Eigen To Instructions Interface

制作者 hao.chen

Contents

Chapter 1

类索引

1.1 类列表

这里列出了所有类、结构、联合以及接口定义等，并附带简要说明:

Chapter 2

文件索引

2.1 文件列表

这里列出了所有文档化的文件，并附带简要说明:

Chapter 3

类说明

3.1 CustomInsns类 参考

custom扩展指令类

```
#include <eigen3_ops.h>
```

Public 成员函数

- CustomInsns ()
- int vecvt_hf_x8_m (int8_t *rs1, half *rd, struct ShapeStride *ss)
- int vecvt_hf_xu8_m (uint8_t *rs1, half *rd, struct ShapeStride *ss)
- int vecvt_hf_x16_m (int16_t *rs1, half *rd, struct ShapeStride *ss)
- int vecvt_hf_xu16_m (uint16_t *rs1, half *rd, struct ShapeStride *ss)
- int veadd_mm (half *rs1, half *rd, half *rs2, struct ShapeStride *ss)
- int veadd_mv (half *rs1, half *rd, half *rs2, struct ShapeStride *ss, int dim)
- int veadd_mf (half *rs1, half *rd, half rs2, struct ShapeStride *ss)
- int vesub_mm (half *rs1, half *rd, half *rs2, struct ShapeStride *ss)
- int vesub_mv (half *rs1, half *rd, half *rs2, struct ShapeStride *ss, int dim)
- int vesub_mf (half *rs1, half *rd, half rs2, struct ShapeStride *ss)
- int veacc_m (half *rs1, half *rd, struct ShapeStride *ss)
- int veacc_m (half *rs1, half *rd, struct ShapeStride *ss, int dim)
- int vemul_mm (half *rs1, half *rs2, half *rd, struct ShapeStride *ss)
- int vemul_mv (half *rs1, half *rs2, half *rd, struct ShapeStride *ss)
- int veemul_mm (half *rs1, half *rd, half *rs2, struct ShapeStride *ss)
- int veemul_mv (half *rs1, half *rd, half *rs2, struct ShapeStride *ss, int dim)
- int veemul_mf (half *rs1, half *rd, half rs2, struct ShapeStride *ss)
- int veemacc_mm (half *rs1, half *rd, half *rs2, struct ShapeStride *ss)
- int veemacc_mm (half *rs1, half *rd, half *rs2, struct ShapeStride *ss, int dim)
- int veemacc_mv (half *rs1, half *rd, half *rs2, struct ShapeStride *ss, int dim)
- int veemacc_mf (half *rs1, half *rd, half rs2, struct ShapeStride *ss, int dim)
- int vemax_mm (half *rs1, half *rd, half *rs2, struct ShapeStride *ss)
- int vemax_m (half *rs1, half *rd, struct ShapeStride *ss, int dim)
- int vemax_m (half *rs1, half *rd, struct ShapeStride *ss)
- int vemax_mf (half *rs1, half *rd, half rs2, struct ShapeStride *ss)
- int vemax_mv (half *rs1, half *rd, half *rs2, struct ShapeStride *ss, int dim)
- int vemin_mm (half *rs1, half *rd, half *rs2, struct ShapeStride *ss)

- int vemin_m (half *rs1, half *rd, struct ShapeStride *ss, int dim)
- int vemin_m (half *rs1, half *rd, struct ShapeStride *ss)
- int vemin_mf (half *rs1, half *rd, half rs2, struct ShapeStride *ss)
- int vemin_mv (half *rs1, half *rd, half *rs2, struct ShapeStride *ss, int dim)
- int velkrelu_mf (half *rs1, half rs2, half *rd, struct ShapeStride *ss)
- int velkrelu_mv (half *rs1, half *rd, half *rs2, struct ShapeStride *ss, int dim)
- int velut_m (uint16_t *rs1, unsigned long rs2, half *rd, struct ShapeStride *ss)
- int vemv_m (half *rs1, half *rd, struct ShapeStride *ss)

Public 属性

- int **debug**

3.1.1 详细描述

custom扩展指令类

包含了全部的custom矩阵扩展指令 可以通过设置其实例的debug字段值来动态控制debug输出

3.1.2 构造及析构函数说明

3.1.2.1 CustomInsns()

```
CustomInsns::CustomInsns ( )
```

CustomInsns() 构造函数

默认不开启debug

3.1.3 成员函数说明

3.1.3.1 veacc_m() [1/2]

```
int CustomInsns::veacc_m (
            half * rs1,
            half * rd,
            struct ShapeStride * ss )
```

veacc_m() veacc.m

矩阵所有元素求和

参数

rs1	M1,源操作矩阵基地址
rd	V,目的向量基地址
ss	矩阵形状描述

返回

执行结果

3.1.3.2 veacc_m() [2/2]

```
int CustomInsns::veacc_m (
            half * rs1,
            half * rd,
            struct ShapeStride * ss,
            int dim )
```

veacc_m() veacc.m

矩阵列元素(行向量)求和s=sum(M1i)， 矩阵行元素(列向量)求和 s=sum(M1j)

参数

rs1	M1,源操作矩阵基地址
rd	V,目的向量基地址
ss	矩阵形状描述
dim	方向 dim = 0 列向量求和， dim = 1 行向量求和

返回

执行结果

3.1.3.3 veadd_mf()

```
int CustomInsns::veadd_mf (
            half * rs1,
            half * rd,
            half rs2,
            struct ShapeStride * ss )
```

veadd_mf() veadd.mf

标量和矩阵元素广播加 M = M1 + f 如果要在原地进行，则rd的stride必须和rs1的stride相同

参数

rs1	M1,源操作矩阵基地址
rd	M,目的矩阵基地址
rs2	f,源标量操作数
ss	矩阵形状描述

返回

执行结果

3.1.3.4 veadd_mm()

```
int CustomInsns::veadd_mm (
            half * rs1,
            half * rd,
            half * rs2,
            struct ShapeStride * ss )
```

veadd_mm() veadd.mm

同维度矩阵和矩阵元素加 M = M1 + M2 stride 一致的情况下运算还可以原地进行, 即rd = rs1 或 rd = rs2

参数

rs1	M1,源操作矩阵一基地址
rs2	M2,源操作矩阵二基地址
rd	M,目的矩阵基地址
ss	矩阵形状描述

返回

执行结果

3.1.3.5 veadd_mv()

```
int CustomInsns::veadd_mv (
            half * rs1,
            half * rd,
            half * rs2,
            struct ShapeStride * ss,
            int dim )
```

veadd_mv() veadd.mv

同维度矩阵和矩阵元素加 M = M1 + v

参数

rs1	M1,源操作矩阵基地址
rs2	M2,源操作向量基地址
rd	M,目的矩阵基地址
ss	矩阵形状描述
dim	方向 dim = 0 v为行向量， dim = 1 v为列向量

返回

执行结果

3.1.3.6 vecvt_hf_x16_m()

```
int CustomInsns::vecvt_hf_x16_m (
            int16_t * rs1,
            half * rd,
            struct ShapeStride * ss )
```

vecvt_hf_x16_m() vecvt.hf.x16.m

将矩阵中的元素由 int16 格式转换为 fp16

参数

rs1	M1,源操作矩阵基地址
rd	M,目的矩阵基地址
ss	矩阵形状描述

返回

执行结果

3.1.3.7 vecvt_hf_x8_m()

```
int CustomInsns::vecvt_hf_x8_m (
            int8_t * rs1,
            half * rd,
            struct ShapeStride * ss )
```

vecvt_hf_x8_m() vecvt.hf.x8.m

将矩阵中的元素由 int8 格式转换为 fp16

参数

rs1	M1,源操作矩阵基地址
rd	M,目的矩阵基地址
ss	矩阵形状描述

返回

执行结果

3.1.3.8 vecvt_hf_xu16_m()

```
int CustomInsns::vecvt_hf_xu16_m (
            uint16_t * rs1,
            half * rd,
            struct ShapeStride * ss )
```

vecvt_hf_xu16_m() vecvt.hf.xu16.m

将矩阵中的元素由 uint16 格式转换为 fp16

参数

rs1	M1,源操作矩阵基地址
rd	M,目的矩阵基地址
ss	矩阵形状描述

返回

执行结果

3.1.3.9 vecvt_hf_xu8_m()

```
int CustomInsns::vecvt_hf_xu8_m (
            uint8_t * rs1,
            half * rd,
            struct ShapeStride * ss )
```

vecvt_hf_xu8_m() vecvt.hf.xu8.m

将矩阵中的元素由 uint8 格式转换为 fp16

参数

rs1	M1,源操作矩阵基地址
rd	M,目的矩阵基地址
ss	矩阵形状描述

返回

执行结果

3.1.3.10 veemacc_mf()

```
int CustomInsns::veemacc_mf (
            half * rs1,
            half * rd,
            half rs2,
            struct ShapeStride * ss,
            int dim )
```

veemacc_mf() veemacc.mf

当dim=0时，浮点标量和矩阵元素广播乘，再列元素(行向量)求和； 当dim=1时，浮点标量和矩阵元素广播乘，再行元素(列向量)求和

参数

rs1	M1,源操作矩阵基地址
rs2	M2,源操作向量基地址
rd	M,目的向量基地址
ss	矩阵形状描述
dim	方向 dim = 0 v为行向量， dim = 1 v为列向量

返回

执行结果

3.1.3.11 veemacc_mm() [1/2]

```
int CustomInsns::veemacc_mm (
            half * rs1,
            half * rd,
            half * rs2,
            struct ShapeStride * ss )
```

veemacc_mm() veemacc.mm

矩阵和矩阵元素乘，再所有元素求和 M = sum(M1ij∗M2ij)

参数

rs1	M1,源操作矩阵一基地址
rs2	M2,源操作矩阵二基地址
rd	s,目的数存放地址
ss	矩阵形状描述

返回

执行结果

3.1.3.12 veemacc_mm() [2/2]

```
int CustomInsns::veemacc_mm (
            half * rs1,
            half * rd,
            half * rs2,
            struct ShapeStride * ss,
            int dim )
```

veemacc_mm() veemacc.mm dim = ?

矩阵和矩阵元素乘，再按照某个方向元素求和

参数

rs1	M1,源操作矩阵一基地址
rs2	M2,源操作矩阵二基地址
rd	s,目的向量存放地址
ss	矩阵形状描述
dim	方向 dim = 0 v为行向量， dim = 1 v为列向量

返回

执行结果

3.1.3.13 veemacc_mv()

```
int CustomInsns::veemacc_mv (
            half * rs1,
            half * rd,
            half * rs2,
```

```
            struct ShapeStride * ss,
            int dim )
```

veemacc_mv() veemacc.mv

当dim=0时，列向量和矩阵元素广播乘，再列元素(行向量)求和； 当dim=1时，行向量和矩阵元素广播乘，再行元素(列向量)求和

参数

rs1	M1,源操作矩阵基地址
rs2	M2,源操作向量基地址
rd	M,目的向量基地址
ss	矩阵形状描述
dim	方向 dim = 0 v为行向量， dim = 1 v为列向量

返回

执行结果

3.1.3.14 veemul_mf()

```
int CustomInsns::veemul_mf (
            half * rs1,
            half * rd,
            half rs2,
            struct ShapeStride * ss )
```

veemul_mf() veemul.mf

标量和矩阵元素广播乘 M = M1 ∗ f

参数

rs1	M1,源操作矩阵基地址
rd	M,目的矩阵基地址
rs2	f,源标量操作数
ss	矩阵形状描述

返回

执行结果

3.1.3.15 veemul_mm()

```
int CustomInsns::veemul_mm (
              half * rs1,
              half * rd,
              half * rs2,
              struct ShapeStride * ss )
```

veemul_mm() veemul.mm

矩阵和矩阵元素乘 M = M1∗M2 stride 一致的情况下运算还可以原地进行, 即rd = rs1 或 rd = rs2

参数

rs1	M1,源操作矩阵一基地址
rs2	M2,源操作矩阵二基地址
rd	M,目的矩阵基地址
ss	矩阵形状描述

返回

执行结果

3.1.3.16 veemul_mv()

```
int CustomInsns::veemul_mv (
              half * rs1,
              half * rd,
              half * rs2,
              struct ShapeStride * ss,
              int dim )
```

veemul_mv() veemul.mv

向量和矩阵元素广播乘 M = V∗M1

参数

rs1	M1,源操作矩阵基地址
rs2	M2,源操作向量基地址
rd	M,目的矩阵基地址
ss	矩阵形状描述
dim	方向 dim = 0 v为行向量， dim = 1 v为列向量

返回

执行结果

3.1.3.17 velkrelu_mf()

```
int CustomInsns::velkrelu_mf (
            half * rs1,
            half rs2,
            half * rd,
            struct ShapeStride * ss )
```

velkrelu_mf() velkrelu.mf

矩阵元素与0比较，小于标量则乘一常数系数

参数

rs1	M1,源操作矩阵基地址
rs2	k, 源标量浮点操作数
rd	V,目的矩阵基地址
ss	矩阵形状描述

返回

执行结果

3.1.3.18 velkrelu_mv()

```
int CustomInsns::velkrelu_mv (
            half * rs1,
            half * rd,
            half * rs2,
            struct ShapeStride * ss,
            int dim )
```

velkrelu_mv() velkrelu.mv

当dim = 0 时，矩阵行元素与0比较，小于标量则乘以一常数行向量； 当dim = 1时，矩阵列元素与0比较，小于标量则乘以一常数列向量

参数

rs1	M1,源操作矩阵基地址
rs2	M2,源操作向量基地址
rd	M,目的矩阵基地址
ss	矩阵形状描述
dim	方向 dim = 0 v为行向量， dim = 1 v为列向量

返回

执行结果

3.1.3.19 velut_m()

```
int CustomInsns::velut_m (
            uint16_t * rs1,
            unsigned long rs2,
            half * rd,
            struct ShapeStride * ss )
```

velut_m() velut.m

以矩阵中所有元素为索引(二进制),查找表中对应的值 M=LUT(M1ij)

参数

rs1	M1,源操作矩阵基地址
rs2	A_base, LUT基地址
rd	V,目的矩阵基地址
ss	矩阵形状描述

返回

执行结果

3.1.3.20 vemax_m() [1/2]

```
int CustomInsns::vemax_m (
            half * rs1,
            half * rd,
            struct ShapeStride * ss,
            int dim )
```

vemax_m() vemax.m

矩阵列元素(行向量)求最大值s=max(M1i)， 矩阵行元素(列向量)求最大值s=max(M1j)

参数

rs1	M1,源操作矩阵基地址
rd	V,目的向量基地址
ss	矩阵形状描述
dim	方向 dim = 0 列向量求最大值， dim = 1 行向量求最大值

返回

执行结果

3.1.3.21 vemax_m() [2/2]

```
int CustomInsns::vemax_m (
            half * rs1,
            half * rd,
            struct ShapeStride * ss )
```

vemax_m() vemax.m

矩阵所有元素求最大值

参数

rs1	M1,源操作矩阵基地址
rd	V,目的数基地址
ss	矩阵形状描述

返回

执行结果

3.1.3.22 vemax_mf()

```
int CustomInsns::vemax_mf (
            half * rs1,
            half * rd,
            half rs2,
            struct ShapeStride * ss )
```

vemax_mf() vemax.mf

矩阵元素与标量比较求最大值 M = max(M1ij, f) 如果要在原地进行，则rd的stride必须和rs1的stride相同

参数

rs1	M1,源操作矩阵基地址
rd	M,目的矩阵基地址
rs2	f,源标量浮点操作数
ss	矩阵形状描述

返回

执行结果

3.1.3.23 vemax_mm()

```
int CustomInsns::vemax_mm (
            half * rs1,
            half * rd,
            half * rs2,
            struct ShapeStride * ss )
```

vemax_mm() vemax.mm

两个矩阵对应元素间求最大值 M = max(M1, M2) stride 一致的情况下运算还可以原地进行, 即rd = rs1 或 rd = rs2

参数

rs1	M1,源操作矩阵一基地址
rs2	M2,源操作矩阵二基地址
rd	M,目的矩阵基地址
ss	矩阵形状描述

返回

执行结果

3.1.3.24 vemax_mv()

```
int CustomInsns::vemax_mv (
            half * rs1,
            half * rd,
            half * rs2,
            struct ShapeStride * ss,
            int dim )
```

vemax_mv() vemax.mv

当dim=0时，矩阵列元素(行向量)与行向量元素比较求最大值 M=max(M1i, V); 当dim=1时，矩阵行元素(列向量)与列向量元素比较求最大值 M=max(M1j, V)

参数

rs1	M1,源操作矩阵基地址
rs2	M2,源操作向量基地址
rd	M,目的矩阵基地址
ss	矩阵形状描述
dim	方向 dim = 0 v为行向量， dim = 1 v为列向量

返回

执行结果

3.1.3.25 vemin_m() [1/2]

```
int CustomInsns::vemin_m (
            half * rs1,
            half * rd,
            struct ShapeStride * ss,
            int dim )
```

vemin_m() vemin.m

矩阵列元素(行向量)求最小值s=min(M1i)， 矩阵行元素(列向量)求最小值s=min(M1j)

参数

rs1	M1,源操作矩阵基地址
rd	V,目的向量基地址
ss	矩阵形状描述
dim	方向 dim = 0 列向量求最小值， dim = 1 行向量求最小值

返回

执行结果

3.1.3.26 vemin_m() [2/2]

```
int CustomInsns::vemin_m (
            half * rs1,
            half * rd,
            struct ShapeStride * ss )
```

vemin_m() vemin.m

矩阵所有元素求最小值

参数

rs1	M1,源操作矩阵基地址
rd	V,目的数基地址
ss	矩阵形状描述

返回

执行结果

3.1.3.27 vemin_mf()

```
int CustomInsns::vemin_mf (
            half * rs1,
            half * rd,
            half rs2,
            struct ShapeStride * ss )
```

vemin_mf() vemin.mf

矩阵元素与标量比较求最小值 M = min(M1ij, f) 如果要在原地进行，则rd的stride必须和rs1的stride相同

参数

rs1	M1,源操作矩阵基地址
rd	M,目的矩阵基地址
rs2	f,源标量浮点操作数
ss	矩阵形状描述

返回

执行结果

3.1.3.28 vemin_mm()

```
int CustomInsns::vemin_mm (
            half * rs1,
            half * rd,
            half * rs2,
            struct ShapeStride * ss )
```

vemin_mm() vemin.mm

两个矩阵对应元素间求最小值 M = min(M1, M2) stride 一致的情况下运算还可以原地进行, 即rd = rs1 或 rd = rs2

参数

rs1	M1,源操作矩阵一基地址
rs2	M2,源操作矩阵二基地址
rd	M,目的矩阵基地址
ss	矩阵形状描述

返回

执行结果

3.1.3.29 vemin_mv()

```
int CustomInsns::vemin_mv (
            half * rs1,
            half * rd,
            half * rs2,
            struct ShapeStride * ss,
            int dim )
```

vemin_mv() vemin.mv

当dim=0时，矩阵列元素(行向量)与行向量元素比较求最小值 M=min(M1i, V); 当dim=1时，矩阵行元素(列向量)与列向量元素比较求最小值 M=min(M1j, V)

参数

rs1	M1,源操作矩阵基地址
rs2	M2,源操作向量基地址
rd	M,目的矩阵基地址
ss	矩阵形状描述
dim	方向 dim = 0 v为行向量， dim = 1 v为列向量

返回

执行结果

3.1.3.30 vemul_mm()

```
int CustomInsns::vemul_mm (
            half * rs1,
            half * rs2,
            half * rd,
            struct ShapeStride * ss )
```

vemul_mm() vemul.mm

矩阵和矩阵算术乘，正常算术运算 M = M1.M2 源操作矩阵一的列值必须和源操作矩阵二的行值相等，如果不等则直接返回错误

参数

rs1	M1,源操作矩阵一基地址
rd	M,目的矩阵基地址
rs2	M2,源操作矩阵二基地址
ss	矩阵形状描述

返回

执行结果

3.1.3.31 vemul_mv()

```
int CustomInsns::vemul_mv (
              half * rs1,
              half * rs2,
              half * rd,
              struct ShapeStride * ss )
```

vemul_mv() vemul.mv

向量和矩阵算数乘 V = V1.M1

参数

rs1	M1,源操作矩阵基地址
rd	V,目的向量基地址
rs2	V1,源操作向量基地址
ss	矩阵形状描述

返回

执行结果

3.1.3.32 vemv_m()

```
int CustomInsns::vemv_m (
              half * rs1,
              half * rd,
              struct ShapeStride * ss )
```

vemv_m() vemv.m

将矩阵从一个地方搬移到另一个地方

参数

rs1	M1,源操作矩阵基地址
rd	V,目的矩阵基地址
ss	矩阵形状描述

返回

执行结果

3.1.3.33 vesub_mf()

```
int CustomInsns::vesub_mf (
            half * rs1,
            half * rd,
            half rs2,
            struct ShapeStride * ss )
```

vesub_mf() vesub.mf

标量和矩阵元素广播减 M = M1 - f 如果要在原地进行，则rd的stride必须和rs1的stride相同

参数

rs1	M1,源操作矩阵基地址
rd	M,目的矩阵基地址
rs2	f,源标量操作数
ss	矩阵形状描述

返回

执行结果

3.1.3.34 vesub_mm()

```
int CustomInsns::vesub_mm (
            half * rs1,
            half * rd,
            half * rs2,
            struct ShapeStride * ss )
```

vesub_mm() vesub.mm

同维度矩阵和矩阵元素减 M = M1 - M2 stride 一致的情况下运算还可以原地进行, 即rd = rs1 或 rd = rs2

参数

rs1	M1,源操作矩阵一基地址
rs2	M2,源操作矩阵二基地址
rd	M,目的矩阵基地址
ss	矩阵形状描述

返回

执行结果

3.1.3.35 vesub_mv()

```
int CustomInsns::vesub_mv (
            half * rs1,
            half * rd,
            half * rs2,
            struct ShapeStride * ss,
            int dim )
```

vesub_mv() vesub.mv

同维度矩阵和矩阵元素减 M = M1 - v

参数

rs1	M1,源操作矩阵基地址
rs2	M2,源操作向量基地址
rd	M,目的矩阵基地址
ss	矩阵形状描述
dim	方向 dim = 0 v为行向量， dim = 1 v为列向量

返回

执行结果

该类的文档由以下文件生成:

- eigen3_ops.h
- eigen3_ops.cc

3.2 ShapeStride结构体 参考

矩阵形状描述结构

```
#include <eigen3_ops.h>
```

Public 属性

- unsigned short **shape1_column**
- unsigned short **shape1_row**
- unsigned short **shape2_column**
- unsigned short **shape2_row**
- unsigned short **stride_rd**
- unsigned short **stride_rs2**
- unsigned short **stride_rs1**

3.2.1 详细描述

矩阵形状描述结构

按照 CSR shape1, shape2, stride1, stride2 进行设计 用于提供输入矩阵和输出矩阵的形状和存储方式的描述

该结构体的文档由以下文件生成:

- eigen3_ops.h

3.3 Vadd< Type, MaskType > 模板类 参考

单宽度向量加法指令

```
#include <eigen3_ops.h>
```

Public 类型

- typedef Map< Matrix< Type, 1, Dynamic > > **VaddVecMap**
- typedef Map< Matrix< MaskType, 1, Dynamic > > **VaddMaskVecMap**

Public 成员函数

- int vadd_vf (Type *vs2, Type rs1, Type *vd, int vm, MaskType *v0, int vl)
- int vadd_vv (Type *vs2, Type *vs1, Type *vd, int vm, MaskType *v0, int vl)

Public 属性

- int **debug**

3.3.1 详细描述

template<typename Type, typename MaskType>
class Vadd< Type, MaskType >

单宽度向量加法指令

目的元素的宽度和源操作数中的元素宽度保持一致， 可以通过Type指定数据类型

3.3.2 成员函数说明

3.3.2.1 vadd_vf()

```
template<typename Type , typename MaskType >
int Vadd< Type, MaskType >::vadd_vf (
            Type * vs2,
            Type rs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vadd_vf() vfadd.vf

参数

vs2	源操作向量基地址
rs1	源标量操作数
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.3.2.2 vadd_vv()

```
template<typename Type , typename MaskType >
int Vadd< Type, MaskType >::vadd_vv (
            Type * vs2,
            Type * vs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vadd_vv() vfadd.vv

参数

vs2	源操作向量二基地址
vs1	源操作向量一基地址
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

该类的文档由以下文件生成:

- eigen3_ops.h

3.4 Vcompare< InType, OutType, MaskType > 模板类 参考

向量比较指令

```
#include <eigen3_ops.h>
```

Public 类型

- typedef Map< Matrix< InType, 1, Dynamic > > **VcompareInVecMap**
- typedef Map< Matrix< OutType, 1, Dynamic > > **VcompareOutVecMap**
- typedef Map< Matrix< MaskType, 1, Dynamic > > **VcompareMaskVecMap**

Public 成员函数

- int veq_vf (InType *vs2, InType rs1, OutType *vd, int vm, MaskType *v0, int vl)
- int veq_vv (InType *vs2, InType *vs1, OutType *vd, int vm, MaskType *v0, int vl)
- int vne_vf (InType *vs2, InType rs1, OutType *vd, int vm, MaskType *v0, int vl)
- int vne_vv (InType *vs2, InType *vs1, OutType *vd, int vm, MaskType *v0, int vl)
- int vlt_vf (InType *vs2, InType rs1, OutType *vd, int vm, MaskType *v0, int vl)
- int vlt_vv (InType *vs2, InType *vs1, OutType *vd, int vm, MaskType *v0, int vl)
- int vle_vf (InType *vs2, InType rs1, OutType *vd, int vm, MaskType *v0, int vl)
- int vle_vv (InType *vs2, InType *vs1, OutType *vd, int vm, MaskType *v0, int vl)
- int vgt_vf (InType *vs2, InType rs1, OutType *vd, int vm, MaskType *v0, int vl)
- int vgt_vv (InType *vs2, InType *vs1, OutType *vd, int vm, MaskType *v0, int vl)
- int vge_vf (InType *vs2, InType rs1, OutType *vd, int vm, MaskType *v0, int vl)
- int vge_vv (InType *vs2, InType *vs1, OutType *vd, int vm, MaskType *v0, int vl)

Public 属性

- int **debug**

3.4.1 详细描述

template<typename InType, typename OutType, typename MaskType>
class Vcompare< InType, OutType, MaskType >

向量比较指令

比较指令的作用通常是为了产生屏蔽向量的值。比较指令包括相等(==),不相等(!=), 大于(>),小于(<),大于等于(>=),小于等于(<=)等类型

3.4.2 成员函数说明

3.4.2.1 veq_vf()

```
template<typename InType , typename OutType , typename MaskType >
int Vcompare< InType, OutType, MaskType >::veq_vf (
            InType * vs2,
            InType rs1,
            OutType * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

veq_vf() vfeq.vf vd, vs2, rs1, vm Compare equal(==)

参数

vs2	源操作向量基地址
rs1	源标量操作数
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.4.2.2 veq_vv()

```
template<typename InType , typename OutType , typename MaskType >
int Vcompare< InType, OutType, MaskType >::veq_vv (
            InType * vs2,
            InType * vs1,
            OutType * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

veq_vv() vfeq.vv vd, vs2, vs1, vm Compare equal(==)

参数

vs2	源操作向量二基地址
vs1	源操作向量一基地址
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.4.2.3 vge_vf()

```
template<typename InType , typename OutType , typename MaskType >
int Vcompare< InType, OutType, MaskType >::vge_vf (
```

```
            InType * vs2,
            InType rs1,
            OutType * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vge_vf() vfge.vf vd, vs2, rs1, vm Compare greater than or equal (>=)

参数

vs2	源操作向量基地址
rs1	源标量操作数
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.4.2.4 vge_vv()

```
template<typename InType , typename OutType , typename MaskType >
int Vcompare< InType, OutType, MaskType >::vge_vv (
            InType * vs2,
            InType * vs1,
            OutType * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vge_vv() vfge.vv vd, vs2, vs1, vm Compare greater than or equal (>=)

参数

vs2	源操作向量二基地址
vs1	源操作向量一基地址
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.4.2.5 vgt_vf()

```
template<typename InType , typename OutType , typename MaskType >
int Vcompare< InType, OutType, MaskType >::vgt_vf (
            InType * vs2,
            InType rs1,
            OutType * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vgt_vf() vfgt.vf vd, vs2, rs1, vm Compare greater than(>)

参数

vs2	源操作向量基地址
rs1	源标量操作数
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.4.2.6 vgt_vv()

```
template<typename InType , typename OutType , typename MaskType >
int Vcompare< InType, OutType, MaskType >::vgt_vv (
            InType * vs2,
            InType * vs1,
            OutType * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vgt_vv() vfgt.vv vd, vs2, vs1, vm Compare greater than(>)

参数

vs2	源操作向量二基地址
vs1	源操作向量一基地址
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.4.2.7 vle_vf()

```
template<typename InType , typename OutType , typename MaskType >
int Vcompare< InType, OutType, MaskType >::vle_vf (
             InType * vs2,
             InType rs1,
             OutType * vd,
             int vm,
             MaskType * v0,
             int vl )  [inline]
```

vle_vf() vfle.vf vd, vs2, rs1, vm Compare less than or equal(<=)

参数

vs2	源操作向量基地址
rs1	源标量操作数
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.4.2.8 vle_vv()

```
template<typename InType , typename OutType , typename MaskType >
int Vcompare< InType, OutType, MaskType >::vle_vv (
             InType * vs2,
             InType * vs1,
             OutType * vd,
             int vm,
             MaskType * v0,
             int vl )  [inline]
```

vle_vv() vfle.vv vd, vs2, vs1, vm Compare less than or equal(<=)

参数

vs2	源操作向量二基地址

参数

vs1	源操作向量一基地址
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.4.2.9 vlt_vf()

```
template<typename InType , typename OutType , typename MaskType >
int Vcompare< InType, OutType, MaskType >::vlt_vf (
            InType * vs2,
            InType rs1,
            OutType * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vlt_vf() vflt.vf vd, vs2, rs1, vm Compare less than(<)

参数

vs2	源操作向量基地址
rs1	源标量操作数
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.4.2.10 vlt_vv()

```
template<typename InType , typename OutType , typename MaskType >
int Vcompare< InType, OutType, MaskType >::vlt_vv (
```

```
            InType * vs2,
            InType * vs1,
            OutType * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vlt_vv() vflt.vv vd, vs2, vs1, vm Compare less than(<)

参数

vs2	源操作向量二基地址
vs1	源操作向量一基地址
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.4.2.11 vne_vf()

```
template<typename InType , typename OutType , typename MaskType >
int Vcompare< InType, OutType, MaskType >::vne_vf (
            InType * vs2,
            InType rs1,
            OutType * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vne_vf() vfne.vf vd, vs2, rs1, vm Compare not equal(!=)

参数

vs2	源操作向量基地址
rs1	源标量操作数
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.4.2.12 vne_vv()

```
template<typename InType , typename OutType , typename MaskType >
int Vcompare< InType, OutType, MaskType >::vne_vv (
             InType * vs2,
             InType * vs1,
             OutType * vd,
             int vm,
             MaskType * v0,
             int vl )  [inline]
```

vne_vv() vfne.vv vd, vs2, vs1, vm Compare not equal(!=)

参数

vs2	源操作向量二基地址
vs1	源操作向量一基地址
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

该类的文档由以下文件生成:

- eigen3_ops.h

3.5 Vext< Type > 模板类 参考

整数提取指令

```
#include <eigen3_ops.h>
```

Public 类型

- typedef Map< Matrix< Type, 1, Dynamic > > **VextVecMap**

Public 成员函数

- int vext_x_v (Type *vs2, Type *rd, uint16_t rs1, int vl)

Public 属性

- int **debug**

3.5.1 详细描述

template<typename Type>
class Vext< Type >

整数提取指令

选取源向量寄存器中的一个元素

3.5.2 成员函数说明

3.5.2.1 vext_x_v()

```
template<typename Type >
int Vext< Type >::vext_x_v (
            Type * vs2,
            Type * rd,
            uint16_t rs1,
            int vl )  [inline]
```

vext_x_v() vext.x.v rd = vs2[rs1]

如果索引超出范围则rd会被置为0，不会认为指令错误

参数

vs2	源操作向量基地址
rs1	元素索引
vd	目的数存放地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

该类的文档由以下文件生成:

- eigen3_ops.h

3.6 Vfcvt< MaskType > 模板类 参考

浮点/整数类型转换指令

```
#include <eigen3_ops.h>
```

Public 类型

- typedef Map< Matrix< uint16_t, 1, Dynamic > > **VfcvtU16VecMap**
- typedef Map< Matrix< int16_t, 1, Dynamic > > **VfcvtI16VecMap**
- typedef Map< Matrix< half, 1, Dynamic > > **VfcvtHalfVecMap**
- typedef Map< Matrix< MaskType, 1, Dynamic > > **VfcvtMaskVecMap**

Public 成员函数

- int vfcvt_f_x_v (int16_t *vs2, half *vd, int vm, MaskType *v0, int vl)
- int vfcvt_f_xu_v (uint16_t *vs2, half *vd, int vm, MaskType *v0, int vl)

Public 属性

- int **debug**

3.6.1 详细描述

template<typename MaskType>
class Vfcvt< MaskType >

浮点/整数类型转换指令

九章处理器只支持 int16/uint16 到 fp16 的转换指令,即支持 vfcvt.f.xu.v 和 vfcvt.f.x.v。 此外,该执行转换指令时 SEW 必须为 16b,否则将触发非法指令异常。

3.6.2 成员函数说明

3.6.2.1 vfcvt_f_x_v()

```
template<typename MaskType >
int Vfcvt< MaskType >::vfcvt_f_x_v (
            int16_t * vs2,
            half * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vfcvt_f_x_v() vfcvt.f.x.v

convert signed integer to fp16 (int16 -> fp16)

参数

vs2	源操作向量基地址
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.6.2.2 vfcvt_f_xu_v()

```
template<typename MaskType >
int Vfcvt< MaskType >::vfcvt_f_xu_v (
            uint16_t * vs2,
            half * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vfcvt_f_xu_v() vfcvt.f.xu.v

convert uinsigned integer to fp16 (uint16 -> fp16)

参数

vs2	源操作向量基地址
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

该类的文档由以下文件生成:

- eigen3_ops.h

3.7 Vma< Type, MaskType > 模板类 参考

单宽度向量乘加(FMA)指令

`#include <eigen3_ops.h>`

Public 类型

- typedef Map< Matrix< Type, 1, Dynamic > > **VmaVecMap**
- typedef Map< Matrix< MaskType, 1, Dynamic > > **VmaMaskVecMap**

Public 成员函数

- int vmacc_vf (Type *vs2, Type rs1, Type *vd, int vm, MaskType *v0, int vl)
- int vmacc_vv (Type *vs2, Type *vs1, Type *vd, int vm, MaskType *v0, int vl)
- int vnmacc_vf (Type *vs2, Type rs1, Type *vd, int vm, MaskType *v0, int vl)
- int vnmacc_vv (Type *vs2, Type *vs1, Type *vd, int vm, MaskType *v0, int vl)
- int vmsac_vf (Type *vs2, Type rs1, Type *vd, int vm, MaskType *v0, int vl)
- int vmsac_vv (Type *vs2, Type *vs1, Type *vd, int vm, MaskType *v0, int vl)
- int vnmsac_vf (Type *vs2, Type rs1, Type *vd, int vm, MaskType *v0, int vl)
- int vnmsac_vv (Type *vs2, Type *vs1, Type *vd, int vm, MaskType *v0, int vl)
- int vmadd_vf (Type *vs2, Type rs1, Type *vd, int vm, MaskType *v0, int vl)
- int vmadd_vv (Type *vs2, Type *vs1, Type *vd, int vm, MaskType *v0, int vl)
- int vnmadd_vf (Type *vs2, Type rs1, Type *vd, int vm, MaskType *v0, int vl)
- int vnmadd_vv (Type *vs2, Type *vs1, Type *vd, int vm, MaskType *v0, int vl)
- int vmsub_vf (Type *vs2, Type rs1, Type *vd, int vm, MaskType *v0, int vl)
- int vmsub_vv (Type *vs2, Type *vs1, Type *vd, int vm, MaskType *v0, int vl)
- int vnmsub_vf (Type *vs2, Type rs1, Type *vd, int vm, MaskType *v0, int vl)
- int vnmsub_vv (Type *vs2, Type *vs1, Type *vd, int vm, MaskType *v0, int vl)
- int vmax_vf (Type *vs2, Type rs1, Type *vd, int vm, MaskType *v0, int vl)
- int vmax_vv (Type *vs2, Type *vs1, Type *vd, int vm, MaskType *v0, int vl)
- int vmin_vf (Type *vs2, Type rs1, Type *vd, int vm, MaskType *v0, int vl)
- int vmin_vv (Type *vs2, Type *vs1, Type *vd, int vm, MaskType *v0, int vl)

Public 属性

- int **debug**

3.7.1 详细描述

template<typename Type, typename MaskType>
class Vma< Type, MaskType >

单宽度向量乘加(FMA)指令

包含乘累加(macc), 乘累减(msac), 乘加(madd)， 乘减(msub) 支持任意数据类型

3.7.2 成员函数说明

3.7.2.1 vmacc_vf()

```
template<typename Type , typename MaskType >
int Vma< Type, MaskType >::vmacc_vf (
            Type * vs2,
            Type rs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vmacc_vf() vfmacc.vf

参数

vs2	源操作向量基地址
rs1	源标量操作数
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.7.2.2 vmacc_vv()

```
template<typename Type , typename MaskType >
int Vma< Type, MaskType >::vmacc_vv (
            Type * vs2,
            Type * vs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vmacc_vv() vfmacc.vv

参数

vs2	源操作向量二基地址
vs1	源操作向量一基地址
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.7.2.3 vmadd_vf()

```
template<typename Type , typename MaskType >
int Vma< Type, MaskType >::vmadd_vf (
```

```
            Type * vs2,
            Type rs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vmadd_vf() vfmadd.vf

vfmadd.vf vd, rs1, vs2, vm # vd[i] = +(vd[i] * f[rs1]) + vs2[i]

参数

vs2	源操作向量基地址
rs1	源标量操作数
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.7.2.4 vmadd_vv()

```
template<typename Type , typename MaskType >
int Vma< Type, MaskType >::vmadd_vv (
            Type * vs2,
            Type * vs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vmadd_vv() vfmadd.vv

vfmadd.vv vd, vs1, vs2, vm # vd[i] = +(vd[i] * vs1[i]) + vs2[i]

参数

vs2	源操作向量二基地址
vs1	源操作向量一基地址
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.7.2.5 vmax_vf()

```
template<typename Type , typename MaskType >
int Vma< Type, MaskType >::vmax_vf (
            Type * vs2,
            Type rs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vmax_vf() vfmax.vf

参数

vs2	源操作向量基地址
rs1	源标量操作数
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.7.2.6 vmax_vv()

```
template<typename Type , typename MaskType >
int Vma< Type, MaskType >::vmax_vv (
            Type * vs2,
            Type * vs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vmax_vv() vfmax.vv

参数

vs2	源操作向量二基地址

参数

vs1	源操作向量一基地址
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.7.2.7 vmin_vf()

```
template<typename Type , typename MaskType >
int Vma< Type, MaskType >::vmin_vf (
            Type * vs2,
            Type rs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vmin_vf() vfmin.vf

参数

vs2	源操作向量基地址
rs1	源标量操作数
vd	目的向量基地址test_vsub
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.7.2.8 vmin_vv()

```
template<typename Type , typename MaskType >
int Vma< Type, MaskType >::vmin_vv (
```

```
            Type * vs2,
            Type * vs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vmin_vv() vfmin.vv

参数

vs2	源操作向量二基地址
vs1	源操作向量一基地址
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.7.2.9 vmsac_vf()

```
template<typename Type , typename MaskType >
int Vma< Type, MaskType >::vmsac_vf (
            Type * vs2,
            Type rs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vmsac_vf() vfmsac.vf

vd[i] = +(f[rs1] * vs2[i]) - vd[i]

参数

vs2	源操作向量基地址
rs1	源标量操作数
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.7.2.10 vmsac_vv()

```
template<typename Type , typename MaskType >
int Vma< Type, MaskType >::vmsac_vv (
            Type * vs2,
            Type * vs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vmsac_vv() vfmsac.vv

vd[i] = +(vs1[i] * vs2[i]) - vd[i]

参数

vs2	源操作向量二基地址
vs1	源操作向量一基地址
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.7.2.11 vmsub_vf()

```
template<typename Type , typename MaskType >
int Vma< Type, MaskType >::vmsub_vf (
            Type * vs2,
            Type rs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vmsub_vf() vfmsub.vf

vfmsub.vf vd, rs1, vs2, vm # vd[i] = +(vd[i] * f[rs1]) - vs2[i]

参数

vs2	源操作向量基地址
rs1	源标量操作数
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.7.2.12 vmsub_vv()

```
template<typename Type , typename MaskType >
int Vma< Type, MaskType >::vmsub_vv (
            Type * vs2,
            Type * vs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vmsub_vv() vfmsub.vv

vfmsub.vv vd, vs1, vs2, vm # vd[i] = +(vd[i] * vs1[i]) - vs2[i]

参数

vs2	源操作向量二基地址
vs1	源操作向量一基地址
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.7.2.13 vnmacc_vf()

```
template<typename Type , typename MaskType >
int Vma< Type, MaskType >::vnmacc_vf (
```

```
            Type * vs2,
            Type rs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vnmacc_vf() vfnmacc.vf

vd[i] = -(f[rs1] * vs2[i]) - vd[i]

参数

vs2	源操作向量基地址
rs1	源标量操作数
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.7.2.14 vnmacc_vv()

```
template<typename Type , typename MaskType >
int Vma< Type, MaskType >::vnmacc_vv (
            Type * vs2,
            Type * vs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vnmacc_vv() vfnmacc.vv

vd[i] = -(vs1[i] * vs2[i]) - vd[i]

参数

vs2	源操作向量二基地址
vs1	源操作向量一基地址
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.7.2.15 vnmadd_vf()

```
template<typename Type , typename MaskType >
int Vma< Type, MaskType >::vnmadd_vf (
            Type * vs2,
            Type rs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vnmadd_vf() vfnmadd.vf

vfnmadd.vf vd, rs1, vs2, vm # vd[i] = -(vd[i] * f[rs1]) - vs2[i]

参数

vs2	源操作向量基地址
rs1	源标量操作数
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.7.2.16 vnmadd_vv()

```
template<typename Type , typename MaskType >
int Vma< Type, MaskType >::vnmadd_vv (
            Type * vs2,
            Type * vs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vnmadd_vv() vfnmadd.vv

vfnmadd.vv vd, vs1, vs2, vm # vd[i] = -(vd[i] * vs1[i]) - vs2[i]

参数

vs2	源操作向量二基地址
vs1	源操作向量一基地址
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.7.2.17 vnmsac_vf()

```
template<typename Type , typename MaskType >
int Vma< Type, MaskType >::vnmsac_vf (
            Type * vs2,
            Type rs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vnmsac_vf() vfnmsac.vf

vd[i] = -(f[rs1] * vs2[i]) + vd[i]

参数

vs2	源操作向量基地址
rs1	源标量操作数
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.7.2.18 vnmsac_vv()

```
template<typename Type , typename MaskType >
int Vma< Type, MaskType >::vnmsac_vv (
```

```
            Type * vs2,
            Type * vs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vnmsac_vv() vfnmsac.vv

d[i] = -(vs1[i] * vs2[i]) + vd[i]

参数

vs2	源操作向量二基地址
vs1	源操作向量一基地址
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.7.2.19 vnmsub_vf()

```
template<typename Type , typename MaskType >
int Vma< Type, MaskType >::vnmsub_vf (
            Type * vs2,
            Type rs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vnmsub_vf() vfnmsub.vf

vfnmsub.vf vd, rs1, vs2, vm # vd[i] = -(vd[i] * f[rs1]) + vs2[i]

参数

vs2	源操作向量基地址
rs1	源标量操作数
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.7.2.20 vnmsub_vv()

```
template<typename Type , typename MaskType >
int Vma< Type, MaskType >::vnmsub_vv (
            Type * vs2,
            Type * vs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vnmsub_vv() vfnmsub.vv

vfnmsub.vv vd, vs1, vs2, vm # vd[i] = -(vd[i] * vs1[i]) + vs2[i]

参数

vs2	源操作向量二基地址
vs1	源操作向量一基地址
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

该类的文档由以下文件生成:

- eigen3_ops.h

3.8 Vmerge< Type, MaskType > 模板类 参考

向量浮点合并指令类

```
#include <eigen3_ops.h>
```

Public 类型

- typedef Map< Matrix< Type, 1, Dynamic > > **VmergeDataVecMap**
- typedef Map< Matrix< MaskType, 1, Dynamic > > **VmergeMaskVecMap**

Public 成员函数

- int vmerge_vf (Type *vs2, Type rs1, Type *vd, int vm, MaskType *v0, int vl)

Public 属性

- int **debug**

3.8.1 详细描述

template<typename Type, typename MaskType>
class Vmerge< Type, MaskType >

向量浮点合并指令类

虽然目前设计文档中仅有一个操作，但本接口实际支持任意数据类型的merge 当然，从接口格式上限制了输入向量，输出向量，输入标量这三者的数据类型必须一致 mask向量数据类型可以独立指定

Type 输入向量，输出向量，输入标量的数据类型 MaskType mask向量的数据类型

3.8.2 成员函数说明

3.8.2.1 vmerge_vf()

```
template<typename Type , typename MaskType >
int Vmerge< Type, MaskType >::vmerge_vf (
            Type * vs2,
            Type rs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vmerge_vf() vfmerge.vf

参数

vs2	源操作向量基地址
rs1	源标量操作数
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

该类的文档由以下文件生成:

- eigen3_ops.h

3.9 Vmul< Type, MaskType > 模板类 参考

单宽度向量乘法指令

```
#include <eigen3_ops.h>
```

Public 类型

- typedef Map< Matrix< Type, 1, Dynamic > > **VmulVecMap**
- typedef Map< Matrix< MaskType, 1, Dynamic > > **VmulMaskVecMap**

Public 成员函数

- int vmul_vf (Type *vs2, Type rs1, Type *vd, int vm, MaskType *v0, int vl)
- int vmul_vv (Type *vs2, Type *vs1, Type *vd, int vm, MaskType *v0, int vl)

Public 属性

- int **debug**

3.9.1 详细描述

template<typename Type, typename MaskType>
class Vmul< Type, MaskType >

单宽度向量乘法指令

目的元素的宽度和源操作数中的元素宽度保持一致， 可以通过Type指定数据类型

3.9.2 成员函数说明

3.9.2.1 vmul_vf()

```
template<typename Type , typename MaskType >
int Vmul< Type, MaskType >::vmul_vf (
             Type * vs2,
             Type rs1,
             Type * vd,
             int vm,
             MaskType * v0,
             int vl )  [inline]
```

vmul_vf() vfmul.vf

参数

vs2	源操作向量基地址
rs1	源标量操作数
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.9.2.2 vmul_vv()

```
template<typename Type , typename MaskType >
int Vmul< Type, MaskType >::vmul_vv (
            Type * vs2,
            Type * vs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vmul_vv() vfmul.vv

参数

vs2	源操作向量二基地址
vs1	源操作向量一基地址
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

该类的文档由以下文件生成:

- eigen3_ops.h

3.10 Vsgnj< Type, MaskType > 模板类 参考

向量浮点符号注入指令

`#include <eigen3_ops.h>`

Public 类型

- typedef Map< Matrix< Type, 1, Dynamic > > **VsgnjVecMap**
- typedef Map< Matrix< MaskType, 1, Dynamic > > **VsgnjMaskVecMap**

Public 成员函数

- int vsgnj_vv (Type *vs2, Type *vs1, Type *vd, int vm, MaskType *v0, int vl)
- int vsgnj_vf (Type *vs2, Type rs1, Type *vd, int vm, MaskType *v0, int vl)
- int vsgnjn_vv (Type *vs2, Type *vs1, Type *vd, int vm, MaskType *v0, int vl)
- int vsgnjn_vf (Type *vs2, Type rs1, Type *vd, int vm, MaskType *v0, int vl)
- int vsgnjx_vv (Type *vs2, Type *vs1, Type *vd, int vm, MaskType *v0, int vl)
- int vsgnjx_vf (Type *vs2, Type rs1, Type *vd, int vm, MaskType *v0, int vl)

Public 属性

- int **debug**

3.10.1 详细描述

template<typename Type, typename MaskType>
class Vsgnj< Type, MaskType >

向量浮点符号注入指令

向量浮点符号注入(Sign-Injection)指令的运算结果的指数和尾数由第一个源操作数 vs2 提供

3.10.2 成员函数说明

3.10.2.1 vsgnj_vf()

```
template<typename Type , typename MaskType >
int Vsgnj< Type, MaskType >::vsgnj_vf (
            Type * vs2,
            Type rs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vsgnj_vf() vfsgnj.vf

参数

vs2	源操作向量基地址
rs1	源标量操作数
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.10.2.2 vsgnj_vv()

```
template<typename Type , typename MaskType >
int Vsgnj< Type, MaskType >::vsgnj_vv (
             Type * vs2,
             Type * vs1,
             Type * vd,
             int vm,
             MaskType * v0,
             int vl )  [inline]
```

vsgnj_vv() vfsgnj.vv

参数

vs2	源操作向量二基地址
vs1	源操作向量一基地址
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.10.2.3 vsgnjn_vf()

```
template<typename Type , typename MaskType >
int Vsgnj< Type, MaskType >::vsgnjn_vf (
             Type * vs2,
             Type rs1,
             Type * vd,
             int vm,
             MaskType * v0,
             int vl )  [inline]
```

vsgnjn_vf() vfsgnjn.vf

参数

vs2	源操作向量基地址

参数

rs1	源标量操作数
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.10.2.4 vsgnjn_vv()

```
template<typename Type , typename MaskType >
int Vsgnj< Type, MaskType >::vsgnjn_vv (
            Type * vs2,
            Type * vs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vsgnjn_vv() vfsgnjn.vv

参数

vs2	源操作向量二基地址
vs1	源操作向量一基地址
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.10.2.5 vsgnjx_vf()

```
template<typename Type , typename MaskType >
int Vsgnj< Type, MaskType >::vsgnjx_vf (
```

```
            Type * vs2,
            Type rs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vsgnjx_vf() vfsgnjx.vf

参数

vs2	源操作向量基地址
rs1	源标量操作数
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.10.2.6 vsgnjx_vv()

```
template<typename Type , typename MaskType >
int Vsgnj< Type, MaskType >::vsgnjx_vv (
            Type * vs2,
            Type * vs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vsgnjx_vv() vfsgnjx.vv

参数

vs2	源操作向量二基地址
vs1	源操作向量一基地址
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

该类的文档由以下文件生成:

- eigen3_ops.h

3.11 Vsub< Type, MaskType > 模板类 参考

单宽度向量减法指令

```
#include <eigen3_ops.h>
```

Public 类型

- typedef Map< Matrix< Type, 1, Dynamic > > **VsubVecMap**
- typedef Map< Matrix< MaskType, 1, Dynamic > > **VsubMaskVecMap**

Public 成员函数

- int vsub_vf (Type *vs2, Type rs1, Type *vd, int vm, MaskType *v0, int vl)
- int vsub_vv (Type *vs2, Type *vs1, Type *vd, int vm, MaskType *v0, int vl)

Public 属性

- int **debug**

3.11.1 详细描述

template<typename Type, typename MaskType>
class Vsub< Type, MaskType >

单宽度向量减法指令

因为芯片本身不支持浮点减法， 所以利用 a + (-b) 来实现 a - b 的操作 目的元素的宽度和源操作数中的元素宽度保持一致， 可以通过Type指定数据类型

3.11.2 成员函数说明

3.11.2.1 vsub_vf()

```
template<typename Type , typename MaskType >
int Vsub< Type, MaskType >::vsub_vf (
            Type * vs2,
            Type rs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vsub_vf() vfsub.vf

参数

vs2	源操作向量基地址
rs1	源标量操作数
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

3.11.2.2 vsub_vv()

```
template<typename Type , typename MaskType >
int Vsub< Type, MaskType >::vsub_vv (
            Type * vs2,
            Type * vs1,
            Type * vd,
            int vm,
            MaskType * v0,
            int vl )  [inline]
```

vsub_vv() vfsub.vv

参数

vs2	源操作向量二基地址
vs1	源操作向量一基地址
vd	目的向量基地址
vm	不可屏蔽标识， vm=0 可屏蔽， vm=1不可屏蔽
v0	mask向量基地址
vl	向量长度(准确的说应该是个数)

返回

执行结果

该类的文档由以下文件生成:

- eigen3_ops.h

Chapter 4

文件说明

4.1 eigen3_ops.cc 文件参考

The Source Code About Eigen3 To Spike Interface

```
#include "eigen3_ops.h"
```

eigen3_ops.cc 的引用(Include)关系图:

宏定义

- #define **MY_MATRIX_DEFINE**(Type)
- #define **SHAPE_STRIDE_INFO**(ss)

类型定义

- typedef Stride< Dynamic, Dynamic > **DynStride**

4.1.1 详细描述

The Source Code About Eigen3 To Spike Interface

Class CustomInsns 包含了所有的custom定制指令，但不包含vector指令， custom指令具有数据类型明确的特点，不需要模板类就能轻松实现，所以这些都放在 一个类里面（实际上没有任何抽象的意义）。vector指令更多的没有指定被操作数的数据 类型，所以为了使代码简洁，使用了大量的模板类，同时，因为无法统一每一个vector指令 的模板参数，所以基本上一类vector指令的实现封装在一个类中（实际上也没有经过抽象，纯粹是 按照不同的vector指令去区分该不该放在一个类里面）

作者

chenhao

4.1.2 宏定义说明

4.1.2.1 MY_MATRIX_DEFINE

```
#define MY_MATRIX_DEFINE(
              Type )
```

值:

```
typedef Matrix<Type, Dynamic, Dynamic, RowMajor> Matrix_##Type;               \
typedef Map<Matrix_##Type, Unaligned, Stride<Dynamic, Dynamic> > Map_##Type;
```

4.1.2.2 SHAPE_STRIDE_INFO

```
#define SHAPE_STRIDE_INFO(
              ss )
```

值:

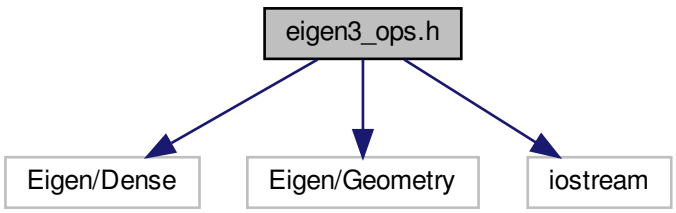
```
do {\
        if (debug) {\
            cout << endl << __FUNCTION__ << endl;\
            shapestride_dbg(ss);\
        }\
    } while(0)
```

4.2 eigen3_ops.h 文件参考

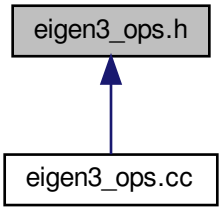
The Source Code About Eigen3 To Spike Interface

```
#include <Eigen/Dense>
#include <Eigen/Geometry>
#include <iostream>
```

eigen3_ops.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



类

- struct ShapeStride
  矩阵形状描述结构
- class CustomInsns
  *custom*扩展指令类
- class Vfcvt< MaskType >
  浮点/整数类型转换指令
- class Vadd< Type, MaskType >
  单宽度向量加法指令
- class Vsub< Type, MaskType >
  单宽度向量减法指令
- class Vmul< Type, MaskType >
  单宽度向量乘法指令
- class Vmerge< Type, MaskType >
  向量浮点合并指令类
- class Vext< Type >
  整数提取指令
- class Vma< Type, MaskType >
  单宽度向量乘加*(FMA)*指令
- class Vsgnj< Type, MaskType >
  向量浮点符号注入指令
- class Vcompare< InType, OutType, MaskType >
  向量比较指令

宏定义

- #define **GLOBAL_DBG** 1
- #define **DBG_VECTOR_VF**
- #define **DBG_VECTOR_VV**
- #define **VEQ_VF**
- #define **VEQ_VV**
- #define **VNE_VF**
- #define **VNE_VV**
- #define **VLT_VF**

- #define **VLT_VV**
- #define **VLE_VF**
- #define **VLE_VV**
- #define **VGT_VF**
- #define **VGT_VV**
- #define **VGE_VF**
- #define **VGE_VV**

枚举

- enum { **BR_OK** = 0, **BR_EPARAM** }
  返回值枚举

4.2.1 详细描述

The Source Code About Eigen3 To Spike Interface

Class CustomInsns 包含了所有的custom定制指令，但不包含vector指令， custom指令具有数据类型明确的特点，不需要模板类就能轻松实现，所以这些都放在 一个类里面（实际上没有任何抽象的意义）。vector指令更多的没有指定被操作数的数据 类型，所以为了使代码简洁，使用了大量的模板类，同时，因为无法统一每一个vector指令 的模板参数，所以基本上一类vector指令的实现封装在一个类中（实际上也没有经过抽象，纯粹是 按照不同的vector指令去区分该不该放在一个类里面）

作者

chenhao

4.2.2 宏定义说明

4.2.2.1 DBG_VECTOR_VF

```
#define DBG_VECTOR_VF
```

值:

```
do {                    \
    if (debug) {                                \
        cout << __FUNCTION__ << endl;           \
        cout << "vs2:\n" << vector_vs2 << endl; \
        cout << "rs1:\n" << rs1 << endl;        \
        cout << "vm:\n" << vm << endl;          \
        cout << "v0:\n" << vector_v0 << endl;   \
        cout << "vd:\n" << vector_vd << endl;   \
    }                                           \
} while(0)
```

4.2.2.2 DBG_VECTOR_VV

```
#define DBG_VECTOR_VV
```

值:

```
do {                    \
    if (debug) {                                    \
        cout << __FUNCTION__ << endl;               \
        cout << "vs2:\n" << vector_vs2 << endl;     \
        cout << "vs1:\n" << vector_vs1 << endl;     \
        cout << "vm:\n" << vm << endl;              \
        cout << "v0:\n" << vector_v0 << endl;       \
        cout << "vd:\n" << vector_vd << endl;       \
    }                                               \
} while(0)
```

4.2.2.3 VEQ_VF

```
#define VEQ_VF
```

值:

```
do {              \
            if (vector_vs2(i) == rs1)         \
                vector_vd(i) = (OutType)1;    \
            else                              \
                vector_vd(i) = (OutType)0;    \
        } while(0)
```

4.2.2.4 VEQ_VV

```
#define VEQ_VV
```

值:

```
do {                    \
            if (vector_vs2(i) == vector_vs1(i))     \
                vector_vd(i) = (OutType)1;          \
            else                                    \
                vector_vd(i) = (OutType)0;          \
        } while(0)
```

4.2.2.5 VGE_VF

```
#define VGE_VF
```

值:

```
do {                  \
            if (vector_vs2(i) >= rs1)             \
                vector_vd(i) = (OutType)1;        \
            else                                  \
                vector_vd(i) = (OutType)0;        \
        } while(0)
```

4.2.2.6 VGE_VV

```
#define VGE_VV
```

值:

```
do {                    \
            if (vector_vs2(i) >= vector_vs1(i))    \
                vector_vd(i) = (OutType)1;         \
            else                                   \
                vector_vd(i) = (OutType)0;         \
        } while(0)
```

4.2.2.7 VGT_VF

```
#define VGT_VF
```

值:

```
do {                \
            if (vector_vs2(i) > rs1)           \
                vector_vd(i) = (OutType)1;     \
            else                               \
                vector_vd(i) = (OutType)0;     \
        } while(0)
```

4.2.2.8 VGT_VV

```
#define VGT_VV
```

值:

```
do {                  \
            if (vector_vs2(i) > vector_vs1(i))    \
                vector_vd(i) = (OutType)1;        \
            else                                  \
                vector_vd(i) = (OutType)0;        \
        } while(0)
```

4.2.2.9 VLE_VF

```
#define VLE_VF
```

值:

```
do {          \
            if (vector_vs2(i) <= rs1)         \
                vector_vd(i) = (OutType)1;    \
            else                              \
                vector_vd(i) = (OutType)0;    \
        } while(0)
```

4.2.2.10 VLE_VV

```
#define VLE_VV
```

值:

```
do {                    \
            if (vector_vs2(i) <= vector_vs1(i))   \
                vector_vd(i) = (OutType)1;        \
            else                                  \
                vector_vd(i) = (OutType)0;        \
        } while(0)
```

4.2.2.11 VLT_VF

```
#define VLT_VF
```

值:

```
do {                \
            if (vector_vs2(i) < rs1)            \
                vector_vd(i) = (OutType)1;      \
            else                                \
                vector_vd(i) = (OutType)0;      \
        } while(0)
```

4.2.2.12 VLT_VV

```
#define VLT_VV
```

值:

```
do {                  \
            if (vector_vs2(i) < vector_vs1(i))    \
                vector_vd(i) = (OutType)1;        \
            else                                  \
                vector_vd(i) = (OutType)0;        \
        } while(0)
```

4.2.2.13 VNE_VF

```
#define VNE_VF
```

值:

```
do {                \
            if (vector_vs2(i) != rs1)           \
                vector_vd(i) = (OutType)1;      \
            else                                \
                vector_vd(i) = (OutType)0;      \
        } while(0)
```

4.2.2.14 VNE_VV

```
#define VNE_VV
```

值:

```
do {                        \
            if (vector_vs2(i) != vector_vs1(i))       \
                vector_vd(i) = (OutType)1;            \
            else                                      \
                vector_vd(i) = (OutType)0;            \
        } while(0)
```