riscv-v-spec0.9和0.8对比

软件所智能软件中心PLCT实验室 王鹏 实习生

2020//05/27

目 录

• 01 指令编码变动

在Vector Load/Store Whole Register指令部分，LOAD-FP主操作码下的Vector Load Whole Register指令和Vector Store Whole Register指令的格式发生了改变。在riscv-v-spec 0.8中的Inst(14-12)=0b111，在riscv-v-spec 0.9中的Inst(14-12)=0b000，如下所示：

Format for Vector Load Whole Register Instructions under LOAD-FP major opcode

31 29	28 26	25	24 20	19 15	14 12	11 7	6 0
nf	000	1	01000	rs1	000	vd	0000111
							VL<nf>R

Format for Vector Store Whole Register Instructions under STORE-FP major opcode

31 29	28 26	25	24 20	19 15	14 12	11 7	6 0
nf	000	1	01000	rs1	000	vs3	0100111
							VS<nf>R

```
Format for Vector Load Instructions under LOAD-FP major opcode
31 29 28 26  25  24      20 19       15 14   12 11      7 6     0
 nf   | mop | vm |  lumop   |    rs1    | width |    vd   |0000111| VL*  unit-stride
 nf   | mop | vm |   rs2    |    rs1    | width |    vd   |0000111| VLS* strided
 nf   | mop | vm |   vs2    |    rs1    | width |    vd   |0000111| VLX* indexed
  3     3     1      5           5         3         5       7

Format for Vector Store Instructions under STORE-FP major opcode
31 29 28 26  25  24      20 19       15 14   12 11      7 6     0
 nf   | mop | vm |  sumop   |    rs1    | width |   vs3   |0100111| VS*  unit-stride
 nf   | mop | vm |   rs2    |    rs1    | width |   vs3   |0100111| VSS* strided
 nf   | mop | vm |   vs2    |    rs1    | width |   vs3   |0100111| VSX* indexed
  3     3     1      5           5         3         5       7
```

riscv-v-spec 0.8

```
Format for Vector Load Instructions under LOAD-FP major opcode
31 29  28  27 26  25  24      20 19       15 14   12 11      7 6     0
 nf  | mew| mop | vm |  lumop   |    rs1    | width |    vd   |0000111| VL*  uni
 nf  | mew| mop | vm |   rs2    |    rs1    | width |    vd   |0000111| VLS* str
 nf  | mew| mop | vm |   vs2    |    rs1    | width |    vd   |0000111| VLX* ind
  3     1    2     1      5           5         3         5       7

Format for Vector Store Instructions under STORE-FP major opcode
31 29  28  27 26  25  24      20 19       15 14   12 11      7 6     0
 nf  | mew| mop | vm |  sumop   |    rs1    | width |   vs3   |0100111| VS*  uni
 nf  | mew| mop | vm |   rs2    |    rs1    | width |   vs3   |0100111| VSS* str
 nf  | mew| mop | vm |   vs2    |    rs1    | width |   vs3   |0100111| VSX* ind
  3     1    2     1      5           5         3         5       7
```

riscv-v-spec 0.9

Vector type register, vtype

vtype是只读的XLEN范围向量类型CSR，它的作用是提供用于解释向量寄存器文件内容的默认类型，并且只能通过vsetvl {i}指令进行更新。

Table 16. vtype register layout

Bits	Name	Description
XLEN-1	vill	Illegal value if set
XLEN-2:7		Reserved (write 0)
6:5	vediv[1:0]	Used by EDIV extension
4:2	vsew[2:0]	Standard element width (SEW) setting
1:0	vlmul[1:0]	Vector register group multiplier (LMUL) setting

riscv-v-spec 0.8

Bits	Name	Description
XLEN-1	vill	Illegal value if set
XLEN-2:8		Reserved (write 0)
7	vma	Vector mask agnostic
6	vta	Vector tail agnostic
5	vlmul[2]	Vector register group multiplier (LMUL) setting (fractional)
4:2	vsew[2:0]	Standard element width (SEW) setting
1:0	vlmul[1:0]	Vector register group multiplier (LMUL) setting

riscv-v-spec 0.9

在执行矢量指令时候，矢量尾部不可知vta修改了目标尾部元素，矢量掩码不可知vma修改了目标非活动掩蔽元素的行为。

1.1 Vector Operands

每个向量操作数都有一个有效元素宽度（effective element width，EEW）和一个有效LMUL（effective LMUL，EMUL），用于确定向量寄存器组中所有元素的大小和位置。默认情况下，对于大多数指令的大多数操作数，EEW = SEW和EMUL = LMUL。

一些向量指令的源向量操作数和目标向量操作数具有相同数量的元素，但宽度不同，因此EEW和EMUL分别不同于SEW和LMUL，但EEW / EMUL = SEW / LMUL。例如，大多数widening arithmetic指令的源组为EEW = SEW和EMUL = LMUL，而destination group是EEW = 2 * SEW和EMUL = 2 * LMUL。Narrowing指令的源操作数为EEW = 2 * SEW和EMUL = 2 * LMUL，但destination group是EEW = SEW和EMUL = LMUL。

向量操作数或结果可能会占用一个或多个向量寄存器，具体取决于EMUL，但始终使用组中编号最小的向量寄存器来指定。否则导致非法指令异常。

指令使用的最大向量寄存器组不能超过8个向量寄存器（即EMUL≤8，并且如果向量指令在一组中需要大于8个向量寄存器，则会引发非法指令异常。例如，当LMUL = 8时尝试进行扩展操作产生扩展的向量寄存器组结果将引发非法指令异常，因为这将意味着结果EMUL = 16。

• 02 寄存器改变

2.1 Vector Load/Store Instruction Encoding

向量加载和存储在标量浮点加载和存储主操作码（LOAD-FP / STORE-FP）中进行编码。向量加载和存储编码重新利用部分标准标量浮点加载/存储12位立即数字段，以提供进一步的向量指令编码，其中位25保留标准向量掩码位。指定内存寻址模式在riscv-v-spec 0.8中是mop[2:0]，在riscv-v-spec 0.9中变为mop[1:0].

新增加了一个Field叫mew，即扩展存储元素的大小。向量存储操作直接在指令中对静态传输的数据进行EEW编码，以减少在混合宽度例程中访问内存时更改“vtype”的次数。索引操作在指令中使用显式EEW编码来设置指令的大小。使用的索引，并使用SEW / LMUL指定数据宽度。

Vector Load/Store Addressing Modes

在riscv-v-spec 0.8中，向量寻址模式使用3位mop [2：0]字段编码。现在，在riscv-v-spec 0.9中向量寻址模式使用2位mop [1：0]字段进行编码。

Table 1. encoding for loads

mop [1:0]		Description	Opcodes
0	0	unit-stride	VLE<EEW>
0	1	reserved	-
1	0	strided	VLSE<EEW>
1	1	indexed	VLXEI<EEW>

Table 2. encoding for stores

mop [1:0]		Description	Opcodes
0	0	unit-stride	VSE<EEW>
0	1	indexed-unordered	VSUXEI<EEW>
1	0	strided	VSSE<EEW>
1	1	indexed-ordered	VSXEI<EEW>

riscv-v-spec 0.9

Vector Load/Store Width Encoding

向量加载和存储将EEW直接编码在指令中。 EMUL的计算公式为EMUL = （EEW / SEW）* LMUL。如果EMUL超出范围（EMUL> 8或EMUL <1/8），则会引发非法指令异常。向量寄存器组必须具有用于所选EMUL的合法寄存器说明符，否则会引发非法指令。

向量加载和存储使用标准标量浮点加载和存储未声明的宽度值进行编码。 mew位（inst [28]）编码128位及以上的扩展内存大小。实现中必须支持向量加载和存储，直到EEW = ELEN。将向量加载/存储与不支持的EEW一起使用会引发非法指令异常。

mew	width [2:0]			Mem bits	Reg bits	Opcodes	
Standard scalar FP	x	0	0	1	16	FLEN	FLH/FSH
Standard scalar FP	x	0	1	0	32	FLEN	FLW/FSW
Standard scalar FP	x	0	1	1	64	FLEN	FLD/FSD
Standard scalar FP	x	1	0	0	128	FLEN	FLQ/FSQ
Vector 8b element	0	0	0	0	8	8	VLxE8/VSxE8
Vector 16b element	0	1	0	1	16	16	VLxE16/VSxE16

mew	width [2:0]			Mem bits	Reg bits	Opcodes	
Vector 32b element	0	1	1	0	32	32	VLxE32/VSxE32
Vector 64b element	0	1	1	1	64	64	VLxE64/VSxE64
Vector 128b element	1	0	0	0	128	128	VLxE128/VSxE128
Vector 256b element	1	1	0	1	256	256	VLxE256/VSxE256
Vector 512b element	1	1	1	0	512	512	VLxE512/VSxE512
Vector 1024b element	1	1	1	1	1024	1024	VLxE1024/VSxE1024

其中，Mem bits是内存中访问的每个元素的大小，Reg bits是寄存器中访问的每个元素的大小。

Vector Unit-Stride Instructions

在Vector unit-stride loads and stores部分，去掉了定义在riscv-v-spec 0.8中的单位步幅指令，增加了定义在riscv-v-spec 0.9中的单位步幅指令。

表 3 定义在 riscv-v-spec 0.8 中的单位步幅指令

vlb.v vd, (rs1), vm # 8b signed	vlwu.v vd, (rs1), vm # 32b unsigned
vlh.v vd, (rs1), vm # 16b signed	vsb.v vs3, (rs1), vm # 8b store
vlw.v vd, (rs1), vm # 32b signed	vsh.v vs3, (rs1), vm # 16b store
vlbu.v vd, (rs1), vm # 8b unsigned	vsw.v vs3, (rs1), vm # 32b store
vlhu.v vd, (rs1), vm # 16b unsigned	vse.v vs3, (rs1), vm # SEW store

表 4 定义在 riscv-v-spec 0.9 中的单位步幅指令

vle32.v vd, (rs1), vm # 32-bit loads
vse64.v vs3, (rs1), vm # 64-bit stores

Vector Indexed Instructions

在Vector Indexed Instructions部分，去掉了定义在riscv-v-spec 0.8中的Vector indexed loads and stores指令，Vector ordered-indexed store指令和Vector unordered-indexed store指令，增加了定义在riscv-v-spec 0.9中的Vector indexed loads and stores指令，Vector ordered -indexed store指令和Vector unordered-indexed store指令。

表 5 定义在 riscv-v-spec 0.8 中的 Vector Indexed 指令

vlxb.v vd, (rs1), vs2, vm # 8b	vlxwu.v vd, (rs1), vs2, vm # 32b unsigned
vlxh.v vd, (rs1), vs2, vm # 16b	vlxe.v vd, (rs1), vs2, vm # SEW
vlxw.v vd, (rs1), vs2, vm # 32b	vssb.v vs3, (rs1), rs2, vm # 8b
vlxbu.v vd, (rs1), vs2, vm # 8b unsigned	vssh.v vs3, (rs1), rs2, vm # 16b
vlxhu.v vd, (rs1), vs2, vm # 16b unsigned	vssw.v vs3, (rs1), rs2, vm # 32b
vsse.v vs3, (rs1), rs2, vm # SEW	

表 6 定义在 riscv-v-spec 0.9 中的 Vector Indexed 指令

vlxei16.v vd, (rs1), vs2, vm # vs2 data EEW = SEW, indices EEW = 16b
vsxei32.v vs3, (rs1), vs2, vm # SEW data, 32b indicess
vsuxei64.v vs3, (rs1), vs2, vm # SEW data, 64b indices

Unit-stride Fault-Only-First Loads

在Unit-stride Fault-Only-First Loads部分，去掉了定义在riscv-v-spec 0.8中的指令，增加了定义在riscv-v-spec 0.9中的指令

表 7 定义在 riscv-v-spec 0.8 中的 Vector Indexed 指令

vlbff.v vd, (rs1), vm # 8b	vlhuff.v vd, (rs1), vm # unsigned 16b
vlhff.v vd, (rs1), vm # 16b	vlwuff.v vd, (rs1), vm # unsigned 32b
vlwff.v vd, (rs1), vm # 32b	vleff.v vd, (rs1), vm # SEW
vlbuff.v vd, (rs1), vm # unsigned 8b	

表 8 定义在 riscv-v-spec 0.9 中的 Vector Indexed 指令

vle8ff.v vd, (rs1), vm

Vector Load/Store Segment Instructions

在 Vector Load/Store Segment Instructions 部分，去掉了定义在 riscv-v-spec 0.8 中的 Vector Unit-Stride Segment Loads and Stores 指令和 Vector Indexed Segment Loads and Stores 指令，Vector Strided Segment Loads and Stores 指令，增加了定义在 riscv-v-spec 0.9 中的 Vector Unit-Stride Segment Loads and Stores 指令和 Vector Indexed Segment Loads and Stores 指令。

表 9 定义在 riscv-v-spec 0.8 中的 Vector Indexed 指令

vlseg<nf>{b,h,w}.v vd, (rs1), vm # Unit-stride signed segment load template
vlseg<nf>e.v vd, (rs1), vm # Unit-stride segment load template
vlseg<nf>{b,h,w}u.v vd, (rs1), vm # Unit-stride unsigned segment load template
vsseg<nf>{b,h,w,e}.v vs3, (rs1), vm # Unit-stride segment store template
vlseg<nf>{b,h,w}ff.v vd, (rs1), vm # Unit-stride signed fault-only-first segment loa
vlseg<nf>eff.v vd, (rs1), vm # Unit-stride fault-only-first segment loads
vlseg<nf>{b,h,w}uff.v vd, (rs1), vm # Unit-stride unsigned fault-only-first segment
vlsseg<nf>{b,h,w}.v vd, (rs1), rs2, vm # Strided signed segment loads

vlsseg<nf>e.v vd, (rs1), rs2, vm # Strided segment loads
vlsseg<nf>{b,h,w}u.v vd, (rs1), rs2, vm # Strided unsigned segment loads
vssseg<nf>{b,h,w,e}.v vs3, (rs1), rs2, vm # Strided segment stores
vlxseg<nf>{b,h,w}.v vd, (rs1), vs2, vm # Indexed signed segment loads
vlxseg<nf>e.v vd, (rs1), vs2, vm # Indexed segment loads
vlxseg<nf>{b,h,w}u.v vd, (rs1), vs2, vm # Indexed unsigned segment loads
vsxseg<nf>{b,h,w,e}.v vs3, (rs1), vs2, vm # Indexed segment stores

表 10 定义在 riscv-v-spec 0.9 中的 Vector Indexed 指令

vlseg<nf>e<eew>.v vd, (rs1), vm # Unit-stride segment load template
vsseg<nf>e<eew>.v vs3, (rs1), vm # Unit-stride segment store template
vlseg<nf>e<eew>ff.v vd, (rs1), vm # Unit-stride fault-only-first segment loads
vlsseg<nf>e<eew>.v vd, (rs1), rs2, vm # Strided segment loads vssseg<nf>e<eew>.v vs3, (rs1), rs2, vm # Strided segment stores
vlxseg<nf>ei<eew>.v vd, (rs1), vs2, vm # Indexed segment loads vsxseg<nf>ei<eew>.v vs3, (rs1), vs2, vm # Indexed segment stores

2.2 Vector AMO Operations

AMO 具有与索引操作相同的索引 EEW 方案，不同之处在于没有'mew'位（假定为零），因此偏移量只能具有 EEW = 8、16、32、64，如表 11 所示。寄存器“ vs2”中字节偏移量的向量被添加到“rs1”中的标量基址寄存器中，以给出 AMO 操作的地址。AMO 主操作码下的 Vector AMO 指令格式不变。数据寄存器“ vs3”使用了动态 SEW 和 MUL 设置。

31 27	26	25	24 20	19 15	14 12	11 7	6 0
amoop	wd	vm	vs2	rs1	width	vs3/vd	0101111 VAMO*
5	1	1	5	5	3	5	7

表 11 Vector AMO width encoding

	Width [2:0]			Index EEW	Mem data bits	Reg data bits	Opcode
Standard scalar AMO	0	1	0	-	32	XLEN	AMO*.W
Standard scalar AMO	0	1	1	-	64	XLEN	AMO*.D
Standard scalar AMO	1	0	0	-	128	XLEN	AMO*.Q
Vector AMO	0	0	0	8	SEW	SEW	VAMO*EI8.V

	Width [2:0]			Index EEW	Mem data bits	Reg data bits	Opcode
Vector AMO	1	0	1	16	SEW	SEW	VAMO*EI16.V
Vector AMO	1	1	0	32	SEW	SEW	VAMO*EI32.V
Vector AMO	1	1	1	64	SEW	SEW	VAMO*EI64.V

其中 Mem 位表示内存中访问元素的大小，Reg 位表示寄存器中访问元素的大小，索引位（Index bits）是偏移量的 EEW。

Vector AMO的指令变换

在riscv-v-spec 0.8中， 32-bit vector AMOs指令和SEW-bit vector AMOs指令都被去掉了，如表12所示。在riscv-v-spec 0.9中，新增Vector AMOs for index EEW=32指令定义，如表13所示：

vamoswapw.v vd, (rs1), v2, vd, v0.t （riscv-v-spec 0.8）

vamoswapei32.v vd, (rs1), v2, vd， v0.t （riscv-v-spec 0.9）

表 12 定义在 riscv-v-spec 0.8 中的 Vector AMO 指令

vamoswapw.v vd, (rs1), v2, vd, v0.t	vamoorw.v vd, (rs1), v2, vd, v0.t
vamoswapw.v x0, (rs1), v2, vs3, v0.t	vamoorw.v x0, (rs1), v2, vs3, v0.t
vamoaddw.v vd, (rs1), v2, vd, v0.t	vamominw.v vd, (rs1), v2, vd, v0.t
vamoaddw.v x0, (rs1), v2, vs3, v0.t	vamominw.v x0, (rs1), v2, vs3, v0.t
vamoxorw.v vd, (rs1), v2, vd, v0.t	vamomaxw.v vd, (rs1), v2, vd, v0.t
vamoxorw.v x0, (rs1), v2, vs3, v0.t	vamomaxw.v x0, (rs1), v2, vs3, v0.t
vamoandw.v vd, (rs1), v2, vd, v0.t	vamominuw.v vd, (rs1), v2, vd, v0.t
vamoandw.v x0, (rs1), v2, vs3, v0.t	vamominuw.v x0, (rs1), v2, vs3, v0.t
vamomaxuw.v vd, (rs1), v2, vd, v0.t	vamomaxuw.v x0, (rs1), v2, vs3, v0.t
vamoswape.v vd, (rs1), v2, vd, v0.t	vamoswape.v x0, (rs1), v2, vs3, v0.t
vamoadde.v vd, (rs1), v2, vd, v0.t	vamoadde.v x0, (rs1), v2, vs3, v0.t
vamoxore.v vd, (rs1), v2, vd, v0.t	vamoxore.v x0, (rs1), v2, vs3, v0.t
vamoande.v vd, (rs1), v2, vd, v0.t	vamoande.v x0, (rs1), v2, vs3, v0.t
vamoore.v vd, (rs1), v2, vd, v0.t	vamoore.v x0, (rs1), v2, vs3, v0.t
vamomine.v vd, (rs1), v2, vd, v0.t	vamomine.v x0, (rs1), v2, vs3, v0.t
vamomaxe.v vd, (rs1), v2, vd, v0.t	vamomaxe.v x0, (rs1), v2, vs3, v0.t
vamominue.v vd, (rs1), v2, vd, v0.t	vamominue.v x0, (rs1), v2, vs3, v0.t
vamomaxue.v vd, (rs1), v2, vd, v0.t	vamomaxue.v x0, (rs1), v2, vs3, v0.t

表 13 定义在 riscv-v-spec 0.9 中的 Vector AMO 指令

vamoswapei32.v vd, (rs1), v2, vd, v0.t	vamoswapei32.v x0, (rs1), v2, vs3, v0.t
vamoaddei32.v vd, (rs1), v2, vd, v0.t	vamoaddei32.v x0, (rs1), v2, vs3, v0.t
vamoxorei32.v vd, (rs1), v2, vd, v0.t	vamoxorei32.v x0, (rs1), v2, vs3, v0.t
vamoandei32.v vd, (rs1), v2, vd, v0.t	vamoandei32.v x0, (rs1), v2, vs3, v0.t
vamoorei32.v vd, (rs1), v2, vd, v0.t	vamoorei32.v x0, (rs1), v2, vs3, v0.t
vamominei32.v vd, (rs1), v2, vd, v0.t	vamominei32.v x0, (rs1), v2, vs3, v0.t
vamomaxei32.v vd, (rs1), v2, vd, v0.t	vamomaxei32.v x0, (rs1), v2, vs3, v0.t
vamominuei32.v vd, (rs1), v2, vd, v0.t	vamominuei32.v x0, (rs1), v2, vs3, v0.t
vamomaxuei32.v vd, (rs1), v2, vd, v0.t	vamomaxuei32.v x0, (rs1), v2, vs3, v0.t

2.3 Vector Integer Extension

向量整数扩展指令对 EEW 小于 SEW 的源向量整数操作数进行零扩展或符号扩展，以填充目标中 SEW 大小的元素。源的 EEW 为目标的 1 / 2、1 / 4 或 1/8，而源的 EMUL 为（EEW / SEW）* LMUL。如果源 EEW 不是受支持的宽度，或者源 EMUL 不是受支持的 LMUL，则会引发非法指令异常。表 14 是 riscv-v-spec 0.9 中新增的 Vector Integer Extension 指令。

表 14 riscv-v-spec 0.9 中 Vector Integer Extension 新增指令

vzext.vf2 vd, vs2, vm	# Zero-extend SEW/2 source to SEW destination
vsext.vf2 vd, vs2, vm	# Sign-extend SEW/2 source to SEW destination
vzext.vf4 vd, vs2, vm	# Zero-extend SEW/4 source to SEW destination
vsext.vf4 vd, vs2, vm	# Sign-extend SEW/4 source to SEW destination
vzext.vf8 vd, vs2, vm	# Zero-extend SEW/8 source to SEW destination
vsext.vf8 vd, vs2, vm	# Sign-extend SEW/8 source to SEW destination

• 03 只改变掩码注释的指令

Vector Integer Merge Instructions

在 riscv-v-spec 0.8 中，Vector Integer Merge 的三条指令的注释 vd[i] = v0[i].LSB 变成了 vd[i] = v0.mask[i]的表示形式。

表 15 riscv-v-spec 0.9 版本的 Vector Integer Merge 注释变化

riscv-v-spec 0.8 版本	vmerge.vvm vd, vs2, vs1, v0 # vd[i] = v0[i].LSB ? vs1[i] : vs2[i]
	vmerge.vxm vd, vs2, rs1, v0 # vd[i] = v0[i].LSB ? x[rs1] : vs2[i]
	vmerge.vim vd, vs2, imm, v0 # vd[i] = v0[i].LSB ? imm : vs2[i]
riscv-v-spec 0.9 版本	vmerge.vvm vd, vs2, vs1, v0 # vd[i] = v0.mask[i] ? vs1[i] : vs2[i]
	vmerge.vxm vd, vs2, rs1, v0 # vd[i] = v0.mask[i] ? x[rs1] : vs2[i]
	vmerge.vim vd, vs2, imm, v0 # vd[i] = v0.mask[i] ? imm : vs2[i]

Vector Floating-Point Merge Instruction

在 riscv-v-spec 0.9 中，Vector Floating-Point Merge 指令的注释 vd[i] = v0[i].LSB 变成了 vd[i] = v0.mask[i]的表示形式：

表 15 riscv-v-spec 0.9 版本的 Vector Floating-Point Merge 指令注释变化

riscv-v-spec 0.8 版本	vfmerge.vfm vd, vs2, rs1, v0 # vd[i] = v0[i].LSB ? f[rs1] : vs2[i]
riscv-v-spec 0.9 版本	vfmerge.vfm vd, vs2, rs1, v0 # vd[i] = v0.mask[i] ? f[rs1] : vs2[i]

Vector Mask-Register Logical Instructions

向量掩码寄存器逻辑操作在掩码寄存器上进行。掩码寄存器中的每个元素都是一个位，因此这些指令都对单个向量寄存器进行操作，而与vtype中的vlmul字段的设置无关。它们不会更改vlmul的值。目的向量寄存器可以与任何一个源向量寄存器相同。

与其他向量指令一样，索引小于vstart的元素保持不变，并且vstart在执行后重置为零。矢量掩码逻辑指令始终处于未掩码状态，因此没有无效元素。根据vtype中的vta设置（节向量尾部不可知和矢量掩码不可知vta和vma），处理超过vl的掩码元素（尾部元素）。

```
vmand.mm vd, vs2, vs1      # vd[i] =   vs2[i].LSB &&  vs1[i].LSB
vmnand.mm vd, vs2, vs1     # vd[i] = !(vs2[i].LSB &&  vs1[i].LSB)
vmandnot.mm vd, vs2, vs1   # vd[i] =   vs2[i].LSB && !vs1[i].LSB
vmxor.mm  vd, vs2, vs1     # vd[i] =   vs2[i].LSB ^^  vs1[i].LSB
vmor.mm  vd, vs2, vs1      # vd[i] =   vs2[i].LSB ||  vs1[i].LSB
vmnor.mm  vd, vs2, vs1     # vd[i] = !(vs2[i[.LSB ||  vs1[i].LSB)
vmornot.mm  vd, vs2, vs1   # vd[i] =   vs2[i].LSB || !vs1[i].LSB
vmxnor.mm vd, vs2, vs1     # vd[i] = !(vs2[i].LSB ^^  vs1[i].LSB)
```

riscv-v-spec 0.8

```
vmand.mm vd, vs2, vs1     # vd[i] =   vs2.mask[i] &&  vs1.mask[i]
vmnand.mm vd, vs2, vs1    # vd[i] = !(vs2.mask[i] &&  vs1.mask[i])
vmandnot.mm vd, vs2, vs1  # vd[i] =   vs2.mask[i] && !vs1.mask[i]
vmxor.mm  vd, vs2, vs1    # vd[i] =   vs2.mask[i] ^^  vs1.mask[i]
vmor.mm  vd, vs2, vs1     # vd[i] =   vs2.mask[i] ||  vs1.mask[i]
vmnor.mm  vd, vs2, vs1    # vd[i] = !(vs2.mask[i] ||  vs1.mask[i])
vmornot.mm  vd, vs2, vs1  # vd[i] =   vs2.mask[i] || !vs1.mask[i]
vmxnor.mm vd, vs2, vs1    # vd[i] = !(vs2.mask[i] ^^  vs1.mask[i])
```

riscv-v-spec 0.9

Vector mask population count vpopc

vpopc.m指令对向量源掩码寄存器的活动元素的值为1的掩码元素进行计数，并将结果写入标量寄存器。

- vpopc.m rd, vs2, v0.t # x[rd] = sum_i (vs2[i].LSB && v0[i].LSB)

+ vpopc.m rd, vs2, v0.t # x[rd] = sum_i (vs2.mask[i] && v0.mask[i])

Vector Mask Instructions

表 18 Vector Mask Instructions 注释改变指令

vmand.mm vd, vs2, vs1	vmor.mm vd, vs2, vs1
vmnand.mm vd, vs2, vs1	vmnor.mm vd, vs2, vs1
vmandnot.mm vd, vs2, vs1	vmornot.mm vd, vs2, vs1
vmxor.mm vd, vs2, vs1	vmxnor.mm vd, vs2, vs1

vd[i] = vs2[i].LSB && vs1[i].LSB riscv-v-spec 0.8

vd[i] = !(vs2.mask[i] ^^ vs1.mask[i]) riscv-v-spec 0.9

diff中的一部分指令是由于调整格式出现的不能，其实指令，寄存器，编码和注释都没有改变。

```
 ----
-vfncvt.xu.f.w vd, vs2, vm   # Convert double-width float to unsigned integer.
-vfncvt.x.f.w  vd, vs2, vm   # Convert double-width float to signed integer.
+vfncvt.xu.f.w vd, vs2, vm   # Convert double-width float to unsigned integer.
+vfncvt.x.f.w  vd, vs2, vm   # Convert double-width float to signed integer.
+
+vfncvt.rtz.xu.f.w vd, vs2, vm   # Convert double-width float to unsigned integer, truncating.
+vfncvt.rtz.x.f.w  vd, vs2, vm   # Convert double-width float to signed integer, truncating.

-vfncvt.f.xu.w vd, vs2, vm   # Convert double-width unsigned integer to float.
-vfncvt.f.x.w  vd, vs2, vm   # Convert double-width signed integer to float.
+vfncvt.f.xu.w vd, vs2, vm   # Convert double-width unsigned integer to float.
+vfncvt.f.x.w  vd, vs2, vm   # Convert double-width signed integer to float.
```

• 04 新增指令

新增 rtz variants

在 riscv-v-spec 0.9 中提供了 rtz variants，它的功能是加速从浮点到整数的截断转换，rtz variants 在 C 和 Java 之类的语言中很常见。

Single-Width Floating-Point/Integer Type-Convert Instructions

在 riscv-v-spec 0.9 中，Single-Width Floating-Point/Integer Type-Convert 指令新增加了 vfcvt.rtz.xu.f.v 和 vfcvt.rtz.x.f.v 两条指令。

表 15 Single-Width Floating-Point/Integer Type-Convert 新增指令

vfcvt.rtz.xu.f.v vd, vs2, vm # Convert float to unsigned integer, truncating.
vfcvt.rtz.x.f.v vd, vs2, vm # Convert float to signed integer, truncating.

Widening Floating-Point/Integer Type-Convert Instructions

在 riscv-v-spec 0.9 中，Widening Floating-Point/Integer Type-Convert 指令新增加了 vfwcvt.rtz.xu.f.v 和 vfwcvt.rtz.x.f.v 两条指令。

表 16 Widening Floating-Point/Integer Type-Convert 新增指令

vfwcvt.rtz.xu.f.v vd, vs2, vm	# Convert float to double-width unsigned integer, truncating.
vfwcvt.rtz.x.f.v vd, vs2, vm	# Convert float to double-width signed integer, truncating.

Widening Floating-Point/Integer Type-Convert Instructions是一组转换指令，它们的作用是可以在较窄的整数和浮点数据类型之间转换为两倍宽度的类型。

Narrowing Floating-Point/Integer Type-Convert Instructions

在 riscv-v-spec 0.9 中，Narrowing Floating-Point/Integer Type-Convert 指令新增加了 vfncvt.rtz.xu.f.w 和 vfncvt.rtz.x.f.w 两条指令。

表 17 Narrowing Floating-Point/Integer Type-Convert 新增指令

vfncvt.rtz.xu.f.w vd, vs2, vm	# Convert double-width float to unsigned integer, truncating.
vfncvt.rtz.x.f.w vd, vs2, vm	# Convert double-width float to signed integer, truncating.

Narrowing Floating-Point/Integer Type-Convert Instructions是一组转换指令，它们的作用是可以将较宽的整数和浮点数据类型转换为宽度的一半

Vector Mask-Register Logical Instructions

-vmcpy.m vd, vs => vmand.mm vd, vs, vs # Copy mask register

+vmmv.m vd, vs => vmand.mm vd, vs, vs #Copy mask register

vmmv.m指令以前称为vmcpy.m，但采用新布局时，它更一致地命名为“mv”，因为位被复制而没有解释。可以保留vmcpy.m汇编程序伪指令以实现兼容性。

Vector Slide1up and Slide1down Instruction

+ vfslide1up.vf vd, vs2, rs1, vm # vd[0]=f[rs1], vd[i+1] = vs2[i]

vfslide1up指令的定义类似，但是它的标量参数来自f寄存器。如果SEW <FLEN并且未正确为SEW位设置NaN框，则将SEW位规范的NaN替换。如果FLEN <SEW，则将标量值NaN装箱（扩展）到SEW位。

+ vfslide1down.vf vd, vs2, rs1, vm # vd[i] = vs2[i+1], vd[vl-1]=f[rs1]

vfslide1down指令的定义类似，但是它的标量参数来自f寄存器。如果SEW <FLEN并且未正确为SEW位设置NaN框，则将SEW位规范的NaN替换。如果FLEN <SEW，则将标量值NaN装箱（扩展）到SEW位。

Whole Vector Register Move

- vmv<nf>r.v vd, vs2 # General form
+ vmv<nr>r.v vd, vs2 # General form

如果vd等于vs2，则该指令为NOP。

vmv <nr> r.v指令复制整个向量寄存器（即所有VLEN位），并且可以复制整个向量寄存器组。不管vtype和vl中的当前设置如何。当EEW = 8，EMUL = nr，有效长度evl = VLEN / 8 * EMUL。指令开始执行。

- 4.1 补充

```
Suggested assembler names used for vtypei setting

e8     #    8b elements
e16    #   16b elements
e32    #   32b elements
e64    #   64b elements
e128   #  128b elements

m1    # Vlmul x1, assumed if m setting absent
m2    # Vlmul x2
m4    # Vlmul x4
m8    # Vlmul x8

d1    # EDIV 1, assumed if d setting absent
d2    # EDIV 2
d4    # EDIV 4
d8    # EDIV 8
```

riscv-v-spec 0.8

```
Suggested assembler names used for vsetvli immediate

e8    # SEW=8b
e16   # SEW=16b
e32   # SEW=32b
e64   # SEW=64b
e128  # SEW=128b
e256  # SEW=256b
e512  # SEW=512b
e1024 # SEW=1024b

mf8  # LMUL=1/8
mf4  # LMUL=1/4
mf2  # LMUL=1/2
m1   # LMUL=1, assumed if m setting absent
m2   # LMUL=2
m4   # LMUL=4
m8   # LMUL=8
```

riscv-v-spec 0.9

向量寄存器分组（vlmul [2：0]）

可以将多个向量寄存器组合在一起，以便单个向量指令可以对多个向量寄存器进行操作。向量寄存器组来是一个或多个向量寄存器，这些向量寄存器用于向量指令的单个操作数。向量寄存器组为长向量提供了更高的执行效率。当向量长度乘数LMUL大于1时，表示向量寄存器的默认数目，LMUL可以具有整数值1,2,4,8。（riscv-v-spec 0.8）

vlmul		LMUL	#groups	VLMAX	Grouped registers
0	0	1	32	VLEN/SEW	vn (single register in group)
0	1	2	16	2*VLEN/SEW	vn, vn+1
1	0	4	8	4*VLEN/SEW	vn, ..., vn+3
1	1	8	4	8*VLEN/SEW	vn, ..., vn+7

LMUL可以具有分数值1 / 2、1 / 4、1 / 8。分数LMUL可以操作混合宽度值，较宽的值可以占用单个向量寄存器，较窄的值可以占用向量寄存器的一部分。（riscv-v-spec 0.9新增）

vlmul			LMUL	#groups	VLMAX	Registers grouped with register n
1	0	0	-	-	-	reserved
1	0	1	1/8	32	VLEN/SEW/8	v n (single register in group)
1	1	0	1/4	32	VLEN/SEW/4	v n (single register in group)
1	1	1	1/2	32	VLEN/SEW/2	v n (single register in group)

3.9. Vector Fixed-Point Fields in fcsr

浮点CSR fcsr中的字段可以访问vxrm和vxsat分离的CSR。 fcsr必须将寄存器添加到不带浮点的系统中，以增加矢量扩展名。

Table 6. fcsr layout

Bits	Name	Description
10:9	vxrm	Fixed-point rounding mode
8	vxsat	Fixed-point accrued saturation flag
7:5	frm	Floating-point rounding mode
4:0	fflags	Floating-point accrued exception flags

riscv-v-spec 0.8

3.9. Vector Control and Status Register vcsr

矢量控制和状态CSR vcsr中的字段可以访问vxrm和vxsat分离的CSR。

Table 5. vcsr layout

Bits	Name	Description
2:1	vxrm[1:0]	Fixed-point rounding mode
0	vxsat	Fixed-point accrued saturation flag

riscv-v-spec 0.9

• 05 参考资料

riscv-v-spec 0.9

https://github.com/riscv/riscv-v-spec/blob/master/v-spec.adoc#vector-control-and-status-register-vcsr

riscv-v-spec 0.8

https://github.com/riscv/riscv-v-spec/releases/tag/0.8

The RISC-V Instruction Set Manual

https://content.riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf

谢 谢

欢迎交流合作

2019/02/25