

[0123] (1)将边界的值15存储在向量寄存器(wreg2)中并且它变为wreg2＝[15,15,15,15]。

[0124] (2)将数组A[3-0]的每个值存储在向量寄存器(wreg1)中并且它变为wreg1＝[4,2,1,0]。

[0125] (3)将wreg1与wreg2进行比较并且变为标志＝[0,0,0,0]并且SR＝0000_0000_0000_0000。

[0126] 步骤2：

[0127] (1)将数组A[7-4]的每个值存储在向量寄存器(wreg1)中;其结果是,wreg1＝[10,8,7,5]。

[0128] (2)将wreg1与wreg2进行比较;其结果是,标志＝[0,0,0,0]并且SR＝0000_0000_0000_0000。

[0129] 步骤3：

[0130] (1)将数组A[11-8]的每个值存储在向量寄存器(wreg1)中;其结果是,wreg1＝[20,16,15,12]。

[0131] (2)将wreg1与wreg2进行比较;其结果是,标志＝[1,1,0,0]并且SR＝1100_0000_0000_0000。

[0132] 步骤4：

[0133] (1)将数组A[15-12]的每个值存储在向量寄存器(wreg1)中;其结果是,wreg1＝[31,30,25,22]。

[0134] (2)将wreg1与wreg2进行比较;其结果是,标志＝[1,1,1,1]并且SR＝1111_1100_0000_0000。

[0135] 在上述处理中,比较结果在数组A中超过值15的位置处倒置;其结果是,发现超过15的数组的索引是10。这可通过一个指令实现:用于使专用寄存器的值移动到通用寄存器的指令或者从通用寄存器的低位连续地检测具有1的位置的指令。

[0136] 在上述示例中,仅执行第四算法之内的最内循环一次;然而,即使当数组A的大小(M)大于16时,每当专用寄存器的大小(K＝16位)被装满时将专用寄存器的值移动到通用寄存器以检查比较结果的附加信息。

[0137] 如上所述,通过使用根据第二示例的向量指令,用于使附加信息移动到通用寄存器的处理不是必需的。在最内循环中,基于比较结果的循环退出的检查变得不必要。

[0138] 从上述原因,根据第二示例的向量指令可有效地使用向量比较指令,从而提高了循环性能。此外,将向量比较的结果存储在专用寄存器中并且将用于插入数据的专用电路组装在专用寄存器中;因此,在比较指令的每个执行中用于更新专用寄存器的值的读取操作不是必需的并且可避免专用寄存器中的RAM危险。只有当检查专用寄存器的值是否是0时,专用寄存器的读取操作才成为必要。

[0139] 另一方面,当使用根据第二示例的向量指令时,检查K位的值,从而确定是否退出循环;因此,在上述与用于当使用根据比较示例的向量指令时利用标量指令一个接一个地比较字来确定是否退出循环的方法之间存在折衷。当要搜索的数组小或者相应索引小于K时,可更好地使用标量指令以更快地搜索索引。然而,当数组的大小较大或者要搜索的索引较大时,通过每K位进行比较的根据第二示例的向量指令可提高循环性能。

[0140] 根据第二示例的向量指令可使得用于从按照升序或降序排列的数组搜索超过一些边界的位置(索引)的算法加速。

[0141] 如上所述，尽管已经基于实施例和示例具体地描述了由本发明人等提出的发明，但是不用说地是本发明并不局限于上述，而是各种修改是可能的。

[0142] 例如，虽然已通过示例的方式描述了包括在半导体器件之中的CPU和存储器，但是存储器可以包括在与包括CPU的半导体器件不同的另一半导体器件之中。虽然已通过示例的方式描述了包括在CPU之中的向量运算单元，但是向量运算单元可以被提供在CPU之外。虽然已对32位宽度的专用寄存器进行了描述，但是可以是诸如16位宽度或者64位宽度这样的任何其它位宽度。虽然已对32位宽度的通用寄存器进行了描述，但是它可以是诸如16位宽度或者64位宽度这样的任何其它位宽度。虽然已对128位宽度的向量寄存器进行了描述，但是它可以是诸如64位宽度或256位宽度这样的任何其它位宽度。虽然已对向量运算单元的四个算术单元进行了描述，但是可以是诸如8个这样的任何其它数目的单元。

[0143] <实施例>

[0144] 如下附加了对实施例而言的附录。

[0145] (附录1)

[0146] 一种半导体器件，所述半导体器件包括能够执行向量指令的数据处理器，

[0147] 其中，所述数据处理器基于执行所述向量指令的运算结果产生附加信息，

[0148] 所述数据处理器包括附加信息存储寄存器，并且

[0149] 所述附加信息存储寄存器根据所述向量指令将用于表示所述附加信息的位组合并存储在通过对表示所述附加信息的位进行移位所引起的空的部分中。

[0150] (附录2)

[0151] 在如(附录1)中所公开的半导体器件中，

[0152] 所述附加信息存储寄存器存储用于表示通过由所述数据处理器执行若干次所产生的所述附加信息的位。

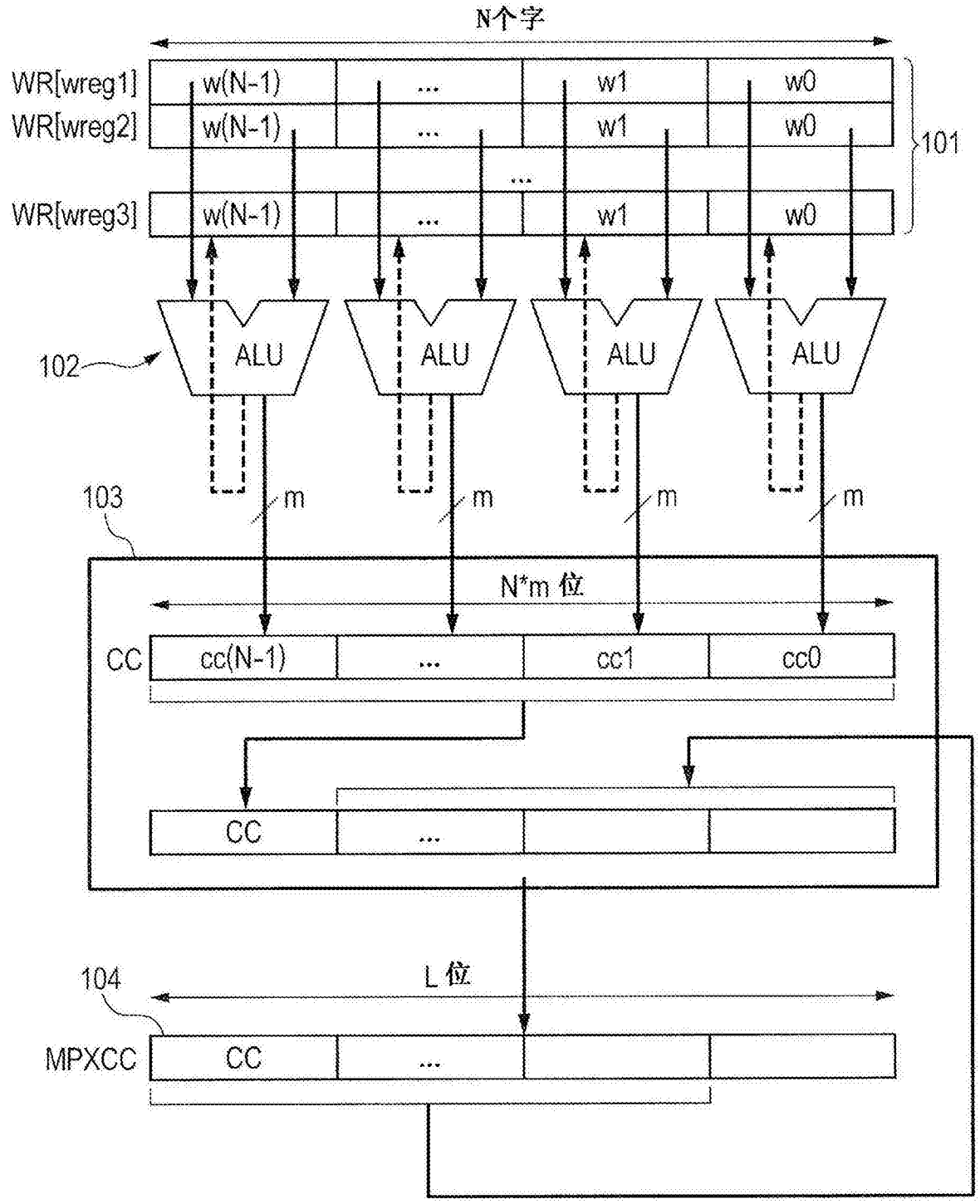

N个字
WR[wreg1]
WR[wreg2]
WR[wreg3]
w(N-1)
...
w1
w0
101
102
ALU
103
m
N*m 位
CC
cc(N-1)
cc1
cc0
104
L 位
MPXCC

图1

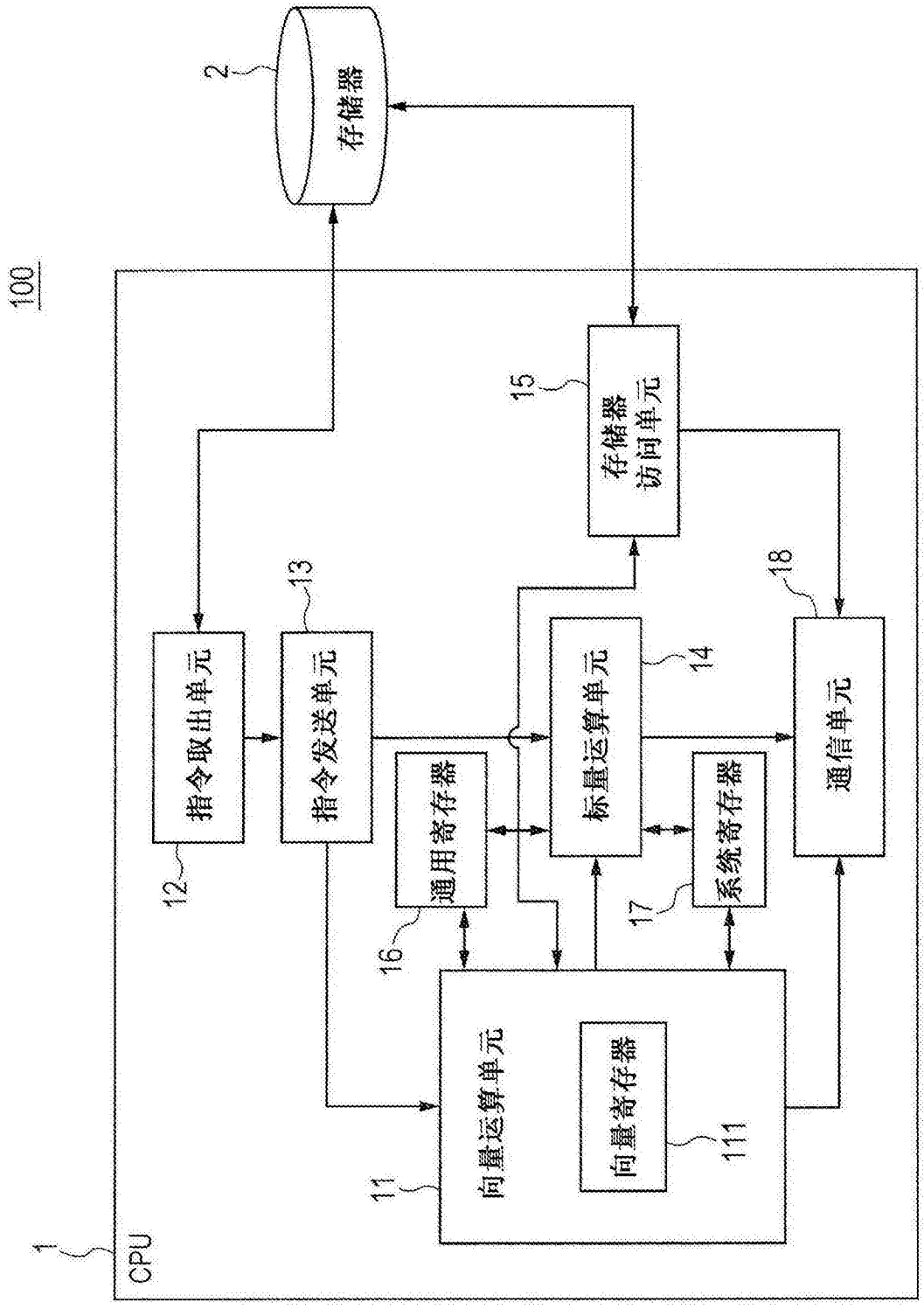

2
存储器
100
15
存储器访问单元
13
18
指令取出单元
指令发送单元
标量运算单元
14
通信单元
通用寄存器
系统寄存器
12
16
17
向量运算单元
向量寄存器
111
11
1
CPU

图2

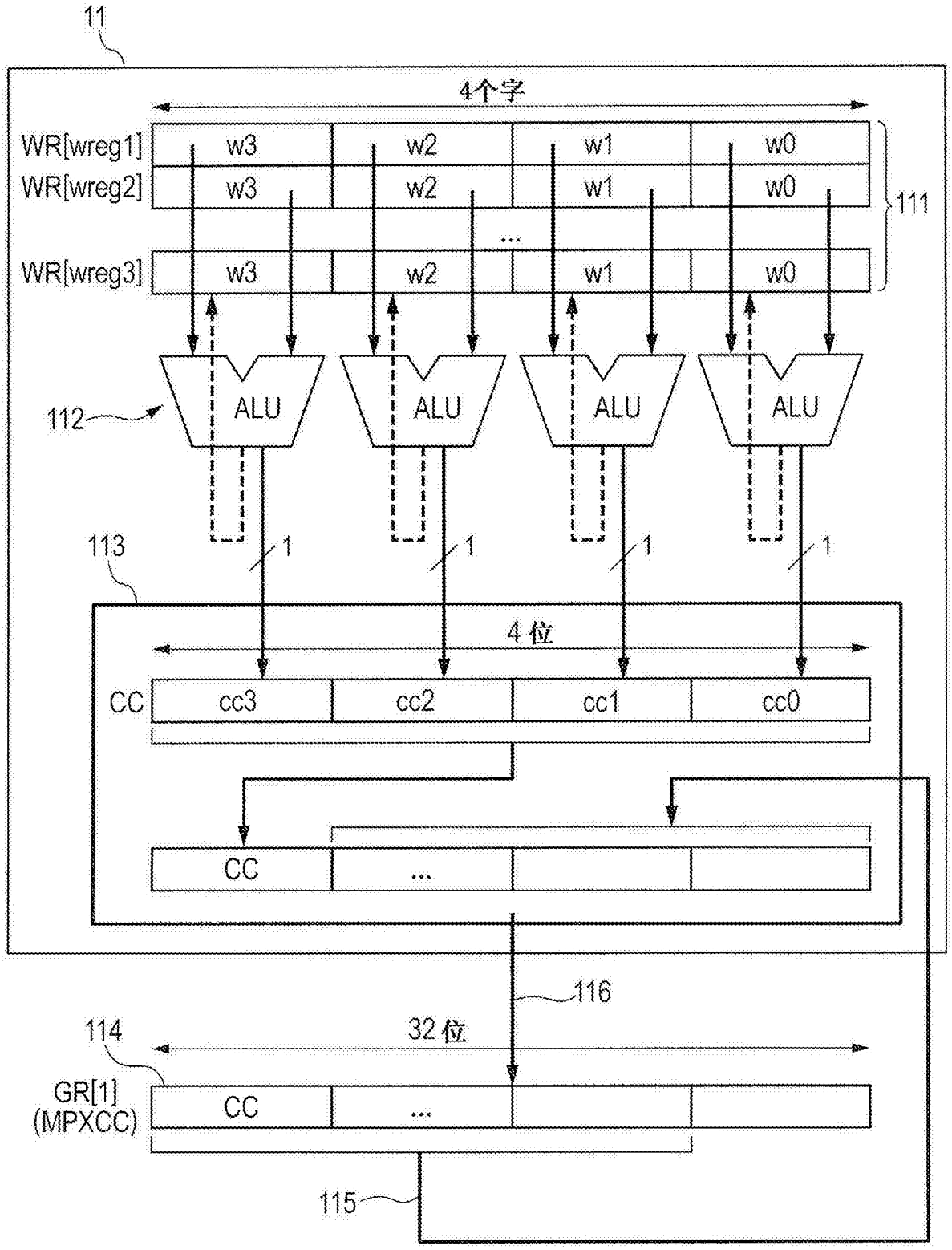

11
4个字
WR[wreg1]
WR[wreg2]
WR[wreg3]
w3
w2
w1
w0
111
112
ALU
113
1
4 位
CC
cc3
cc2
cc1
cc0
116
114
32 位
GR[1]
(MPXCC)
115

图3

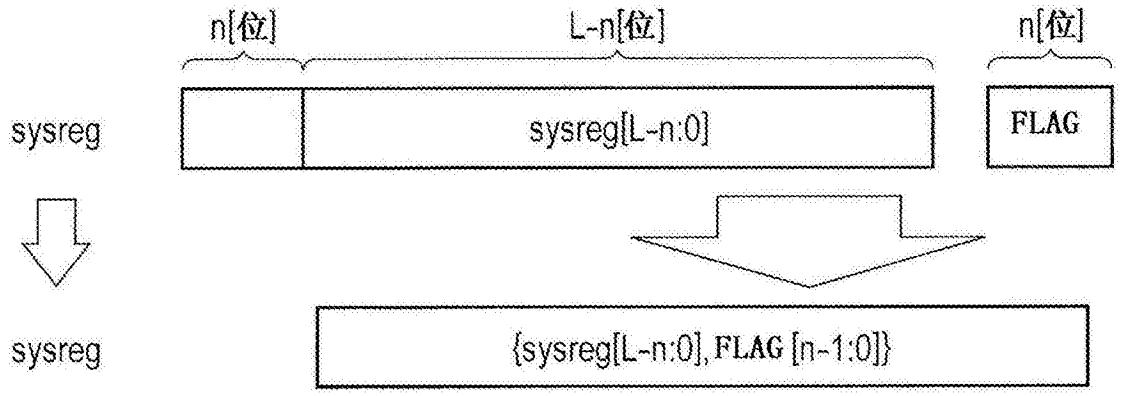

n[位]
L-n[位]
n[位]
sysreg
sysreg[L-n:0]
FLAG
sysreg
{sysreg[L-n:0], FLAG [n-1:0]}

图4

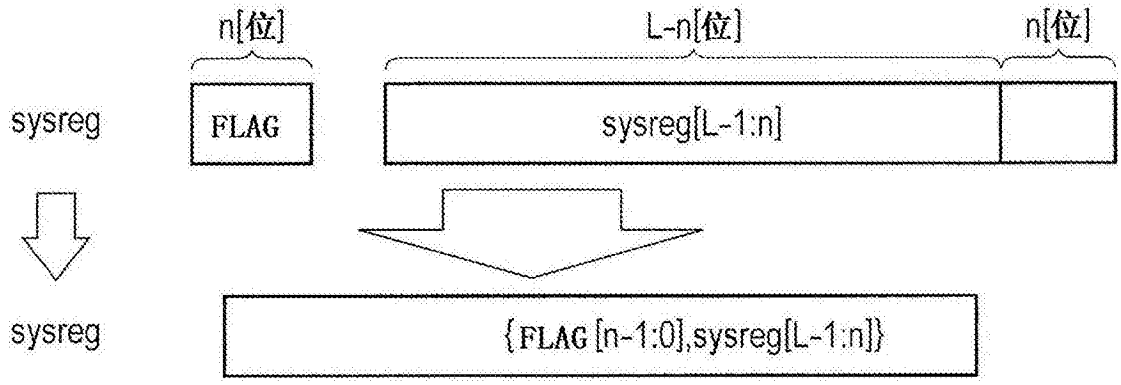

n[位]
L-n[位]
n[位]
sysreg
FLAG
sysreg[L-1:n]
sysreg
{FLAG [n-1:0],sysreg[L-1:n]}

图5

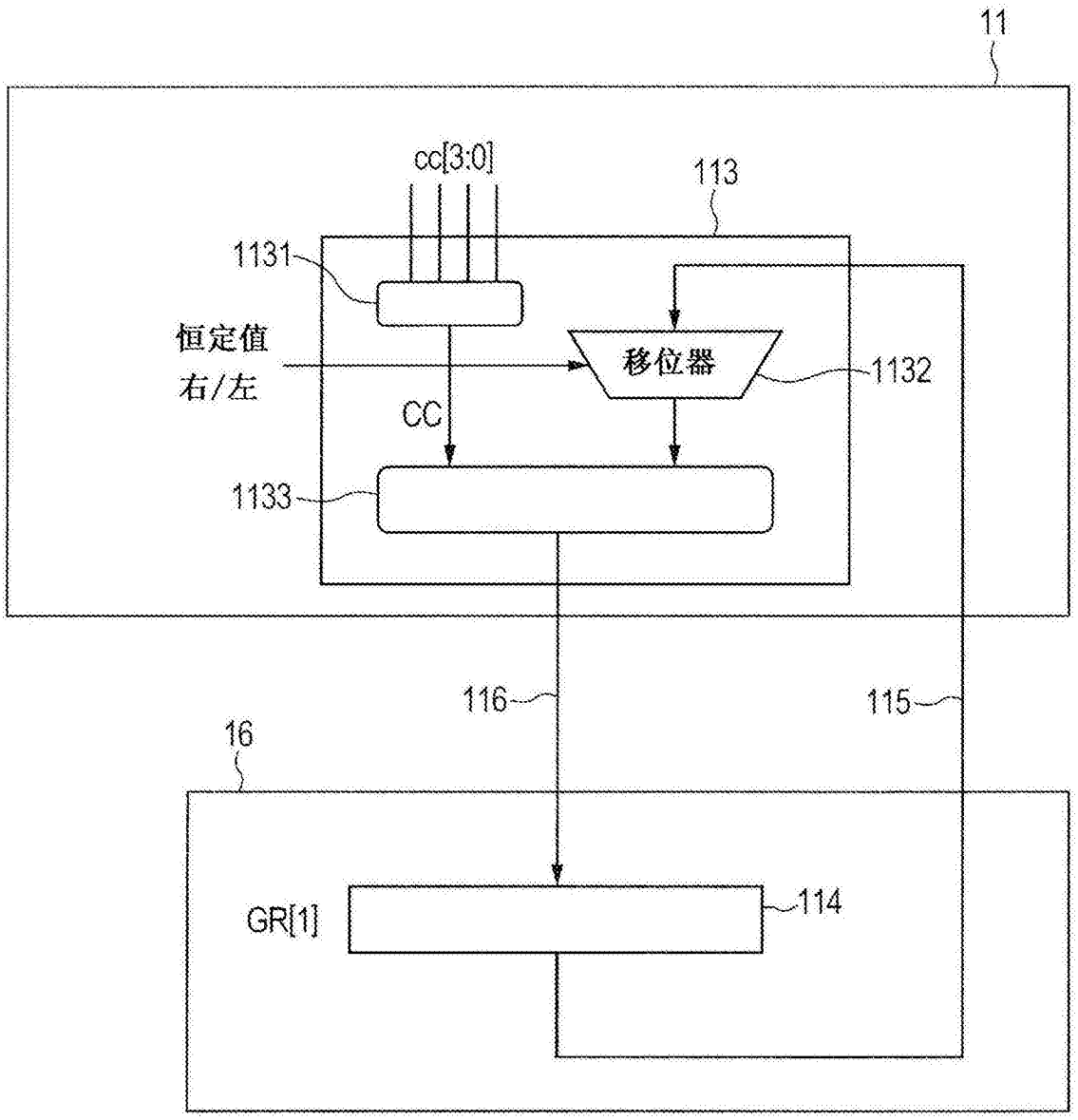


图6

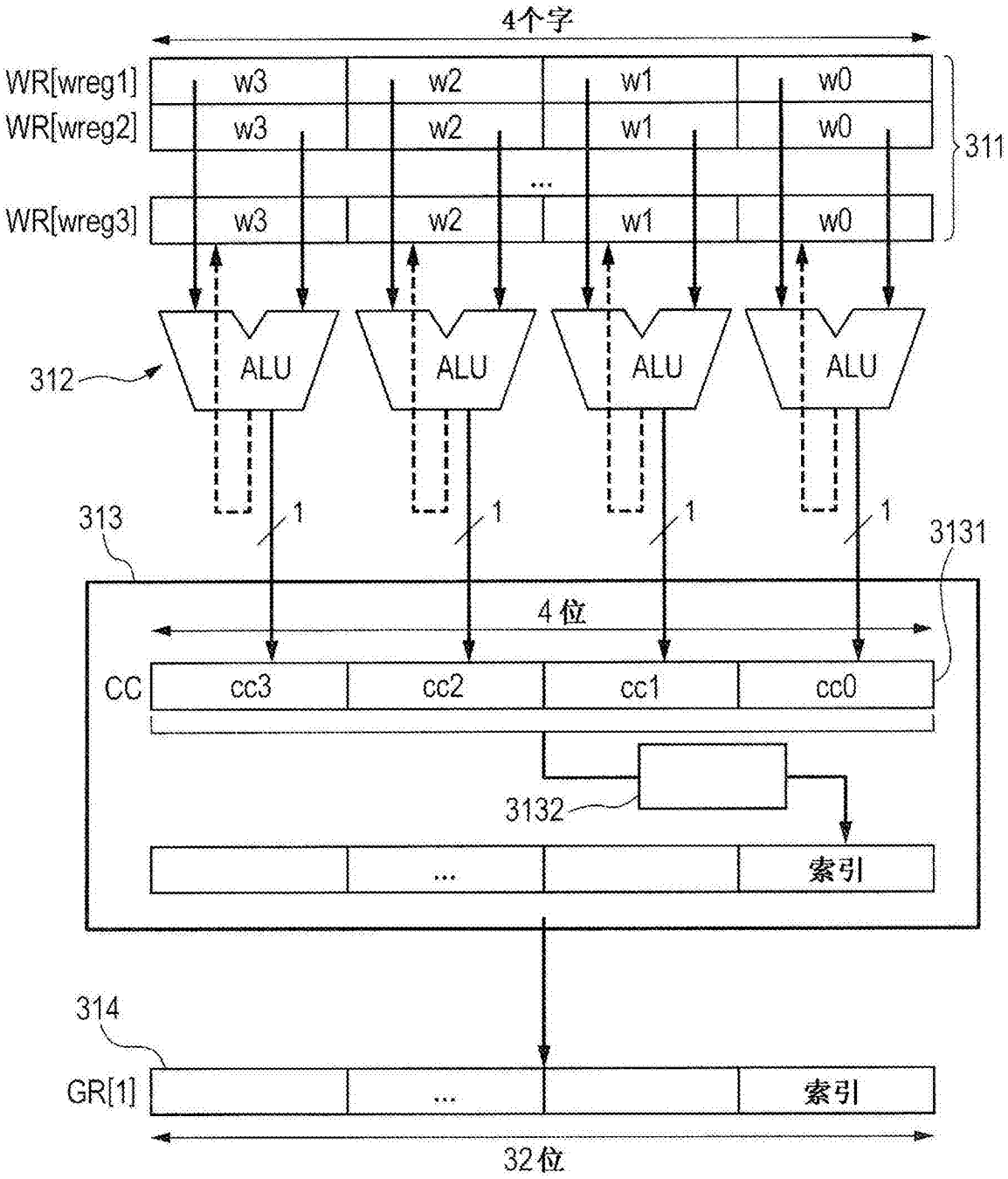

4个字
WR[wreg1]
WR[wreg2]
WR[wreg3]
w3
w2
w1
w0
311
312
ALU
313
1
3131
4位
CC
cc3
cc2
cc1
cc0
3132
索引
314
GR[1]
32位

图7

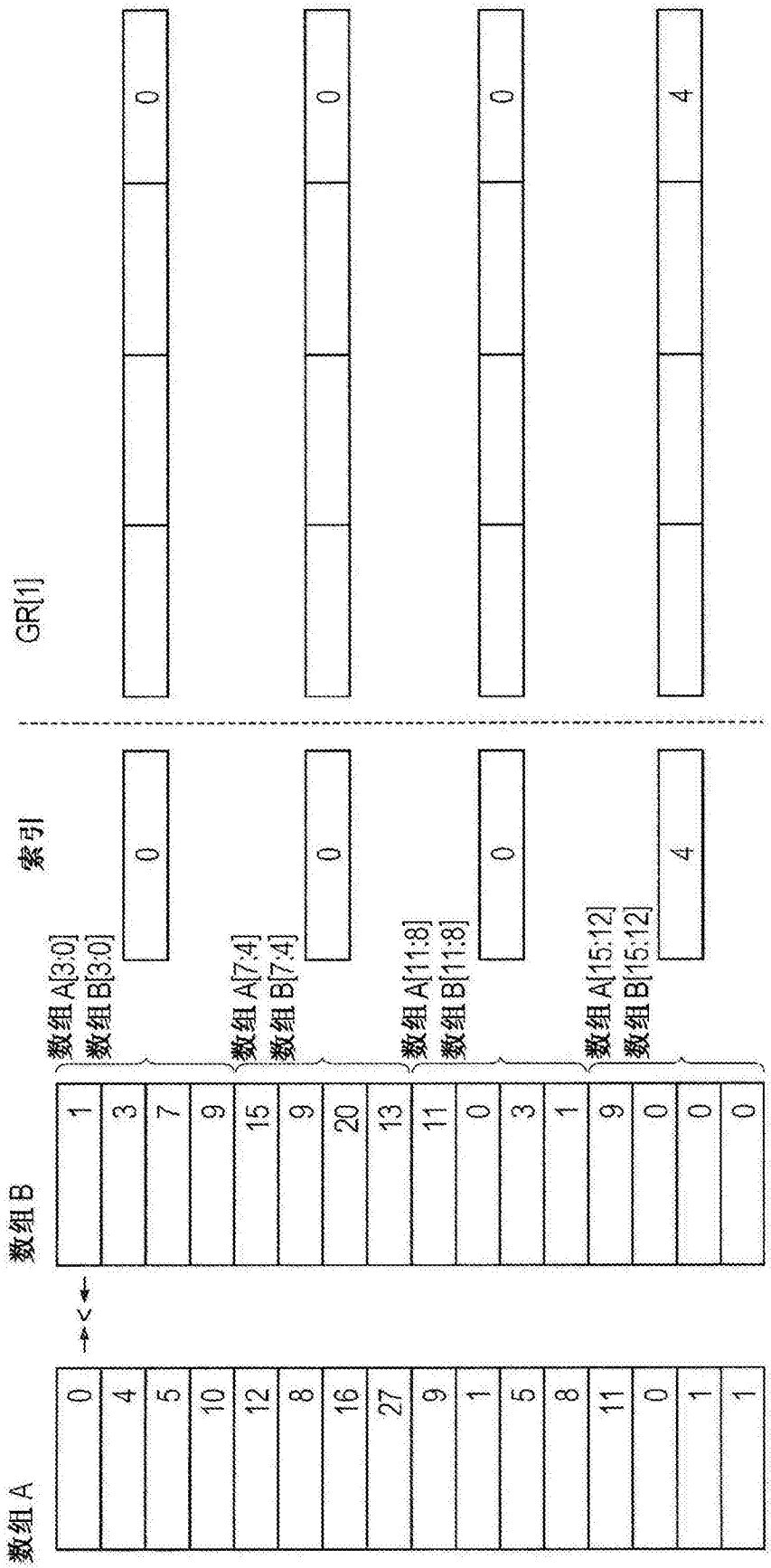

数组 A
数组 B
索引
GR[1]
数组 A[3:0]
数组 B[3:0]
数组 A[7:4]
数组 B[7:4]
数组 A[11:8]
数组 B[11:8]
数组 A[15:12]
数组 B[15:12]
0 4 5 10 12 8 16 27 9 1 5 8 11 0 1 1
1 3 7 9 15 9 20 13 11 0 3 1 9 0 0 0
0
0
0
4
0
0
0
4

图8

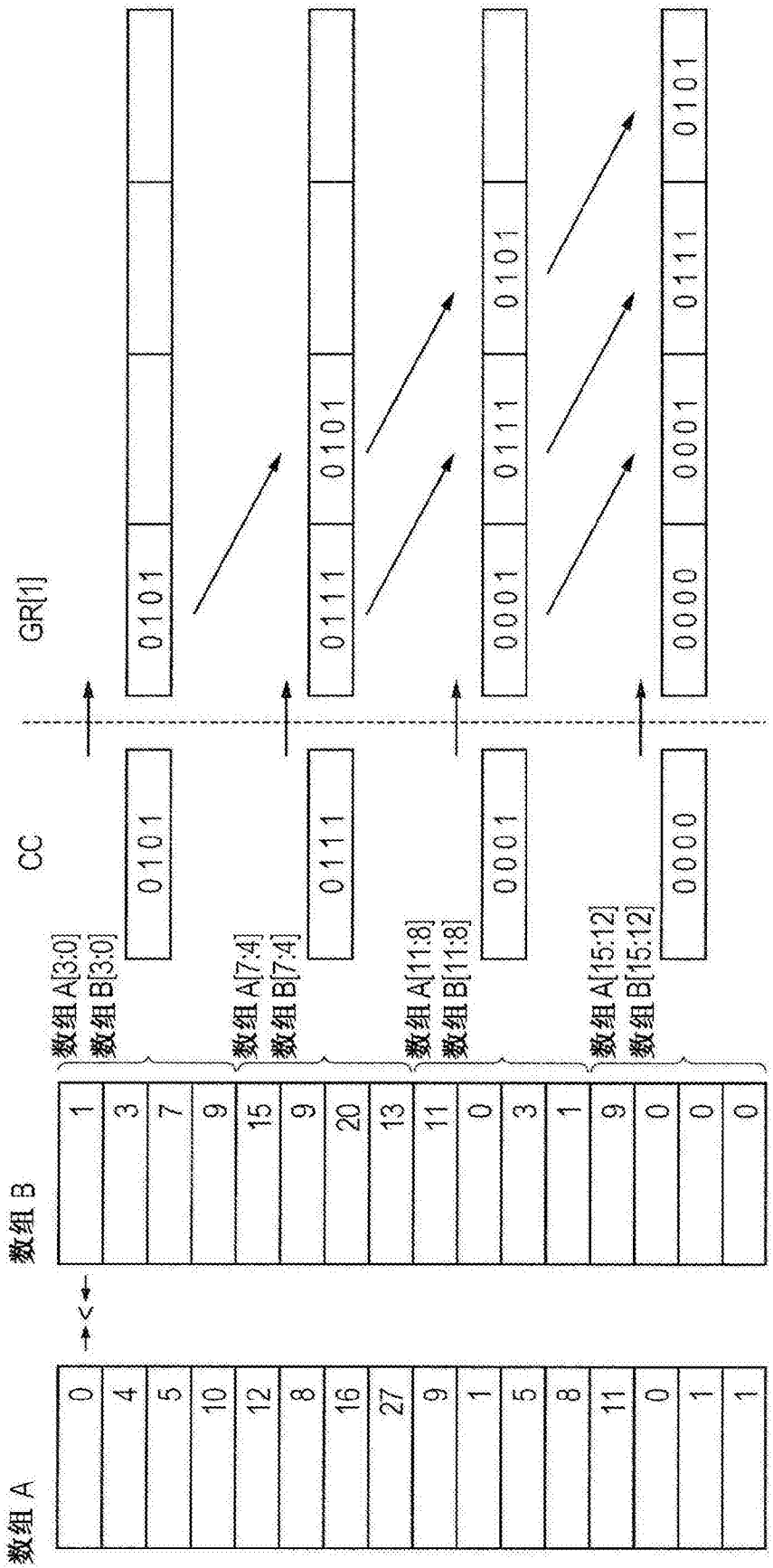

数组 A
数组 B
CC
GR[1]
数组 A[3:0]
数组 B[3:0]
数组 A[7:4]
数组 B[7:4]
数组 A[11:8]
数组 B[11:8]
数组 A[15:12]
数组 B[15:12]
0 4 5 10 12 8 16 27 9 1 5 8 11 0 1 1
1 3 7 9 15 9 20 13 11 0 3 1 9 0 0 0
0101
0111
0001
0000
0101
0111
0001
0000
0101
0111
0101

图9

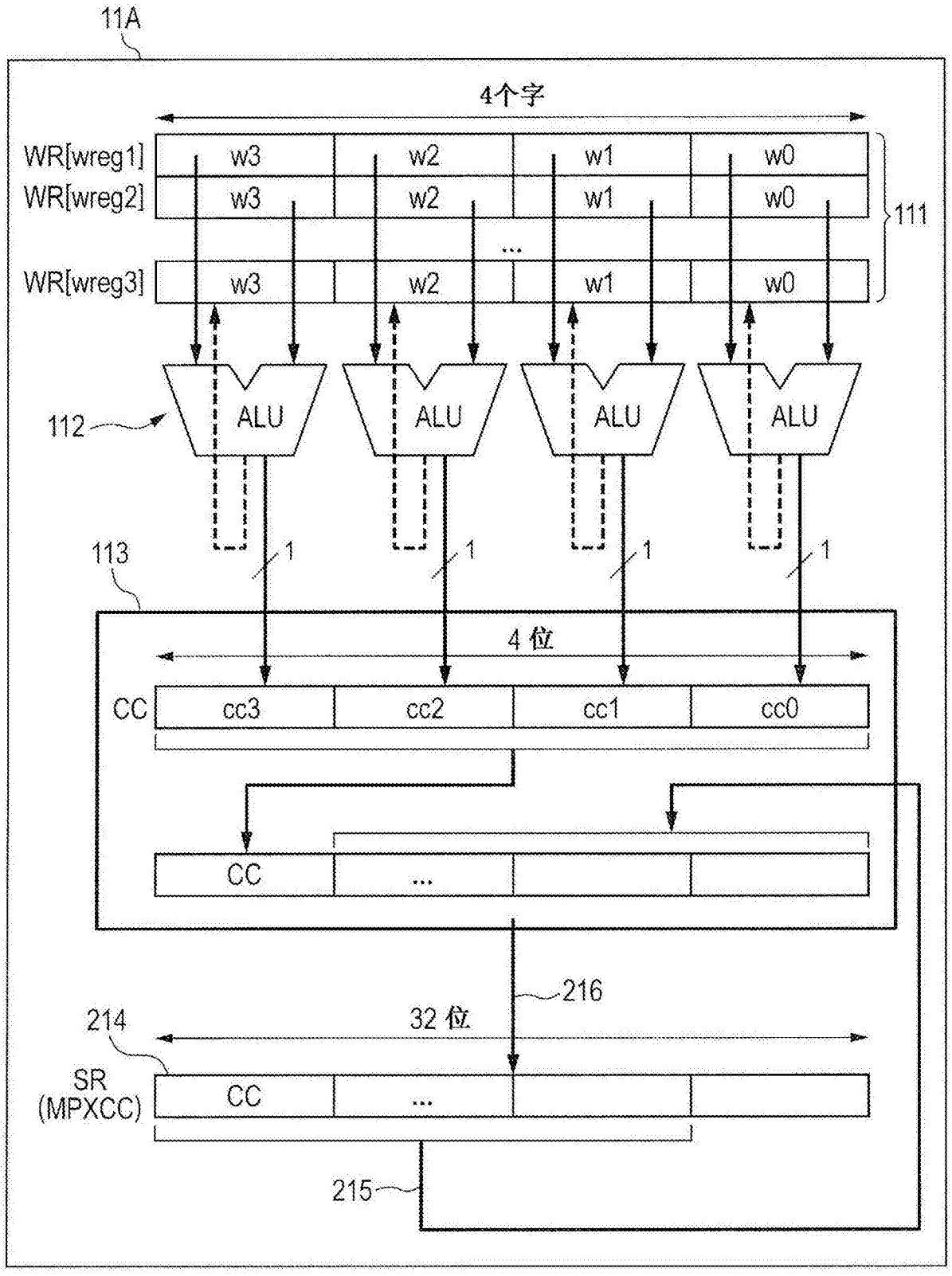

11A
4个字
WR[wreg1]
WR[wreg2]
WR[wreg3]
w3
w2
w1
w0
111
112
ALU
113
1
4 位
CC
cc3
cc2
cc1
cc0
216
214
32 位
SR
(MPXCC)
215

图10

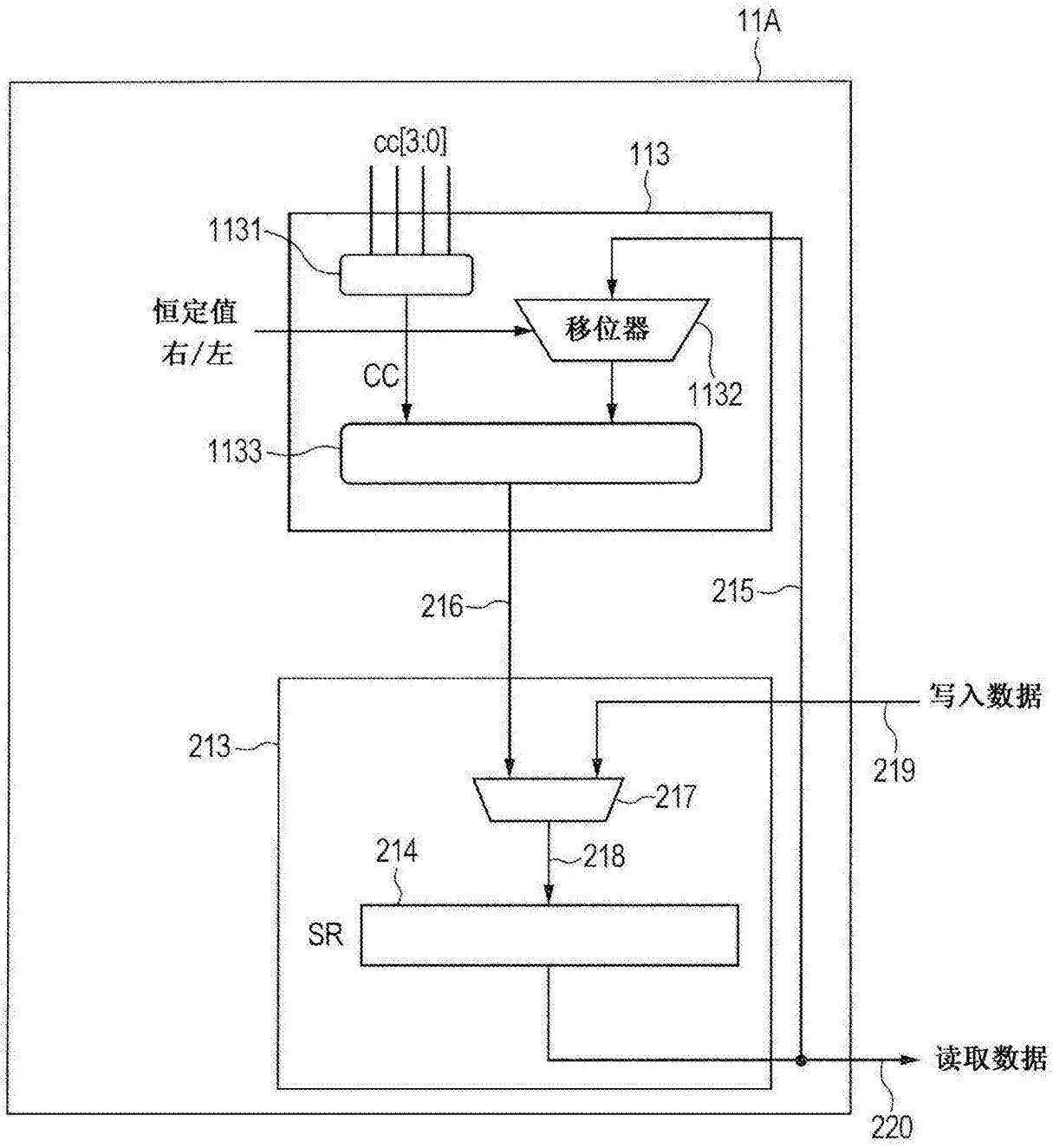

11A
cc[3:0]
113
1131
恒定值
右/左
移位器
CC
1132
1133
216
215
写入数据
213
219
217
214
218
SR
读取数据
220

图11

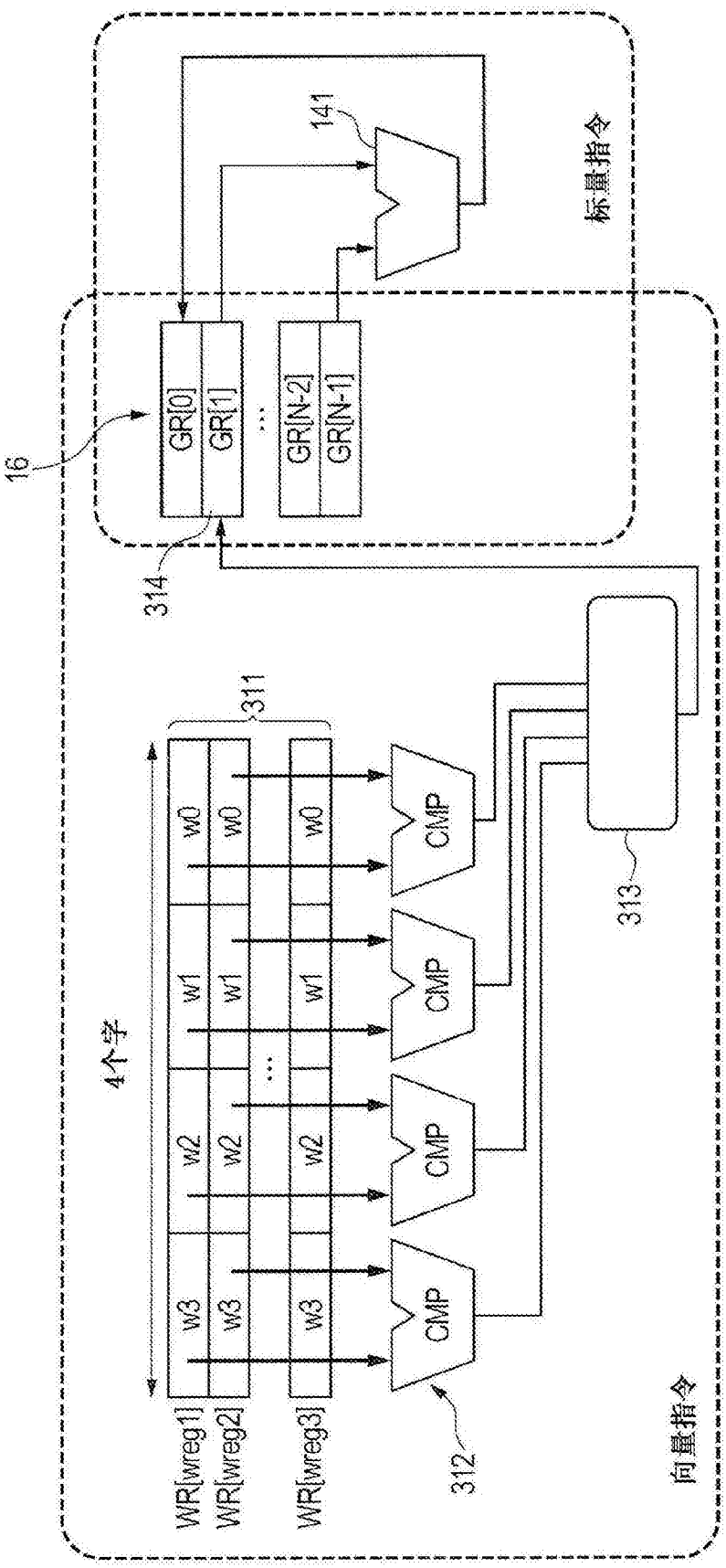

标量指令
141
16
GR[0]
GR[1]
GR[N-2]
GR[N-1]
314
311
4个字
w0
w1
w2
w3
CMP
313
312
WR[wreg1]
WR[wreg2]
WR[wreg3]
向量指令

图12

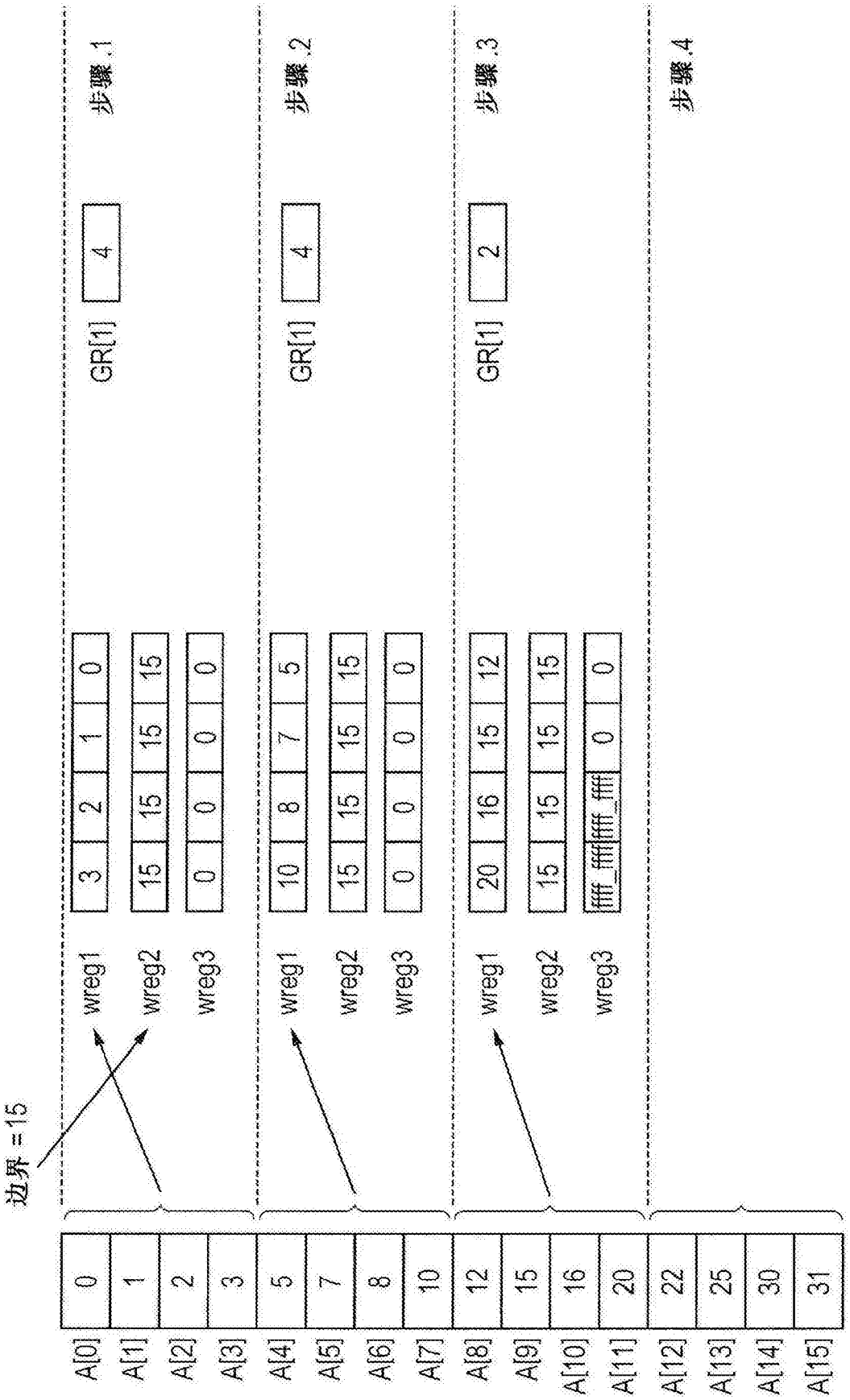

步骤.1
步骤.2
步骤.3
步骤.4
GR[1]
4
GR[1]
4
GR[1]
2
wreg1
wreg2
wreg3
wreg1
wreg2
wreg3
wreg1
wreg2
wreg3
3 2 1 0
15 15 15 15
0 0 0 0
10 8 7 5
15 15 15 15
0 0 0 0
20 16 15 12
15 15 15 15
ffff_ffff ffff_ffff 0 0
边界 =15
0 1 2 3 5 7 8 10 12 15 16 20 22 25 30 31
A[0] A[1] A[2] A[3] A[4] A[5] A[6] A[7] A[8] A[9] A[10] A[11] A[12] A[13] A[14] A[15]

图13

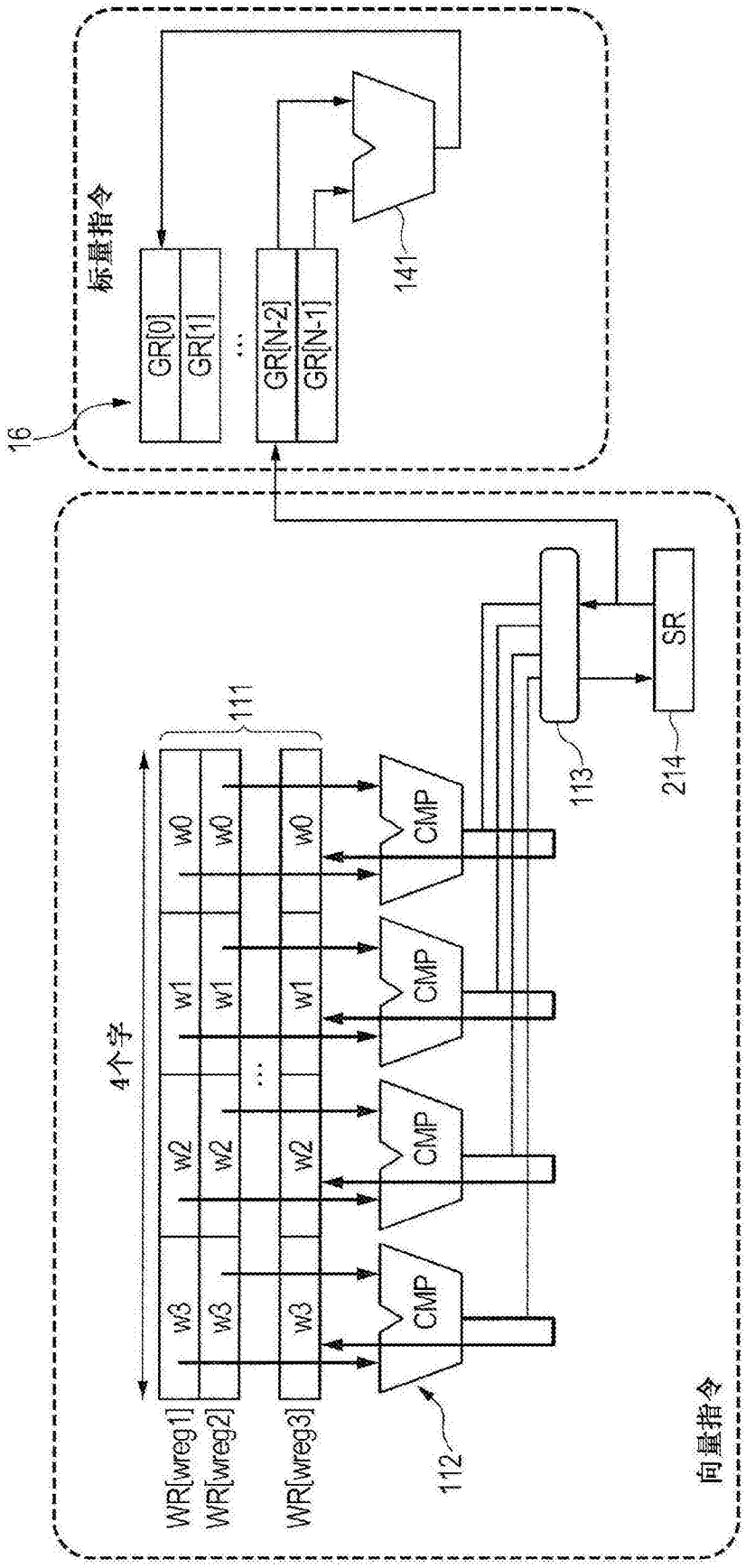

标量指令
GR[0]
GR[1]
GR[N-2]
GR[N-1]
141
16
向量指令
SR
214
113
111
4个字
w0
w1
w2
w3
CMP
112
WR[wreg1]
WR[wreg2]
WR[wreg3]

图14

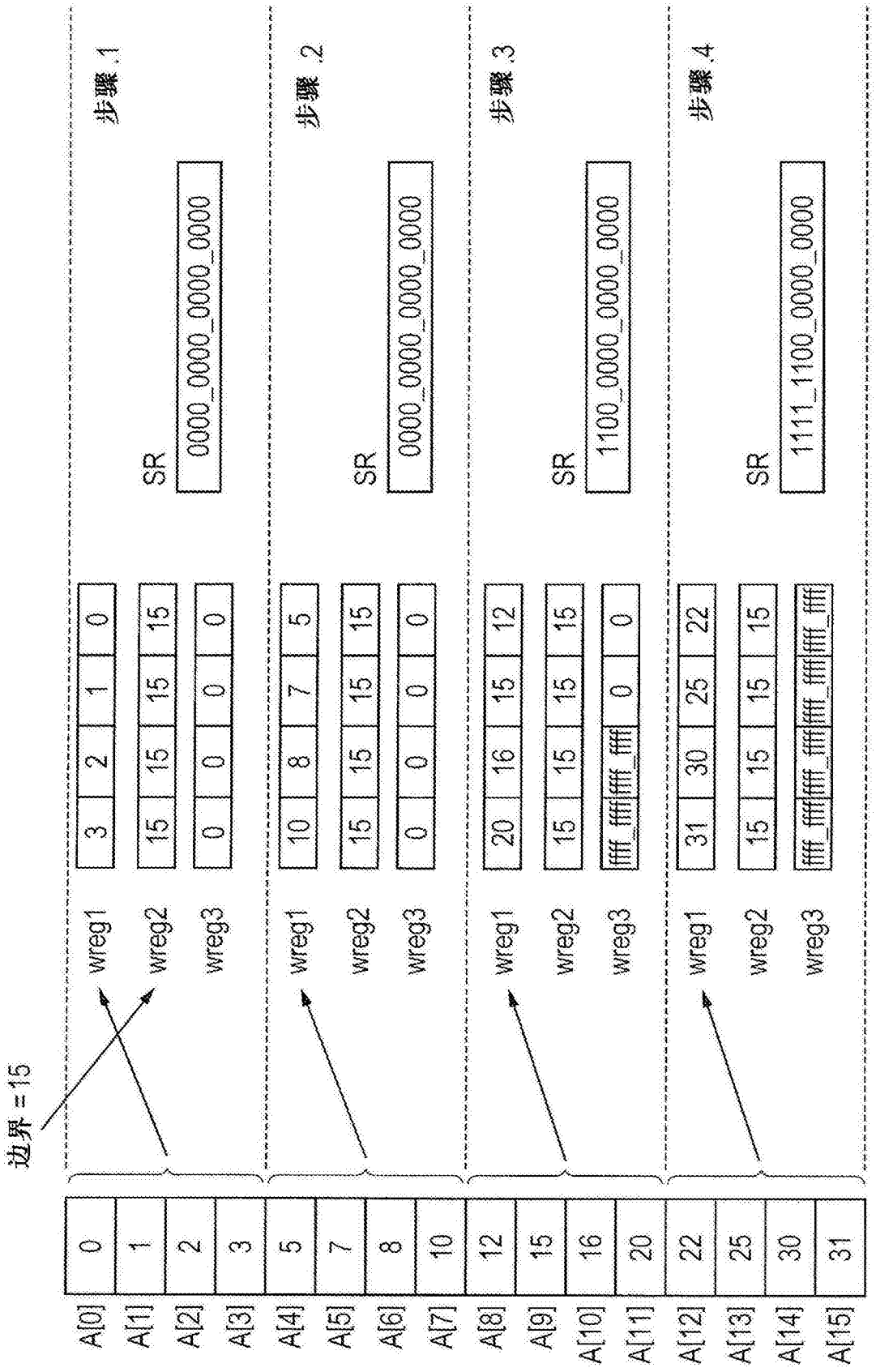

步骤.1
步骤.2
步骤.3
步骤.4
SR
0000_0000_0000_0000
0000_0000_0000_0000
1100_0000_0000_0000
1111_1100_0000_0000
边界=15
wreg1
wreg2
wreg3
A[0]
A[1]
A[2]
A[3]
A[4]
A[5]
A[6]
A[7]
A[8]
A[9]
A[10]
A[11]
A[12]
A[13]
A[14]
A[15]

图15