基础编译器 SIMD 用户编程手册-C 语言

V1.3.2

2020 年 9 月

目　　录

第一章 概述 ……… - 1 -
1.1 背景 ……… - 1 -
1.2 有关 SIMD 的几个基本概念 ……… - 1 -
1.2.1 标量与向量 ……… - 1 -
1.2.2 SW处理器主核支持的几种 SIMD 操作 ……… - 1 -
1.2.3 SW处理器从核支持的几种 SIMD 操作 ……… - 1 -
1.2.4 SIMD 的主要特点 ……… - 2 -
1.3 SW处理器主核、从核 SIMD 实现概述 ……… - 3 -
1.4 编译器对 C 语言的 SIMD 扩展 ……… - 3 -
1.5 SIMD 扩展的适用范围 ……… - 3 -
1.6 使用 SIMD 可以预期获得多大的性能提升 ……… - 3 -
第二章 快速入门 ……… - 5 -
2.1 两个简单的例子 ……… - 5 -
2.2 变量声明 ……… - 6 -
2.3 将标准类型的数据赋给扩展类型 ……… - 6 -
2.4 扩展类型变量的运算 ……… - 7 -
2.5 扩展类型的类型转换 ……… - 7 -
2.6 扩展类型的打印 ……… - 7 -
第三章 SIMD 对 C 语言的扩展 ……… - 8 -
3.1 主核数据类型的扩展 ……… - 8 -
3.2 从核数据类型的扩展 ……… - 8 -
3.3 主核对界（Host Alignment） ……… - 9 -
3.4 从核对界（Slave Alignment） ……… - 9 -
3.5 SIMD 对 C 操作的扩展（主核） ……… - 10 -
3.6 SIMD 对 C 操作的扩展（从核） ……… - 12 -
3.7 SIMD 对 C 内部函数的扩展（主核） ……… - 13 -
3.8 SIMD 对 C 内部函数的扩展（从核） ……… - 15 -
3.9 扩展类型的类型转换（主核） ……… - 17 -
3.10 扩展类型的类型转换（从核） ……… - 19 -
3.11 扩展类型与标准类型之间的数据交换（主核） ……… - 20 -
3.12 扩展类型与标准类型之间的数据交换（从核） ……… - 21 -
第四章 串行程序的向量化 ……… - 23 -
4.1 程序形式对向量化的限制 ……… - 23 -
4.1.1 相关性的限制 ……… - 23 -
4.1.2 循环结构的限制 ……… - 24 -
4.1.3 循环退出条件的限制 ……… - 24 -
4.2 循环向量化步骤 ……… - 24 -
4.2.1 循环展开和分裂 ……… - 25 -
4.2.2 变量替换 ……… - 26 -
4.2.3 操作替换 ……… - 26 -
4.3 如何更好地向量化 ……… - 26 -
4.3.1 合理使用数组 ……… - 26 -

4.3.2 对函数调用的处理 - 27 -
4.3.3 更有效的使用 Cache - 27 -
4.3.4 对界问题的处理 - 29 -
第五章 C 内部函数接口 - 33 -
5.1 使用的符号说明 - 33 -
5.2 主核内部函数接口 - 33 -
5.2.1 装入/存储函数接口 - 33 -
5.2.2 定点向量运算函数接口 - 38 -
5.2.3 浮点向量运算函数接口 - 52 -
5.2.4 数据整理函数接口 - 61 -
5.2.5 赋值函数接口 - 65 -
5.2.6 打印函数接口 - 67 -
5.2.7 向量归约接口 - 71 -
5.3 从核内部函数接口 - 73 -
5.3.1 装入/存储函数接口 - 73 -
5.3.2 定点向量运算函数接口 - 77 -
5.3.3 浮点向量运算函数接口 - 87 -
5.3.4 数据整理函数接口 - 104 -
5.3.5 赋值函数接口 - 110 -
5.3.6 打印函数接口 - 113 -
5.3.7 向量归约接口 - 118 -
修订记录 - 120 -

第一章 概述

1.1 背景

SW处理器主核支持 256 位 SIMD 扩展指令，从核支持 512 位 SIMD 扩展指令，该功能的增加，不仅能够降低功耗，而且在一定程度上提升了指令级并行的能力。

1.2 有关 SIMD 的几个基本概念

SIMD（Single Instruction Multiple Data）是单指令流多数据流的缩写，SW处理器主核支持的 SIMD 处理长度为 256 位，从核支持的 SIMD 处理长度为 512 位。

1.2.1 标量与向量

标量：运算粒度为单个元素

向量：运算粒度为一组有序的标量

1.2.2 SW处理器主核支持的几种 SIMD 操作

a)64*4 的双精度浮点运算：

即一次操作处理 4 个双精度浮点运算

b)64*4 的单精度浮点运算

即一次操作处理 4 个单精度浮点运算

c) 32*8 的定点运算

即一次操作处理 8 个 32 位的定点运算

d) 256*1 的定点运算

即一次操作处理 1 个 256 位的长整型运算

该运算可在部分情况下支持一次处理 4 个 64 位的长整形运算

1.2.3 SW处理器从核支持的几种 SIMD 操作

a)64*8 的双精度浮点运算：

即一次操作处理 8 个双精度浮点运算

b)64*8 的单精度浮点运算

即一次操作处理 8 个单精度浮点运算

c) 32*16 的定点运算

即一次操作处理 16 个 32 位的定点运算

d) 512*1 的定点运算

即一次操作处理 1 个 512 位的长整型运算

该运算可在部分情况下支持一次处理 8 个 64 位的长整形运算

e)16*32 的半精度浮点运算

即一次操作处理 32 个 16 位的半精度浮点运算

1.2.4 SIMD 的主要特点

我们知道，一个短向量中各个元素是互不相关的，对当前向量元素的操作结果不影响到其他向量元素。比如有一个数组，我们要使数组中的每一个元素 Ai 都乘以一个标量 b，那么 b 与 A1 相乘的结果不影响 A2 的结果，各自独立。

一条 SIMD 指令相当于一个小的循环，所以可以减少指令数，从而可以降低对指令访问带宽的要求。并且减少了由循环引起的控制相关。

例如主核：

```
int a (N),b (N),c (N)
for(i=0; i < N; i++) {
      a[i]=b[i]+c[i]
}
```

如果采用 SIMD 技术，则最终的实现相当于：

```
for(i=0; i<N; i=i+8){
      a[i+7:i]=b[i+7:i]+c[i+7:i]
}
```

例如从核：

```
int a (N),b (N),c (N)
for(i=0; i < N; i++) {
      a[i]=b[i]+c[i]
}
```

如果采用 SIMD 技术，则最终的实现相当于：

```
for(i=0; i<N; i=i+16){
      a[i+15:i]=b[i+15:i]+c[i+15:i]
}
```

可以发现，主核循环迭代次数减少为原来的八分之一，该循环的总指令数减少到原来的八分之一左右；从核循环迭代次数减少为原来的十六分之一，该循环的总指令数减少到原来的十六分之一左右，都提高了单机的效率。

然而，SW处理器主核上的 SIMD 需要满足以下一些要求：

a) 仅在内存中连续的数据可以被取入向量寄存器进行向量运算；

b) 数据在内存中要求 32 字节对界（64*4 单精度浮点要求 16 字节对界），不对界的向量访存会引起异常，然后由操作系统模拟，性能上有很大的降低；

SW处理器从核上的 SIMD 需要满足以下一些要求：

a) 仅在内存中连续的数据可以被取入向量寄存器进行向量运算；

b) 数据在内存中要求 64 字节对界（64*8 单精度浮点要求 32 字节对界），不对界的向量访存会引起异常，从核会直接报错退出。

1.3 SW处理器主核、从核 SIMD 实现概述

SW处理器主核与从核支持的 SIMD 功能，借鉴了 SIMD 技术及专用机设计理念——主核：首先把浮点流水线设计成为单指令流多数据流部件（SIMD 部件），以增强其浮点运算能力，同时增加 256 位专用部件流水线，提高整数处理能力，增强了对实际应用的能力。

从核：首先把浮点流水线设计成为单指令流多数据流部件（SIMD 部件），以增强其浮点运算能力，同时增加 512 位专用部件流水线，提高整数处理能力，增强了对实际应用的能力。

SW处理器主从核各提供了一百余条指令来实现 SIMD 的功能。用户可以直接通过汇编调用这些指令，但十分不方便。为了用户更好的使用 SIMD，我们提供了一整套方案，通过直接定义数据类型、调用内部函数，或者通过编译器自动的向量识别，来满足用户的需求。

1.4 编译器对 C 语言的 SIMD 扩展

我们在 SW处理器的配套编译器 SWGCC 上实现了一整套 SIMD 支持，针对C 语言扩展了一系列 SIMD 数据类型和函数接口，通过在 C 语言中使用这些数据类型和接口进行编程，用户就相当于直接使用了对应的 SIMD 功能部件。

扩展的函数接口与 SIMD 指令一一对应，在编译的时候由 SWGCC 编译器 inline 进来，使用这些函数接口编程可以在 C 语言一级获得与汇编编程一样的性能。

1.5 SIMD 扩展的适用范围

SIMD 解决的是 SW处理器主核与从核上的短向量问题，属于串行程序的向量化的范畴。我们知道，神威 E 级高性能计算机的并行语言用户可以使用 MPI、OPENACC等并行语言进行编程，在这些并行程序中也可以使用 SIMD 扩展编程，并且最终这些扩展将交由 SWGCC 编译器来处理。

1.6 使用 SIMD 可以预期获得多大的性能提升

SIMD 对性能的提升可由程序本身的特性和程序编写技巧等多方面因素决定。对于完全 SIMD 向量化的程序，对主核来说，32*8 的向量运算性能可以达到标量的 8 倍，64*4 的向量运算性能可以达到标量的 4 倍，如果浮点使用向量乘加部件，性能还可以提高到 8 倍。对从核来说，32*16 的向量运算性能可以达到标量的 16 倍，64*8 的向量运算性能可以达到标

量的 8 倍，16*32 的半精度浮点向量运算性能可以达到标量的 32 倍，如果半精度浮点使用向量乘加部件，性能还可以提高到 64 倍。

第二章 快速入门

本章以 C 语言为例，给出一个简单的 SIMD 程序，用来说明语言的某些基本特征。本章没有详尽地描述SIMD的所有特征，目的在于让你对 SIMD 编程先有一个大致的概念。

2.1 两个简单的例子

例 1．下面是用 C 语言编写的一个 SIMD 程序（主核）:

```
#include <simd.h>            //包含 SIMD 头文件
main()
{
      int arr[8] __attribute__ ((aligned (32)))= {1,2,3,4,5,6,7,8};
      int i,res[8],t=0;
      intv8 va,vb,vi;
      simd_load (va,arr);
      for (i=16;i>=1;i>>=1) {
               vi = simd_set_intv8(i,i,i,i,i,i,i,i);     //从标准类型向扩展类型赋值
               va ^= simd_vsraw(va,i); //扩展类型的变量使用运算符号
      }
      vb=simd_veqvw(va,vi); //扩展类型的变量使用扩展的内部函数接口
      simd_print_intv8(vb);     //用 intv8 类型的格式打印
      simd_print_intv8(va);     //用 intv8 类型的格式打印
      simd_store(va, res);        //intv8 类型的存储
      for (i=0; i<8; i++)
               t =t+ res[i];
      printf("%d\n",t);
}
```

例 2．下面是用 C 语言编写的一个 SIMD 程序（从核）:

```
#include <simd.h>            //包含 SIMD 头文件
main()
{
      int arr[16] __attribute__ ((aligned (64)))= {1,2,3,4,5,6,7,8,10,11,12,13,14,15};
      int i,res[16],t=0;
      intv16 va,vb,vi;
      simd_load (va,arr);
      for (i=16;i>=1;i>>=1) {
               vi = simd_set_intv16(i,i,i,i,i,i,i,i,i,i,i,i,i,i,i,i);   //从标准类型向扩展类型赋值
               va ^= simd_vsraw(va,i); //扩展类型的变量使用运算符号
```

```
        }
        double f=1.0;
        vb=simd_vconw (va,vi,f); //扩展类型的变量使用扩展的内部函数接口
        simd_print_intv16(vb);     //用 intv16 类型的格式打印
        simd_print_intv16(va);     //用 intv16 类型的格式打印
        simd_store(va, res);        //intv16 类型的存储
        for (i=0; i<16; i++)
                t =t+ res[i];
        printf("%d\n",t);
    }
```

2.2 变量声明

在 SIMD 程序中，变量的声明方式与 C 中是一致的，只是在标准 C 的基础上扩展了一些基本类型。

主核：上面的例 1 中使用了:

intv8 vi；

声明了一个 intv8 类型的 SIMD 变量，表示 vi 是 8 个 32 位整型组成的类型变量。

vi:

255 224	223 192	191 160	159 128	127 96	95 64	63 32	31 0

从核：上面的例 2 中使用了:

intv16 vi；

声明了一个 intv16 类型的 SIMD 变量，表示 vi 是 16 个 32 位整型组成的类型变量。

vi:

511 480	479 448	447 416	415 384	383 352	351 320	319 288	287 256	255 224	223 192	191 160	159 128	127 96	95 64	63 32	31 0

2.3 将标准类型的数据赋给扩展类型

SWGCC 扩展了一些内部函数用于将标准类型的数据赋给扩展类型。

主核：上面的例 1 中：

vi = simd_set_intv8 (i, i, i, i, i, i, i, i);

表示将 8 个 int 类型的数赋给变量 vi，这个语句执行后，vi 中的值为：

vi:

i	i	i	i	i	i	i	i
255 224	223 192	191 160	159 128	127 96	95 64	63 32	31 0

从核：上面的例 2 中：

vi = simd_set_intv16 (i, i, i, i, i, i, i, i, i, i, i, i, i, i, i, i);

表示将 16 个 int 类型的数赋给变量 vi，这个语句执行后，vi 中的值为：

vi:

i	i	i	i	i	i	i	i	i	i	i	i	i	i	i	i
511 480	479 448	447 416	415 384	383 352	351 320	319 288	287 256	255 224	223 192	191 160	159 128	127 96	95 64	63 32	31 0

2.4 扩展类型变量的运算

SWGCC 还对运算符进行了扩展，可以使用 C 语言中的一些运算符对扩展类型的变量进行运算。

如主核：

va ^= va>>1;

对 va 中的 8 个整型分别右移 1 位后再分别与原来 va 中的 8 个整型进行异或运算。va 进行的右移操作可以同时完成 8 个 32 位整型的移位操作。

有些运算不能用运算符表示，只能使用扩展的内部函数。

例如例子中的“等效”运算：

vb = simd_veqvw(va,vi);

它表示：将 va 和 vi 中 8 个 32 位字分别对应按位逻辑等效。

如从核：

va ^= va>>1;

对 va 中的 16 个整型分别右移 1 位后再分别与原来 va 中的 16 个整型进行异或运算。va 进行的右移操作可以同时完成 16 个 32 位整型的移位操作。

有些运算不能用运算符表示，只能使用扩展的内部函数。

例如例子中的“拼接”运算：

vb = simd_vconw(va,vi,f);

它表示：将 va 和 vi 中 16 个 32 位字分别按 f 的值进行高低位拼接。

2.5 扩展类型的类型转换

在 SWGCC 中支持几种扩展类型的转换，SWGCC 支持的几种类型转换和它们各自的用法在后面的章节中有详细的介绍。

2.6 扩展类型的打印

SWGCC 为每一种扩展的数据类型扩展了四个内部函数用于打印，例如 intv8 类型的四个打印函数为：simd_print_intv8(intv8)、simd_fprint_intv8(FILE *, intv8)、simd_print_intv8_X(intv8)、simd_fprint_intv8_X(FILE *, intv8)。

上面的例 1 中：

```
simd_print_intv8(va);    //用 intv8 类型的格式打印
```

输出如下：

```
[ 15, 5, 4, 6, 7, 2, 3, 1 ]
```

第三章 SIMD 对 C 语言的扩展

本章主要介绍 SWGCC 编译器针对 C 语言进行的 SIMD 扩展。

3.1 主核数据类型的扩展

在标准 C 的基础上扩展了 6 种数据类型：

- intv8：32*8 有符号整型
- uintv8：32*8 无符号整型
- int256：256*1 有符号长整型
- uint256：256*1 无符号长整型
- floatv4：64*4 单精度浮点
- doublev4：64*4 双精度浮点

表 3-1 扩展数据类型

扩展类型	说明	值的范围
intv8	8 个 int	$-2^{31}…2^{31}-1$
uintv8	8 个 unsigned int	$0…2^{32}-1$
int256	256 位有符号长整型	—
uint256	256 位有符号长整型	—
floatv4	4 个 float	1.17549435e-38…3.40232847e38
doublev4	4 个 double	2.2250738585072013e-308 …1.7976931328623158e308

3.2 从核数据类型的扩展

在标准 C 的基础上扩展了 7 种数据类型：

- intv16：32*16 有符号整型
- uintv16：32*16 无符号整型
- int512：512*1 有符号长整型
- uint512：512*1 无符号长整型
- floatv8：64*8 单精度浮点
- doublev8：64*8 双精度浮点
- float16v32：16*32 半精度浮点

表 3-2 扩展数据类型

扩展类型	说明	值的范围
intv16	16 个 int	$-2^{31}…2^{31}-1$
uintv16	16 个 unsigned int	$0…2^{32}-1$
int512	512 位有符号长整型	—
uint512	512 位无符号长整型	—

floatv8	8 个 float	1.17549435e-38…3.40232847e38
doublev8	8 个 double	2.2250738585072013e-308 …1.7976931328623158e308
float16v32	32 个_Float16	6.103515625e-5…65504

3.3 主核对界（Host Alignment）

本节描述主核 SIMD 的对界要求，当使用 SIMD 数据进行编程的时候，用户必须清楚这些对界问题，因为编译器不会在对界方面产生报警或错误信息，向量类型的数据在使用过程中是否出现了不对界问题需要用户判断。

主核向量类型的数据项在内存中除了 floatv4 类型是 16 字节对界，其余都是 32 字节对界。编译器保证用向量类型定义的变量在内存中是 32 字节（16 字节）对界的。包含主核向量类型的结构体（struct）/联合（union），编译器保证它们是 32 字节（或 16 字节）对界的（必要的时候在内部进行了垫塞）。

例如：

struct st { doublev4 vb; int a; }

sizeof (struct st)返回值是 64。

在 SW处理器主核架构中，不对界的 Load/Store 会引发异常，操作系统收到异常信号后就会将这些 Load/Store 拆分成标准类型的 Load/Store，从而大大降低性能。因此在向量编程时，编译程序一般需要添加-faddress_align=n 选项来保证所有类型数组、结构体、联合体等的首地址是 n 字节对界。一般情况下，主核向量程序需要设置 n=32，但是对于主从混合程序，为保证主从程序对界一致，编译主从程序时都需要添加-faddress_align=64 选项，保证所有类型数组、结构体、联合体等的首地址是 64 字节对界的。

C 库中的 malloc 函数分配的空间首地址为 16 字节对界，为满足主核向量对界要求并与从核向量对界一致，新增 libc_aligned_malloc 函数，该函数分配的空间首地址为 64 字节对界，需配合 libc_aligned_free 函数使用。

3.4 从核对界（Slave Alignment）

本节描述从核 SIMD 的对界要求，当使用 SIMD 数据进行编程的时候，用户必须清楚这些对界问题，因为编译器不会在对界方面产生报警或错误信息，向量类型的数据在使用过程中是否出现了不对界问题需要用户判断。

从核向量类型的数据项在内存中除了 floatv8 类型是 32 字节对界，其余都是 64 字节对界。编译器保证用从核向量类型定义的变量在内存中是 64 字节（32 字节）对界的。包含从核向量类型的结构体（struct）/联合（union），编译器保证它们是 64 字节（或 32 字节）对界的（必要的时候在内部进行了垫塞）。

例如：

struct st { doublev8 vb; int a; }

sizeof (struct st)返回值是 128。

在 SW处理器从核构架中，不对界的 Load/Store 会引发不对界访存异常，程

序会直接退出。因此在从核向量编程时，编译程序一般需要添加-faddress_align=64 选项，保证所有类型数组、结构体、联合体等的首地址是 64 字节对界的。

为满足从核向量对界要求，从核 C 库新增 libc_aligned_malloc 函数，该函数分配的空间首地址为 64 字节对界，需配合 libc_aligned_free 函数使用。

使用-faddress_align=*n* 后，编译器会将所有的数组、结构体、联合体、向量的首地址进行对界设置，即使这些数组、结构体、联合体与向量的访存操作无关，这在一定程度上会增加程序的内存需求。对性能要求较高的用户，可以不使用这个选项，而是在程序中手工设置其对界属性，具体方法可参见《基础编译系统用户手册》中对界部分相关内容。

3.5 SIMD 对 C 操作的扩展（主核）

假设定义了下面的一些变量：

intv8 x;   int y;   long la;   floatv4 fv;   doublev4 *p, dv.

float f;   double d;

a) sizeof()

sizeof(x)和 sizeof(*p)的返回值都是 32。

b) 赋值

扩展类型可以在定义变量的时候像数组一样的赋初值，例如：

intv8 va = {1,2,3,4,5,6,7,8};   赋值的结果为：

va:	8	7	6	5	4	3	2	1
	255 224	223 192	191 160	159 128	127 96	95 64	63 32	31 0

或者

floatv4 vf = {2.323}; 赋值的结果为：

vf:	0	0	0	2.323
	255 192	191 128	127 64	63 0

另外，除了向量浮点类型和向量整型之间不能互相赋值之外，其他的赋值都是允许的。比如：

fv = 3;

x = 2.345;

*p = dv;

其中，fv=3 的赋值结果为：

fv:	3.0	3.0	3.0	3.0
	255 192	191 128	127 64	63 0

x=2.345 的赋值结果为：

x:	2	2	2	2	2	2	2	2
	255 224	223 192	191 160	159 128	127 96	95 64	63 32	31 0

等号左/右侧的类型不一样隐含着强制类型转换，强制类型转换的具体含义参见 3.9 节《扩展类型的类型转换》。

这里需要注意，下面的两种赋值方法都是允许的，但它们是有区别的：

i)　intv8 va = {2};

ii)　intv8 vb = 2;

其中，i)表示给 va 的最低 32 位赋初值 2，va 的其他部分为缺省值 0。而 ii)表示将 int 类型的常数强制转换为 intv8 类型，这种强制类型转换是扩展式的，vb 中的八个部分都是 2。

i)的结果

	255 224	223 192	191 160	159 128	127 96	95 64	63 32	31 0
va:	0	0	0	0	0	0	0	2

ii)的结果：

	255 224	223 192	191 160	159 128	127 96	95 64	63 32	31 0
vb:	2	2	2	2	2	2	2	2

c) 取地址符号

对向量类型的取地址符号是允许的，&x 表示取变量 x 的地址。

d) 指针

指向扩展类型的指针运算与标准类型的相同。例如 p+1 表示指向下一个扩展类型变量的指针。指针引用*p 隐含着从 p 的地址进行 256 位的装入操作。

e) 运算符的扩充

同时，本文本对标准 C 中的运算符进行了扩充，例如“+、-、*、/”等运算符现在都可以用于扩展类型数据之间的运算。下面列出了所有的扩展情况，运算符后跟的数据类型表示用这些类型定义的变量可以进行相应的运算操作（“：”前是标准 C 中的运算符号，“：”后面是该运算符扩展到的数据类型）：

+：intv8，uintv8，int256，uint256，floatv4，doublev4

-：intv8，uintv8，int256，uint256，floatv4，doublev4

*：floatv4，doublev4

/：floatv4，doublev4

&：intv8，uintv8，int256，uint256，

|：intv8，uintv8，int256，uint256，

^：intv8，uintv8，int256，uint256，

>>：intv8，uintv8，int256，uint256，

<<：intv8，uintv8，int256，uint256，

==：intv8，uintv8，floatv4，doublev4

<=：intv8，uintv8，floatv4，doublev4

<：intv8，uintv8，floatv4，doublev4

>=：intv8，uintv8

在编写 SIMD 程序时可以直接使用这些运算符来进行扩展数据类型上的运算。在使用的时候需要注意以下几点：

（1）对 intv8 类型扩展的“>>”符号，与标准 C 中的约定一样，如果“>>”左边的操作数是有符号的，那么最终生成的指令为算术右移，如果是无符号的，最终生成的指令为逻辑右移。

（2）int256 类型支持的加减法相当于 64*4 的四个 long 类型的向量加减法，因为 SW处理器主核指令集里面针对 64*4 这种长整型向量的操作仅包含加法和减法，

而 int256 本身也没有算术意义，从编译器具体实现的角度，我们认为没有必要再单独提供一种新类型，因而采用 int256 类型来实现 64*4 的长整型向量的加减。

（3）“>>”运算符仅支持 uint256 类型，为逻辑右移，不支持 int256 类型。

3.6 SIMD 对 C 操作的扩展（从核）

假设定义了下面的一些变量：

intv16 x; int y; long la; floatv8 fv; doublev8 *p, dv.

float f; double d;

a) sizeof()

sizeof(x)和 sizeof(*p)的返回值都是 64。

b) 赋值

扩展类型可以在定义变量的时候像数组一样的赋初值，例如：

intv16 va = {1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8}; 赋值的结果为：

va:	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
	511 480	479 448	447 416	415 384	383 352	351 320	319 288	287 256	255 224	223 192	191 160	159 128	127 96	95 64	63 32	31 0

或者

floatv8 vf = {2.323}; 赋值的结果为：

vf:	0	0	0	0	0	0	0	2.323
	511 448	447 384	383 320	319 256	255 192	191 128	127 64	63 0

另外，除了向量浮点类型和向量整型之间不能互相赋值之外，其他的赋值都是允许的。比如：

fv = 3;

x = 2.345;

*p = dv;

其中，fv=3 的赋值结果为：

fv:	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0
	511 448	447 384	383 320	319 256	255 192	191 128	127 64	63 0

x=2.345 的赋值结果为：

x:	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
	511 480	479 448	447 416	415 384	383 352	351 320	319 288	287 256	255 224	223 192	191 160	159 128	127 96	95 64	63 32	31 0

等号左/右侧的类型不一样隐含着强制类型转换，强制类型转换的具体含义参见 3.10 节《扩展类型的类型转换》。

intv16 va = {2};

表示给 va 的最低 32 位赋初值 2，va 的其他部分为缺省值 0。

va:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
	511 480	479 448	447 416	415 384	383 352	351 320	319 288	287 256	255 224	223 192	191 160	159 128	127 96	95 64	63 32	31 0

c) 取地址符号

对向量类型的取地址符号是允许的，&x 表示取变量 x 的地址。

d) 指针

指向扩展类型的指针运算与标准类型的相同。例如 p+1 表示指向下一个扩展类型变量的指针。指针引用*p 隐含着从 p 的地址进行 512 位的装入操作。

e) 运算符的扩充

同时，本文本对标准 C 中的运算符进行了扩充，例如“+、-、*”等运算符现在都可以用于扩展类型数据之间的运算。下面列出了所有的扩展情况，运算符后跟的数据类型表示用这些类型定义的变量可以进行相应的运算操作（“：”前是标准 C 中的运算符号，“：”后面是该运算符扩展到的数据类型）：

+：intv16，uintv16，int512，uint512，floatv8，doublev8，float16v32

-：intv16，uintv16，int512，uint512，floatv8，doublev8，float16v32

*：floatv8，doublev8，float16v32

>>：intv16，uintv16

<<：intv16，uintv16

==：intv16，uintv16，floatv8，doublev8，float16v32

<=：intv16，uintv16，floatv8，doublev8，float16v32

< ：intv16，uintv16，floatv8，doublev8，float16v32

在编写 SIMD 程序时可以直接使用这些运算符来进行扩展数据类型上的运算。在使用的时候需要注意以下几点：

（1）对 intv16 类型扩展的“>>”符号，与标准 C 中的约定一样，如果“>>”左边的操作数是有符号的，那么最终生成的指令为算术右移，如果是无符号的，最终生成的指令为逻辑右移。

（2）int512 类型支持的加减法相当于 64*8 的四个 long 类型的向量加减法，因为 SW处理器从核指令集里面针对 64*8 这种长整型向量的操作仅包含加法和减法，而 int512 本身也没有算术意义，从编译器具体实现的角度，我们认为没有必要再单独提供一种新类型，因而采用 int512 类型来实现 64*8 的长整型向量的加减。

3.7 SIMD 对 C 内部函数的扩展（主核）

对于扩展数据类型上的每一种运算操作我们都扩充了相应的内部函数或宏定义接口，供编程时调用。编译器成功安装后，只需要将“simd.h” include 到程序中就可以使用这些扩展数据类型和扩充的内部函数或宏定义了。

例如，要将一段普通的 C 程序向量化经常需要将标准类型的数据先映射到 SIMD 变量中，然后才进行运算。本文本扩充了 9 个 SIMD 宏定义接口，用于进行扩展数据类型与标准类型之间的映射操作。这 9 个宏定义是：

simd_load() —— 对界装入，将标准类型的数据从向量要求的对界内存地址映射到扩展类型变量

simd_loadu() —— 不对界装入，将标准类型的数据从不对界内存地址映射到扩展类型变量

simd_load_u() —— 强制对界装入，将不对界的内存地址低位清零形成强制对界地址，然后将标准类型的数据从对界内存映射到扩展类型变量

simd_loade() —— 装入并扩展，将标准类型的数据从内存映射到扩展类型变量的低位并扩展到高位

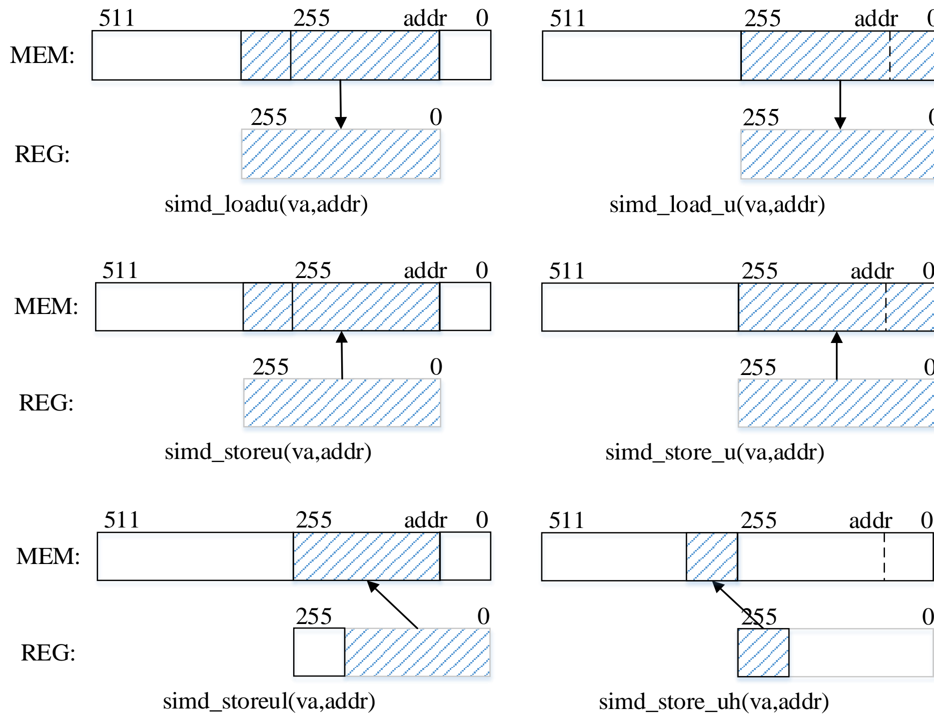
simd_store() —— 对界存储，将扩展类型变量中的数据映射到向量要求的对界内存中

simd_storeu() —— 不对界存储，将扩展类型变量中的数据映射到不对界内存中

simd_store_u() —— 强制对界存储，将不对界的内存地址低位清零形成强制对界地址，然后将扩展类型变量中的数据映射到对界内存中

simd_storeul() —— 不对界存储，将扩展类型变量中的低位数据映射到不对界内存的低位中

simd_storeuh() —— 不对界存储，将扩展类型变量中的高位数据映射到不对界内存的高位中

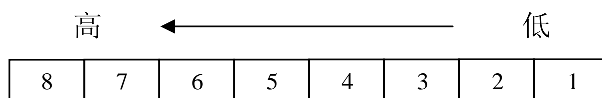


其中 simd_load、simd_loade、simd_store 适用于所有扩展数据类型的装入/存储，simd_loadu、simd_load_u、simd_storeu、simd_storeul、simd_storeuh 支持 intv8、uintv8、floatv4、doublev4 类型的不对界装入/存储。这些宏定义有两个参数，第一个是扩展类型，第二个是标准数据类型的指针，参数具体允许的类型请参照第五章《C 内部函数接口》。

在编写 SIMD 程序的过程中还经常需要给这些扩展类型的变量赋初值，本文提供了一些宏定义来做这件事情。例如定义一个 intv8 类型的变量 va，要给该变量赋初值，该初值应该包含八个整型常量，比如为{1,2,3,4,5,6,7,8}，那么可以这样编写：

```
intv8 va = simd_set_intv8 (1, 2, 3, 4,5,6,7,8);
```

在变量 va 中，这 8 个值的存放位置是这样的：



相应的 uintv8、int256、uint256、doublev4 和 floatv4 类型也有类似的函数：simd_set_uintv8、simd_set_int256、simd_set_uint256、simd_set_doublev4 和 simd_set_floatv4，也可以用“=”符号直接为该类型的变量赋初值，例如：

doublev4 vlint;

vlint = 123.456;

对于扩展数据类型，许多操作是用标准 C 的运算符号或库函数所无法表示的，这样的操作必须靠调用内部函数或宏定义来实现，例如：

simd_vinsfd () —— D 浮点向量插入

simd_vextfd () —— D 浮点向量提取

simd_vcpyfd () —— D 浮点向量拷贝

simd_vfseleqd () —— D 浮点向量等于选择

simd_vfselltd () —— D 浮点向量小于选择

simd_vfselled () ——D 浮点向量小于等于选择

扩展类型的打印，我们提供了 24 个打印函数，分别用于打印 6 种扩展类型，具体的打印函数见 5.2.6 节《打印函数接口》。

例如，

va = simd_set_intv8 (1,2,3,4,5,6,7,8);

simd_print_intv8(va);

打印的结果是：

$> [8, 7, 6, 5, 4, 3, 2, 1]

另外，还有一些操作，一般由编译器自动生成，这里也提供了一些内部函数接口供编程时直接调用，主要有扩展浮点类型的乘加、乘减、负乘加和负乘减操作等。

所有扩展运算符的运算操作也都提供了内部函数接口。

这里只是对扩充内部函数接口进行了简单的介绍，详细的描述请查阅第五章《C 内部函数接口》。

3.8 SIMD 对 C 内部函数的扩展（从核）

对于扩展数据类型上的每一种运算操作我们都扩充了相应的内部函数或宏定义接口，供编程时调用。编译器成功安装后，只需要将“simd.h” include 到程序中就可以使用这些扩展数据类型和扩充的内部函数或宏定义了。

例如，要将一段普通的 C 程序向量化经常需要将标准类型的数据先映射到 SIMD 变量中，然后才进行运算。本文本扩充了 7 个 SIMD 宏定义接口，用于进行扩展数据类型与标准类型之间的映射操作。这 7 个宏定义是：

simd_load() —— 对界装入，将标准类型的数据从向量要求的对界内存地址映射到扩展类型变量

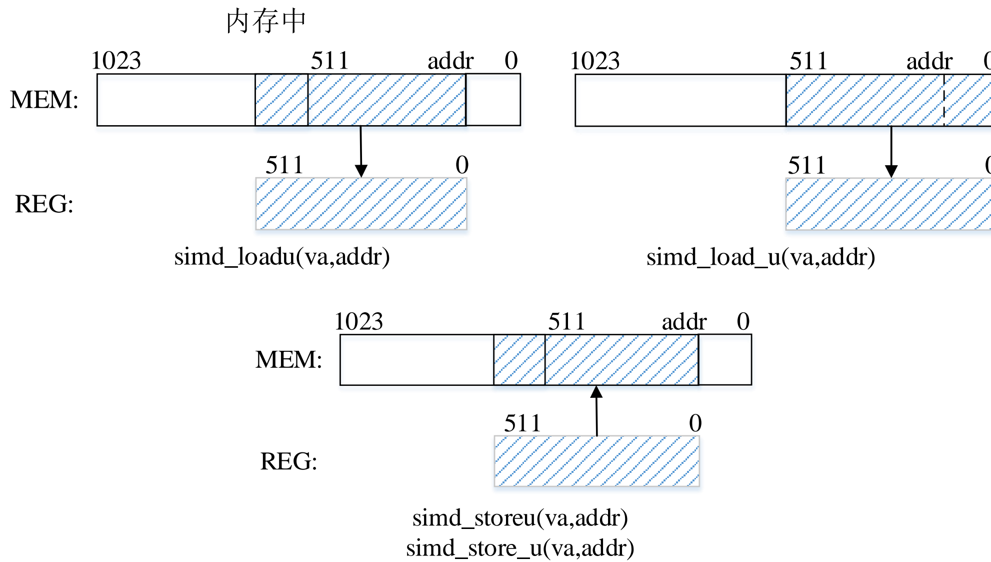
simd_loade() —— 装入并扩展，将标准类型的数据从内存映射到扩展类型变量的低位并扩展到高位

simd_loadu() —— 不对界装入，将标准类型的数据从不对界内存地址映射到扩展类型变量

simd_load_u() —— 强制对界装入，将不对界的内存地址低位清零形成强制对界地址，然后将标准类型的数据从对界内存映射到扩展类型变量

simd_store() —— 对界存储，将扩展类型变量中的数据映射到向量要求的对界内存中

simd_storeu()/simd_store_u() —— 不对界存储，将扩展类型变量中的数据映射到不对界内存中

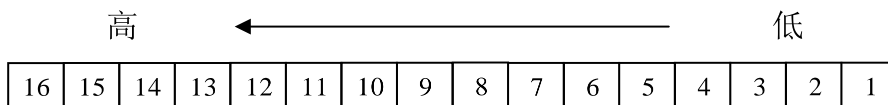


其中 simd_load、simd_store 适用于所有扩展数据类型的装入/存储，simd_loadu、simd_load_u、simd_storeu 支持 intv16、uintv16、floatv8、doublev8 类型的不对界装入/存储，simd_loade 仅支持 float16v32 类型的装入并扩展。这些宏定义有两个参数，第一个是扩展类型，第二个是标准数据类型的指针，参数具体允许的类型请参照第五章《C 内部函数接口》。

在编写 SIMD 程序的过程中还经常需要给这些扩展类型的变量赋初值，本文提供了一些宏定义来做这件事情。例如定义一个 intv16 类型的变量 va，要给该变量赋初值，该初值应该包含八个整型常量，比如为{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16}，那么可以这样编写：

intv16 va = simd_set_intv16 (1, 2, 3, 4,5,6,7,8,9,10,11,12,13,14,15,16);

在变量 va 中，这 8 个值的存放位置是这样的：



相应的 uintv16、int512、uint512、doublev8、floatv8、float16v32 类型也有类似的函数：simd_set_uintv16、simd_set_int512、simd_set_uint512、simd_set_doublev8、simd_set_floatv8 和 simd_set_float16v32，也可以用“=”符号直接为该类型的变量赋初值，例如：

doublev8 vlint;

vlint = 123.456;

对于扩展数据类型，许多操作是用标准 C 的运算符号或库函数所无法表示的，这样的操作必须靠调用内部函数或宏定义来实现，例如：

simd_vinsfd () —— D 浮点向量插入

simd_vextfd () —— D 浮点向量提取

simd_vcpyfd () —— D 浮点向量拷贝

simd_vfseleqd () —— D 浮点向量等于选择

simd_vfselltd () —— D 浮点向量小于选择

simd_vfselled () ——D 浮点向量小于等于选择

扩展类型的打印，我们提供了 28 个打印函数，分别用于打印 7 种扩展类型：具体的打印函数见 5.3.6 节《打印函数接口》。

例如，

```
va = simd_set_intv16 (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16);
simd_print_intv16(va);
```

打印的结果是：

```
$> [16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 ]
```

另外，还有一些操作，一般由编译器自动生成，这里也提供了一些内部函数接口供编程时直接调用，主要有扩展浮点类型的乘加、乘减、负乘加和负乘减操作等。

所有扩展运算符的运算操作也都提供了内部函数接口。

这里只是对扩充内部函数接口进行了简单的介绍，详细的描述请查阅第五章《C 内部函数接口》。

3.9 扩展类型的类型转换（主核）

在扩展类型中，SWGCC 支持如下几种转换：

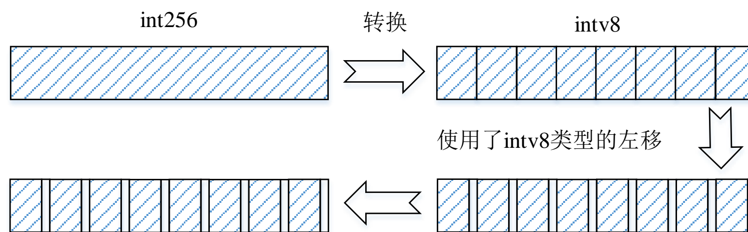
a）扩展浮点类型单、双精度之间的转换

扩展浮点类型 floatv4 和 doublev4 之间的转换，意义上与标准类型的单、双精度的类型转换一样，例如：

```
floatv4     fv;         doublev4     dv;
fv = dv;
```

b）扩展整数类型 intv8 和 int256 之间的转换

这两种类型之间的转换，变量中的数据不会产生任何的变化。编译器之所以支持这两种类型之间的转换，只是为了有效利用系统提供的指令。例如下面的例子中，一个 int256 的数转换成 intv8 后，使用 intv8 的移位操作可以达到使用 int256 类型很多条操作才能达到的效果：



同样的，intv8 类型的数据，转换成 int256 类型，并使用 int256 类型的操作后也可以收到事半功倍的效果。例如下图中先将 intv8 转换为 int256，使用 int256 的移位操作后可以很轻松的将最高的 32 位清空：

在编程过程中，用户灵活运用这两种类型的转换往往可以收到事半功倍的效果。

c）标准类型到扩展类型的转换

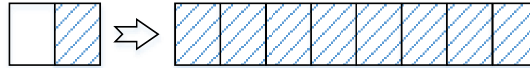
（1）值扩展

标准类型到 intv8，floatv4，doublev4 的转换是值的扩展。

例如：

```
intv8 va;        int x;
va = x;
```

把 x 的值复制 8 份放到 va 中：



在 SW处理器主核中，32 位的 int 数据是放在一个 64 位定点寄存器的低 32位的。所以上面的图中，左边是一个 64 位的寄存器，其中低 32 位是有效数据，扩展到右边一个 256 位的向量寄存器中。

再例如：

```
floatv4 fv;
fv = 3245.345;
```

结果是：

3245.345	3245.345	3245.345	3245.345

long 到 intv8 的转换，例如：intv8 vx;   long l;

```
vx = l;
```

相当于：

```
vx = (int)l;
```

float 到 intv8 的转换，例如：float f;   intv8 va;

```
va = f;
```

相当于：

```
va = (int)f;
```

（2）符号扩展

标准类型到 int256 的转换是值的扩展。

例如：

```
int256 va;        long x=-2;
va = x;
```

把 x 的值复制到 va 的低 64 位中，同时把符号位扩展到高 192，结果是：

ffff...fff	ffff...fff	ffff...fff	ffff...ffe

int 到 int256 的转换，例如：int256 vx;   int l;

```
vx = l;
```

相当于：

```
vx = (long)l;
```

d) 扩展类型到标准类型的转换

只是取扩展类型变量中最低的那部分。

例如：

```
int a;    intv8 va = {1,2,3,4,5,6,7,8};
a = va;
 printf (“%d\n”,a);
```

打印结果为：

```
1
```

3.10 扩展类型的类型转换（从核）

在扩展类型中，SWGCC 支持如下几种转换：

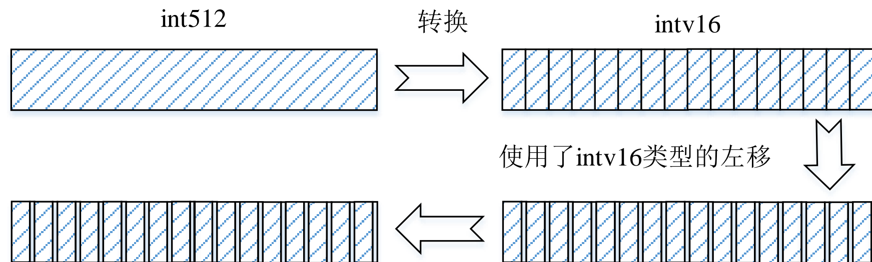
a）扩展浮点类型单、双精度之间的转换

扩展浮点类型 floatv8 和 doublev8 之间的转换，意义上与标准类型的单、双精度的类型转换一样，例如：

```
floatv8    fv;        doublev8    dv;
fv = dv;
```

b）扩展整数类型 intv16 和 int512 之间的转换

这两种类型之间的转换，变量中的数据不会产生任何的变化。编译器之所以支持这两种类型之间的转换，只是为了有效利用系统提供的指令。例如下面的例子中，一个 int512 的数转换成 intv16 后，使用 intv16 的移位操作可以达到使用 int512 类型很多条操作才能达到的效果：



同样的，intv16 类型的数据，转换成 int512 类型，并使用 int512 类型的操作后也可以收到事半功倍的效果。例如下图中先将 intv16 转换为 int512，使用 int512 的移位操作后可以很轻松的将最高的 32 位清空：

在编程过程中，用户灵活运用这两种类型的转换往往可以收到事半功倍的效果。

c) 扩展类型到标准类型的转换

只是取扩展类型变量中最低的那部分。

例如：

```
int a;    intv16 va = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};
a = va;
 printf (“%d\n”,a);
```

打印结果为：

1

d) 扩展浮点类型半精度、单精度之间的转换

floatv8 simd_vfcvths (float16v32 __A, const int __B)：扩展浮点类型 float16v32 到 floatv8 之间的转换，根据第二个 int 类型参数的值，从 float16v32 输入参数的每个 64 位中选择 1 个 16 位 HP 浮点数分量，转换成 64 位的 SP 浮点数扩展格式，将转换后的 8 个 64 位 SP 浮点数写入返回值。

float16v32 simd_vfcvtsh (floatv8 __A, const int __B)：扩展浮点类型 floatv8 到 float16v32 之间的转换, 将 floatv8 输入参数中的 8 个 64 位 SP 扩展格式浮点数，转换成 8 个 16 位 HP 格式浮点数，然后根据第二个 int 类型参数的值写入 float16v32 返回值指定位置，并将其它位置清 0。

e）扩展浮点类型与扩展长字整形之间的转换

int512 可以作为扩展的长字整形向量存在，寄存器中的 512 位被划分为 8 个 64 位，每个 64 位存放一个长整形数据，此时支持它与扩展浮点类型 floatv8/doublev8 之间的转换，意义上与标准类型的长字整形/浮点类型的转换一样。例如：

int512 va; floatv8 vb; doublev8 vc;

编译器支持 int512 类型到 floatv8/doublev8 类型之间的转换：

vb = va; vc = va;

编译器支持 doublev8 类型到 int512 类型的转换：

va = vc；

因为从核没有直接的单精度浮点向量到长字向量的转换指令，编译器不支持 floatv8 类型到 int512 类型的转换，即下面的语句编译会报错：

va = vb；

3.11 扩展类型与标准类型之间的数据交换（主核）

SWGCC 可以通过以下两种方法在标准数据类型与扩展类型的变量之间进行数据交换：（1）通过内存交换；（2）通过 SW处理器主核提供的一些特殊操作如：vcpyf，vextf 等，上一节中的标准类型到扩展类型的强制类型转换编译器内部也是使用了这些操作完成的。后一种的性能一般要好于前一种，但是后一种一般由编译器优化使用。例如要将一个整型数组的8个连续整型数据传递到一个intv8类型的向量变量中，下面的写法是错误的：

```
intv8 vsum;
int vertex[8];
vsum=(intv8)vertex[0]; /* 错误写法 */
```

由于 vertex[0]表示一个 int 类型的值，所以这种写法相当于上一节中的强制类型转换，是扩展方式的：

vertex[0]	vertex[0]	vertex[0]	vertex[0]	vertex[0]	vertex[0]	vertex[0]	vertex[0]

而我们想要的结果实际上是：

vertex[7]	vertex[6]	vertex[5]	vertex[4]	vertex[3]	vertex[2]	vertex[1]	vertex[0]

正确的方法主要有三种：

a) 使用指针：

```
intv8 vsum, *p;
int vertex[8];
p = (intv8 *) (&(vertex[0]));
vsum = *p;
```

b) 使用内部访存函数：

```
/*把以&(vertex[0])为起址，长度为256位的内容拷贝到vsum中*/
intv8 vsum;    int vertex[8]
simd_load(vsum, &(vertex[0]));
```

c) 使用内部赋值函数：

```
intv8 vsum;    int vertex[8];
vsum = simd_set_intv8 (vertex[0], vertex[1], vertex[2], vertex[3]，vertex[4]，vertex[5]，
vertex[6]，vertex[7]);
```

目前三种方法都使用了内存作为交换通道，在性能上没有差异。在本例子中，因为前两种方法都只需要一次访存，而第三种方法需要多次访存，前两种方法更好。但如果要传递的数据不在连续的内存区域中，比如，要将8个int类型的变量传递到一个intv8变量中，第三种方法就更好一些，例如：

```
intv8 va;
int x,y,z,w,m,n,s,t;
va = simd_set_intv8 (x, y, z, w, m, n, s, t);
```

3.12 扩展类型与标准类型之间的数据交换（从核）

SWGCC 可以通过以下两种方法在标准数据类型与扩展类型的变量之间进行数据交换：（1）通过内存交换；（2）通过 SW处理器从核提供的一些特殊操作如：vcpyf，vextf 等，上一节中的标准类型到扩展类型的强制类型转换编译器内部也是使用了这些操作完成的。后一种的性能一般要好于前一种，但是后一种一般由编译器优化使用。例如要将一个整型数组的16个连续整型数据传递到一个intv16类型的向量变量中，下面的写法是错误的：

```
intv16 vsum;
int vertex[16];
vsum=(intv16)vertex[0]; /*  错误写法  */
```

正确用法：

d) 使用指针：

```
intv16 vsum, *p;
int vertex[16];
p = (intv16 *) (&(vertex[0]));
vsum = *p;
```

e) 使用内部访存函数：

/*把以&(vertex[0])为起址，长度为 512 位的内容拷贝到 vsum 中*/

```
intv16 vsum;     int vertex[16]
simd_load(vsum, &(vertex[0]));
```

f) 使用内部赋值函数：

```
intv16 vsum;     int vertex[16];
vsum = simd_set_intv16 (vertex[0], vertex[1], vertex[2], vertex[3]，vertex[4]，vertex[5]，
vertex[6]，vertex[7]，vertex[8], vertex[9], vertex[10], vertex[11]，vertex[12]，vertex[13]，
vertex[14] , vertex[15]);
```

目前三种方法都使用了内存作为交换通道，在性能上没有差异。在本例子中，因为前两种方法都只需要一次访存，而第三种方法需要多次访存，前两种方法更好。但如果要传递的数据不在连续的内存区域中，比如，要将 16 个 int 类型的变量传递到一个 intv16 变量中，第三种方法就更好一些，例如：

```
intv16 va;
int x,y,z,w,m,n,s,t,a,s,d,f,g,h,j,k;
va = simd_set_intv16 (x, y, z, w, m, n, s, t, a, s, d, f, g, h, j, k);
```

第四章 串行程序的向量化

在原有程序的基础上，要使用系统的 SIMD 功能，最直接的办法是先对程序进行向量化。

向量化时首先要考虑到一些限制：

主核硬件限制：

向量化受限于硬件的限制。在 SIMD 扩展中，向量化访存操作必须是地址连续的，且要求 32 字节（单精度浮点向量为 16 字节）对界的。意味着虽然有些循环是可向量化的，但具体针对 SW处理器主核体系结构，需要进行程序变换后才能向量化。另外，由于扩展定点类型的指令集是不完备的，例如 8*32 的乘法、除法等，这样的循环在 SW处理器主核上没有办法向量化。

从核硬件限制：

向量化受限于硬件的限制。在 SIMD 扩展中，向量化访存操作必须是地址连续的，且要求 64 字节（单精度浮点向量为 32 字节）对界的。意味着虽然有些循环是可向量化的，但具体针对 SW处理器从核体系结构，需要进行程序变换后才能向量化。另外，由于扩展定点类型的指令集是不完备的，例如 16*32 的乘法、除法等，这样的循环在 SW处理器从核上没有办法向量化。

程序形式：

循环体中有复杂的关键字、操作、数据访问和内存操作等，都会影响对程序的相关性分析和向量化分析，从而不利于用户或编译器对程序进行向量化。

了解了这些限制，你可以修改程序，避开这些限制，实现高效向量化。下面我们主要针对 C 语言介绍一些针对循环进行向量化的技术。

4.1 程序形式对向量化的限制

4.1.1 相关性的限制

向量化的基础是相关性分析。例如

例 3.

```
float data[N];
int i;
for (i=1; i<N-1; i++){
    data[i] = data[i-1]*0.25 + data[i]*0.5 + data[i+1]*0.25;
}
```

上述循环不能向量化，因为写当前元素 data[i]时依赖于使用前面的元素 data[i-1]，而它在前一个迭代发生了改变。为了更清楚一点，下面列出头两个迭代的数组访问图：

```
i=1：    READ data[0]
         READ data[1]
         READ data[2]
```

```
              WRITE data[1]
    i=2：     READ data[1]
              READ data[2]
              READ data[3]
              WRITE data[2]
```

按照循环的执行顺序，第二次迭代读 data[1]的值就是第一次迭代所写的值。而向量化必须要求迭代间可以并行执行，且不改变原始循环的语义。

数据相关性分析包括寻找内存访问可能重叠的条件。给定程序的两个访问，相关的条件如下：

a) 访问的变量在内存中是同一个（或者重叠）区域的别名

b) 对数组访问，分析下标的关系是否可能相等

4.1.2 循环结构的限制

循环可以是常用的 for，while-do 或 repeat-until 结构，也可以是使用 goto 和标号组成的。然而，循环必须是只有一个入口和一个出口时才能被向量化。

例 4．一个可向量化的循环：

```
double a[100], b[100], c[100];
while ( i<n ){
      a[i]= b[i] * c[i];
      i++;
}
```

例 5．一个不可向量化的循环：

```
while ( i<n ){
      if ( condition ) break;
      /*  循环体的另一个出口  */
      i++;
}
```

4.1.3 循环退出条件的限制

循环退出条件决定循环执行的迭代次数。向量化需要迭代次数是可以计算的，也就是说迭代次数必须是下述表达式：

a) 常数

b) 循环不变量

c) 外层循环控制变量的线性函数

4.2 循环向量化步骤

通常，确定循环可以向量化后，向量化一个循环可以分以下步骤：循环展开和分裂、变量替换、操作替换。

4.2.1 循环展开和分裂

例 6．对循环进行循环分裂（主核）

```
int vertex[n], sum;
for (i = 0; i < n; i++){
    sum=sum+vertex[i];
};
```

变为：

```
int sum_temp[8], vertex[n], sum;
for (i = 0; i < n/8; i += 8){
    for(j = 0; j < 8; j++){
        sum_temp[j] = sum_temp[j] + vertex[8*i+j];
    }
};
for(j = 0; j < 8; j++)
    sum = sum + sum_temp[j];
for (i = (n/8)*8; i < n; i++){
    sum=sum+vertex[i];
};
```

例如上面的例子中如果 n 不能被 8 整除，将最后不够一次向量操作的部分分裂出来，前面的部分就可以向量化了。

例 7．对循环进行循环分裂（从核）

```
int vertex[n], sum;
for (i = 0; i < n; i++){
    sum=sum+vertex[i];
};
```

变为：

```
int sum_temp[16], vertex[n], sum;
for (i = 0; i < n/16; i += 16){
    for(j = 0; j < 16; j++){
        sum_temp[j] = sum_temp[j] + vertex[16*i+j];
    }
};
for(j = 0; j < 16; j++)
    sum = sum + sum_temp[j];
for (i = (n/16)*16; i < n; i++){
    sum=sum+vertex[i];
};
```

例如上面的例子中如果 n 不能被 16 整除，将最后不够一次向量操作的部分分裂出来，

前面的部分就可以向量化了。

4.2.2 变量替换

声明扩展类型变量有两种策略，一种是在算法一级把关键变量都替换成扩展类型变量，这种方法改造程序彻底，但对程序改动较大；一种方法是在需要向量化的地方才引入扩展类型变量用于 SIMD 操作，该方法牵涉的地方较小，有利于快速向量化和调试。

主核：

int vsum[8]; → intv8 vsum_temp;

int vertex[n]; → intv8 tx;

从核：

int vsum[16]; → intv16 vsum_temp;

int vertex[n]; → intv16 tx;

4.2.3 操作替换

若不是全局替换，在用 SIMD 操作替换普通操作之前，需要把标准类型变量映射（或复制）到扩展类型变量上，操作完之后，再把扩展类型变量映射（或复制）回标准类型变量。如：

主核：

```
simd_load (tx, &(vertex[8*i]));    //将标准类型映射到扩展类型
simd_store (tx ,&(vertex[8*i]));    //将扩展类型映射到标准类型
```

从核：

```
simd_load (tx, &(vertex[16*i]));    //将标准类型映射到扩展类型
simd_store (tx ,&(vertex[16*i]));    //将扩展类型映射到标准类型
```

4.3 如何更好地向量化

4.3.1 合理使用数组

为了有效的进行向量化，程序中对数组的操作需要尽量的连续访问，这意味着对于多维数组，尽量选择在最右边的维（C 语言）进行向量化。

另外，尽量让数组作为结构的一个域，而不是使用结构类型的数组。例如下面的例子中，右边的更好向量化。

表 5-1 数组的向量化比较

Array of Structures	Structure of Arrays

struct
{ float a, b, c, d, e, f }s[99];
int i, n;
for (i=0; i < n; i++)
    x += s[i].a * s[i].e;
	struct { float a[99], b[99], c[99],
	d[99], e[99], f[99] } s;
	int i, n;
	for (i=0; i < n; i++)
	    x += s.a[i] * s.e[i];

4.3.2 对函数调用的处理

如果要向量化的循环中包含一个函数调用，占用很大开销，可以通过将该函数 inline 进循环体或者将该函数体向量化的方法来解决。例如下面的一段程序：

表 5-2 函数体的向量化比较

原来的程序	inline	函数体向量化

unsigned fc[FCL],q,k;
for (k=0;k<FCL;k++)
  if (func(fc[k]&q))
    s--;
  else s++;
int func(unsigned s) {
  int i;
 	
for(i=16;i>=1;i>>=1)
    s ^= s>>i;
  return s%2;
}
	int fc[FCL],q,tmp[8],k,i;
	intv8 v,vq,vi;
	vq=simd_set_intv8(q,q,q,q,q,q,q,q);
	for(k=0;k<FCL;k+=8){
	  simd_load(v,&fc[k]);
	  v=v&vq;
	  for(i=16;i>=1;i>>=1) {
	    vi=
	    simd_set_intv8(i,i,i,i,i,i,i,i);
	    v^=simd_vsraw(v,vi)
	    simd_store(v,tmp);
	    for(i=0;i<8;i++)
	      if (tmp[i]%2) s--;
	      else s++;
	  }
	}
		unsigned fc[FCL],q,t[8],k;
		intv8 vq,v;
		vq=simd_set_intv8(q,q,q,q);
		for(k=0;k<FCL;k+=8) {
		  simd_load(v,&fc[k]);
		  v=v&vq;
		  vfunc(v,t);
		  for(i=0;i<8;i++)
		    if(t[i]%2) s--;
		    else s++;
		}
		void vfunc(intv8 v,int*r){
		  int i; intv8 vi;
		  for(i=16;i>=1;i>>=1) {
		    vi=
		    simd_set_intv8(i,i,i,i,i,i,i,i);
		    v^=simd_vsraw(v,vi)
		  }
		  simd_store(v,r);
		}

在上面的例子中，扩展类型变量 vs（程序中是 v）作为函数的参数传递进 vfunc。在主核 SIMD 扩展中，扩展类型在函数调用过程中的使用约定与浮点类型的一致。最左边的六个参数通过寄存器$f16-$f21 传递，第七个往后的参数通过栈传递。返回值放在$f0 中。在从核 SIMD 扩展中，扩展类型在函数调用过程的中，整数和浮点数最左边的六个参数通过寄存器$16-$21 传递，第七个往后的参数通过栈传递，返回值在$0 中。而向量类型最左边的六个参数通过寄存器$48-$53 传递，返回值放在$32 中。

4.3.3 更有效的使用 Cache

高效地使用各级 Cache 对于性能的提高是极为重要的，到主存储器中访问数据要比在一级 Cache 中访问数据慢数十倍。为了更好地使用 Cache，程序需要尽量使用同一个 Cache 行的所有数据而不是各不同 Cache 行的部分数据，而且程序最好能在数据被替换出 Cache 以前尽量多的重用这些数据。当然，为了从 SIMD 部件中获得性能的提升，也要求程序最好访问连续的内存区域，这一点来讲，Cache 与 SIMD 部件对程序的要求是一样的。

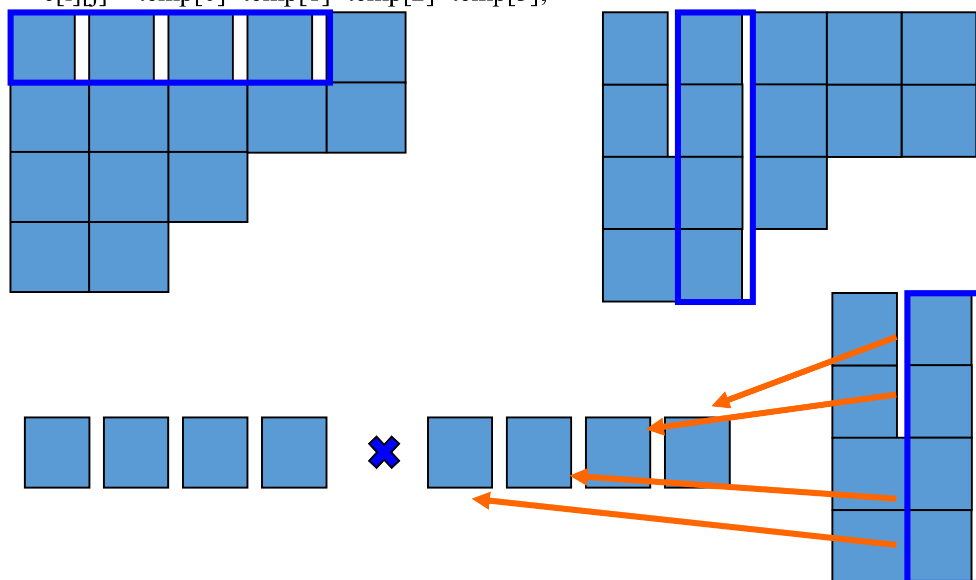
例如在矩阵乘的例子中，第一个矩阵通常按照行优先的顺序访问，第二个矩阵则要按照列优先的顺序访问，显然对第一个矩阵的访问具有空间局部性，因为访问的是连续地址，而对第二个矩阵的访问每一次访问的经常是一个新的 Cache 行。为了实施向量化，通常需要将

第二个矩阵的某一列数据逐一装入，然后重新组织到向量寄存器中，这样会增加很大的开销。我们以程序的形式来说明：

```
for (i=0;i<n;i++)
    for(j=0;j<n;j++)
        for(k=0;k<n;k++)
          c[i][j] = c[i][j] + a[i][k]*b[k][j];
```

如果直接进行向量化，则程序变换为：

```
for(k=0;k<n;k += 4) {
    simd_load(va， &a[i][k]);
    vb <- (b[k][j], b[k+1][j], b[k+2][j], b[k+3][j])
    vt += va*vb;
}
simd_store(vt,temp);
c[i][j] = temp[0]+temp[1]+temp[2]+temp[3];
```



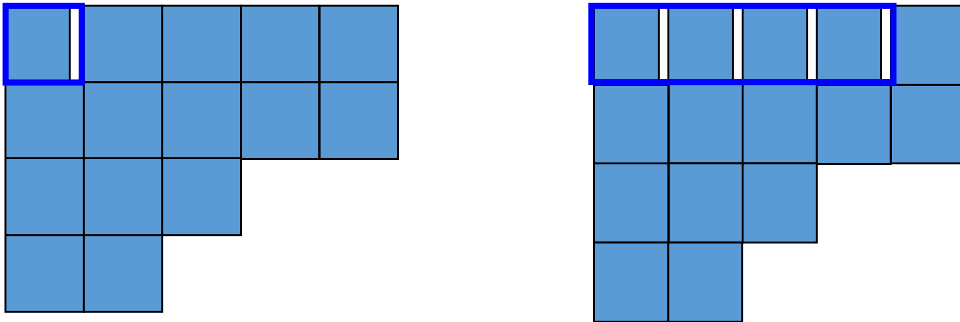
向量化的时候，数组 A 连续的 4 个元素可以直接装载到向量中，而数组 B 对应作乘法的元素在内存中并不连续，相当于是对多个 cache 行的访问，这样就需要花费较大的开销去拼接。但是，如果把程序做一下变换，就可以很好的避开访问不同 cache 行的开销，把 k 级循环和 j 级循环互换，然后再做向量化，如下：

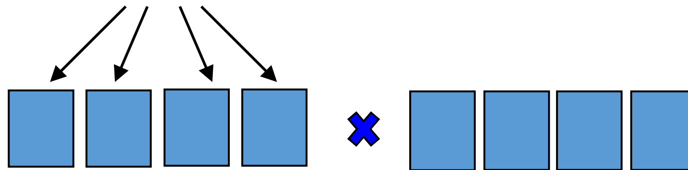
```
for (i=0;i<n;i++)
    for(j=0;j<n;j++)
        for(k=0;k<n;k++)
          c[i][j] = c[i][j] + a[i][k]*b[k][j];
```

向量化后程序变换为：

```
for(i=0;i<n;i++) {
    for (k=0;k<n;k++) {
        simd_loade(va, &a[i][k]);
```

```
            for (j=0;j<n;j += 4) {
                simd_load(vb,&b[k][j]);
                vt = va*vb;
                simd_load(vs, &c[i][j]);
                vs += vt;
                simd_store ( vs,&c[i][j])
            }
        }
    }
```





由于 j 是最内层循环，因此对 B[k][j]的访问是连续的，而 A[i][k]在 j 级循环里面仅相当于一个常数，这样就可以用数组 A 的一个元素同时去乘数组 B 的连续四个元素，完全屏蔽了不连续的 cache 访问。

4.3.4 对界问题的处理

主核：在 4.2.3 节介绍的操作替换过程中，使用 simd_load 或 simd_store 操作进行变量映射的时候，需要保证标准类型变量为 32 字节对界（对于 floatv4，只需 16 字节对界）。如果出现了不对界的访存，执行的时候操作系统将最终处理该访存引起的异常，这将付出更昂贵的代价。

编译器可以通过选项-faddress_align=32 来控制数组、结构体、union 联合体等的起始地址为 32 字节对界，其余要求用户保证。

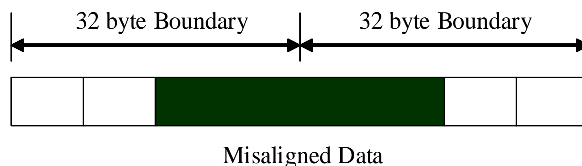


图 4-1 数据跨 32 字节的不对界访存

从核：在 4.2.3 节介绍的操作替换过程中，使用 simd_load 或 simd_store 操作进行变量映

射的时候，需要保证标准类型变量为 64 字节对界（对于 floatv8，只需 32 字节对界）。如果出现了不对界的访存，执行的时候处理器硬件会直接报错退出。

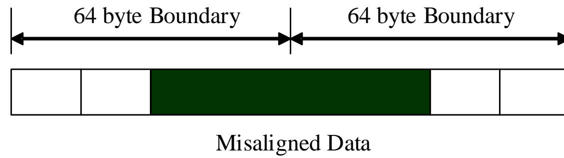


图 4-2 数据跨 64 字节的不对界访存

编译器可以通过选项-faddress_align=64 来控制数组、结构体、union 联合体等的起始地址为 64 字节对界，其余要求用户保证。

对于有些不对界的情况，用户可以进行调整，例如如下主核例子：

例 8．处理扩展类型的对界问题

```
double a[n],b[n+2];
for ( i=0; i<n; i++) {
      a[i]=a[i]+b[i+2];
}
```

在向量化该循环时，用户若保证数组 a 的对界要求，必然就不能保证数组 b 的对界要求，我们可以采用两种方法来解决该问题。

1、采用 SIMD 整理指令对数组 b 进行拼接

由于对数组 b 的访问不能保证 32 字节对界的要求，因此需要降低对数组 b 的访问粒度，将 b[i+2]、b[i+3]、b[i+4]、b[i+5]分别取出，然后组合成向量与数组 a 进行向量运算。改写后的 SIMD 程序如下：

```
#include <simd.h>
double   a[n],b[n+2];
doublev4   aa, bb,bb1,vtmp;
for ( i=0; i<(n/4)*4; i +=4 ) {
      simd_load(aa, &(a[i]));
      simd_load(bb, &(b[i]));
      vtmp=simd_vextfd2(bb);
      bb1=simd_vinsfd0(vtmp,bb1);
      vtmp=simd_vextfd3(bb);
      bb1=simd_vinsfd1(vtmp,bb1);
      simd_load(bb, &(b[i+4]));
      vtmp=simd_vextfd0(bb);
      bb1=simd_vinsfd2(vtmp,bb1);
      vtmp=simd_vextfd1(bb);
      bb1=simd_vinsfd3(vtmp,bb1);
      aa=aa+bb1;
      simd_store(aa, &(a[i]));
}
```

```
for ( i=(n/4)*4; i<n; i++) {
        a[i]=a[i]+b[i+2];
}
```

调整后，分别在&b[i]和&b[i+4]处使用两次对界的装入，然后拼接成从&b[i+2]开始的连续的 4 个 64 位双精度浮点数进行运算。

2、使用不对界访存接口直接处理

根据向量不对界访存指令，主核 SIMD 我们提供了五个不对界访存接口，分别是向量不对界取 simd_loadu、simd_load_u 和向量不对界存 simd_storeu、simd_storeul、simd_storeuh。使用不对界访存指令可以不考虑向量类型 32 字节对界（floatv4 为 16 字节对界）的要求，只需满足存取基本单元的对界要求即可，如 int 和 float 类型需要 4 字节对界，double 类型需要 8 字节对界。从核 SIMD 我们提供了三个不对界访存接口，分别是向量不对界取 simd_loadu、simd_load_u 和向量不对界存 simd_storeu。同样使用不对界访存指令可以不考虑向量类型 64 字节对界（floatv4 为 32 字节对界）的要求，只需满足存取基本单元的对界要求即可，如 int 和 float 类型需要 4 字节对界，double 类型需要 8 字节对界。上述主核程序转换如下：

```
#include <simd.h>
double    a[n],b[n+2];
doublev4    aa, bb;
for ( i=0; i<(n/4)*4; i += 4) {
      simd_load(aa, &(a[i]));
      simd_loadu(bb, &(b[i+2]));
      aa=aa+bb;
      simd_store(aa,&a[i]);
}
for ( i=(n/4)*4; i<n; i++) {
        a[i]=a[i]+b[i+2];
}
```

调用不对界访存接口直接装入 b[i+2]，然后直接做向量加法。第二种方法在性能上要优于第一种。

3、使用不对界访存接口和拼接指令组合处理

simd_loadu(bb, &(b[i+2]))实际上是由 simd_load_u(vtmp1, &(b[i+2]))、simd_load_u(vtmp2, &(b[i+6]))和 bb=simd_vcond（vtmp1，vtmp2，&(b[i+2]) ）三条语句实现，，连续使用 simd_loadu 取连续的数组数据时，相当于除了首尾数据外，都使用了 simd_load_u 取了两次，造成了访存浪费。用户可根据具体情况具体实现 simd_loadu 功能。上述主核程序可转化如下。

```
#include <simd.h>
double    a[n],b[n+2];
doublev4    aa, bb, vtmp1, vtmp2;
simd_load_u(vtmp1, &(b[2]));
for ( i=0; i<(n/4)*4; i += 4) {
```

```
        simd_load(aa, &(a[i]));
        simd_load_u(vtmp2, &(b[i+6]));
        bb=simd_vcond（vtmp1， vtmp2， &(b[i+2])）;
        aa=aa+bb;
        simd_store(aa,&a[i]);
        vtmp1= vtmp2;
}
for ( i=(n/4)*4; i<n; i++) {
            a[i]=a[i]+b[i+2];
}
```

第五章 C 内部函数接口

5.1 使用的符号说明

VRav：第一个向量参数

VRbv：第二个向量参数

VRcv：第三个向量参数

VRdv：第四个向量参数

VRc：向量返回值

Rbv：无符号长整型变量，一般用于表示地址

Va：一段内存地址中的数据 f

fp：FILE*类型的变量

5.2 主核内部函数接口

5.2.1 装入/存储函数接口

表 5-1 装入/存储操作的宏定义

宏定义	操作	参数说明	
		VRav	**Rbv**
simd_load	装入	扩展类型	int*,long*,float*,double*
simd_loade	装入并扩展	扩展类型	int*,float*,double*
simd_store	存储	扩展类型	int*,long*,float*,double*
simd_loadu	不对界装入	扩展类型	int* ,float*,double*
simd_load_u	强制对界装入	扩展类型	int* ,float*,double*
simd_storeu	不对界存储	扩展类型	int*,float*,double*
simd_store_u	强制对界存储	扩展类型	int*,float*,double*
simd_storeul	不对界存储	扩展类型	int*,long*,float*,double*
simd_storeuh	不对界存储	扩展类型	int*,long*,float*,double*

说明：当 Rbv 为 int *时，VRav 只能是 intv8；当 Rbv 为 long *类型时，VRav 只能是 int256；Rbv 为 float *时，VRav 只能是 floatv4 类型；Rbv 为 double *类型时，VRav 只能是 doublev4 类型。如果上述类型不匹配，编译器将报错。

函数名

simd_load —— 扩展类型装入

参数说明

a) intv8 类型的装入

```
[u]intv8 va;
[unsigned] int *addr; //intv8 类型的装入
```

b)　int256 类型的装入

[u]int256 va;

[[unsigned] long *addr;]; //int256 类型的装入

c)　floatv4 类型的装入

floatv4 va;

float *addr; //floatv4 类型的装入

d)　doublev4 类型的装入

doublev4 va;

double *addr; //doublev4 类型的装入

装入结果：

a)　intv8 类型的装入

[u]intv8 //intv8 类型的装入

b)　int256 类型的装入

[u]int256 //int256 类型的装入

c)　floatv4 类型的装入

floatv4; //floatv4 类型的装入

d)　doublev4 类型的装入

doublev4; //doublev4 类型的装入

功能说明

扩展类型的装入，将 32 字节（floatv4 类型是 16 字节）长度的数据从连续内存区域中装入到一个向量变量中。

函数名

simd_loade —— 扩展类型装入并扩展

参数说明

[u]intv8 va;

[unsigned] int *addr; //intv8 类型的装入

floatv4 va;

float *addr; //floatv4 类型的装入

doublev4 va;

double *addr; //doublev4 类型的装入

装入结果

[u]intv8 //intv8 类型的装入

floatv4; //floatv4 类型的装入

doublev4; //doublev4 类型的装入

功能说明

扩展类型的装入并扩展，将 8 字节（floatv4 类型是 4 字节）长度的数据从连续内存区域中装入到一个向量变量的低 64 位，并且将该数据扩展到高 192 位，使向量寄存器的四个寄

存器具有相同的值。intv8 是取出 4 字节，然后扩展成 8 个分量。

函数名

simd_store —— 扩展类型存储

参数说明

[u]intv8 va;

[unsigned] int *addr;[[unsigned] long *addr;] //intv8 类型的存储

floatv4 va;

float *addr; //floatv4 类型的存储

doublev4 va;

double *addr; //doublev4 类型的存储

返回值

无

功能说明

扩展类型的存储，将一个向量变量中的数据存储到 32 字节（floatv4 类型是 16 字节）连续内存区域中。

函数名

simd_loadu —— 扩展类型不对界装入

参数说明

a) intv8 类型的装入

[u]intv8 va;

[unsigned] int *addr; //intv8 类型的装入

b) floatv4 类型的装入

floatv4 va;

float *addr; //floatv4 类型的装入

c) doublev4 类型的装入

doublev4 va;

double *addr; //doublev4 类型的装入

装入结果

a) intv8 类型的装入

[u]intv8 //intv8 类型的装入

b) floatv4 类型的装入

floatv4; //floatv4 类型的装入

c) doublev4 类型的装入

doublev4; //doublev4 类型的装入

功能说明

扩展类型的不对界装入，将 32 字节（floatv4 类型是 16 字节）长度的数据，从任意地址

开始的内存区域中装入到一个向量变量中。

函数名

simd_load_u —— 扩展类型强制对界装入

参数说明

d) intv8 类型的装入

[u]intv8 va;

[unsigned] int *addr; //intv8 类型的装入

e) floatv4 类型的装入

floatv4 va;

float *addr; //floatv4 类型的装入

f) doublev4 类型的装入

doublev4 va;

double *addr; //doublev4 类型的装入

装入结果

d) intv8 类型的装入

[u]intv8 //intv8 类型的装入

e) floatv4 类型的装入

floatv4; //floatv4 类型的装入

f) doublev4 类型的装入

doublev4; //doublev4 类型的装入

功能说明

扩展类型的强制对界装入，将不对界地址 addr 低位清零形成强制对界地址，将 32 字节（floatv4 类型是 16 字节）长度的数据从强制对界地址开始的连续内存区域中装入到一个向量变量中。

函数名

simd_storeu —— 扩展类型的不对界存储

参数说明

[u]intv8 va;

[unsigned] int *addr; //intv8 类型的存储

floatv4 va;

float *addr; //floatv4 类型的存储

doublev4 va;

double *addr; //doublev4 类型的存储

返回值

无

功能说明

扩展类型的不对界存储，将一个向量变量中的数据存储到 32 字节（floatv4 类型是 16 字节）连续内存区域中。

函数名

simd_store_u —— 扩展类型的强制对界存储

参数说明

[u]intv8 va;

[unsigned] int *addr; //intv8 类型的存储

floatv4 va;

float *addr; //floatv4 类型的存储

doublev4 va;

double *addr; //doublev4 类型的存储

返回值

无

功能说明

扩展类型的强制对界存储，不对界地址低位清零形成强制对界地址，将一个向量变量中的数据存储到强制对界地址开始的 32 字节（floatv4 类型是 16 字节）连续内存区域中。

函数名

simd_storeul —— 扩展类型的不对界存储

参数说明

[u]intv8 va;

[unsigned] int *addr; //intv8 类型的存储

[u]int256 va;

[[unsigned] long *addr;]; //int256 类型的存储

floatv4 va;

float *addr; //floatv4 类型的存储

doublev4 va;

double *addr; //doublev4 类型的存储

返回值

无

功能说明

扩展类型的不对界存储，将一个向量变量中的低位数据存储到不对界地址的低位连续内存区域中。

函数名

simd_storeuh —— 扩展类型的不对界存储

参数说明

[u]intv8 va;

[unsigned] int *addr; //intv8 类型的存储

[u]int256 va;

[[unsigned] long *addr;]; //int256 类型的存储

floatv4 va;

float *addr; //floatv4 类型的存储

doublev4 va;

double *addr; //doublev4 类型的存储

返回值

无

功能说明

扩展类型的不对界存储，将一个向量变量中的高位数据存储到不对界地址的高位连续内存区域中。

5.2.2 定点向量运算函数接口

表 5-2 整数运算操作函数

内部函数	操作	参数说明			
		返回值	VRav	VRbv	VRcv
simd_vaddw	+	intv8	intv8	intv8/lit8	—
simd_vsubw	-	intv8	intv8	intv8/lit8	—
simd_vandw	&	intv8	intv8	intv8/lit8	—
simd_vbicw	与非	intv8	intv8	intv8/lit8	—
simd_vbisw	|	intv8	intv8	intv8/lit8	—
simd_vornotw	或非	intv8	intv8	intv8/lit8	—
simd_vxorw	^	intv8	intv8	intv8/lit8	—
simd_veqvw	等效	intv8	intv8	intv8/lit8	—
simd_vsllw simd_vsllwi	<<	intv8	intv8	int/lit5	—
simd_vsrlw simd_vsrlwi	>>	intv8	intv8	int/lit5	—
simd_vsraw simd_vsrawi	算术右移	intv8	intv8	int/lit5	—
simd_vcmpgew simd_vcmpgewi	大于等于比较	int	intv8	intv8/lit8	—
simd_vcmpeqw simd_vcmpeqwi	等于比较	intv8	intv8	intv8/lit8	—
simd_vcmplew	小于等于比较	intv8	intv8	intv8/lit8	—

simd_vcmplewi					
simd_vcmpltw simd_vcmpltwi	小于比较	intv8	intv8	intv8/lit8	—
simd_vcmpulew simd_vcmpulewi	无符号小于等于比较	intv8	intv8	intv8/lit8	—
simd_vcmpultw simd_vcmpultwi	无符号小于比较	intv8	intv8	intv8/lit8	—
simd_vrolw simd_vrolwi	循环左移	intv8	intv8	int/lit5	—
simd_vaddl simd_vaddli	长字向量加法	int256	int256	int256/lit8	—
simd_vsubl simd_vsubli	长字向量减法	int256	int256	int256/lit8	—
simd_sllow simd_sllowi	八倍字逻辑左移	int256	int256	int/lit8	—
simd_srlow simd_srlowi	八倍字逻辑右移	int256	int256	int/lit8	—
simd_ctpopow	八倍字数 1	int	int256	—	—
simd_ctlzow	八倍字数头 0	int	int256	—	—
simd_vucaddw simd_vucaddwi	字饱和加	intv8	intv8	intv8/lit8	—
simd_vucsubw simd_vucsubwi	字饱和减	intv8	intv8	intv8/lit8	—
simd_vucaddh simd_vucaddhi	半字饱和加	intv8	intv8	intv8/lit8	—
simd_vucsubh simd_vucsubhi	半字饱和减	intv8	intv8	intv8/lit8	—
simd_vucaddb simd_vucaddbi	字节饱和加	intv8	intv8	intv8/lit8	—
simd_vucsubb simd_vucsubbi	字节饱和减	intv8	intv8	intv8/lit8	—
simd_vseleqw simd_vseleqwi	等于比较选择	intv8	intv8	intv8	intv8/ lit5
simd_vselltw simd_vselltwi	小于比较选择	intv8	intv8	intv8	intv8/ lit5
simd_vsellew simd_vsellewi	小于等于比较选择	intv8	intv8	intv8	intv8/ lit5
simd_vsellbcw	低位为 0 比较选择	intv8	intv8	intv8	intv8/

simd_vsellbcwi					lit5
simd_smaxw	大值比较选择	intv8	intv8	intv8	
simd_umaxw	无符号大值比较选择	uintv8	uintv8	uintv8	
simd_sminw	小值比较选择	intv8	intv8	intv8	
simd_uminw	无符号小值比较选择	uintv8	uintv8	uintv8	

内部函数	**操作**	**参数说明**				
		VRe（返回值）	**VRav**	**VRbv**	**VRcv**	**VRdv**
simd_vlog	可重构逻辑	intv8/ int256	0xzz	intv8/ int256	intv8/ int256	intv8/ int256

说明：

a) VRav、VRbv、VRcv 分别为第一、第二、第三个参数，VRc 为返回值；可重构逻辑接口 VRav、VRbv、VRcv、VRdv 分别为第一、第二、第三、第四个参数，VRe 为返回值。

b) "－"表示对应接口的该参数无效

c) 比较指令、移位指令、字向量和长字向量加减法均支持运算符操作

d) simd_vlog 的 VRav 参数要求格式为 16 进制数，它指定参数里的 zz 域

函数名

simd_vaddw —— intv8 的加法

参数说明

[u]intv8 va;

[u]intv8 vb(int8 b);

返回值

[u]intv8 //intv8 类型的加法

功能说明

intv8 类型的加法，将 va 中的 8 个 int 类型的整型数据分别与 vb 中的 8 个 int 类型的整型数据或 8 位立即数 b 相加。可以使用"+"符号替换。

函数名

simd_vsubw —— intv8 的减法

参数说明

[u]intv8 va;

[u]intv8 vb（int8 b）;

返回值

[u]intv8 //intv8 类型的减法

功能说明

intv8 类型的减法，将 va 中的 8 个 int 类型的整型数据分别与 vb 中的 8 个 int 类型的整

型数据或 8 位立即数 b 相减。可以使用“-”符号替换。

函数名

simd_vandw —— intv8 的逻辑与

参数说明

[u]intv8 va;

[u]intv8 vb（int8 b）;

返回值

[u]intv8

功能说明

intv8 类型的逻辑与，将 va 中的 8 个 int 类型的整型数据分别与 vb 中的 8 个 int 类型的整型数据或 8 位立即数 b 进行逻辑与运算。可以使用“&”符号替换。

函数名

simd_vbicw —— intv8 的逻辑与非

参数说明

[u]intv8 va;

[u]intv8 vb（int8 b）;

返回值

[u]intv8

功能说明

intv8 类型的逻辑与非，将 va 中的 8 个 int 类型的整型数据分别与 vb 中的 8 个 int 类型的整型数据或 8 位立即数 b 进行逻辑与非运算。可以使用“&”和“~”符号替换。例如：

vc = simd_vbicw(va,vb);等价于：

vc = va & (~vb);

函数名

simd_vbisw —— intv8 的逻辑或

参数说明

[u]intv8 va;

[u]intv8 vb（int8 b）;

返回值

[u]intv8

功能说明

intv8 类型的逻辑或，将 va 中的 8 个 int 类型的整型数据分别与 vb 中的 8 个 int 类型的整型数据或 8 位立即数 b 进行逻辑或运算。可以使用“|”符号替换。

函数名

simd_vornotw —— intv8 的逻辑或非

参数说明

[u]intv8 va;

[u]intv8 vb（int8 b）;

返回值

[u]intv8

功能说明

intv8 类型的逻辑或非，将 va 中的 8 个 int 类型的整型数据分别与 vb 中的 8 个 int 类型的整型数据或 8 位立即数 b 进行逻辑与非运算。可以使用“|”和“~”符号替换。例如：

vc = simd_vornotw(va,vb);等价于：

vc = va | (~vb);

函数名

simd_vxorw —— intv8 的逻辑异或

参数说明

[u]intv8 va;

[u]intv8 vb（int8 b）;

返回值

[u]intv8

功能说明

intv8 类型的逻辑异或，将 va 中的 8 个 int 类型的整型数据分别与 vb 中的 8 个 int 类型的整型数据或 8 位立即数 b 进行逻辑异或运算。可以使用“^”符号替换。

函数名

simd_vsllw —— intv8 的逻辑左移

simd_vsllwi —— intv8 的逻辑左移（立即数格式）

参数说明

[u]intv8 va;

int/lit5 b;

返回值

[u]intv8 //intv8 类型的逻辑左移

功能说明

intv8 类型的逻辑左移，将 va 中的 8 个 int 类型的整型数据进行逻辑左移，移出的空位用“0”补充，八个数移位的位数相同，由 b 中的最低 5 位或者立即数决定。可以使用“<<”符号替换。

函数名

simd_vsrlw —— intv8 的逻辑右移

simd_vsrlwi —— intv8 的逻辑右移（立即数格式）

参数说明

[u]intv8 va;

int/lit5 b;

返回值

[u]intv8 //intv8 类型的逻辑右移

功能说明

intv8 类型的逻辑右移，将 va 中的 8 个 int 类型的整型数据进行逻辑右移，移出的空位用"0"补充，八个数移位的位数相同，由 b 中的最低 5 位或者 5 位立即数决定。可以使用">>"符号替换。使用">>"符号的时候，如果操作数是有符号的，生成算术右移指令，如果是无符号的，生成逻辑右移指令。

函数名

simd_vsraw —— intv8 的算术右移

simd_vsrawi —— intv8 的算术右移（立即数格式）

参数说明

[u]intv8 va;

int/lit5 b;

返回值

[u]intv8 //intv8 类型的算术右移

功能说明

intv8 类型的算术右移，将 va 中的 8 个 int 类型的整型数据进行算术右移，移出的空位分别用 8 个字的符号位填充，八个数移位的位数相同，由 b 中的最低 5 位或者 5 位立即数决定。可以使用">>"符号替换。使用">>"符号的时候，如果操作数是有符号的，生成算术右移指令，如果是无符号的，生成逻辑右移指令。

函数名

simd_veqvw —— intv8 的逻辑等效

参数说明

intv8 va;

intv8 vb（int8 b）;

返回值

intv8

功能说明

intv8 类型的逻辑等效，将 va 中的 8 个 int 类型的整型数据和 vb 中的 8 个 int 类型的整型数据或 8 位立即数 b 分别对应按位逻辑等效（同或），结果保存到返回值中。

函数名

simd_vcmpgew —— intv8 的等于比较运算

simd_vcmpgewi —— intv8 的等于比较运算（立即数格式）

参数说明

[u]intv8 va;

[u]intv8 vb(int8 b);

返回值

int c

功能说明

对Va中8个32位的字整数向量元素和Vb中对应8个32位的字整数向量元素（或8位立即数# b）分别进行有符号的大于等于比较，只要有一个32位字整数元素满足比较条件，则c写入“1”，否则返回“0”。

函数名

simd_vcmpeqw —— intv8 的等于比较运算

simd_vcmpeqwi —— intv8 的等于比较运算（立即数格式）

参数说明

[u]intv8 va;

[u]intv8 vb(int8 b);

返回值

[u]intv8 vc

功能说明

对Va中8个32位的字整数向量元素和Vb中对应8个32位的字整数向量元素（或8位立即数# b）分别进行等于比较运算，若满足指定关系，则Vc中对应的字元素为“1”，否则为“0”。

函数名

simd_vcmplew —— intv8 的小于等于比较运算

simd_vcmplewi —— intv8 的小于等于比较运算（立即数格式）

参数说明

[u]intv8 va;

[u]intv8 vb(int8 b);

返回值

[u]intv8 vc

功能说明

对Va中8个32位的字整数向量元素和Vb中对应8个32位的字整数向量元素（或8位立即数# b）分别进行小于等于比较运算，若满足指定关系，则Vc中对应的字元素为“1”，否则为“0”。

函数名

simd_vcmpltw —— intv8 的小于比较运算

simd_vcmpltwi —— intv8 的小于比较运算（立即数格式）

参数说明

[u]intv8 va;

[u]intv8 vb(int8 b);

返回值

[u]intv8 vc

功能说明

对Va中8个32位的字整数向量元素和Vb中对应8个32位的字整数向量元素（或8位立即数# b）分别进行小于比较运算，若满足指定关系，则Vc中对应的字元素为"1"，否则为"0"。

函数名

simd_vcmpulew —— intv8 的无符号小于等于比较运算

simd_vcmpulewi —— intv8 的无符号小于等于比较运算（立即数格式）

参数说明

[u]intv8 va;

[u]intv8 vb(int8 b);

返回值

[u]intv8 vc

功能说明

对Va中8个32位的字整数向量元素和Vb中对应8个32位的字整数向量元素（或8位立即数# b）分别进行无符号小于等于比较运算，若满足指定关系，则Vc中对应的字元素为"1"，否则为"0"。

函数名

simd_vcmpultw —— intv8 的无符号小于比较运算

simd_vcmpultwi —— intv8 的无符号小于比较运算（立即数格式）

参数说明

[u]intv8 va;

[u]intv8 vb(int8 b);

返回值

[u]intv8 vc

功能说明

对Va中8个32位的字整数向量元素和Vb中对应8个32位的字整数向量元素（或8位立即数# b）分别进行无符号小于比较运算，若满足指定关系，则Vc中对应的字元素为"1"，否则为"0"。

函数名

simd_vrolw —— intv8 的循环左移

simd_vrolwi —— intv8 的循环左移（立即数格式）

参数说明

[u]intv8 va;

int/lit5 b;

返回值

[u]intv8

功能说明

intv8类型的循环左移，将va中的8个int类型的整型数据进行循环左移，移出的空位用移出位补充，八个数移位的位数相同，由b的低5位或5位立即数决定。

函数名

simd_sllow —— int256 的逻辑左移

simd_sllowi —— int256 的逻辑左移（立即数格式）

参数说明

[u]int256 va;

int b; （lit8 b）

返回值

[u]int256

功能说明

将va中256位整体左移，移出的空位用“0”填充，移位位数由b低8位或8位立即数确定，结果写入Vc中。

函数名

simd_srlow —— int256 的逻辑右移

simd_srlowi —— int256 的逻辑右移（立即数格式）

参数说明

[u]int256 va;

int b; （lit8 b）

返回值

[u]int256

功能说明

将va中256位整体右移，移出的空位用“0”填充，移位位数由b低8位或8位立即数确定，结果写入Vc中。

函数名

simd_vaddl —— 长字向量加法

simd_vaddli—— 长字向量加法（立即数格式）

参数说明

[u]int256 va;

[u]int256 vb(int8 b);

返回值

[u]int256

功能说明

将Va和Vb中4个64位长字（或8位立即数b）分别对应相加，结果保存到Vc中对应位置。

函数名

simd_vsubl —— 长字向量减法

simd_vsubli —— 长字向量减法（立即数格式）

参数说明

[u]int256 va;

[u]int256 vb(int8 b);

返回值

[u]int256

功能说明

将Va和Vb中4个64位长字（或8位立即数b）分别对应相减，结果保存到Vc中对应位置。

函数名

simd_ctpopow —— int256 的数 1

参数说明

[u]int256 va;

返回值

int vc

功能说明

返回va中的256位八倍字整数中“1”的个数。

函数名

simd_ctlzow —— int256 的数头 0

参数说明

[u]int256 va;

返回值

int vc

功能说明

返回Va中的256位八倍字整数从最高位起连续“0”的个数。

函数名

simd_vucaddw —— intv8 的饱和加运算

simd_vucaddwi —— intv8 的饱和加运算（立即数格式）

参数说明

[u]intv8 va;

[u]intv8 vb(int8 b);

返回值

[u]intv8 vc

功能说明

将Va和Vb中对应的8个32位字整数（或8位立即数#b零扩展的字整数）分别进行32位的有符号加，取低32位结果保存到Vc。若结果产生上溢，则将对应的32位结果置为0x7fff,ffff；若产生下溢，则将对应的32位结果置为0x8000,0000。

函数名

simd_vucsubw —— intv8 的饱和减运算

simd_vucsubwi —— intv8 的饱和减运算（立即数格式）

参数说明

[u]intv8 va;

[u]intv8 vb(int8 b);

返回值

[u]intv8 vc

功能说明

将Va和Vb中对应的8个32位字整数（或8位立即数#b零扩展的字整数）分别进行32位的有符号减，取低32位结果保存到Vc。若结果产生上溢，则将对应的32位结果置为0x7fff,ffff；若产生下溢，则将对应的32位结果置为0x8000,0000。

函数名

simd_vucaddh —— 有符号半字向量的饱和加运算

simd_vucaddhi —— 有符号半字向量的饱和加运算（立即数格式）

参数说明

[u]intv8 va;

[u]intv8 vb(int8 b);

返回值

[u]intv8 vc

功能说明

将Va和Vb中对应的16个16位半字整数（或8位立即数#b零扩展的半字整数）分别进行16位的有符号加，取低16位结果保存到Vc。若结果产生上溢，则将对应的16位结果置为0x7fff；若产生下溢，则将对应的16位结果置为0x8000。

函数名

simd_vucsubh —— 有符号半字向量的饱和减运算

simd_vucsubhi —— 有符号半字向量的饱和减运算（立即数格式）

参数说明

[u]intv8 va;

[u]intv8 vb(int8 b);

返回值

[u]intv8 vc

功能说明

将Va和Vb中对应的16个16位半字整数（或8位立即数#b零扩展的半字整数）分别进行16位的有符号减，取低16位结果保存到Vc。若结果产生上溢，则将对应的16位结果置为0x7fff；若产生下溢，则将对应的16位结果置为0x8000。

函数名

simd_vucaddb —— 有符号字节向量的饱和加运算

simd_vucaddbi —— 有符号字节向量的饱和加运算（立即数格式）

参数说明

[u]intv8 va;

[u]intv8 vb(int8 b);

返回值

[u]intv8 vc

功能说明

将Va和Vb中对应的32个8位字节整数（或8位立即数#b）分别进行8位的有符号加，取低8位结果保存到Vc。若结果产生上溢，则将对应的8位结果置为0x7f；若产生下溢，则将对应的8位结果置为0x80。

函数名

simd_vucsubb —— 有符号字节向量的饱和减运算

simd_vucsubbi —— 有符号字节向量的饱和减运算（立即数格式）

参数说明

[u]intv8 va;

[u]intv8 vb(int8 b);

返回值

[u]intv8 vc

功能说明

将Va和Vb中对应的32个8位字节整数（或8位立即数#b）分别进行8位的有符号减，取低8位结果保存到Vc。若结果产生上溢，则将对应的8位结果置为0x7f；若产生下溢，则将对应的8位结果置为0x80。

函数名

simd_vseleqw ——intv8 的等于选择运算

simd_vseleqwi ——intv8 的等于选择运算（立即数格式）

参数说明

[u]intv8 va;

[u]intv8 vb;

[u]intv8 vc(int5 b);

返回值

[u]intv8 vd

功能说明

对Va中每个字元素与0进行等于比较测试，若条件成立，则将向量Vb对应的字元素写入Vd的对应位置，否则将向量Vc对应的字元素（或“0”扩展5位立即数#c形成的字整数）写入Vd的对应位置。

函数名

simd_vsellew ——intv8 的小于等于选择运算

simd_vsellewi ——intv8 的小于等于选择运算（立即数格式）

参数说明

[u]intv8 va;

[u]intv8 vb;

[u]intv8 vc(int5 b);

返回值

[u]intv8 vd

功能说明

对Va中每个字元素与0进行小于等于比较测试，若条件成立，则将向量Vb对应的字元素写入Vd的对应位置，否则将向量Vc对应的字元素（或“0”扩展5位立即数#c形成的字整数）写入Vd的对应位置。

函数名

simd_vselltw ——intv8 的小于选择运算

simd_vselltwi ——intv8 的小于选择运算（立即数格式）

参数说明

[u]intv8 va;

[u]intv8 vb;

[u]intv8 vc(int5 b);

返回值

[u]intv8 vd

功能说明

对Va中每个字元素与0进行小于比较测试，若条件成立，则将向量Vb对应的字元素写入Vd的对应位置，否则将向量Vc对应的字元素（或“0”扩展5位立即数#c形成的字整数）写

入Vd的对应位置。

函数名

simd_vsellbcw ——intv8 的低位为 0 选择运算

simd_vsellbcwi ——intv8 的低位为 0 选择运算（立即数格式）

参数说明

[u]intv8 va;

[u]intv8 vb;

[u]intv8 vc(int5 b);

返回值

[u]intv8 vd

功能说明

对Va中每个字元素的低位进行测试，若为0，则将向量Vb对应的字元素写入Vd的对应位置，否则将向量Vc对应的字元素（或“0”扩展5位立即数#c形成的字整数）写入Vd的对应位置。

函数名

simd_vlog —— intv8 的可重构逻辑运算

参数说明

zz;

[u]intv8 va;

[u]intv8 vb;

[u]intv8 vc;

返回值

[u]intv8 vd

功能说明

将Va、Vb和Vc中对应的每一位，组成3位的二进制数，从两位16进制参数zz的8位二进制值中选出一位，写入Vd的对应位中。

函数名

simd_smaxw ——intv8 的大值比较选择

simd_umaxw ——uintv8 的无符号大值比较选择

参数说明

[u]intv8 va;

[u]intv8 vb;

返回值

[u]intv8 vc

功能说明

对Va与Vb中每个字元素进行大于比较测试，若条件成立，则将向量Va对应的字元素写

入Vc的对应位置，否则将向量Vb对应的字元素写入Vc的对应位置。

函数名

simd_sminw ——intv8 的小值比较选择

simd_uminw ——uintv8 的无符号小值比较选择

参数说明

[u]intv8 va;

[u]intv8 vb;

返回值

[u]intv8 vc

功能说明

对Va与Vb中每个字元素进行小于比较测试，若条件成立，则将向量Va对应的字元素写入Vc的对应位置，否则将向量Vb对应的字元素写入Vc的对应位置。

5.2.3 浮点向量运算函数接口

表 5-3 64*4 浮点运算操作函数

内部函数	操作	参数说明			
		VRc（返回值）	VRav	VRbv	VRcv
simd_vadds	+	floatv4	floatv4	floatv4	/
simd_vsubs	-	floatv4	floatv4	floatv4	/
simd_vmuls	*	floatv4	floatv4	floatv4	/
simd_vmas	乘加	floatv4	floatv4	floatv4	floatv4
simd_vmss	乘减	floatv4	floatv4	floatv4	floatv4
simd_vnmas	负乘加	floatv4	floatv4	floatv4	floatv4
simd_vnmss	负乘减	floatv4	floatv4	floatv4	floatv4
simd_vaddd	+	doublev4	doublev4	doublev4	/
simd_vsubd	-	doublev4	doublev4	doublev4	/
simd_vmuld	*	doublev4	doublev4	doublev4	/
simd_vmad	乘加	doublev4	doublev4	doublev4	doublev4
simd_vmsd	乘减	doublev4	doublev4	doublev4	doublev4
simd_vnmad	负乘加	doublev4	doublev4	doublev4	doublev4
simd_vnmsd	负乘减	doublev4	doublev4	doublev4	doublev4
simd_vfseleqs simd_vfseleqd	条件选择	floatv4 doublev4	floatv4 doublev4	floatv4 doublev4	floatv4 doublev4
simd_vfsellts simd_vfselltd		floatv4 doublev4	floatv4 doublev4	floatv4 doublev4	floatv4 doublev4
simd_vfselles		floatv4	floatv4	floatv4	floatv4

simd_vfselled		doublev4	doublev4	doublev4	doublev4
simd_vcpyss simd_vcpysd	符号拷贝	floatv4 doublev4	floatv4 doublev4	floatv4 doublev4	/
simd_vcpyses simd_vcpysed		floatv4 doublev4	floatv4 doublev4	floatv4 doublev4	/
simd_vcpysns simd_vcpysnd		floatv4 doublev4	floatv4 doublev4	floatv4 doublev4	/
simd_vdivs	除法	floatv4	floatv4	floatv4	—
simd_vdivd		doublev4	doublev4	doublev4	—
simd_vsqrts	求平方根	floatv4	floatv4	—	—
simd_vsqrtd		doublev4	doublev4	—	—
simd_vfcmpeqs simd_vfcmpeqd	等于比较	floatv4 doublev4	floatv4 doublev4	floatv4 doublev4	—
simd_vfcmples simd_vfcmpled	小于等于比较	floatv4 doublev4	floatv4 doublev4	floatv4 doublev4	—
simd_vfcmplts simd_vfcmpltd	小于比较	floatv4 doublev4	floatv4 doublev4	floatv4 doublev4	—
simd_vfcmpuns simd_vfcmpund	无序比较	floatv4 doublev4	floatv4 doublev4	floatv4 doublev4	—
simd_vsllsn simd_vslldn (n=1,2,3)	浮点整体 逻辑左移	floatv4 doublev4	floatv4 doublev4	—	—
simd_vsrlsn simd_vsrldn (n=1,2,3)	浮点整体 逻辑右移	floatv4 doublev4	floatv4 doublev4	—	—
simd_smaxs simd_smaxd	大值比较选择	floatv4 doublev4	floatv4 doublev4	floatv4 doublev4	—
simd_smins simd_smind	小值比较选择	floatv4 doublev4	floatv4 doublev4	floatv4 doublev4	—

说明：

a) VRav、VRbv、VRcv 分别为第一、第二、第三个参数，VRc 为返回值；

b) “—”表示对应接口的该参数无效

c) 加法、减法、乘法、除法、乘加类运算支持运算符操作，浮点逻辑移位不支持运算符操作。

函数名

simd_vaddd —— doublev4 的加法运算

simd_vadds —— floatv4 的加法运算

参数说明

doublev4/floatv4 va;

doublev4/floatv4 vb;

返回值

doublev4/floatv4

功能说明

加法运算，将 va 中的 4 个浮点数分别与 vb 中的 4 个浮点数进行加法运算。可以用“+”符号替换。

函数名

simd_vsubd —— doublev4 的减法运算

simd_vsubs —— floatv4 的减法运算

参数说明

doublev4/floatv4 va;

doublev4/floatv4 vb;

返回值

doublev4/floatv4

功能说明

减法运算，将 va 中的 4 个浮点数分别与 vb 中的 4 个浮点数进行减法运算。可以用“-”符号替换。

函数名

simd_vmuld —— doublev4 的乘法运算

simd_vmuls —— floatv4 的乘法运算

参数说明

doublev4/floatv4 va;

doublev4/floatv4 vb;

返回值

doublev4/floatv4

功能说明

乘法运算，将 va 中的 4 个浮点数分别与 vb 中的 4 个浮点数进行乘法运算。可以用“*”符号替换。

函数名

simd_vmad —— doublev4 的乘加运算

simd_vmas —— floatv4 的乘加运算

参数说明

doublev4/floatv4 va;

doublev4/floatv4 vb;

doublev4/floatv4 vc;

返回值

doublev4/floatv4

功能说明

乘加运算，将 va 中的 4 个浮点数和 vb 中的 4 个浮点数以及 vc 中的 4 个浮点数分别进行乘加运算。可以用“*”和“+”符号替换。

ret = va*vb + vc;

函数名

simd_vmsd —— doublev4 的乘减运算

simd_vmss —— floatv4 的乘减运算

参数说明

doublev4/floatv4 va;

doublev4/floatv4 vb;

doublev4/floatv4 vc;

返回值

doublev4/floatv4

功能说明

乘减运算，将 va 中的 4 个浮点数和 vb 中的 4 个浮点数以及 vc 中的 4 个浮点数分别进行乘减运算。可以用“*”和“-”符号替换。

ret=va*vb-vc；

函数名

simd_vnmad —— doublev4 的负乘加运算

simd_vnmas —— floatv4 的负乘加运算

参数说明

doublev4/floatv4 va;

doublev4/floatv4 vb;

doublev4/floatv4 vc;

返回值

doublev4/floatv4

功能说明

负乘加运算，将 va 中的 4 个浮点数和 vb 中的 4 个浮点数以及 vc 中的 4 个浮点数分别进行负乘加运算。可以用“*”和“+”和“-”符号替换。

ret = -va*vb + vc；

函数名

simd_vnmsd —— doublev4 的负乘减运算

simd_vnmss —— floatv4 的负乘减运算

参数说明

doublev4/floatv4 va;

doublev4/floatv4 vb;

doublev4/floatv4 vc;

返回值

doublev4/floatv4

功能说明

负乘减运算，将 va 中的 4 个浮点数和 vb 中的 4 个浮点数以及 vc 中的 4 个浮点数分别进行负乘减运算。可以用“*”和“+”和“-”符号替换。

ret = -va*vb - vc；

函数名

simd_vfseleqd —— doublev4 的等于选择运算

simd_vfseleqs —— floatv4 的等于选择运算

参数说明

doublev4/floatv4 va;

doublev4/floatv4 vb;

doublev4/floatv4 vc;

返回值

doublev4/floatv4

功能说明

对 doublev4/floatv4 类型的参数 va 中的对应浮点向量元素进行比较，如果等于 0，则返回 vb 中对应浮点向量的值；否则返回 vc 中对应浮点向量的值。

函数名

simd_vfselltd —— doublev4 的小于选择运算

simd_vfsellts —— floatv4 的小于选择运算

参数说明

doublev4/floatv4 va;

doublev4/floatv4 vb;

doublev4/floatv4 vc;

返回值

doublev4/floatv4

功能说明

对 doublev4/floatv4 类型的参数 va 中的对应浮点向量元素进行比较，如果小于 0，则返回 vb 中对应浮点向量的值；否则返回 vc 中对应浮点向量的值。

函数名

simd_vfselled —— doublev4 的小于等于选择运算

simd_vfselles —— floatv4 的小于等于选择运算

参数说明

doublev4/floatv4 va;

doublev4/floatv4 vb;

doublev4/floatv4 vc;

返回值

doublev4/floatv4

功能说明

对 doublev4/floatv4 类型的参数 va 中的对应浮点向量元素进行比较，如果小于等于 0，则返回 vb 中对应浮点向量的值；否则返回 vc 中对应浮点向量的值。

函数名

simd_vcpysd —— doublev4 的拷贝符号

simd_vcpyss —— floatv4 的拷贝符号

参数说明

doublev4/floatv4 va;

doublev4/floatv4 vb;

返回值

doublev4/floatv4

功能说明

浮点拷贝符号，返回值中每个浮点向量的符号位是 va 中对应浮点向量的符号位，返回值中每个浮点向量的指数位和尾数位是 vb 中对应浮点向量的指数位和尾数位。

函数名

simd_vcpysed —— doublev4 的拷贝符号和指数

simd_vcpyses —— floatv4 的拷贝符号和指数

参数说明

doublev4/floatv4 va;

doublev4/floatv4 vb;

返回值

doublev4/floatv4

功能说明

浮点拷贝符号，返回值中每个浮点向量的符号位和指数位是 va 中对应浮点向量的符号位和指数位，返回值中每个浮点向量的尾数位是 vb 中对应浮点向量的尾数位。

函数名

simd_vcpysnd —— doublev4 的拷贝符号反码

simd_vcpysns —— floatv4 的拷贝符号反码

参数说明

doublev4/floatv4 va;

doublev4/floatv4 vb;

返回值

doublev4/floatv4

功能说明

浮点拷贝符号，返回值中每个浮点向量的符号位是 va 中对应浮点向量的符号位的反码，返回值中每个浮点向量的指数位和尾数位是 vb 中对应浮点向量的指数位和尾数位。

函数名

simd_vdivd —— doublev4 的除法运算

simd_vdivs —— floatv4 的除法运算

参数说明

doublev4/floatv4 va;

doublev4/floatv4 vb;

返回值

doublev4/floatv4

功能说明

除法运算，将 va 中的 4 个浮点数分别与 vb 中的 4 个浮点数进行除法运算。

函数名

simd_vsqrtd —— doublev4 的求平方根运算

simd_vsqrts —— floatv4 的求平方根运算

参数说明

doublev4/floatv4 va;

返回值

doublev4/floatv4

功能说明

求平方根运算，将 va 中的 4 个浮点数分别求平方根。

函数名

simd_vfcmpeqd —— doublev4 的等于比较运算

simd_vfcmpeqs —— floatv4 的等于比较运算

参数说明

doublev4/floatv4 va;

doublev4/floatv4 vb;

返回值

doublev4/floatv4 vc

功能说明

将Va和Vb中的对应浮点向量元素进行等于比较，如果条件成立，则将非“0”浮点值（1.0）写入vc的对应位置，否则将真“0”写入vc的对应位置。

函数名

simd_vfcmpled —— doublev4 的小于等于比较运算

simd_vfcmples —— floatv4 的小于等于比较运算

参数说明

doublev4/floatv4 va;

doublev4/floatv4 vb;

返回值

doublev4/floatv4 vc

功能说明

将Va和Vb中的对应浮点向量元素进行小于等于比较，如果条件成立，则将非“0”浮点值（1.0）写入vc的对应位置，否则将真“0”写入vc的对应位置。

函数名

simd_vfcmpltd —— doublev4 的小于比较运算

simd_vfcmplts —— floatv4 的小于比较运算

参数说明

doublev4/floatv4 va;

doublev4/floatv4 vb;

返回值

doublev4/floatv4 vc

功能说明

将Va和Vb中的对应浮点向量元素进行小于比较，如果条件成立，则将非“0”浮点值（1.0）写入vc的对应位置，否则将真“0”写入vc的对应位置。

函数名

simd_vfcmpund —— doublev4 的无序比较运算

simd_vfcmpuns —— floatv4 的无序比较运算

参数说明

doublev4/floatv4 va;

doublev4/floatv4 vb;

返回值

doublev4/floatv4 vc

功能说明

将Va和Vb中的对应浮点向量元素进行无序比较，如果条件成立，则将非“0”浮点值（1.0）写入vc的对应位置，否则将真“0”写入vc的对应位置。

函数名

simd_vslldn—— D 浮点整体逻辑左移运算

simd_vsllsn—— S 浮点整体逻辑左移运算

参数说明

doublev4/floatv4 va;

返回值

doublev4/floatv4

功能说明

将浮点向量参数 va 整体左移 n*64 位（n 的有效值为 1～3），同时空位补 0，组成并返回新的浮点向量。

该接口实际上对应3个接口，分别是simd_vslld1、simd_vslld2、simd_vslld3，比如：

vb=simd_vslld1(va)

表示 va 整体左移 64 位，同时低 64 位数据补 0，组成新的向量返回给 vb。

函数名

simd_vsrldn—— D 浮点整体逻辑右移运算

simd_vsrlsn—— S 浮点整体逻辑右移运算

参数说明

doublev4/floatv4 va;

返回值

doublev4/floatv4

功能说明

将浮点向量参数 va 整体右移 n*64 位（n 的有效值为 1～3），同时空位补 0，组成并返回新的浮点向量。

该接口实际上对应3个接口，分别是simd_vsrld1、simd_vsrld2、simd_vsrld3，比如：

vb=simd_vsrld1(va)

表示 va 整体右移 64 位，同时高 64 位数据补 0，组成新的向量返回给 vb。

函数名

simd_smaxs ——floatv4 的大值比较选择

simd_smaxd ——doublev4 的大值比较选择

参数说明

floatv4/doublev4 va;

floatv4/doublev4 vb;

返回值

floatv4/doublev4 vc

功能说明

对Va与Vb中每个浮点元素进行大于比较测试，若条件成立，则将向量Va对应的浮点元素写入Vc的对应位置，否则将向量Vb对应的浮点元素写入Vc的对应位置。

函数名

simd_smins ——floatv4 的小值比较选择

simd_smind ——doublev4 的小值比较选择

参数说明

floatv4/doublev4 va;

floatv4/doublev4 vb;

返回值

floatv4/doublev4 vc

功能说明

对Va与Vb中每个浮点元素进行小于比较测试，若条件成立，则将向量Va对应的浮点元素写入Vc的对应位置，否则将向量Vb对应的浮点元素写入Vc的对应位置。

5.2.4 数据整理函数接口

表 5-4 向量数据整理操作函数

内部函数	操作	参数说明			
		VRc（返回值）	**VRav**	**VRbv**	**VRcv**
simd_vinsw*n* n=(0,1,…,7)	字向量插入	intv8	int	intv8	/
simd_vinsfd*n* simd_vinsfs*n* n=(0,1,2,3)	浮点向量插入	floatv4 doublev4	float double	floatv4 doublev4	/
simd_vextw*n* n=(0,1,…,7)	字向量提取	int	intv8	/	/
simd_vextfs*n* simd_vextfd*n* n=(0,1,2,3)	浮点向量提取	float double	floatv4 doublev4	/	/
simd_vcpyw	字向量拷贝	intv8	int	/	/
simd_vcpyfs simd_vcpyfd	浮点向量拷贝	floatv4 doublev4	float double	/	/
simd_vconw	字向量元素拼接	intv8	intv8	intv8	void*
simd_vcons	浮点向量元素拼接	floatv4	floatv4	floatv4	void*

simd_vcond		doublev4	doublev4	doublev4	void*
simd_vshfw	字向量全混洗	intv8	intv8	intv8	double

说明：

a) simd_vinsw*n* 和 simd_vextw*n* 中 n 的取值只能是 0、1、2、3、4、5、6、7；simd_vinsfd*n* 和 simd_vextfd*n* 中 n 的取值只能是 0、1、2、3。

函数名

simd_vinsw*n*—— 字向量元素插入运算

参数说明

int a;

intv8 vb;

返回值

intv8

功能说明

将 32 位字整数 a 替代 vb 中由 n（有效值为 0～7）指定 32 位字元素，组成并返回新的字整数向量。

该接口实际上对应 8 个接口，分别是 simd_vinsw0、simd_vinsw1、simd_vinsw2、simd_vinsw3、simd_vinsw4、simd_vinsw5、simd_vinsw6、simd_vinsw7，比如：

vc=simd_vinsw2(a,vb)

表示将 32 位字整数 a 替代 vb 的[95:64]这 32 位的字元素，组成新的 vb 返回给 vc。

函数名

simd_vinsfd*n*—— D 浮点向量元素插入运算

simd_vinsfs*n*—— S 浮点向量元素插入运算

参数说明

double/float a;

doublev4/floatv4 vb;

返回值

doublev4/floatv4

功能说明

将浮点寄存器 a 的 64 位浮点数据替代 vb 中由 n（有效值为 0～3）指定的 64 位浮点元素，组成并返回新的浮点向量。

该接口实际上对应 4 个接口，分别是 simd_vinsfd0、simd_vinsfd1、simd_vinsfd2、simd_vinsfd3，比如：

vc=simd_vinsfd1(a,vb)

表示 a 的 64 位浮点数据替代 vb 的[127:64]这 64 位的浮点元素，组成新的 vb 返回给 vc。

函数名

simd_vextwn—— 字向量元素提取运算

参数说明

intv8 va;

返回值

int a

功能说明

将 va 中由 n（有效值为 0～7）指定的 32 位字元素，变为 64 位寄存器格式的字整数写入目的寄存器 a。

该接口实际上对应 8 个接口，分别是 simd_vextw0、simd_vextw1、simd_vextw2、simd_vextw3、simd_vextw4、simd_vextw5、simd_vextw6、simd_vextw7，比如：

a=simd_vextw2(va)

表示 va 的 [95:64]这 32 位的字元素，变为 64 位寄存器格式的字整数写入 a。

函数名

simd_vextfdn—— D 浮点向量元素提取运算

simd_vextfsn—— S 浮点向量元素提取运算

参数说明

doublev4/floatv4 va;

返回值

double/float a

功能说明

将 va 中由 n（有效值为 0～3）指定的 S 浮点或 D 浮点元素存入目的浮点寄存器 a 中。此指令可实现 S 浮点和 D 浮点的提取。

该接口实际上对应4个接口，分别是simd_vextfd0、simd_vextfd1、simd_vextfd2、simd_vextfd3，比如：

a=simd_vextfd1(va)

表示 va 的 [127:64]这 64 位的浮点元素存入 a 的<63:0>中。

函数名

simd_vcpyw—— 字向量元素拷贝运算

参数说明

int a;

返回值

intv8

功能说明

将 32 位字整数 a 复制成 8 个相同元素组成新的字整数向量写入目的寄存器。

函数名

simd_vcpyfd—— D 浮点向量元素拷贝运算

simd_vcpyfs—— S 浮点向量元素拷贝运算

参数说明

double/float a;

返回值

doublev4/floatv4

功能说明

将浮点数 a 复制成 4 个相同元素，组成新的浮点向量写入目的寄存器。

函数名

simd_vconw—— 字向量元素拼接

参数说明

intv8 va;

intv8 vb;

void*;

返回值

intv8 vc

功能说明

字向量拼接，以字分量为单位，va 存放需拼接的字向量低段，vb 存放需拼接的字向量高段，addr 存放不对界访存地址低 5 位（忽略其低 2 位）。根据不对界访存地址低 5 位，将 vb 的低段数据放入目标向量的高段，将 va 的高段数据放入目标向量的低段，拼接后形成新的字向量，存放到 vc 中。

函数名

simd_vcons—— S 浮点向量元素拼接

参数说明

floatv4 va;

floatv4 vb;

void*;

返回值

floatv4 vc

功能说明

S 浮点向量拼接，以 64 位分量为单位，va 存放需拼接的 S 浮点向量低段，vb 存放需拼接的 S 浮点向量高段，addr 存放不对界访存地址低 4 位（忽略其低 2 位）。根据不对界访存地址低 4 位，将 vb 的低段数据放入目标向量的高段，将 va 的高段数据放入目标向量的低段，拼接后形成新的 S 浮点向量，存放到 vc 中。

函数名

simd_vcond—— D 浮点向量元素拼接

参数说明

doublev4 va;

doublev4 vb;

void*;

返回值

doublev4 vc

功能说明

D 浮点向量拼接，以 64 位分量为单位，va 存放需拼接的 D 浮点向量低段，vb 存放需拼接的 D 浮点向量高段，addr 存放不对界访存地址低 5 位（忽略其低 3 位）。根据不对界访存地址低 5 位，将 vb 的低段数据放入目标向量的高段，将 va 的高段数据放入目标向量的低段，拼接后形成新的 D 浮点向量，存放到 vc 中。

函数名

simd_vshfw—— 字向量全混洗

参数说明

intv8 va;

intv8 vb;

double fc;

返回值

intv8 vd

功能说明

操作数寄存器为Va、Vb、Fc和Vd。Vd的八个32bit的数由Va和Vb装填，具体如何装填由Fc说明：

Fc的<31:28>指出vd的<255:224>来自Va或Vb的位置

Fc的<27:24>指出vd的<223:192>来自Va或Vb的位置

Fc的<23:20>指出vd的<191:160>来自Va或Vb的位置

Fc的<19:16>指出vd的<159:128>来自Va或Vb的位置

Fc的<15:12>指出vd的<127:96>来自Va或Vb的位置

Fc的<11:8>指出vd的<95:64>来自Va或Vb的位置

Fc的<7:4>指出vd的<63:32>来自Va或Vb的位置

Fc的<3:0>指出vd的<31:0>来自Va或Vb的位置

Fc 相应的 4bit 位置的最高位为“0”时，Vd 相应的数来自 Va；为“1”时，Vd 相应的数来自 Vb。Va/Vb 分成自然对界的 8 个 32bit，Fc 相应的 4bit 位置的低 3bit（0~7）用于指示是第几个 32bit 数据，为“0”表示第 0 个 32bit（[31:0]），依次类推。

5.2.5 赋值函数接口

表 5-5 赋/取值运算操作宏定义

宏定义	操作	参数说明

		返回值	参数
simd_set_intv8	赋值	32*8 向量类型	8 个 int 类型数
simd_set_uintv8	赋值	32*8 向量类型	8 个 unsigned int 类型数
simd_set_int256	赋值	256 位八倍字	4 个 long 类型数
simd_set_uint256	赋值	256 位八倍字	4 个 unsigned long 类型数
simd_set_floatv4	赋值	floatv4 类型	4 个单精浮点
simd_set_doublev4	赋值	doublev4 类型	4 个双精浮点

函数名

simd_set_doublev4—— doublev4 赋值函数

simd_set_floatv4 —— floatv4 赋值函数

参数说明

double/float a;

double/float b;

double/float c;

double/float d;

返回值

doublev4/floatv4

功能说明

doublev4/floatv4 类型的赋值函数，将 4 个 double/float 类型的数据传递到向量变量中。

例如：doublev4 va=simd_set_doublev4(1.0，2.0，3.0，4.0);

va 中的值是这样的：

4.0	3.0	2.0	1.0
255 192	191 128	127 64	63 0

函数名

simd_set_intv8 —— intv8 赋值函数

simd_set_uintv8 —— uintv8 赋值函数

参数说明

int/ unsigned int a;

int/ unsigned int b;

int/ unsigned int c;

int/ unsigned int d;

int/ unsigned int e;

int/ unsigned int f;

int/ unsigned int g;

int/ unsigned int h;

返回值

intv8/uintv8

功能说明

intv8/uintv8 类型的赋值函数，将 8 个 int/ unsigned int 类型的数据传递到向量变量中。

例如：va=simd_set_intv8(1,2,3,4,5,6,7,8);

va 中的值是这样的：

8	7	6	5	4	3	2	1
255 224	223 192	191 160	159 128	127 96	95 64	63 32	31 0

函数名

simd_set_int256 —— int256 赋值函数

simd_set_uint256 —— uint256 赋值函数

参数说明

long/ unsigned long a;

long/ unsigned long b;

long/ unsigned long c;

long/ unsigned long d;

返回值

int256/uint256

功能说明

int256/uint256 类型的赋值函数，将 4 个 long/unsigned long 类型的数据传递到向量变量中。

例如：va=simd_set_int256(1,2,3,4);

va 中的值是这样的：

4	3	2	1
255 192	191 128	127 64	63 0

5.2.6 打印函数接口

表 5-6 扩展类型的打印函数

函数	操作	参数说明	
		返回值	参数
simd_fprint_intv8	打印	/	FILE*, intv8
simd_fprint_uintv8	打印	/	FILE *, uint8
simd_fprint_int256	打印	/	FILE*, int256
simd_fprint_uint256	打印	/	FILE *, uint256
simd_fprint_floatv4	打印	/	FILE*, floatv4

simd_fprint_doublev4	打印	/	FILE*, doublev4
simd_print_intv8	打印	/	intv8
simd_print_uintv8	打印	/	uintv8
simd_print_int256	打印	/	int256
simd_print_uint256	打印	/	uint256
simd_print_floatv4	打印	/	floatv4
simd_print_doublev4	打印	/	doublev4
simd_fprint_intv8_X	打印	/	FILE*, intv8
simd_fprint_uintv8_X	打印	/	FILE *, uint8
simd_fprint_int256_X	打印	/	FILE*, int256
simd_fprint_uint256_X	打印	/	FILE *, uint256
simd_fprint_floatv4_X	打印	/	FILE*, floatv4
simd_fprint_doublev4_X	打印	/	FILE*, doublev4
simd_print_intv8_X	打印	/	intv8
simd_print_uintv8_X	打印	/	uintv8
simd_print_int256_X	打印	/	int256
simd_print_uint256_X	打印	/	uint256
simd_print_floatv4_X	打印	/	floatv4
simd_print_doublev4_X	打印	/	doublev4

函数名

simd_fprint_intv8 —— intv8 打印到文件

simd_fprint_uintv8 —— uintv8 打印到文件

simd_fprint_intv8_X —— intv8 以 16 进制打印到文件

simd_fprint_uintv8_X —— uintv8 以 16 进制打印到文件

参数说明

FILE *file;

intv8/uintv8 va;

返回值

无

功能说明

intv8/uintv8 类型的数据以 8 个 int/unsigned int 的形式打印到文件 file 中，其中 simd_fprint_intv8_X、simd_fprint_uintv8_X 以 16 进制格式打印。

例如：

```
va=simd_set_intv8(1,2,3,4,5,6,7,8);
simd_fprint_intv8(stderr, va);
simd_fprint_intv8_X (stderr, va);
```

打印的结果是这样的：

[8，7，6，5，4, 3, 2, 1]

[0x8，0x7，0x6，0x5，0x4, 0x3, 0x2, 0x1]

函数名

simd_fprint_int256 —— int256 打印到文件

simd_fprint_uint256—— uint256 打印到文件

simd_fprint_int256_X —— int256 以 16 进制打印到文件

simd_fprint_uint256_X—— uint256 以 16 进制打印到文件

参数说明

FILE *file;

int256/uint256 va;

返回值

无

功能说明

int256/uint256 类型的数据以 4 个 long/unsigned long 的形式打印到文件 file 中，其中 simd_fprint_int256_X、simd_fprint_uint256_X 以 16 进制格式打印。

例如： va=simd_set_uint256(1,2,3,4);

simd_fprint_uint256(stderr, va);

simd_fprint_uint256_X (stderr, va);

打印的结果是这样的：

[4, 3, 2, 1]

[0x4, 0x3, 0x2, 0x1]

函数名

simd_fprint_floatv4 —— floatv4 打印到文件

simd_fprint_doublev4 —— doublev4 打印到文件

simd_fprint_floatv4_X —— floatv4 以 16 进制打印到文件

simd_fprint_doublev4_X —— doublev4 以 16 进制打印到文件

参数说明

FILE *file;

floatv4/doublev4 va;

返回值

无

功能说明

floatv4/doublev4 类型的数据以 4 个 float/double 的形式打印到文件 file 中，其中 simd_fprint_floatv4_X、simd_fprint_doublev4_X 以十六进制格式打印。

例如： va=simd_set_floatv4(1.0,2.0,3.0,4.0);

simd_fprint_floatv4(stderr, va);

vb=simd_set_doublev4(1.0,2.0,3.0,4.0);

simd_fprint_floatv4_X (stderr, va);

simd_fprint_doublev4_X (stderr, vb);

打印的结果是这样的：

[3.0，4.0，2.0, 1.0]

[0x40800000, 0x40400000, 0x40000000, 0x3f800000]

[0x4010000000000000, 0x4008000000000000, 0x4000000000000000, 0x3ff0000000000000]

函数名

simd_print_intv8 —— intv8 打印到屏幕

simd_print_uintv8 —— uintv8 打印到屏幕

simd_print_intv8_X —— intv8 以 16 进制打印到屏幕

simd_print_uintv8_X —— uintv8 以 16 进制打印到屏幕

参数说明

intv8/uintv8 va;

返回值

无

功能说明

intv8/uintv8 类型的数据以 8 个 int/unsigned int 的形式打印到屏幕，其中 simd_print_intv8_X、simd_print_uintv8_X 以十六进制格式打印。

例如： va=simd_set_intv8(1,2,3,4,5,6,7,8);

simd_print_intv8(va);

simd_print_intv8_X (va);

打印的结果是这样的：

[8，7，6，5，4, 3, 2, 1]

[0x8，0x7，0x6，0x5，0x4, 0x3, 0x2, 0x1]

函数名

simd_print_int256 —— int256 打印到屏幕

simd_print_uint256—— uint256 打印到屏幕

simd_print_int256_X —— int256 以 16 进制打印到屏幕

simd_print_uint256_X—— uint256 以 16 进制打印到屏幕

参数说明

int256/uint256 va;

返回值

无

功能说明

int256/uint256 类型的数据以 4 个 long/unsigned long 的形式打印到屏幕，其中

simd_print_int256_X、simd_print_uint256_X 以十六进制格式打印。

例如： va=simd_set_int256(1,2,3,4);

simd_print_int256(va);

simd_print_int256_X (va);

打印的结果是这样的：

[4, 3, 2, 1]

[0x4, 0x3, 0x2, 0x1]

函数名

simd_print_floatv4 —— floatv4 打印到屏幕

simd_print_doublev4 —— doublev4 打印到屏幕

simd_print_floatv4_X —— floatv4 以 16 进制打印到屏幕

simd_print_doublev4_X —— doublev4 以 16 进制打印到屏幕

参数说明

floatv4/doublev4 va;

返回值

无

功能说明

floatv4/doublev4 类型的数据以 4 个 float/double 的形式打印到屏幕，其中 simd_print_floatv4_X、simd_print_doublev4_X 以十六进制格式打印。

例如： va=simd_set_floatv4(1.0,2.0,3.0,4.0);

simd_print_floatv4(va);

simd_print_floatv4_X (va);

vb=simd_set_doublev4(1.0,2.0,3.0,4.0);

simd_print_doublev4_X (vb);

打印的结果是这样的：

[3.0，4.0，2.0, 1.0]

[0x40800000, 0x40400000, 0x40000000, 0x3f800000]

[0x4010000000000000, 0x4008000000000000, 0x4000000000000000, 0x3ff0000000000000]

5.2.7 向量归约接口

表 5-7 向量归约函数

函数	操作	参数说明	
		返回值	参数
simd_reduc_plusw	加法归约	int	intv8
simd_reduc_pluss		float	floatv4

<table>
<tr><td>simd_reduc_plusd</td><td></td><td>double</td><td>doublev4</td></tr>
<tr><td>simd_reduc_umaxw</td><td>无符号最大值归约</td><td>unsigned int</td><td>uintv8</td></tr>
<tr><td>simd_reduc_smaxw</td><td rowspan="3">最大值归约</td><td>int</td><td>intv8</td></tr>
<tr><td>simd_reduc_smaxs</td><td>float</td><td>floatv4</td></tr>
<tr><td>simd_reduc_smaxd</td><td>double</td><td>doublev4</td></tr>
<tr><td>simd_reduc_uminw</td><td>无符号最小值归约</td><td>unsigned int</td><td>uintv8</td></tr>
<tr><td>simd_reduc_sminw</td><td rowspan="3">最小值归约</td><td>int</td><td>intv8</td></tr>
<tr><td>simd_reduc_smins</td><td>float</td><td>floatv4</td></tr>
<tr><td>simd_reduc_smind</td><td>double</td><td>doublev4</td></tr>
</table>

函数名

simd_reduc_plusw —— intv8 的加法归约

参数说明

intv8 va;

返回值

int

功能说明

intv8 类型的加法归约，将 va 中的 8 个 int 类型的整型数据归约求和。

函数名

simd_reduc_pluss —— floatv4 的加法归约

simd_reduc_plusd —— doublev4 的加法归约

参数说明

floatv4/doublev4 va;

返回值

float/double

功能说明

加法归约，将 va 中的 4 个浮点数据归约求和。

函数名

simd_reduc_smaxw —— intv8 的最大值归约

simd_reduc_umaxw —— uintv8 的无符号最大值归约

参数说明

intv8/uintv8 va;

返回值

int/unsigned int

功能说明

最大值归约，求 va 中的 8 个整型数据中的最大值。

函数名

simd_reduc_smaxs —— floatv4 的最大值归约

simd_reduc_smaxd —— doublev4 的最大值归约

参数说明

floatv4/doublev4 va;

返回值

float/double

功能说明

最大值归约，求 va 中的 4 个浮点数据中的最大值。

函数名

simd_reduc_sminw —— intv8 的最小值归约

simd_reduc_uminw —— uintv8 的无符号最小值归约

参数说明

intv8/uintv8 va;

返回值

int/unsigned int

功能说明

最小值归约，求 va 中的 8 个整型数据中的最小值。

函数名

simd_reduc_smins —— floatv4 的最小值归约

simd_reduc_smind —— doublev4 的最小值归约

参数说明

floatv4/doublev4 va;

返回值

float/double

功能说明

最小值归约，求 va 中的 4 个浮点数据中的最小值。

5.3 从核内部函数接口

5.3.1 装入/存储函数接口

表 5-8 装入/存储操作的宏定义

宏定义	操作	参数说明	
		VRav	**Rbv**
simd_load	装入	扩展类型	int*,long*,float*,double*,_Float16*
simd_loade	装入并扩展	float16v32	_Float16*

simd_store	存储	扩展类型	int*,float*,double*,_Float16*
simd_loadu	不对界装入	扩展类型	int* ,float*,double*,_Float16*
simd_load_u	强制对界装入	扩展类型	int* ,float*,double*,_Float16*
simd_storeu simd_store_u	不对界存储	扩展类型	int*,float*,double*,_Float16*

说明：当 Rbv 为 int * 类型时，VRav 只能是 intv16，当 Rbv 为 long *类型时，VRav 只能是 int512；Rbv 为 float *时，VRav 只能是 floatv8 类型；Rbv 为 double *类型时，VRav 只能是 doublev8 类型；Rbv 为_Float16*类型时，VRav 只能是 float16v32 类型。如果上述类型不匹配，编译器将报错。

函数名

simd_load —— 扩展类型装入

参数说明

a) intv16 类型的装入

[u]intv16 va;

[unsigned] int *addr;

b) int512 类型的装入

[u]int512 va;

[[unsigned] long *addr;] //int512 类型的装入

c) floatv8 类型的装入

floatv8 va;

float *addr; //floatv8 类型的装入

d) doublev8 类型的装入

doublev8 va;

double *addr; //doublev8 类型的装入

e) float16v32 类型的装入

float16v32 va;

_Float 16 *addr; //float16v32 类型的装入

装入结果

a) intv16 类型的装入

[u]intv16 //intv16 类型的装入

b) int512 类型的装入

[u]int512; //int512 类型的装入

c) floatv8 类型的装入

floatv8; //floatv8 类型的装入

d) doublev8 类型的装入

doublev8; //doublev8 类型的装入

e) float16v32 类型的装入

float16v32;//float16v32 类型的装入

功能说明

扩展类型的装入，将 64 字节（floatv8 类型是 32 字节）长度的数据从连续内存区域中装入到一个向量变量中。

函数名

simd_loade —— 半精度浮点类型装入并扩展

参数说明

float16v32 va;

_Float 16 *addr; //float16v32 类型的装入

装入结果

float16v32;//float16v32 类型的装入

功能说明

半精度浮点类型的装入并扩展，将 2 字节长度的数据从连续内存区域中装入到一个向量变量的低 16 位，并且将该数据扩展到高 496 位，使向量寄存器的 32 个半精度分量具有相同的值。从核仅支持半精度浮点向量的装入并扩展。

函数名

simd_store —— 扩展类型存储

参数说明

[u]intv16 va;

[unsigned] int *addr; //intv16 类型的存储

floatv8 va;

float *addr; //floatv8 类型的存储

doublev8 va;

double *addr; //doublev8 类型的存储

float16v32 va;

_Float *addr; //float16v32 类型的存储

返回值

无

功能说明

扩展类型的存储，将一个向量变量中的数据存储到 64 字节（floatv8 类型是 32 字节）连续内存区域中。

函数名

simd_loadu —— 扩展类型不对界装入

参数说明

a) intv16 类型的装入

[u]intv16 va;

[unsigned] int *addr; //intv16 类型的装入

b) floatv8 类型的装入

floatv8 va;

float *addr; //floatv8 类型的装入

c) doublev8 类型的装入

doublev8 va;

double *addr; //doublev8 类型的装入

d) float16v32 类型的装入

float16v32 va;

_Float16 *addr; //float16v32 类型的装入

装入结果

a) intv16 类型的装入

[u]intv16 //intv16 类型的装入

b) floatv8 类型的装入

floatv8; //floatv8 类型的装入

c) doublev8 类型的装入

doublev8; //doublev8 类型的装入

d) float16v32 类型的装入

float16v32; //float16v32 类型的装入

功能说明

扩展类型的不对界装入，将 64 字节（floatv8 类型是 32 字节）长度的数据，从任意地址（须满足 4 字节对界）开始的内存区域中装入到一个向量变量中。

函数名

simd_load_u —— 扩展类型强制对界装入

参数说明

a) intv16 类型的装入

[u]intv16 va;

[unsigned] int *addr; //intv16 类型的装入

b) floatv8 类型的装入

floatv8 va;

float *addr; //floatv8 类型的装入

c) doublev8 类型的装入

doublev8 va;

double *addr; //doublev8 类型的装入

d) float16v32 类型的装入

float16v32 va;

_Float16 *addr; //float16v32 类型的装入

装入结果

a) intv16 类型的装入

[u]intv16 //intv16 类型的装入

b) floatv8 类型的装入

floatv8; //floatv8 类型的装入

c) doublev8 类型的装入

doublev8; //doublev8 类型的装入

d) float16v32 类型的装入

float16v32; //float16v32 类型的装入

功能说明

扩展类型的强制对界装入，将不对界地址 addr 低位清零形成强制对界地址，将 64 字节（floatv8 类型是 32 字节）长度的数据从强制对界地址开始的连续内存区域中装入到一个向量变量中。

函数名

simd_storeu/ simd_store_u —— 扩展类型的不对界存储

参数说明

[u]intv16 va;

[unsigned] int *addr; //intv16 类型的存储

floatv8 va;

float *addr; //floatv8 类型的存储

doublev8 va;

double *addr; //doublev4 类型的存储

float16v32 va;

_Float16*addr; //float16v32 类型的存储

返回值

无

功能说明

扩展类型的不对界存储，将一个向量变量中的数据存储到 64 字节（floatv8 类型是 32 字节）连续内存区域中。

5.3.2 定点向量运算函数接口

表 5-9 整数运算操作函数

内部函数	操作	参数说明			
		返回值	VRav	VRbv	VRcv
simd_vaddw	+	intv16	intv16	intv16/lit8	—

<table>
<tr><td>simd_vsubw</td><td>-</td><td>intv16</td><td>intv16</td><td>intv16/lit8</td><td>—</td></tr>
<tr><td>simd_vsllw
simd_vsllwi</td><td><<</td><td>intv16</td><td>intv16</td><td>intv16/lit5</td><td>—</td></tr>
<tr><td>simd_vsrlw
simd_vsrlwi</td><td>>></td><td>intv16</td><td>intv16</td><td>intv16/lit5</td><td>—</td></tr>
<tr><td>simd_vsraw
simd_vsrawi</td><td>算术右移</td><td>intv16</td><td>intv16</td><td>intv16/lit5</td><td>—</td></tr>
<tr><td>simd_vrolw
simd_vrolwi</td><td>循环左移</td><td>intv16</td><td>intv16</td><td>intv16/lit5</td><td>—</td></tr>
<tr><td>simd_vaddl
simd_vaddli</td><td>长字向量加法</td><td>int512</td><td>int512</td><td>int512/lit8</td><td>—</td></tr>
<tr><td>simd_vsubl
simd_vsubli</td><td>长字向量减法</td><td>int512</td><td>int512</td><td>int512/lit8</td><td>—</td></tr>
<tr><td>simd_vornotw</td><td>或非</td><td>intv16</td><td>intv16</td><td>intv16/lit8</td><td>—</td></tr>
<tr><td>simd_vxorw</td><td>^</td><td>intv16</td><td>intv16</td><td>intv16/lit8</td><td>—</td></tr>
<tr><td>simd_vandw</td><td>&</td><td>intv16</td><td>intv16</td><td>intv16/lit8</td><td>—</td></tr>
<tr><td>simd_vbicw</td><td>与非</td><td>intv16</td><td>intv16</td><td>intv16/lit8</td><td>—</td></tr>
<tr><td>simd_vbisw</td><td>|</td><td>intv16</td><td>intv16</td><td>intv16/lit8</td><td>—</td></tr>
<tr><td>simd_vcmpeqw
simd_vcmpeqwi</td><td>等于比较</td><td>intv16</td><td>intv16</td><td>intv16/lit8</td><td>—</td></tr>
<tr><td>simd_vcmplew
simd_vcmplewi</td><td>小于等于比较</td><td>intv16</td><td>intv16</td><td>intv16/lit8</td><td>—</td></tr>
<tr><td>simd_vcmpltw
simd_vcmpltwi</td><td>小于比较</td><td>intv16</td><td>intv16</td><td>intv16/lit8</td><td>—</td></tr>
<tr><td>simd_vcmpulew
simd_vcmpulewi</td><td>无符号小于等于比较</td><td>uintv16</td><td>uintv16</td><td>uintv16/lit8</td><td>—</td></tr>
<tr><td>simd_vcmpultw
simd_vcmpultwi</td><td>无符号小于比较</td><td>uintv16</td><td>uintv16</td><td>uintv16/lit8</td><td>—</td></tr>
<tr><td>simd_sllx</td><td>八倍字逻辑左移</td><td>int512</td><td>int512</td><td>int/lit9</td><td></td></tr>
<tr><td>simd_srlx</td><td>八倍字逻辑右移</td><td>int512</td><td>int512</td><td>int/lit9</td><td></td></tr>
</table>

内部函数	**操作**	**参数说明**				
		VRe（返回值）	**VRav**	**VRbv**	**VRcv**	**VRdv**
simd_vlog2xi	可重构逻辑	intv16	intv16	intv16	0xzz	—
simd_vlog2xi_i	可重构逻辑	intv16	intv16	lit8	0xzz	—
simd_vlog2xx	可重构逻辑	int512	int512	int512	0xzz	—
simd_vlog2xx_i	可重构逻辑	int512	int512	lit8	0xzz	—

simd_vlog3ri	可重构逻辑	intv16	intv16	intv16	intv16	0xzz
simd_vlog3rx	可重构逻辑	int512	int512	int512	int512	0xzz

说明：

e) VRav、VRbv、VRcv 分别为第一、第二、第三个参数，VRc 为返回值；可重构逻辑接口 VRav、VRbv、VRcv、VRdv 分别为第一、第二、第三、第四个参数，VRe 为返回值。

f) “—”表示对应接口的该参数无效

g) 比较指令、移位指令、字向量和长字向量加减法均支持运算符操作

h) simd_vlog 的 VRav 参数要求格式为 16 进制数，它指定参数里的 zz 域

函数名

simd_vaddw —— intv16 的加法

参数说明

[u]intv16 va;

[u]intv16 vb(int8 b);

返回值

[u]intv16 //intv16 类型的加法

功能说明

intv16 类型的加法，将 va 中的 16 个 int 类型的整型数据分别与 vb 中的 16 个 int 类型的整型数据或 8 位立即数 b 相加。可以使用“+”符号替换。

函数名

simd_vsubw —— intv16 的减法

参数说明

[u]intv16 va;

[u]intv16 vb（int8 b）；

返回值

[u]intv16 //intv16 类型的减法

功能说明

intv16 类型的减法，将 va 中的 16 个 int 类型的整型数据分别与 vb 中的 16 个 int 类型的整型数据或 8 位立即数 b 相减。可以使用“-”符号替换。

函数名

simd_vsllw —— intv16 的逻辑左移

simd_vsllwi —— intv16 的逻辑左移（立即数格式）

参数说明

[u]intv16 va;

intv16/lit5 vb/b;

返回值

[u]intv16 //intv16 类型的逻辑左移

功能说明

intv16 类型的逻辑左移，将 va 中的 16 个 int 类型的整型数据进行逻辑左移，移出的空位用“0”补充，移位位数由 vb 中对应位置字分量的低 5 位或立即数 b 的低 5 位确定。可以使用“<<”符号替换。

函数名

simd_vsrlw —— intv16 的逻辑右移

simd_vsrlwi —— intv16 的逻辑右移（立即数格式）

参数说明

[u]intv16 va;

intv16/lit5 vb/b;

返回值

[u]intv16 //intv16 类型的逻辑右移

功能说明

intv16 类型的逻辑右移，将 va 中的 16 个 int 类型的整型数据进行逻辑右移，移出的空位用“0”补充，移位位数由 vb 中对应位置字分量的低 5 位或立即数 b 的低 5 位确定。可以使用“>>”符号替换。使用“>>”符号的时候，如果操作数是有符号的，生成算术右移指令，如果是无符号的，生成逻辑右移指令。

函数名

simd_vsraw —— intv16 的算术右移

simd_vsrawi —— intv16 的算术右移（立即数格式）

参数说明

[u]intv16 va;

intv16/lit5 vb/b;

返回值

[u]intv16 //intv16 类型的算术右移

功能说明

intv16 类型的算术右移，将 va 中的 16 个 int 类型的整型数据进行算术右移，移出的空位分别用 16 个 int 类型的整型数据的符号位填充，移位位数由 vb 中对应位置字分量的低 5 位或立即数 b 的低 5 位确定。可以使用“>>”符号替换。使用“>>”符号的时候，如果操作数是有符号的，生成算术右移指令，如果是无符号的，生成逻辑右移指令。

函数名

simd_vrolw —— intv16 的循环左移

simd_vrolwi —— intv16 的循环左移（立即数格式）

参数说明

[u]intv16 va;

intv16/lit5 vb/b;

返回值

[u]intv16

功能说明

intv16 类型的循环左移，将 va 中的 16 个 int 类型的整型数据进行循环左移，移出的空位用移出位补充，移位位数由 vb 中对应位置字分量的低 5 位或立即数 b 的低 5 位确定。

函数名

simd_vaddl —— 长字向量加法

simd_vaddli —— 长字向量加法（立即数格式）

参数说明

[u]int512 va;

[u]int512 vb(int8 b);

返回值

[u]int512

功能说明

将Va和Vb中8个64位长字（或8位立即数b）分别对应相加，结果保存到Vc中对应位置。

函数名

simd_vsubl —— 长字向量减法

simd_vsubli —— 长字向量减法（立即数格式）

参数说明

[u]int512 va;

[u]int512 vb(int8 b);

返回值

[u]int512

功能说明

将Va和Vb中8个64位长字（或8位立即数b）分别对应相减，结果保存到Vc中对应位置。

函数名

simd_vornotw —— intv16 的逻辑或非

参数说明

[u]intv16 va;

[u]intv16 vb（int16 b）;

返回值

[u]intv16

功能说明

intv16 类型的逻辑或非，将 va 中的 16 个 int 类型的整型数据分别与 vb 中的 16 个 int 类型的整型数据或 8 位立即数 b 进行逻辑与非运算。可以使用“|”和“~”符号替换。例如：

vc = simd_vornotw(va,vb);等价于：

vc = va | (~vb);

函数名

simd_vxorw —— intv16 的逻辑异或

参数说明

[u]intv16 va;

[u]intv16 vb（int16 b）；

返回值

[u]intv16

功能说明

intv16 类型的逻辑异或，将 va 中的 16 个 int 类型的整型数据分别与 vb 中的 16 个 int 类型的整型数据或 8 位立即数 b 进行逻辑异或运算。可以使用“^”符号替换。

函数名

simd_vbicw —— intv16 的逻辑与非

参数说明

[u]intv16 va;

[u]intv16 vb（int16 b）；

返回值

[u]intv16

功能说明

intv16 类型的逻辑与非，将 va 中的 16 个 int 类型的整型数据分别与 vb 中的 16 个 int 类型的整型数据或 8 位立即数 b 进行逻辑与非运算。可以使用“&”和“~”符号替换。例如：

vc = simd_vbicw(va,vb);等价于：

vc = va & (~vb);

函数名

simd_vbisw —— intv16 的逻辑或

参数说明

[u]intv16 va;

[u]intv16 vb（int8 b）；

返回值

[u]intv16

功能说明

intv16 类型的逻辑或，将 va 中的 16 个 int 类型的整型数据分别与 vb 中的 16 个 int 类型的整型数据或 8 位立即数 b 进行逻辑或运算。可以使用“|”符号替换。

函数名

simd_vlog2xi —— 2 操作数 intv16 的可重构逻辑运算

参数说明

[u]intv16 va;

[u]intv16 vb;

const int zz;

返回值

[u]intv16 vc

功能说明

将 Va 和 Vb（或 8 位立即数）中对应的每一位，组成 2 位的二进制数， zz 表示的 4 位二进制值中选出一位，写入 Vc 的对应位中。

对立即数格式的vlog2x指令，先将8位立即数零扩展为字整数，再分别与Va寄存器中字整数向量的每个元素进行运算。

函数名

simd_vlog2xi_i —— 2 操作数 intv16 的可重构逻辑运算

参数说明

[u]intv16 va;

int8 b;

const int zz;

返回值

[u]intv16 vc

功能说明

将 Va 和 8 位立即数 b 中对应的每一位，组成 2 位的二进制数， zz 表示的 4 位二进制值中选出一位，写入 Vc 的对应位中。

对立即数格式的vlog2x指令，先将8位立即数零扩展为字整数，再分别与Va寄存器中字整数向量的每个元素进行运算。

函数名

simd_vlog3ri —— 3 操作数 intv16 的可重构逻辑运算

参数说明

[u]intv16 va;

[u]intv16 vb;

[u]intv16 vc;

const int zz；

返回值

[u]intv16 vd

功能说明

根据zz值，从相应的配置寄存器取出真值表<TRUTH>，然后将Va、Vb、Vc中对应的每一位，组成3位的二进制数，从真值表<TRUTH>中的8位二进制值中选出一位，写入Vd的对应位中。SW处理器从核提供16个真值表配置寄存器（与LOG3R指令共用），zz有效值为0~15。zz超出有效范围，将引起非法指令异常。

函数名

simd_vlog2xx —— 2 操作数 int512 的可重构逻辑运算

参数说明

[u]int512 va;

[u]int512 vb;

const int zz；

返回值

[u]int512 vc

功能说明

将 Va 和 Vb（或 8 位立即数）中对应的每一位，组成 2 位的二进制数，从 zz 表示的 4 位二进制值中选出一位，写入 Vc 的对应位中。

对立即数格式的vlog2x指令，先将8位立即数零扩展为字整数，再分别与Va寄存器中字整数向量的每个元素进行运算。

函数名

simd_vlog2xx_i —— 2 操作数 int512 的可重构逻辑运算

参数说明

[u]int512 va;

int8 b;

const int zz；

返回值

[u]int512 vc

功能说明

将 Va 和 8 位立即数 b 中对应的每一位，组成 2 位的二进制数，从 zz 表示的 4 位二进制值中选出一位，写入 Vc 的对应位中。

对立即数格式的vlog2x指令，先将8位立即数零扩展为字整数，再分别与Va寄存器中字整数向量的每个元素进行运算。

函数名

simd_vlog3rx —— 3 操作数 int512 的可重构逻辑运算

参数说明

[u]int512 va;

[u]int512 vb;

[u]int512 vc;

const int zz;

返回值

[u]int512 vd

功能说明

根据zz值，从相应的配置寄存器取出真值表<TRUTH>，然后将Va、Vb、Vc中对应的每一位，组成3位的二进制数，从真值表<TRUTH>中的8位二进制值中选出一位，写入Vd的对应位中。SW处理器从核提供16个真值表配置寄存器（与LOG3R指令共用），zz有效值为0~15。zz超出有效范围，将引起非法指令异常。

函数名

simd_vcmpeqw —— intv16的等于比较运算

simd_vcmpeqwi —— intv16的等于比较运算（立即数格式）

参数说明

[u]intv16 va;

[u]intv16 vb(int8 b);

返回值

[u]intv16 vc

功能说明

对Va中16个32位的字整数向量元素和Vb中对应16个32位的字整数向量元素（或8位立即数#b）分别进行等于比较运算，若满足指定关系，则Vc中对应的字元素为“1”，否则为“0”。

函数名

simd_vcmplew —— intv16的小于等于比较运算

simd_vcmplewi —— intv16的小于等于比较运算（立即数格式）

参数说明

[u]intv16 va;

[u]intv16 vb(int8 b);

返回值

[u]intv16 vc

功能说明

对Va中16个32位的字整数向量元素和Vb中对应16个32位的字整数向量元素（或8位立即

数# b）分别进行小于等于比较运算，若满足指定关系，则Vc中对应的字元素为“1”，否则为“0”。

函数名

simd_vcmpltw —— intv16的小于比较运算

simd_vcmpltwi —— intv16的小于比较运算（立即数格式）

参数说明

[u]intv16 va;

[u]intv16 vb(int8 b);

返回值

[u]intv16 vc

功能说明

对Va中16个32位的字整数向量元素和Vb中对应16个32位的字整数向量元素（或8位立即数# b）分别进行小于比较运算，若满足指定关系，则Vc中对应的字元素为“1”，否则为“0”。

函数名

simd_vcmpulew —— intv16的无符号小于等于比较运算

simd_vcmpulewi —— intv16的无符号小于等于比较运算（立即数格式）

参数说明

[u]intv16 va;

[u]intv16 vb(int8 b);

返回值

[u]intv16 vc

功能说明

对Va中16个32位的字整数向量元素和Vb中对应16个32位的字整数向量元素（或8位立即数# b）分别进行无符号小于等于比较运算，若满足指定关系，则Vc中对应的字元素为“1”，否则为“0”。

函数名

simd_vcmpultw —— intv16的无符号小于比较运算

simd_vcmpultwi —— intv16的无符号小于比较运算（立即数格式）

参数说明

[u]intv16 va;

[u]intv16 vb(int8 b);

返回值

[u]intv16 vc

功能说明

对Va中16个32位的字整数向量元素和Vb中对应16个32位的字整数向量元素（或8位立即数# b）分别进行无符号小于比较运算，若满足指定关系，则Vc中对应的字元素为“1”，否则为“0”。

函数名

simd_sllx —— int512 的逻辑左移

simd_srlx —— int512 的逻辑右移

参数说明

[u]int512 va;

int b; （lit9 b）

返回值

[u]int512 vc

功能说明

将Va中512位整体左移/右移，移出的空位用“0”填充，移位位数由b低9位或9位立即数确定，结果写入Vc中。

5.3.3 浮点向量运算函数接口

表 5-10 向量浮点运算操作函数

内部函数	操作	参数说明			
		VRc（返回值）	VRav	VRbv	VRcv
simd_vadds	+	floatv8	floatv8	floatv8	—
simd_vsubs	-	floatv8	floatv8	floatv8	—
simd_vmuls	*	floatv8	floatv8	floatv8	—
simd_vmas	乘加	floatv8	floatv8	floatv8	floatv8
simd_vmss	乘减	floatv8	floatv8	floatv8	floatv8
simd_vnmas	负乘加	floatv8	floatv8	floatv8	floatv8
simd_vnmss	负乘减	floatv8	floatv8	floatv8	floatv8
simd_vaddd	+	doublev8	doublev8	doublev8	—
simd_vsubd	-	doublev8	doublev8	doublev8	—
simd_vmuld	*	doublev8	doublev8	doublev8	—
simd_vmad	乘加	doublev8	doublev8	doublev8	doublev8
simd_vmsd	乘减	doublev8	doublev8	doublev8	doublev8
simd_vnmad	负乘加	doublev8	doublev8	doublev8	doublev8
simd_vnmsd	负乘减	doublev8	doublev8	doublev8	doublev8

simd_vfseleqs simd_vfseleqd	条件选择	floatv8 doublev8	floatv8 doublev8	floatv8 doublev8	floatv8 doublev8
simd_vfsellts simd_vfselltd		floatv8 doublev8	floatv8 doublev8	floatv8 doublev8	floatv8 doublev8
simd_vfselles simd_vfselled		floatv8 doublev8	floatv8 doublev8	floatv8 doublev8	floatv8 doublev8
simd_vcpyss simd_vcpysd	符号拷贝	floatv8 doublev8	floatv8 doublev8	floatv8 doublev8	—
simd_vcpyses simd_vcpysed		floatv8 doublev8	floatv8 doublev8	floatv8 doublev8	—
simd_vcpysns simd_vcpysnd		floatv8 doublev8	floatv8 doublev8	floatv8 doublev8	—
simd_vfcmpeqs simd_vfcmpeqd	等于比较	floatv8 doublev8	floatv8 doublev8	floatv8 doublev8	—
simd_vfcmples simd_vfcmpled	小于等于比较	floatv8 doublev8	floatv8 doublev8	floatv8 doublev8	—
simd_vfcmplts simd_vfcmpltd	小于比较	floatv8 doublev8	floatv8 doublev8	floatv8 doublev8	—
simd_vfcmpuns simd_vfcmpund	无序比较	floatv8 doublev8	floatv8 doublev8	floatv8 doublev8	—
simd_smaxs simd_smaxd	大值比较选择	floatv8 doublev8	floatv8 doublev8	floatv8 doublev8	—
simd_smins simd_smind	小值比较选择	floatv8 doublev8	floatv8 doublev8	floatv8 doublev8	—
simd_vdivs simd_vdivd	除法	floatv8 doublev8	floatv8 doublev8	floatv8 doublev8	—
simd_vfrecs simd_vfrecd	近似倒数	floatv8 doublev8	floatv8 doublev8	—	—
simd_vfrecps simd_vfrecpd	精确倒数	floatv8 doublev8	floatv8 doublev8	—	—
simd_vsqrts simd_vsqrtd	平方根	floatv8 doublev8	floatv8 doublev8	—	—
simd_vrsqrts simd_vrsqrtd	平方根倒数	floatv8 doublev8	floatv8 doublev8	—	—
simd_vaddh	+	float16v32	float16v32	float16v32	—
simd_vsubh	-	float16v32	float16v32	float16v32	—
simd_vmulh	*	float16v32	float16v32	float16v32	—

simd_vdivh	除法	float16v32	float16v32	float16v32	—
simd_vsqrth	平方根	float16v32	float16v32	—	—
simd_vmah	乘加	float16v32	float16v32	float16v32	float16v32
simd_vmsh	乘减	float16v32	float16v32	float16v32	float16v32
simd_vnmah	负乘加	float16v32	float16v32	float16v32	float16v32
simd_vnmsh	负乘减	float16v32	float16v32	float16v32	float16v32
simd_vseleqh	条件选择	float16v32	float16v32	float16v32	float16v32
simd_vsellth		float16v32	float16v32	float16v32	float16v32
simd_vselleh		float16v32	float16v32	float16v32	float16v32
simd_vcpysh	符号拷贝	float16v32	float16v32	float16v32	—
simd_vcpyseh		float16v32	float16v32	float16v32	—
simd_vcpysnh		float16v32	float16v32	float16v32	—
simd_vfcmpeqh	等于比较	float16v32	float16v32	float16v32	—
simd_vfcmpleh	小于等于比较	float16v32	float16v32	float16v32	—
simd_vfcmplth	小于比较	float16v32	float16v32	float16v32	—
simd_vfcmpunh	无序比较	float16v32	float16v32	float16v32	—
simd_smaxh	大值比较选择	float16v32	float16v32	float16v32	—
simd_sminh	小值比较选择	float16v32	float16v32	float16v32	—
simd_vfcvtsh	向量转换	float16v32	floatv8	lit2	—
simd_vfcvths	向量转换	floatv8	float16v32	lit2	—

说明：

d) VRav、VRbv、VRcv 分别为第一、第二、第三个参数，VRc 为返回值；

e) “—”表示对应接口的该参数无效

f) 加法、减法、乘法、除法、乘加类运算支持运算符操作。

函数名

simd_vaddd —— doublev8 的加法运算

simd_vadds —— floatv8 的加法运算

参数说明

doublev8/floatv8 va;

doublev8/floatv8 vb;

返回值

doublev8/floatv8

功能说明

加法运算，将 va 中的 8 个浮点数分别与 vb 中的 8 个浮点数进行加法运算。可以用“+”符号替换。

函数名

simd_vsubd —— doublev8 的减法运算

simd_vsubs —— floatv8 的减法运算

参数说明

doublev8/floatv8 va;

doublev8/floatv8 vb;

返回值

doublev8/floatv8

功能说明

减法运算，将 va 中的 8 个浮点数分别与 vb 中的 8 个浮点数进行减法运算。可以用“-”符号替换。

函数名

simd_vmuld —— doublev8 的乘法运算

simd_vmuls —— floatv8 的乘法运算

参数说明

doublev8/floatv8 va;

doublev8/floatv8 vb;

返回值

doublev8/floatv8

功能说明

乘法运算，将 va 中的 8 个浮点数分别与 vb 中的 8 个浮点数进行乘法运算。可以用“*”符号替换。

函数名

simd_vmad —— doublev8 的乘加运算

simd_vmas —— floatv8 的乘加运算

参数说明

doublev8/floatv8 va;

doublev8/floatv8 vb;

doublev8/floatv8 vc;

返回值

doublev8/floatv8

功能说明

乘加运算，将 va 中的 8 个浮点数和 vb 中的 8 个浮点数以及 vc 中的 8 个浮点数分别进行乘加运算。可以用“*”和“+”符号替换。

ret = va*vb + vc;

函数名

simd_vmsd —— doublev8 的乘减运算

simd_vmss —— floatv8 的乘减运算

参数说明

doublev8/floatv8 va;

doublev8/floatv8 vb;

doublev8/floatv8 vc;

返回值

doublev8/floatv8

功能说明

乘减运算，将 va 中的 8 个浮点数和 vb 中的 8 个浮点数以及 vc 中的 8 个浮点数分别进行乘减运算。可以用“*”和“-”符号替换。

ret=va*vb-vc；

函数名

simd_vnmad —— doublev8 的负乘加运算

simd_vnmas —— floatv8 的负乘加运算

参数说明

doublev8/floatv8 va;

doublev8/floatv8 vb;

doublev8/floatv8 vc;

返回值

doublev8/floatv8

功能说明

负乘加运算，将 va 中的 8 个浮点数和 vb 中的 8 个浮点数以及 vc 中的 8 个浮点数分别进行负乘加运算。可以用“*”和“+”和“-”符号替换。

ret = -va*vb + vc；

函数名

simd_vnmsd —— doublev8 的负乘减运算

simd_vnmss —— floatv8 的负乘减运算

参数说明

doublev8/floatv8 va;

doublev8/floatv8 vb;

doublev8/floatv8 vc;

返回值

doublev8/floatv8

功能说明

负乘减运算，将 va 中的 8 个浮点数和 vb 中的 8 个浮点数以及 vc 中的 8 个浮点数分别进行负乘减运算。可以用“*”和“+”和“-”符号替换。

ret = -va*vb - vc；

函数名

simd_vfseleqd —— doublev8 的等于选择运算

simd_vfseleqs —— floatv8 的等于选择运算

参数说明

doublev8/floatv8 va;

doublev8/floatv8 vb;

doublev8/floatv8 vc;

返回值

doublev8/floatv8

功能说明

对 doublev8/floatv8 类型的参数 va 中的对应浮点向量元素进行比较，如果等于 0，则返回 vb 中对应浮点向量的值；否则返回 vc 中对应浮点向量的值。

函数名

simd_vfselltd —— doublev8 的小于选择运算

simd_vfsellts —— floatv8 的小于选择运算

参数说明

doublev8/floatv8 va;

doublev8/floatv8 vb;

doublev8/floatv8 vc;

返回值

doublev8/floatv8

功能说明

对 doublev8/floatv8 类型的参数 va 中的对应浮点向量元素进行比较，如果小于 0，则返回 vb 中对应浮点向量的值；否则返回 vc 中对应浮点向量的值。

函数名

simd_vfselled —— doublev8 的小于等于选择运算

simd_vfselles —— floatv8 的小于等于选择运算

参数说明

doublev8/floatv8 va;

doublev8/floatv8 vb;

doublev8/floatv8 vc;

返回值

doublev8/floatv8

功能说明

对 doublev8/floatv8 类型的参数 va 中的对应浮点向量元素进行比较，如果小于等于 0，则返回 vb 中对应浮点向量的值；否则返回 vc 中对应浮点向量的值。

函数名

simd_vcpysd —— doublev8 的拷贝符号

simd_vcpyss —— floatv8 的拷贝符号

参数说明

doublev8/floatv8 va;

doublev8/floatv8 vb;

返回值

doublev8/floatv8

功能说明

浮点拷贝符号，返回值中每个浮点向量的符号位是 va 中对应浮点向量的符号位，返回值中每个浮点向量的指数位和尾数位是 vb 中对应浮点向量的指数位和尾数位。

函数名

simd_vcpysed —— doublev8 的拷贝符号和指数

simd_vcpyses —— floatv8 的拷贝符号和指数

参数说明

doublev8/floatv8 va;

doublev8/floatv8 vb;

返回值

doublev8/floatv8

功能说明

浮点拷贝符号，返回值中每个浮点向量的符号位和指数位是 va 中对应浮点向量的符号位和指数位，返回值中每个浮点向量的尾数位是 vb 中对应浮点向量的尾数位。

函数名

simd_vcpysnd —— doublev8 的拷贝符号反码

simd_vcpysns —— floatv8 的拷贝符号反码

参数说明

doublev8/floatv8 va;

doublev8/floatv8 vb;

返回值

doublev8/floatv8

功能说明

浮点拷贝符号，返回值中每个浮点向量的符号位是 va 中对应浮点向量的符号位的反码，返回值中每个浮点向量的指数位和尾数位是 vb 中对应浮点向量的指数位和尾数位。

函数名

simd_vfcmpeqd —— doublev8 的等于比较运算

simd_vfcmpeqs —— floatv8 的等于比较运算

参数说明

doublev8/floatv8 va;

doublev8/floatv8 vb;

返回值

doublev8/floatv8 vc

功能说明

将 Va 和 Vb 中的对应浮点向量元素进行等于比较，如果条件成立，则将非“0”浮点值（1.0）写入 vc 的对应位置，否则将真“0”写入 vc 的对应位置。

函数名

simd_vfcmpled —— doublev8 的小于等于比较运算

simd_vfcmples —— floatv8 的小于等于比较运算

参数说明

doublev8/floatv8 va;

doublev8/floatv8 vb;

返回值

doublev8/floatv8 vc

功能说明

将 Va 和 Vb 中的对应浮点向量元素进行小于等于比较，如果条件成立，则将非“0”浮点值（1.0）写入 vc 的对应位置，否则将真“0”写入 vc 的对应位置。

函数名

simd_vfcmpltd —— doublev8 的小于比较运算

simd_vfcmplts —— floatv8 的小于比较运算

参数说明

doublev8/floatv8 va;

doublev8/floatv8 vb;

返回值

doublev8/floatv8 vc

功能说明

将 Va 和 Vb 中的对应浮点向量元素进行小于比较，如果条件成立，则将非“0”浮点值（1.0）写入 vc 的对应位置，否则将真“0”写入 vc 的对应位置。

函数名

simd_vfcmpund —— doublev8 的无序比较运算

simd_vfcmpuns —— floatv8 的无序比较运算

参数说明

doublev8/floatv8 va;

doublev8/floatv8 vb;

返回值

doublev8/floatv8 vc

功能说明

将 Va 和 Vb 中的对应浮点向量元素进行无序比较，如果条件成立，则将非“0”浮点值（1.0）写入 vc 的对应位置，否则将真“0”写入 vc 的对应位置。

函数名

simd_smaxs ——floatv8 的大值比较选择

simd_smaxd ——doublev8 的大值比较选择

参数说明

floatv8/doublev8 va;

floatv8/doublev8 vb;

返回值

floatv8/doublev8 vc

功能说明

对 Va 与 Vb 中每个浮点元素进行大于比较测试，若条件成立，则将向量 Va 对应的浮点元素写入 Vc 的对应位置，否则将向量 Vb 对应的浮点元素写入 Vc 的对应位置。

函数名

simd_smins ——floatv8 的小值比较选择

simd_smind ——doublev8 的小值比较选择

参数说明

floatv8/doublev8 va;

floatv8/doublev8 vb;

返回值

floatv8/doublev8 vc

功能说明

对 Va 与 Vb 中每个浮点元素进行小于比较测试，若条件成立，则将向量 Va 对应的浮点元素写入 Vc 的对应位置，否则将向量 Vb 对应的浮点元素写入 Vc 的对应位置。

函数名

simd_vdivs ——floatv8 的除法

simd_vdivd ——doublev8 的除法

参数说明

floatv8/doublev8 va;

floatv8/doublev8 vb;

返回值

floatv8/doublev8 vc

功能说明

除法运算，将 va 中的 8 个浮点数分别与 vb 中的 8 个浮点数进行除法运算。

函数名

simd_vfrecs ——floatv8 的近似倒数

simd_vfrecd ——doublev8 的近似倒数

参数说明

floatv8/doublev8 va;

返回值

floatv8/doublev8 vc

功能说明

求近似倒数运算，将 va 中的 8 个浮点数分别求近似倒数。

函数名

simd_vfrecps ——floatv8 的精确倒数

simd_vfrecpd ——doublev8 的精确倒数

参数说明

floatv8/doublev8 va;

返回值

floatv8/doublev8 vc

功能说明

求精确倒数运算，将 va 中的 8 个浮点数分别求精确倒数。

函数名

simd_vsqrts ——floatv8 的平方根

simd_vsqrtd ——doublev8 的平方根

参数说明

floatv8/doublev8 va;

返回值

floatv8/doublev8 vc

功能说明

求平方根运算，将 va 中的 8 个浮点数分别求平方根。

函数名

simd_vrsqrts ——floatv8 的平方根倒数

simd_vrsqrtd ——doublev8 的平方根倒数

参数说明

floatv8/doublev8 va;

返回值

floatv8/doublev8 vc

功能说明

求平方根倒数运算，将 va 中的 8 个浮点数分别求平方根倒数。

函数名

simd_vaddh —— float16v32 的加法运算

参数说明

float16v32 va;

float16v32 vb;

返回值

float16v32

功能说明

float16v32 类型的加法运算，将 va 中的 32 个_Float16 浮点数分别与 vb 中的 32 个_Float16 浮点数进行加法运算。可以用"+"符号替换。

函数名

simd_vsubh —— float16v32 的减法运算

参数说明

float16v32 va;

float16v32 vb;

返回值

float16v32

功能说明

float16v32 类型的减法运算，将 va 中的 32 个_Float16 浮点数分别与 vb 中的 32 个_Float16 浮点数进行减法运算。可以用"-"符号替换。

函数名

simd_vmulh —— float16v32 的乘法运算

参数说明

float16v32 va;

float16v32 vb;

返回值

float16v32

功能说明

float16v32 类型的乘法运算，将 va 中的 32 个_Float16 浮点数分别与 vb 中的 32 个_Float16 浮点数进行乘法运算。可以用“*”符号替换。

函数名

simd_vdivh ——float16v32 的除法

参数说明

float16v32 va;

float16v32 vb;

返回值

float16v32 vc

功能说明

除法运算，将 va 中的 32 个_Float16 浮点数分别与 vb 中的 32 个_Float16 浮点数进行除法运算。

函数名

simd_vsqrth ——float16v32 的平方根

参数说明

float16v32 va;

返回值

float16v32 vc

功能说明

求平方根运算，将 va 中的 32 个_Float16 浮点数分别求平方根。

函数名

simd_vmah —— float16v32 的乘加运算

参数说明

float16v32 va;

float16v32 vb;

float16v32 vc;

返回值

float16v32

功能说明

float16v32 类型的乘加运算，将 va 中的 32 个_Float16 浮点数和 vb 中的 32 个_Float16 浮点数以及 vc 中的 32 个_Float16 浮点数分别进行乘加运算。可以用“*”和“+”符号替换。

ret = va*vb + vc;

函数名

simd_vmsh —— float16v32 的乘减运算

参数说明

float16v32 va;

float16v32 vb;

float16v32 vc;

返回值

float16v32

功能说明

float16v32 类型的乘减运算，将 va 中的 32 个_Float16 浮点数和 vb 中的 32 个_Float16 浮点数以及 vc 中的 32 个_Float16 浮点数分别进行乘减运算。可以用“*”和“-”符号替换。

ret=va*vb-vc;

函数名

simd_vnmah —— float16v32 的负乘加运算

参数说明

float16v32 va;

float16v32 vb;

float16v32 vc;

返回值

float16v32

功能说明

float16v32 类型的负乘加运算，将 va 中的 32 个_Float16 浮点数和 vb 中的 32 个_Float16 浮点数以及 vc 中的 32 个_Float16 浮点数分别进行负乘加运算。可以用“*”和“+”和“-”符号替换。

ret = -va*vb + vc；

函数名

simd_vnmsh —— float16v32 的负乘减运算

参数说明

float16v32 va;

float16v32 vb;

float16v32 vc;

返回值

float16v32

功能说明

float16v32 类型的负乘减运算，将 va 中的 32 个_Float16 浮点数和 vb 中的 32 个_Float16 浮点数以及 vc 中的 32 个_Float16 浮点数分别进行负乘减运算。可以用“*”和“+”和“-”符号替换。

ret = -va*vb - vc；

函数名

simd_vseleqh —— float16v32 的等于选择运算

参数说明

float16v32 va;

float16v32 vb;

float16v32 vc;

返回值

float16v32

功能说明

对 float16v32 类型的参数 va 中的对应浮点向量元素进行比较，如果等于 0，则返回 vb 中对应浮点向量的值；否则返回 vc 中对应浮点向量的值。

函数名

simd_vsellth —— float16v32 的小于选择运算

参数说明

float16v32 va;

float16v32 vb;

float16v32 vc;

返回值

float16v32

功能说明

对 float16v32 类型的参数 va 中的对应浮点向量元素进行比较，如果小于 0，则返回 vb 中对应浮点向量的值；否则返回 vc 中对应浮点向量的值。

函数名

simd_vselleh —— float16v32 的小于等于选择运算

参数说明

float16v32 va;

float16v32 vb;

float16v32 vc;

返回值

float16v32

功能说明

对 float16v32 类型的参数 va 中的对应浮点向量元素进行比较，如果小于等于 0，则返回 vb 中对应浮点向量的值；否则返回 vc 中对应浮点向量的值。

函数名

simd_vcpysh —— float16v32 的拷贝符号

参数说明

float16v32 va;

float16v32 vb;

返回值

float16v32

功能说明

浮点拷贝符号，返回值中每个浮点向量的符号位是 va 中对应浮点向量的符号位，返回值中每个浮点向量的指数位和尾数位是 vb 中对应浮点向量的指数位和尾数位。

函数名

simd_vcpyseh —— float16v32 的拷贝符号和指数

参数说明

float16v32 va;

float16v32 vb;

返回值

float16v32

功能说明

浮点拷贝符号，返回值中每个浮点向量的符号位和指数位是 va 中对应浮点向量的符号位和指数位，返回值中每个浮点向量的尾数位是 vb 中对应浮点向量的尾数位。

函数名

simd_vcpysnh—— float16v32 的拷贝符号反码

参数说明

float16v32 va;

float16v32 vb;

返回值

float16v32

功能说明

浮点拷贝符号，返回值中每个浮点向量的符号位是 va 中对应浮点向量的符号位的反码，

返回值中每个浮点向量的指数位和尾数位是 vb 中对应浮点向量的指数位和尾数位。

函数名

simd_vfcmpeqh —— float16v32 的等于比较运算

参数说明

float16v32 va;

float16v32 vb;

返回值

float16v32 vc

功能说明

将 Va 和 Vb 中的对应浮点向量元素进行等于比较，如果条件成立，则将非“0”浮点值（1.0）写入 vc 的对应位置，否则将真“0”写入 vc 的对应位置。

函数名

simd_vfcmpleh —— float16v32 的小于等于比较运算

参数说明

float16v32 va;

float16v32 vb;

返回值

float16v32 vc

功能说明

将 Va 和 Vb 中的对应浮点向量元素进行小于等于比较，如果条件成立，则将非“0”浮点值（1.0）写入 vc 的对应位置，否则将真“0”写入 vc 的对应位置。

函数名

simd_vfcmplth —— float16v32 的小于比较运算

参数说明

float16v32 va;

float16v32 vb;

返回值

float16v32 vc

功能说明

将 Va 和 Vb 中的对应浮点向量元素进行小于比较，如果条件成立，则将非“0”浮点值（1.0）写入 vc 的对应位置，否则将真“0”写入 vc 的对应位置。

函数名

simd_vfcmpunh —— float16v32 的无序比较运算

参数说明

float16v32 va;

float16v32 vb;

返回值

float16v32 vc

功能说明

将 Va 和 Vb 中的对应浮点向量元素进行无序比较，如果条件成立，则将非“0”浮点值（1.0）写入 vc 的对应位置，否则将真“0”写入 vc 的对应位置。

函数名

simd_smaxh ——float16v32 的大值比较选择

参数说明

float16v32 va;

float16v32 vb;

返回值

float16v32 vc

功能说明

对 Va 与 Vb 中每个浮点元素进行大于比较测试，若条件成立，则将向量 Va 对应的浮点元素写入 Vc 的对应位置，否则将向量 Vb 对应的浮点元素写入 Vc 的对应位置。

函数名

simd_sminh ——float16v32 的小值比较选择

参数说明

float16v32 va;

float16v32 vb;

返回值

float16v32 vc

功能说明

对 Va 与 Vb 中每个浮点元素进行小于比较测试，若条件成立，则将向量 Va 对应的浮点元素写入 Vc 的对应位置，否则将向量 Vb 对应的浮点元素写入 Vc 的对应位置。

函数名

simd_vfcvtsh ——floatv8 向量到 float16v32 向量的转换

参数说明

floatv8 va;

lit2 b;

返回值

float16v32 vc

功能说明

va 中的 8 个 64 位单精度扩展格式浮点数，转换成 8 个 16 位半精度格式浮点数，然后根据立即数 b 的值写入 vc 指定位置，并将 vc 其它位置清 0。

例如： floatv8 va=simd_set_floatv8(1.0，2.0，3.0，4.0，5.0，6.0，7.0，8.0);

float16v32 vc = simd_vfcvtsh(va, 1);

simd_print_float16v32(vc);

打印的结果为： [0.0, 0.0, 8.0, 0.0, 0.0, 0.0, 7.0, 0.0, 0.0, 0.0, 6.0, 0.0, 0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 4.0, 0.0, 0.0, 0.0, 3.0, 0.0, 0.0, 0.0, 2.0, 0.0, 0.0, 0.0, 1.0, 0.0]

函数名

simd_vfcvths ——float16v32 向量到 floatv8 向量的转换

参数说明

float16v32 va;

lit2 b;

返回值

floatv8 vc

功能说明

根据立即数#b 的值，从 va 的每个 64 位中选择 1 个 16 位半精度浮点数分量，转换成 64 位的单精度浮点数扩展格式，将转换后的 8 个 64 位 SP 浮点数写入 vc。

例如： float16v32 va=simd_set_float16v32 (0.0,1.0,2.0,3.0,4.0,5.0,6.0,7.0,
8.0,9.0,10.0,11.0,12.0,13.0,14.0,15.0,
16.0,17.0,18.0,19.0,20.0,21.0,22.0,23.0,
24.0,25.0,26.0,27.0,28.0,29.0,30.0,31.0);

floatv8 vc = simd_vfcvths(va, 1);

simd_print_float16v32(vc);

打印的结果为： [29.0, 25.0, 21.0, 17.0, 13.0, 9.0, 5.0, 1.0]

5.3.4 数据整理函数接口

表 5-11 向量数据操作函数

内部函数	操作	参数说明			
		VRc（返回值）	VRav	VRbv	VRcv
simd_vinsw*n* n=(0,1,…,15)	字向量插入	intv16	int	intv16	/
simd_vinsfd*n* simd_vinsfs*n* n=(0,1…,7)	浮点向量插入	floatv8 doublev8	float double	floatv8 doublev8	/
simd_vextw*n* n=(0,1,…,15)	字向量提取	int	intv16	/	/

simd_vshfw	字向量混洗	intv16	intv16	intv16	intv16
simd_vextfs*n* simd_vextfd*n* n=(0,1…,7)	浮点向量提取	float double	floatv8 doublev8	/	/
simd_vcpyw	字向量拷贝	intv16	int	/	/
simd_vcpyfs simd_vcpyfd	浮点向量拷贝	floatv8 doublev8	float double	/	/
simd_vconw	字向量元素拼接	intv16	intv16	intv16	void*
simd_vcons	浮点向量元素拼接	floatv8	floatv8	floatv8	void*
simd_vcond		doublev8	doublev8	doublev8	void*
simd_vinsh	半精度向量插入	float16v32	_Float16	float16v32	int
simd_vexth	半精度向量提取	_Float16	float16v32	int	/
simd_vcpyh	半精度向量拷贝	float16v32	_Float16	/	/
simd_vshfh	半精度向量混洗	float16v32	float16v32	float16v32	intv16

说明：

a) simd_vinsw*n* 和 simd_vextw*n* 中 n 的取值只能是 0、1、2、3、4、5、6、7、8、9、10、11、12、13、14、15；simd_vinsfd*n* 和 simd_vextfd*n* 中 n 的取值只能是 0、1、2、3、4、5、6、7。

函数名

simd_vinsw*n*—— 字向量元素插入运算

参数说明

int a;

intv16 vb;

返回值

intv16

功能说明

将32位字整数a替代vb中由n（有效值为0～15）指定32位字元素，组成并返回新的字整数向量。

该接口实际上对应16个接口，分别是simd_vinsw0、simd_vinsw1、simd_vinsw2、simd_vinsw3、simd_vinsw4、simd_vinsw5、simd_vinsw6、simd_vinsw7、simd_vinsw8、simd_vinsw9、simd_vinsw10、simd_vinsw11、simd_vinsw12、simd_vinsw13、simd_vinsw14、simd_vinsw15，比如：

vc=simd_vinsw2(a,vb)

表示将32位字整数a替代vb的[95:64]这32位的字元素，组成新的vb返回给vc。

函数名

simd_vinsfd*n*—— D 浮点向量元素插入运算

simd_vinsfs*n*—— S 浮点向量元素插入运算

参数说明

double/float a;

doublev8/floatv8 vb;

返回值

doublev8/floatv8

功能说明

将浮点寄存器a的64位浮点数据替代vb中由n（有效值为0～7）指定的64位浮点元素，组成并返回新的浮点向量。

该接口实际上对应8个接口，分别是simd_vinsfd0、simd_vinsfd1、simd_vinsfd2、simd_vinsfd3、simd_vinsfd4、simd_vinsfd5、simd_vinsfd6、simd_vinsfd7，比如：

vc=simd_vinsfd1(a,vb)

表示a的64位浮点数据替代vb的[127:64]这64位的浮点元素，组成新的vb返回给vc。

函数名

simd_vextw*n*—— 字向量元素提取运算

参数说明

intv16 va;

返回值

int a

功能说明

将va中由n（有效值为0～15）指定的32位字元素，变为64位寄存器格式的字整数写入目的寄存器a。

该接口实际上对应16个接口，分别是simd_vextw0、simd_vextw1、simd_vextw2、simd_vextw3、simd_vextw4、simd_vextw5、simd_vextw6、simd_vextw7、simd_vextw8、simd_vextw9、simd_vextw10、simd_vextw11、simd_vextw12、simd_vextw13、simd_vextw14、simd_vextw15，比如：

a=simd_vextw2(va)

表示va的 [95:64]这32位的字元素，变为64位寄存器格式的字整数写入a。

函数名

simd_vextfd*n*—— D 浮点向量元素提取运算

simd_vextfs*n*—— S 浮点向量元素提取运算

参数说明

doublev8/floatv8 va;

返回值

double/float a

功能说明

将va中由n（有效值为0～7）指定的S浮点或D浮点元素存入目的浮点寄存器a中。此指令

可实现S浮点和D浮点的提取。

该接口实际上对应8个接口，分别是simd_vextfd0、simd_vextfd1、simd_vextfd2、simd_vextfd3、simd_vextfd4、simd_vextfd5、simd_vextfd6、simd_vextfd7，比如：

a=simd_vextfd1(va)

表示va的 [127:64]这64位的浮点元素存入a的<63:0>中。

函数名

simd_vcpyw—— 字向量元素拷贝运算

参数说明

int a;

返回值

intv16

功能说明

将32位字整数a复制成16个相同元素组成新的字整数向量写入目的寄存器。

函数名

simd_vcpyfd——D 浮点向量元素拷贝运算

simd_vcpyfs——S 浮点向量元素拷贝运算

参数说明

double/float a;

返回值

doublev8/floatv8

功能说明

将浮点数a复制成8个相同元素，组成新的浮点向量写入目的寄存器。

函数名

simd_vconw—— 字向量元素拼接

参数说明

intv16 va;

intv16 vb;

void* addr;

返回值

intv16 vc

功能说明

字向量拼接，以字分量为单位，va存放不对界高段，vb存放不对界低段，addr存放不对界地址。根据不对界地址低6位（忽略其低2位），将va的高段数据和vb的低段数据拼接后形成新的向量，存放到vc中。

函数名

simd_vcons——S 浮点向量元素拼接

参数说明

floatv8 va;

floatv8 vb;

void* addr;

返回值

floatv8 vc

功能说明

S浮点向量拼接，以64位分量为单位，va存放不对界高段，vb存放不对界低段，addr存放不对界地址。根据不对界地址低5位（忽略其低2位），将va的高段数据和vb的低段数据拼接后形成新的向量，存放到vc中

函数名

simd_vcond——D 浮点向量元素拼接

参数说明

doublev8 va;

doublev8 vb;

void* addr;

返回值

doublev8 vc

功能说明

D浮点向量拼接，以64位分量为单位，va存放不对界高段，vb存放不对界低段，addr存放不对界地址。根据不对界地址低6位（忽略其低2位），将va的高段数据和vb的低段数据拼接后形成新的向量，存放到vc中。

函数名

simd_vinsh—— 半精度浮点向量元素插入运算

参数说明

_Float16 a;

float16v32 vb;

const int c;

返回值

float16v32

功能说明

将16位半精度浮点数a替代vb中由c（有效值为0～31）指定的16位半精度浮点元素，组成并返回新的半精度浮点数向量。

函数名

simd_vexth——半精度浮点向量元素提取运算

参数说明

float16v32 va;

const int b;

返回值

_Float16 a

功能说明

将va中由b（有效值为0～31）指定的16位半精度浮点元素，变为16位寄存器格式的半精度浮点数写入目的寄存器a。

函数名

simd_vcpyh——半精度浮点向量元素拷贝运算

参数说明

_Float16 a;

返回值

float16v32

功能说明

将16位半精度浮点数a复制成32个相同元素组成新的半精度浮点数向量写入目的寄存器。

函数名

simd_vshfw—— 字向量全混洗

参数说明

intv16 va;

intv16 vb;

intv16 vc;

返回值

intv16 vd

功能说明

操作数寄存器为 Va、Vb、Vc 和 Vd。Vd 的十六个 32bit 的数由 Va 和 Vb 装填，具体如何装填由 Vc 说明：

Vc 的<79:75>指出 Vd 的<511:480>来自 Va 或 Vb 的位置

Vc 的<74:70>指出 Vd 的<479:448>来自 Va 或 Vb 的位置

……

Vc 的<14:10>指出 Vd 的<95:64>来自 Va 或 Vb 的位置

Vc 的<9:5>指出 Vd 的<63:32>来自 Va 或 Vb 的位置

Vc 的<4:0>指出 Vd 的<31:0>来自 Va 或 Vb 的位置

Vc 的 5bit 分量表示 Vd 对应分量的来源，5bit 分量的最高位为 0 时，Vd 相应的数来自 Va；为 1 时，Vd 相应的数来自 Vb。5bit 分量位置的低 4bit（0~15）含义为，Va/Vb 分成自然对界的 16 个 32bit，位置为 0 表示第 0 个 32bit（[31:0]），依次类推。

Vc 的 [511:80]位无意义。

函数名

simd_vshfh—— 半精度向量全混洗

参数说明

float16v32 va;

float16v32 vb;

intv16 vc;

返回值

float16v32 vd

功能说明

操作数寄存器为 Va、Vb、Vc 和 Vd。Vd 的 32 个 16bit 的数由 Va 和 Vb 装填，具体如何装填由 Vc 说明：

Vc 的<191:186>指出 Vd 的<511:496>来自 Va 或 Vb 的位置

Vc 的<185:180>指出 Vd 的<495:480>来自 Va 或 Vb 的位置

……

Vc 的<11:6>指出 Vd 的<31:16>来自 Va 或 Vb 的位置

Vc 的<5:0>指出 Vd 的<15:0>来自 Va 或 Vb 的位置

Vc 的[191:0]位，拆分成 32 个 6bit 位置分量。每个 6bit 位置分量的最高位为 0 时，Vd 相应的数来自 Va；为 1 时，Vd 相应的数来自 Vb。6bit 位置的低 5bit（0~31）含义为，Va/Vb 分成自然对界的 32 个 16bit，位置为 0 表示第 0 个 16bit（[15:0]），依次类推。

Vc 的[511:192]位无意义。

5.3.5 赋值函数接口

表 5-12 赋/取值运算操作宏定义

宏定义	操作	参数说明	
		返回值	参数
simd_set_intv16	赋值	intv16 类型	16 个 int 类型数
simd_set_uintv16	赋值	uintv16 类型	16 个 unsigned int 类型数
simd_set_int512	赋值	512 位八倍字	8 个 long
simd_set_uint512	赋值	512 位八倍字	8 个 unsigned long
simd_set_floatv8	赋值	floatv8 类型	8 个单精度浮点
simd_set_doublev8	赋值	doublev8 类型	8 个双精度浮点
simd_set_float16v32	赋值	float16v32 类型	32 个半精度浮点

函数名

simd_set_doublev8—— doublev8 赋值函数

simd_set_floatv8 —— floatv8 赋值函数

参数说明

double/float K;

double/float L;

double/float M;

double/float N;

double/float O;

double/float P;

double/float Q;

double/float R;

返回值

doublev8/floatv8

功能说明

doublev8/floatv8 类型的赋值函数，将 8 个 double/float 类型的数据传递到向量变量中。

例如：doublev8 va=simd_set_doublev8(1.0，2.0，3.0，4.0，5.0，6.0，7.0，8.0);

va 中的值是这样的：

8.0	7.0	6.0	5.0	4.0	3.0	2.0	1.0
512　448	447　384	383　320	319　256	255　192	191　128	127　64	63　0

函数名

simd_set_intv16 —— intv16 赋值函数

simd_set_uintv16 —— uintv16 赋值函数

参数说明

int/ unsigned int a0;

int/ unsigned int a1;

int/ unsigned int a2;

…………

int/ unsigned int a14;

int/ unsigned int a15;

返回值

intv16/uintv16

功能说明

intv16/uintv16 类型的赋值函数，将 16 个 int/unsigned int 类型的数据传递到向量变量中。

例如：va=simd_set_intv16(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16);

va 中的值是这样的：

16	15	……	3	2	1
511　480	479　448		95　64	63　32	31　0

函数名

simd_set_int512 —— int512 赋值函数

simd_set_uint512 —— uint512 赋值函数

参数说明

long/ unsigned long a;

long/ unsigned long b;

long/ unsigned long c;

long/ unsigned long d;

long/ unsigned long e;

long/ unsigned long f;

long/ unsigned long d;

long/ unsigned long h;

返回值

int512/uint512

功能说明

int512/uint512 类型的赋值函数，将 8 个 long/ unsigned long 类型的数据传递到向量变量中。

例如：va=simd_set_int512(1,2,3,4,5,6,7,8);

va 中的值是这样的：

8	7	6	5	4	3	2	1
512 448	447 384	383 320	319 256	255 192	191 128	127 64	63 0

函数名

simd_set_float16v32 ——float16v32 赋值函数

参数说明

_Float16 a0；

_Float16 a1；

_Float16 a2；

…………；

_Float16 a30；

_Float16 a31；

返回值

float16v32

功能说明

float16v32 类型的赋值函数，将 32 个_Float16 类型的数据传递到向量变量中。

例如：va=simd_set_float16v32 (0.0,1.0,2.0,3.0,4.0,5.0,6.0,7.0,

8.0,9.0,10.0,11.0,12.0,13.0,14.0,15.0,

16.0,17.0,18.0,19.0,20.0,21.0,22.0,23.0,
24.0,25.0,26.0,27.0,28.0,29.0,30.0,31.0);

va 中的值是这样的：

31.0	30.0	······	2.0	1.0	0.0

511　496 495　480　　47　32 31　16　15　0

5.3.6 打印函数接口

表 5-13 扩展类型的打印函数

函数	操作	参数说明	
		返回值	参数
simd_fprint_intv16	打印	/	FILE*, intv16
simd_fprint_uintv16	打印	/	FILE *, uint16
simd_fprint_int512	打印	/	FILE*, int512
simd_fprint_uint512	打印	/	FILE *, uint512
simd_fprint_floatv8	打印	/	FILE*, floatv8
simd_fprint_doublev8	打印	/	FILE*, doublev8
simd_fprint_float16v32	打印	/	FILE*, float16v32
simd_print_intv16	打印	/	intv16
simd_print_uintv16	打印	/	uintv16
simd_print_int512	打印	/	int512
simd_print_uint512	打印	/	uint512
simd_print_floatv8	打印	/	floatv8
simd_print_doublev8	打印	/	doublev8
simd_print_float16v32	打印	/	float16v32
simd_fprint_intv16_X	打印	/	FILE*, intv16
simd_fprint_uintv16_X	打印	/	FILE *, uint16
simd_fprint_int512_X	打印	/	FILE*, int512
simd_fprint_uint512_X	打印	/	FILE *, uint512
simd_fprint_floatv8_X	打印	/	FILE*, floatv8
simd_fprint_doublev8_X	打印	/	FILE*, doublev8
simd_fprint_float16v32_X	打印	/	FILE*, float16v32
simd_print_intv16_X	打印	/	intv16
simd_print_uintv16_X	打印	/	uintv16
simd_print_int512_X	打印	/	int512
simd_print_uint512_X	打印	/	uint512
simd_print_floatv8_X	打印	/	floatv8

simd_print_doublev8_X	打印	/	doublev8
simd_print_float16v32_X	打印	/	float16v32

函数名

simd_fprint_intv16 —— intv16 打印到文件

simd_fprint_uintv16 —— uintv16 打印到文件

simd_fprint_intv16_X —— intv16 以 16 进制打印到文件

simd_fprint_uintv16_X —— uintv16 以 16 进制打印到文件

参数说明

FILE *file;

intv16/uintv16 va;

返回值

无

功能说明

intv16/uintv16 类型的数据以 16 个 int/ unsigned int 的形式打印到文件 file 中，其中 simd_fprint_intv16_X、simd_fprint_uintv16_X 以 16 进制格式打印。

例如： va=simd_set_intv16(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16);

simd_fprint_intv16(stderr, va);

simd_fprint_intv16_X(stderr, va);

打印的结果是这样的：

[16, 15, 14, 13, 12, 11, 10, 9, 8， 7， 6， 5， 4, 3, 2, 1]

[0x10, 0xf, 0xe, 0xd, 0xc, 0xb, 0xa, 0x9, 0x8，0x7，0x6，0x5，0x4, 0x3, 0x2, 0x1]

函数名

simd_fprint_int512 —— int512 打印到文件

simd_fprint_uint512—— uint512 打印到文件

simd_fprint_int512_X —— int512 以 16 进制打印到文件

simd_fprint_uint512_X—— uint512 以 16 进制打印到文件

参数说明

FILE *file;

int512/uint512 va;

返回值

无

功能说明

int512/uint512 类型的数据以 8 个 long/ unsigned long 的形式打印到文件 file 中，其中 simd_fprint_int512_X、simd_fprint_uint512_X 以 16 进制格式打印。

例如： va=simd_set_int512(1,2,3,4,5,6,7,8);

simd_fprint_int512(stderr, va);

simd_fprint_int512_X(stderr, va);

打印的结果是这样的：

[8，7，6，5，4, 3, 2, 1]

[0x8, 0x7, 0x6, 0x5, 0x4, 0x3, 0x2, 0x1]

函数名

simd_fprint_floatv8 —— floatv8 打印到文件

simd_fprint_doublev8 —— doublev8 打印到文件

simd_fprint_floatv8_X —— floatv8 以 16 进制打印到文件

simd_fprint_doublev8_X —— doublev8 以 16 进制打印到文件

参数说明

FILE *file;

floatv8/doublev8 va;

返回值

无

功能说明

floatv8/doublev8 类型的数据以 8 个 float/double 的形式打印到文件 file 中，其中 simd_fprint_floatv8_X、simd_fprint_doublev8_X 以 16 进制格式打印。

例如： va=simd_set_floatv8(1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0);

simd_fprint_floatv8(stderr, va);

simd_fprint_floatv8_X(stderr, va);

vb=simd_set_doublev8(1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0);

simd_fprint_doublev8_X(stderr, vb);

打印的结果是这样的：

[8.0, 7.0, 6.0, 5.0，4.0，3.0, 2.0, 1.0]

[0x41000000, 0x40e00000, 0x40c00000, 0x40a00000, 0x40800000, 0x40400000, 0x40000000, 0x3f800000]

[0x4020000000000000, 0x401c000000000000, 0x4018000000000000, 0x4014000000000000, 0x4010000000000000, 0x4008000000000000, 0x4000000000000000, 0x3ff0000000000000]

函数名

simd_fprint_float16v32 —— float16v32 打印到文件

simd_fprint_float16v32_X —— float16v32 以 16 进制打印到文件

参数说明

FILE *file;

float16v32 va;

返回值

无

功能说明

float16v32 类型的数据以 32 个_Float16 的形式打印到文件 file 中，其中 simd_fprint_float16v32_X 以 16 进制格式打印。

例如： va=simd_set_float16v32 (0.0,1.0,2.0,3.0,4.0,5.0,6.0,7.0,
8.0,9.0,10.0,11.0,12.0,13.0,14.0,15.0,
16.0,17.0,18.0,19.0,20.0,21.0,22.0,23.0,
24.0,25.0,26.0,27.0,28.0,29.0,30.0,31.0);

simd_fprint_float16v32(stderr, va);

simd_fprint_float16v32_X(stderr, va);

打印的结果是这样的：

[31.0, 30.0, 29.0, 28.0, 27.0, 26.0, 25.0, 24.0, 23.0, 22.0, 21.0, 20.0, 19.0, 18.0, 17.0, 16.0, 15.0，14.0，13.0, 12.0, 11.0, 10.0, 9.0, 8.0, 7.0, 6.0, 5.0, 4.0, 3.0, 2.0, 1.0, 0.0]

[0x4fc0, 0x4f80, 0x4f40, 0x4f00, 0x4ec0, 0x4e80, 0x4e40, 0x4e00, 0x4dc0, 0x4d80, 0x4d40, 0x4d00, 0x4cc0, 0x4c80, 0x4c40, 0x4c00, 0x4b80, 0x4b00, 0x4a80, 0x4a00, 0x4980, 0x4900, 0x4880, 0x4800, 0x4700, 0x4600, 0x4500, 0x4400, 0x4200, 0x4000, 0x3c00, 0x0]

函数名

simd_print_intv16 —— intv16 打印到屏幕

simd_print_uintv16 —— uintv16 打印到屏幕

simd_print_intv16_X —— intv16 以 16 进制打印到屏幕

simd_print_uintv16_X —— uintv16 以 16 进制打印到屏幕

参数说明

intv16/uintv16 va;

返回值

无

功能说明

intv16/uintv16 类型的数据以 16 个 int/ unsigned int 的形式打印到屏幕，其中 simd_print_intv16_X、simd_print_uintv16_X 以 16 进制格式打印。

例如： va=simd_set_intv16(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16);

simd_print_intv16(va);

simd_print_intv16_X(va);

打印的结果是这样的：

[16, 15, 14, 13, 12, 11, 10, 9，8，7，6，5，4, 3, 2, 1]

[0x10, 0xf, 0xe, 0xd, 0xc, 0xb, 0xa, 0x9, 0x8，0x7，0x6，0x5，0x4, 0x3, 0x2, 0x1]

函数名

simd_print_int512 —— int512 打印到屏幕

simd_print_uint512—— uint512 打印到屏幕

simd_print_int512_X —— int512 以 16 进制打印到屏幕

simd_print_uint512_X—— uint512 以 16 进制打印到屏幕

参数说明

int512/uint512 va;

返回值

无

功能说明

int512/uint512 类型的数据以 8 个 long/ unsigned long 的形式打印到屏幕，其中 simd_print_int512_X、simd_print_uint512_X 以 16 进制格式打印。

例如： va=simd_set_int512(1,2,3,4,5,6,7,8);

simd_print_int512(va);

simd_print_int512_X(va);

打印的结果是这样的：

[8，7，6，5，4, 3, 2, 1]

[0x8, 0x7, 0x6, 0x5, 0x4, 0x3, 0x2, 0x1]

函数名

simd_print_floatv8 —— floatv8 打印到屏幕

simd_print_doublev8 —— doublev8 打印到屏幕

simd_print_floatv8_X —— floatv8 以 16 进制打印到屏幕

simd_print_doublev8_X —— doublev8 以 16 进制打印到屏幕

参数说明

floatv8/doublev8 va;

返回值

无

功能说明

floatv8/doublev8 类型的数据以 8 个 float/double 的形式打印到屏幕，其中 simd_print_floatv8_X、simd_print_doublev8_X 以 16 进制格式打印。

例如： va=simd_set_floatv8(1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0);

simd_print_floatv8(va);

simd_print_floatv8_X(va);

vb=simd_set_doublev8(1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0);

simd_print_doublev8_X(vb);

打印的结果是这样的：

[8.0,7.0，6.0,5.0，4.0，3.0，2.0, 1.0]

[0x41000000, 0x40e00000, 0x40c00000, 0x40a00000, 0x40800000, 0x40400000, 0x40000000, 0x3f800000]

[0x4020000000000000, 0x401c000000000000, 0x4018000000000000, 0x4014000000000000, 0x4010000000000000, 0x4008000000000000, 0x4000000000000000, 0x3ff0000000000000]

函数名

simd_print_float16v32 —— float16v32 打印到屏幕

simd_print_float16v32_X —— float16v32 以 16 进制打印到屏幕

参数说明

float16v32 va;

返回值

无

功能说明

float16v32 类型的数据以 32 个 _Float16 的形式打印到屏幕中，其中 simd_print_float16v32_X 以 16 进制格式打印。

例如：
```
va=simd_set_float16v32 (0.0,1.0,2.0,3.0,4.0,5.0,6.0,7.0,
                        8.0,9.0,10.0,11.0,12.0,13.0,14.0,15.0,
                        16.0,17.0,18.0,19.0,20.0,21.0,22.0,23.0,
                        24.0,25.0,26.0,27.0,28.0,29.0,30.0,31.0);
simd_print_float16v32(va);
simd_print_float16v32_X(va);
```

打印的结果是这样的：

[31.0, 30.0, 29.0, 28.0, 27.0, 26.0, 25.0, 24.0, 23.0, 22.0, 21.0, 20.0, 19.0, 18.0, 17.0, 16.0, 15.0，14.0，13.0, 12.0, 11.0, 10.0, 9.0, 8.0, 7.0, 6.0, 5.0, 4.0, 3.0, 2.0, 1.0, 0.0]

[0x4fc0, 0x4f80, 0x4f40, 0x4f00, 0x4ec0, 0x4e80, 0x4e40, 0x4e00, 0x4dc0, 0x4d80, 0x4d40, 0x4d00, 0x4cc0, 0x4c80, 0x4c40, 0x4c00, 0x4b80, 0x4b00, 0x4a80, 0x4a00, 0x4980, 0x4900, 0x4880, 0x4800, 0x4700, 0x4600, 0x4500, 0x4400, 0x4200, 0x4000, 0x3c00, 0x0]

5.3.7 向量归约接口

表 5-14 向量归约函数

函数	操作	参数说明	
		返回值	参数
simd_reduc_plusw	加法归约	int	intv16
simd_reduc_pluss		float	floatv8
simd_reduc_plusd		double	doublev8
simd_reduc_plush		_Float16	float16v32
simd_reduc_smaxs	最大值归约	float	floatv8
simd_reduc_smaxd		double	doublev8

simd_reduc_smaxh		_Float16	float16v32
simd_reduc_smins	最小值归约	float	floatv8
simd_reduc_smind		double	doublev8
simd_reduc_sminh		_Float16	float16v32

函数名

simd_reduc_plusw —— intv16 的加法归约

参数说明

intv16 va;

返回值

int

功能说明

intv16 类型的加法归约，将 va 中的 16 个 int 类型的整型数据归约求和。

函数名

simd_reduc_pluss —— floatv8 的加法归约

simd_reduc_plusd —— doublev8 的加法归约

参数说明

floatv8/doublev8 va;

返回值

float/double

功能说明

加法归约，将 va 中的 8 个 float/doublet 类型的浮点数据归约求和。

函数名

simd_reduc_smaxs —— floatv8 的最大值归约

simd_reduc_smaxd —— doublev8 的最大值归约

参数说明

floatv8/doublev8 va;

返回值

float/double

功能说明

最大值归约，求 va 中的 8 个 float/double 类型浮点数据中的最大值。

函数名

simd_reduc_smins —— floatv8 的最小值归约

simd_reduc_smind —— doublev8 的最小值归约

参数说明

floatv8/doublev8 va;

返回值

float/double

功能说明

最小值归约，求 va 中的 8 个 float/double 类型浮点数据中的最小值。

函数名

simd_reduc_plush —— float16v32 的加法归约

参数说明

Float16v32 va;

返回值

_Float16

功能说明

float16v32 类型的加法归约，将 va 中的 32 个_Float16 类型的浮点数据归约求和。

函数名

simd_reduc_smaxh —— float16v32 的最大值归约

参数说明

float16v32 va;

返回值

_Float16

功能说明

float16v32 类型的最大值归约，求 va 中的 32 个_Float16 类型浮点数据中的最大值。

函数名

simd_reduc_sminh —— float16v32 的最小值归约

参数说明

float16v32 va;

返回值

_Float16

功能说明

float16v32 类型的最小值归约，求 va 中的 32 个_Float16 类型浮点数据中的最小值。

修订记录

时间	修订内容	负责人
2018.5.20	3.3 节增加 libc_aligned_malloc 和 libc_aligned_free 主核函数。	钱宏
2018.5.30	1、新增 5.2.7 和 5.3.7 节向量规约加接口	钱宏

2018.6.13	1、删除 5.2.3 节和 5.3.3 节中 simd_vbiss、simd_ vbisd 浮点逻辑或接口 2、5.2.2、5.2.3 和 5.3.3 节新增 simd_smaxw 等大值/小值比较选择接口 3、5.2.7、5.3.7 节新增 simd_reduc_smaxw 等向量规约最大/最小值接口	钱宏
2018.10.02	修改 3.3、3.4 对界章节	管茂林
2018.11.22	增加 16 进制向量打印函数接口	管茂林
2019.2.20	5.3.3 节增加从核浮点向量除法和平方根接口	钱宏
2019.03.22	增加从核半精向量相关接口	钱宏
2019.05.22	增加半精向量大值比较、小值比较及规约接口	钱宏
2020.03.17	完善 SW处理器新增向量相关接口描述 3.10 节增加扩展浮点类型与扩展长字整形之间的转换描述 5.3.1 节增加 simd_loade 接口 5.3.2 节增加 simd_sllx、simd_srlx 接口 5.3.3 节增加 simd_vfcvtsh、simd_vfcvths 接口 5.3.4 节增加向量混洗的接口	管茂林
2020.05.05	5.5.3 节增加从核浮点向量近似倒数(simd_vfrec[sd]) 和精确倒数 (simd_vfrecp[sd]) 接口	王恒
2020.05.22	5.2.2、5.3.2 节增加 simd_vaddli、simd_vsubli 接口	管茂林
2020.05.22	5.3.3 节增加 simd_vrsqrt[sd] 接口	王恒
2020.06.11	4.2 节修改示例，修改部分错别字笔误等	管茂林
2020.06.15	统一文档名称，修改部分文字描述等	吴伟
2020.07.02	4.3 节修改示例程序中的笔误	王恒
2020.07.10	统一相关术语表述等	管茂林
2020.09.23	5.2.1 节 simd_storeuh 的参数类型增加 long*	樊行健
2020.10.12	5.3.3 节增加 simd_vdivh、simd_vsqrth 接口	钱宏
2020.12.15	修改编译器名	杨涛