RISC-V 向量指令扩展（V扩展及RVA23额外向量扩展）运算指令编码与功能整理表

作者：陈修壮，李潇宇　山东大学集成电路学院&山东大学智能创新研究院　2025/12/05

extension	class	group	inst	func6	vm	op2_bit	op1_bit	func3_seg	dst_seg	opcode	function	note
V	Vector Integer Arithmetic Instructions	Vector Single-Width Integer Add and Subtract	vadd.vv vd, vs2, vs1, vm	000000	vm	vs2	vs1	000	vd	1010111	vd[i] = vs2[i] + vs1[i]	
V			vadd.vx vd, vs2, rs1, vm	000000	vm	vs2	rs1	100	vd	1010111	vd[i] = vs2[i] + x[rs1]	
V			vadd.vi vd, vs2, imm, vm	000000	vm	vs2	simm5	011	vd	1010111	vd[i] = simm5 + vs2[i]	
V			vsub.vv vd, vs2, vs1, vm	000010	vm	vs2	vs1	000	vd	1010111	vd[i] = vs2[i] - vs1[i]	
V			vsub.vx vd, vs2, rs1, vm	000010	vm	vs2	rs1	100	vd	1010111	vd[i] = vs2[i] - x[rs1]	
V			vrsub.vx vd, vs2, rs1, vm	000011	vm	vs2	rs1	100	vd	1010111	vd[i] = x[rs1] - vs2[i]	
V			vrsub.vi vd, vs2, imm, vm	000011	vm	vs2	simm5	011	vd	1010111	vd[i] = simm5 - vs2[i]	
V		Vector Widening Integer Add/Subtract	vwaddu.vv vd, vs2, vs1, vm	110000	vm	vs2	vs1	010	vd	1010111	SEW位宽的vs2的元素跟SEW位宽的vs1进行无符号加法，2*SEW位宽的结果存入vd	Widening unsigned integer add/subtract, 2*SEW = SEW +/- SEW
V			vwaddu.vx vd, vs2, rs1, vm	110000	vm	vs2	rs1	110	vd	1010111	SEW位宽的vs2的元素跟SEW位宽的x[rs1]进行无符号加法，2*SEW位宽的结果存入vd	
V			vwsubu.vv vd, vs2, vs1, vm	110010	vm	vs2	vs1	010	vd	1010111	SEW位宽的vs2的元素跟SEW位宽的vs1进行无符号减法，2*SEW位宽的结果存入vd	
V			vwsubu.vx vd, vs2, rs1, vm	110010	vm	vs2	rs1	110	vd	1010111	SEW位宽的vs2的元素跟SEW位宽的x[rs1]进行无符号减法，2*SEW位宽的结果存入vd	
V			vwadd.vv vd, vs2, vs1, vm	110001	vm	vs2	vs1	010	vd	1010111	SEW位宽的vs2的元素跟SEW位宽的vs1进行有符号加法，2*SEW位宽的结果存入vd	Widening signed integer add/subtract, 2*SEW = SEW +/- SEW
V			vwadd.vx vd, vs2, rs1, vm	110001	vm	vs2	rs1	110	vd	1010111	SEW位宽的vs2的元素跟SEW位宽的x[rs1]进行有符号加法，2*SEW位宽的结果存入vd	
V			vwsub.vv vd, vs2, vs1, vm	110011	vm	vs2	vs1	010	vd	1010111	SEW位宽的vs2的元素跟SEW位宽的vs1进行有符号减法，2*SEW位宽的结果存入vd	
V			vwsub.vx vd, vs2, rs1, vm	110011	vm	vs2	rs1	110	vd	1010111	SEW位宽的vs2的元素跟SEW位宽的x[rs1]进行有符号减法，2*SEW位宽的结果存入vd	
V			vwaddu.wv vd, vs2, vs1, vm	110100	vm	vs2	vs1	010	vd	1010111	2*SEW位宽的vs2的元素跟SEW位宽的vs1进行无符号加，2*SEW位宽的结果存入vd	Widening unsigned integer add/subtract, 2*SEW = 2*SEW +/- SEW
V			vwaddu.wx vd, vs2, rs1, vm	110100	vm	vs2	rs1	110	vd	1010111	2*SEW位宽的vs2的元素跟SEW位宽的x[rs1]进行无符号加，2*SEW位宽的结果存入vd	
V			vwsubu.wv vd, vs2, vs1, vm	110110	vm	vs2	vs1	010	vd	1010111	2*SEW位宽的vs2的元素跟SEW位宽的vs1进行无符号减法，2*SEW位宽的结果存入vd	
V			vwsubu.wx vd, vs2, rs1, vm	110110	vm	vs2	rs1	110	vd	1010111	2*SEW位宽的vs2的元素跟SEW位宽的x[rs1]进行无符号减法，2*SEW位宽的结果存入	
V			vwadd.wv vd, vs2, vs1, vm	110101	vm	vs2	vs1	010	vd	1010111	2*SEW位宽的vs2的元素跟SEW位宽的vs1进行有符号加，2*SEW位宽的结果存入vd	Widening signed integer add/subtract, 2*SEW = 2*SEW +/- SEW
V			vwadd.wx vd, vs2, rs1, vm	110101	vm	vs2	rs1	110	vd	1010111	2*SEW位宽的vs2的元素跟SEW位宽的x[rs1]进行由符号加，2*SEW位宽的结果存入vd	
V			vwsub.wv vd, vs2, vs1, vm	110111	vm	vs2	vs1	010	vd	1010111	2*SEW位宽的vs2的元素跟SEW位宽的vs1进有符号减法，2*SEW位宽的结果存入vd	
V			vwsub.wx vd, vs2, rs1, vm	110111	vm	vs2	rs1	110	vd	1010111	2*SEW位宽的vs2的元素跟SEW位宽的x[rs1]进行有符号减法，2*SEW位宽的结果存入	
V		Vector Integer Extension	vzext.vf2 vd, vs2, vm	010010	vm	vs2	00110	010	vd	1010111	Zero-extend SEW/2 source to SEW destination	
V			vsext.vf2 vd, vs2, vm	010010	vm	vs2	00111	010	vd	1010111	Sign-extend SEW/2 source to SEW destination	
V			vzext.vf4 vd, vs2, vm	010010	vm	vs2	00100	010	vd	1010111	Zero-extend SEW/4 source to SEW destination	
V			vsext.vf4 vd, vs2, vm	010010	vm	vs2	00101	010	vd	1010111	Sign-extend SEW/4 source to SEW destination	
V			vzext.vf8 vd, vs2, vm	010010	vm	vs2	00010	010	vd	1010111	Zero-extend SEW/8 source to SEW destination	
V			vsext.vf8 vd, vs2, vm	010010	vm	vs2	00011	010	vd	1010111	Sign-extend SEW/8 source to SEW destination	
V		Vector Integer Add-with-Carry / Subtract-with-Borrow Instructions	vadc.vvm vd, vs2, vs1, v0	010000	0	vs2	vs1	000	vd	1010111	vd[i] = vs2[i] + vs1[i] + v0.mask[i]	Produce sum with carry.
V			vadc.vxm vd, vs2, rs1, v0	010000	0	vs2	rs1	100	vd	1010111	vd[i] = vs2[i] + x[rs1] + v0.mask[i]	Produce sum with carry.
V			vadc.vim vd, vs2, imm, v0	010000	0	vs2	simm5	011	vd	1010111	vd[i] = vs2[i] + imm + v0.mask[i]	Produce sum with carry.
V			vmadc.vvm vd, vs2, vs1, v0	010001	0	vs2	vs1	000	vd	1010111	vd.mask[i] = carry_out(vs2[i] + vs1[i] + v0.mask[i])	Produce carry out in mask register format
V			vmadc.vxm vd, vs2, rs1, v0	010001	0	vs2	rs1	100	vd	1010111	vd.mask[i] = carry_out(vs2[i] + x[rs1] + v0.mask[i])	Produce carry out in mask register format
V			vmadc.vim vd, vs2, imm, v0	010001	0	vs2	simm5	011	vd	1010111	vd.mask[i] = carry_out(vs2[i] + imm + v0.mask[i])	Produce carry out in mask register format
V			vmadc.vv vd, vs2, vs1	010001	1	vs2	vs1	000	vd	1010111	vd.mask[i] = carry_out(vs2[i] + vs1[i])，no carry-in	Produce carry out in mask register format
V			vmadc.vx vd, vs2, rs1	010001	1	vs2	rs1	100	vd	1010111	vd.mask[i] = carry_out(vs2[i] + x[rs1])，no carry-in	Produce carry out in mask register format
V			vmadc.vi vd, vs2, imm	010001	1	vs2	simm5	011	vd	1010111	vd.mask[i] = carry_out(vs2[i] + imm)，no carry-in	Produce carry out in mask register format
V			vsbc.vvm vd, vs2, vs1, v0	010010	0	vs2	vs1	000	vd	1010111	vd[i] = vs2[i] - vs1[i] - v0.mask[i]	Produce difference with borrow.
V			vsbc.vxm vd, vs2, rs1, v0	010010	0	vs2	rs1	100	vd	1010111	vd[i] = vs2[i] - x[rs1] - v0.mask[i]	Produce difference with borrow.
V			vmsbc.vvm vd, vs2, vs1, v0	010011	0	vs2	vs1	000	vd	1010111	vd.mask[i] = borrow_out(vs2[i] - vs1[i] - v0.mask[i])	Produce borrow out in mask register format
V			vmsbc.vxm vd, vs2, rs1, v0	010011	0	vs2	rs1	100	vd	1010111	vd.mask[i] = borrow_out(vs2[i] - x[rs1] - v0.mask[i])	Produce borrow out in mask register format
V			vmsbc.vv vd, vs2, vs1	010011	1	vs2	vs1	000	vd	1010111	vd.mask[i] = borrow_out(vs2[i] - vs1[i]),no borrow-in	Produce borrow out in mask register format
V			vmsbc.vx vd, vs2, rs1	010011	1	vs2	rs1	100	vd	1010111	vd.mask[i] = borrow_out(vs2[i] - x[rs1]),no borrow-in	Produce borrow out in mask register format
V												
V		Vector Bitwise Logical Instructions	vand.vv vd, vs2, vs1, vm	001001	vm	vs2	vs1	000	vd	1010111	vd[i] = vs2[i] and vs1[i]	
V			vand.vx vd, vs2, rs1, vm	001001	vm	vs2	rs1	100	vd	1010111	vd[i] = vs2[i] and x[rs1]	
V			vand.vi vd, vs2, imm, vm	001001	vm	vs2	imm[4:0]	011	vd	1010111	vd[i] = vs2[i] and imm	
V			vor.vv vd, vs2, vs1, vm	001010	vm	vs2	vs1	000	vd	1010111	vd[i] = vs2[i] or vs1[i]	
V			vor.vx vd, vs2, rs1, vm	001010	vm	vs2	rs1	100	vd	1010111	vd[i] = vs2[i] or x[rs1]	
V			vor.vi vd, vs2, imm, vm	001010	vm	vs2	imm[4:0]	011	vd	1010111	vd[i] = vs2[i] or imm	
V			vxor.vv vd, vs2, vs1, vm	001011	vm	vs2	vs1	000	vd	1010111	vd[i] = vs2[i] xor vs1[i]	
V			vxor.vx vd, vs2, rs1, vm	001011	vm	vs2	rs1	100	vd	1010111	vd[i] = vs2[i] xor x[rs1]	
V			vxor.vi vd, vs2, imm, vm	001011	vm	vs2	imm[4:0]	011	vd	1010111	vd[i] = vs2[i] xor imm	
V												
V		Vector Single-Width Shift Instructions	vsll.vv vd, vs2, vs1, vm	100101	vm	vs2	vs1	000	vd	1010111	Vector-vector，logical shift left	
V			vsll.vx vd, vs2, rs1, vm	100101	vm	vs2	rs1	100	vd	1010111	vector-scalar,logical shift left	
V			vsll.vi vd, vs2, uimm, vm	100101	vm	vs2	zimm5	011	vd	1010111	vector-immediate,logical shift left	
V			vsrl.vv vd, vs2, vs1, vm	101000	vm	vs2	vs1	000	vd	1010111	Vector-vector, logical shift right,zero-extending	
V			vsrl.vx vd, vs2, rs1, vm	101000	vm	vs2	rs1	100	vd	1010111	vector-scalar, logical shift right,zero-extending	
V			vsrl.vi vd, vs2, uimm, vm	101000	vm	vs2	zimm5	011	vd	1010111	vector-immediate, logical shift right,zero-extending	
V			vsra.vv vd, vs2, vs1, vm	101001	vm	vs2	vs1	000	vd	1010111	Vector-vector,arithmetic shift right,sign-extending	
V			vsra.vx vd, vs2, rs1, vm	101001	vm	vs2	rs1	100	vd	1010111	vector-scalar,arithmetic shift right,sign-extending	
V			vsra.vi vd, vs2, uimm, vm	101001	vm	vs2	zimm5	011	vd	1010111	vector-immediate,arithmetic shift right,sign-extending	
V		Vector Narrowing Integer	vnsrl.wv vd, vs2, vs1, vm	101100	vm	vs2	vs1	000	vd	1010111	将2*SEW位宽的vs2中的数据，向右移vs1中对应元素的量(只取低log2(2*SEW)位)，高位使用0填充，得到的SEW位宽的结果存入到vd	
V			vnsrl.wx vd, vs2, rs1, vm	101100	vm	vs2	rs1	100	vd	1010111	将2*SEW位宽的vs2中的数据，向右移x[rs1]中对应元素的量(只取低log2(2*SEW)位)，高位使用0填充，得到的SEW位宽的结果存入到vd	
V			vnsrl.wi vd, vs2, uimm, vm	101100	vm	vs2	zimm5	011	vd	1010111	将2*SEW位宽的vs2中的数据，向右移立即数中对应元素的量(只取低log2(2*SEW)位)，高位使用0填充，得到的SEW位宽的结果存入到vd	

V		Right Shift Instructions	vnsra.wv vd, vs2, vs1, vm	101101	vm	vs2	vs1	000	vd	1010111	将2*SEW位宽的vs2中的数据，向右移vs1中对应元素的量(只取低log2(2*SEW)位)，高位使用符号位填充，得到的SEW位宽的结果存入到vd	
V			vnsra.wx vd, vs2, rs1, vm	101101	vm	vs2	rs1	100	vd	1010111	将2*SEW位宽的vs2中的数据，向右移x[rs1]中对应元素的量(只取低log2(2*SEW)位)，高位使用符号位填充，得到的SEW位宽的结果存入到vd	
V			vnsra.wi vd, vs2, uimm, vm	101101	vm	vs2	zimm5	011	vd	1010111	将2*SEW位宽的vs2中的数据，向右移立即数中对应元素的量(只取低log2(2*SEW)位)，高位使用符号位填充，得到的SEW位宽的结果存入到vd	
V												
V												
V												
V												
V		Vector Integer Compare Instructions	vmseq.vv vd, vs2, vs1, vm	011000	vm	vs2	vs1	000	vd	1010111	res = vs2 == vs1	res是vd中相应的位vd[i]，运算取vs2[i]跟vs1[i]或者x[rs1]
V			vmseq.vx vd, vs2, rs1, vm	011000	vm	vs2	rs1	100	vd	1010111	res = rs1 == vs2	
V			vmseq.vi vd, vs2, imm, vm	011000	vm	vs2	simm	011	vd	1010111	res = imm == vs2	
V			vmsne.vv vd, vs2, vs1, vm	011001	vm	vs2	vs1	000	vd	1010111	res = vs2 != vs1	
V			vmsne.vx vd, vs2, rs1, vm	011001	vm	vs2	rs1	100	vd	1010111	res = vs2 != rs1	
V			vmsne.vi vd, vs2, imm, vm	011001	vm	vs2	simm	011	vd	1010111	res = vs2 != imm	
V			vmsltu.vv vd, vs2, vs1, vm	011010	vm	vs2	vs1	000	vd	1010111	res = vs2 < vs1（无符号）	
V			vmsltu.vx vd, vs2, rs1, vm	011010	vm	vs2	rs1	100	vd	1010111	res = vs2 < rs1（无符号）	
V			vmslt.vv vd, vs2, vs1, vm	011011	vm	vs2	vs1	000	vd	1010111	res = vs2 < vs1	
V			vmslt.vx vd, vs2, rs1, vm	011011	vm	vs2	rs1	100	vd	1010111	res = vs2 < rs1	
V			vmsleu.vv vd, vs2, vs1, vm	011100	vm	vs2	vs1	000	vd	1010111	res = vs2 <= vs1（无符号）	
V			vmsleu.vx vd, vs2, rs1, vm	011100	vm	vs2	rs1	100	vd	1010111	res = vs2 <= rs1（无符号）	
V			vmsleu.vi vd, vs2, imm, vm	011100	vm	vs2	simm	011	vd	1010111	res = vs2 <= 符号扩展的立即数	
V			vmsle.vv vd, vs2, vs1, vm	011101	vm	vs2	vs1	000	vd	1010111	res = vs2 <= vs1	
V			vmsle.vx vd, vs2, rs1, vm	011101	vm	vs2	rs1	100	vd	1010111	res = vs2 <= rs1	
V			vmsle.vi vd, vs2, imm, vm	011101	vm	vs2	simm	011	vd	1010111	res = vs2 <= imm	
V			vmsgtu.vx vd, vs2, rs1, vm	011110	vm	vs2	rs1	100	vd	1010111	res = vs2 > rs1（无符号）	
V			vmsgtu.vi vd, vs2, imm, vm	011110	vm	vs2	simm	011	vd	1010111	res = vs2 > simm	
V			vmsgt.vx vd, vs2, rs1, vm	011111	vm	vs2	rs1	100	vd	1010111	res = vs2 > rs1	
V			vmsgt.vi vd, vs2, imm, vm	011111	vm	vs2	simm	011	vd	1010111	res = vs2 > imm	
V												
V		Vector Integer Min/Max Instructions	vminu.vv vd, vs2, vs1, vm	000100	vm	vs2	vs1	000	vd	1010111	将vs1、vs2中最小的量写入vd（无符号）	
V			vminu.vx vd, vs2, rs1, vm	000100	vm	vs2	rs1	100	vd	1010111	将rs1、vs2中最小的量写入vd（无符号）	
V			vmin.vv vd, vs2, vs1, vm	000101	vm	vs2	vs1	000	vd	1010111	将vs1、vs2中最小的量写入vd（有符号）	
V			vmin.vx vd, vs2, rs1, vm	000101	vm	vs2	rs1	100	vd	1010111	将rs1、vs2中最小的量写入vd（有符号）	
V			vmaxu.vv vd, vs2, vs1, vm	000110	vm	vs2	vs1	000	vd	1010111	将vs1、vs2中最大的量写入vd（无符号）	
V			vmaxu.vx vd, vs2, rs1, vm	000110	vm	vs2	rs1	100	vd	1010111	将rs1、vs2中最大的量写入vd（无符号）	
V			vmax.vv vd, vs2, vs1, vm	000111	vm	vs2	vs1	000	vd	1010111	将vs1、vs2中最大的量写入vd（有符号）	
V			vmax.vx vd, vs2, rs1, vm	000111	vm	vs2	rs1	100	vd	1010111	将rs1、vs2中最大的量写入vd（有符号）	
V												
V												
V		Vector Single-Width Integer Multiply Instructions	vmul.vv vd, vs2, vs1, vm	100101	vm	vs2	vs1	010	vd	1010111	Signed multiply, returning low bits of product. vs2与vs1	
V			vmul.vx vd, vs2, rs1, vm	100101	vm	vs2	rs1	110	vd	1010111	Signed multiply, returning low bits of product vs2与x[rs1]	
V			vmulh.vv vd, vs2, vs1, vm	100111	vm	vs2	vs1	010	vd	1010111	Signed multiply, returning high bits of product	
V			vmulh.vx vd, vs2, rs1, vm	100111	vm	vs2	rs1	110	vd	1010111	Signed multiply, returning high bits of product	
V			vmulhu.vv vd, vs2, vs1, vm	100100	vm	vs2	vs1	010	vd	1010111	Unsigned multiply, returning high bits of product	
V			vmulhu.vx vd, vs2, rs1, vm	100100	vm	vs2	rs1	110	vd	1010111	Unsigned multiply, returning high bits of product	
V			vmulhsu.vv vd, vs2, vs1, vm	100110	vm	vs2	vs1	010	vd	1010111	Signed(vs2)-Unsigned multiply, returning high bits of product	
V			vmulhsu.vx vd, vs2, rs1, vm	100110	vm	vs2	rs1	110	vd	1010111	Signed(vs2)-Unsigned multiply, returning high bits of product	
V												
V		Vector Integer Divide Instructions	vdivu.vv vd, vs2, vs1, vm	100000	vm	vs2	vs1	010	vd	1010111	vd = vs2 / vs1；分母为0时vd=-1	
V			vdivu.vx vd, vs2, rs1, vm	100000	vm	vs2	rs1	110	vd	1010111	vd = vs2 / rs1；分母为0时vd=-1	
V			vdiv.vv vd, vs2, vs1, vm	100001	vm	vs2	vs1	010	vd	1010111	vd = vs2 / vs1；分母为0时vd=-1；最小负数除以-1时结果等于vs2	
V			vdiv.vx vd, vs2, rs1, vm	100001	vm	vs2	rs1	110	vd	1010111	vd = vs2 / rs1；分母为0时vd=-1；最小负数除以-1时结果等于vs2	
V			vremu.vv vd, vs2, vs1, vm	100010	vm	vs2	vs1	010	vd	1010111	vd = vs2 % vs1；vs1为0时vd=vs2	
V			vremu.vx vd, vs2, rs1, vm	100010	vm	vs2	rs1	110	vd	1010111	vd = vs2 % rs1；rs1=0时vd=vs2	
V			vrem.vv vd, vs2, vs1, vm	100011	vm	vs2	vs1	010	vd	1010111	vd = vs2 % vs1；vs1为0时vd=vs2；最小负数对-1取余时结果等于0	
V			vrem.vx vd, vs2, rs1, vm	100011	vm	vs2	rs1	110	vd	1010111	vd = vs2 % rs1；rs1=0时vd=vs2；最小负数对-1取余时结果等于0	
V												
V		Vector Widening Integer Multiply Instructions	vwmul.vv vd, vs2, vs1, vm	111011	vm	vs2	vs1	010	vd	1010111	vs1与vs2有符号乘，将结果完整的2*SEW位存入vd	
V			vwmul.vx vd, vs2, rs1, vm	111011	vm	vs2	rs1	110	vd	1010111	vs2与x[rs1]有符号乘，将结果完整的2*SEW位存入vd	
V			vwmulu.vv vd, vs2, vs1, vm	111000	vm	vs2	vs1	010	vd	1010111	vs1与vs2无符号乘，将结果完整的2*SEW位存入vd	
V			vwmulu.vx vd, vs2, rs1, vm	111000	vm	vs2	rs1	110	vd	1010111	vs2与x[rs1]无符号乘，将结果完整的2*SEW位存入vd	
V			vwmulsu.vv vd, vs2, vs1, vm	111010	vm	vs2	vs1	010	vd	1010111	vs2(有符号)和vs1（无符号)乘，将结果完整的2*SEW位存入vd	
V			vwmulsu.vx vd, vs2, rs1, vm	111010	vm	vs2	rs1	110	vd	1010111	vs2(有符号)和x[rs1]（无符号)乘，将结果完整的2*SEW位存入vd	
V												
V		Vector Single-Width Integer Multiply-Add Instructions	vmacc.vv vd, vs1, vs2, vm	101101	vm	vs2	vs1	010	vd	1010111	vd[i] = +(vs1[i] * vs2[i]) + vd[i]	Integer multiply-add, overwrite addend
V			vmacc.vx vd, rs1, vs2, vm	101101	vm	vs2	rs1	110	vd	1010111	vd[i] = +(x[rs1] * vs2[i]) + vd[i]	
V			vnmsac.vv vd, vs1, vs2, vm	101111	vm	vs2	vs1	010	vd	1010111	vd[i] = -(vs1[i] * vs2[i]) + vd[i]	Integer multiply-sub, overwrite minuend
V			vnmsac.vx vd, rs1, vs2, vm	101111	vm	vs2	rs1	110	vd	1010111	vd[i] = -(x[rs1] * vs2[i]) + vd[i]	
V			vmadd.vv vd, vs1, vs2, vm	101001	vm	vs2	vs1	010	vd	1010111	vd[i] = (vs1[i] * vd[i]) + vs2[i]	Integer multiply-add, overwrite multiplicand
V			vmadd.vx vd, rs1, vs2, vm	101001	vm	vs2	rs1	110	vd	1010111	vd[i] = (x[rs1] * vd[i]) + vs2[i]	
V			vnmsub.vv vd, vs1, vs2, vm	101011	vm	vs2	vs1	010	vd	1010111	vd[i] = -(vs1[i] * vd[i]) + vs2[i]	Integer multiply-sub, overwrite multiplicand
V			vnmsub.vx vd, rs1, vs2, vm	101011	vm	vs2	rs1	110	vd	1010111	vd[i] = -(x[rs1] * vd[i]) + vs2[i]	

V		Vector Widening Integer Multiply-Add Instructions	vwmaccu.vv vd, vs1, vs2, vm	111100	vm	vs2	vs1	010	vd	1010111	vd[i] = +(vs1[i] * vs2[i]) + vd[i],Widening unsigned-integer multiply-add, overwrite addend	vs2,vs1,x[rs1]都是SEW，乘积是2*SEW 最后将结果(2*SEW)存入vd
V			vwmaccu.vx vd, rs1, vs2, vm	111100	vm	vs2	rs1	110	vd	1010111	vd[i] = +(x[rs1] * vs2[i]) + vd[i],Widening unsigned-integer multiply-add, overwrite addend	
V			vwmacc.vv vd, vs1, vs2, vm	111101	vm	vs2	vs1	010	vd	1010111	vd[i] = +(vs1[i] * vs2[i]) + vd[i],Widening signed-integer multiply-add, overwrite addend	
V			vwmacc.vx vd, rs1, vs2, vm	111101	vm	vs2	rs1	110	vd	1010111	vd[i] = +(x[rs1] * vs2[i]) + vd[i],Widening signed-integer multiply-add, overwrite addend	
V			vwmaccsu.vv vd, vs1, vs2, vm	111111	vm	vs2	vs1	010	vd	1010111	vd[i] = +(signed(vs1[i]) * unsigned(vs2[i])) + vd[i],Widening signed-unsigned-integer multiply-add, overwrite addend	
V			vwmaccsu.vx vd, rs1, vs2, vm	111111	vm	vs2	rs1	110	vd	1010111	vd[i] = +(signed(x[rs1]) * unsigned(vs2[i])) + vd[i],Widening signed-unsigned-integer multiply-add, overwrite addend	
V			vwmaccus.vx vd, rs1, vs2, vm	111110	vm	vs2	rs1	110	vd	1010111	vd[i] = +(unsigned(x[rs1]) * signed(vs2[i])) + vd[i],# Widening unsigned-signed-integer multiply-add, overwrite addend	
V		Vector Integer Merge Instructions	vmerge.vvm vd, vs2, vs1, v0	010111	0	vs2	vs1	000	vd	1010111	vd[i] = v0.mask[i] ? vs1[i] : vs2[i]	
V			vmerge.vxm vd, vs2, rs1, v0	010111	0	vs2	rs1	100	vd	1010111	vd[i] = v0.mask[i] ? x[rs1] : vs2[i]	
V			vmerge.vim vd, vs2, imm, v0	010111	0	vs2	simm5	011	vd	1010111	vd[i] = v0.mask[i] ? imm : vs2[i]	
V		Vector Integer Move Instructions	vmv.v.v vd, vs1	010111	1	00000	vs1	000	vd	1010111	vd[i] = vs1[i]	
V			vmv.v.x vd, rs1	010111	1	00000	rs1	100	vd	1010111	vd[i] = x[rs1]	
V	Fixed-Point Arithmetic Instructions	Vector Single-Width Saturating Add and Subtract	vsaddu.vv vd, vs2, vs1, vm	100000	vm	vs2	vs1	000	vd	1010111	向量-向量无符号饱和加法	Saturating adds of unsigned integers
V			vsaddu.vx vd, vs2, rs1, vm	100000	vm	vs2	rs1	100	vd	1010111	向量-标量无符号饱和加法	
V			vsaddu.vi vd, vs2, imm, vm	100000	vm	vs2	simm5	011	vd	1010111	向量-立即数无符号饱和加法	
V			vsadd.vv vd, vs2, vs1, vm	100001	vm	vs2	vs1	000	vd	1010111	Vector-vector，有符号饱和加法	Saturating adds of signed integers.
V			vsadd.vx vd, vs2, rs1, vm	100001	vm	vs2	rs1	100	vd	1010111	vector-scalar，有符号饱和加法	
V			vsadd.vi vd, vs2, imm, vm	100001	vm	vs2	simm5	011	vd	1010111	vector-immediate，有符号饱和加法	
V			vssubu.vv vd, vs2, vs1, vm	100010	vm	vs2	vs1	000	vd	1010111	对向量元素与向量执行逐元素无符号饱和减法	Saturating subtract of unsigned integers
V			vssubu.vx vd, vs2, rs1, vm	100010	vm	vs2	rs1	100	vd	1010111	对向量元素与标量执行逐元素无符号饱和减法	
V			vssub.vv vd, vs2, vs1, vm	100011	vm	vs2	vs1	000	vd	1010111	对向量元素与向量执行逐元素有符号饱和减法，结果写入目的向量	Saturating subtract of signed integers
V			vssub.vx vd, vs2, rs1, vm	100011	vm	vs2	rs1	100	vd	1010111	对向量元素与标量执行逐元素有符号饱和减法，结果写入目的向量	
V		Vector Single-Width Averaging Add and Subtract	vaaddu.vv vd, vs2, vs1, vm	001000	vm	vs2	vs1	010	vd	1010111	roundoff_unsigned(vs2[i] + vs1[i], 1),Averaging adds of unsigned integers.	roundoff_(un)signed，将结果根据第二个参数右移，并根据vxrm决定舍入
V			vaaddu.vx vd, vs2, rs1, vm	001000	vm	vs2	rs1	110	vd	1010111	roundoff_unsigned(vs2[i] + x[rs1], 1),roundoff_unsigned(vs2[i] + x[rs1], 1)	
V			vaadd.vv vd, vs2, vs1, vm	001001	vm	vs2	vs1	010	vd	1010111	roundoff_signed(vs2[i] + vs1[i], 1),Averaging adds of signed integers	
V			vaadd.vx vd, vs2, rs1, vm	001001	vm	vs2	rs1	110	vd	1010111	roundoff_signed(vs2[i] + x[rs1], 1),Averaging adds of signed integers	
V			vasubu.vv vd, vs2, vs1, vm	001010	vm	vs2	vs1	010	vd	1010111	roundoff_unsigned(vs2[i] - vs1[i], 1),Averaging subtract of unsigned integers	
V			vasubu.vx vd, vs2, rs1, vm	001010	vm	vs2	rs1	110	vd	1010111	roundoff_unsigned(vs2[i] - x[rs1], 1),Averaging subtract of unsigned integers.	
V			vasub.vv vd, vs2, vs1, vm	001011	vm	vs2	vs1	010	vd	1010111	roundoff_signed(vs2[i] - vs1[i], 1),Averaging subtract of signed integers	
V			vasub.vx vd, vs2, rs1, vm	001011	vm	vs2	rs1	110	vd	1010111	roundoff_signed(vs2[i] - x[rs1], 1),Averaging subtract of signed integers	
V		Vector Single-Width Fractional Multiply with Rounding and Saturation	vsmul.vv vd, vs2, vs1, vm	100111	vm	vs2	vs1	000	vd	1010111	vd[i] = clip(roundoff_signed(vs2[i]*vs1[i], SEW-1))	结果右移以后，saturates the result to fit into SEW bits
V			vsmul.vx vd, vs2, rs1, vm	100111	vm	vs2	rs1	100	vd	1010111	vd[i] = clip(roundoff_signed(vs2[i]*x[rs1], SEW-1))	
V		Vector Single-Width Scaling Shift Instructions	vssrl.vv vd, vs2, vs1, vm	101010	vm	vs2	vs1	000	vd	1010111	vd[i] = roundoff_unsigned(vs2[i], vs1[i])	Scaling shift right logical
V			vssrl.vx vd, vs2, rs1, vm	101010	vm	vs2	rs1	100	vd	1010111	vd[i] = roundoff_unsigned(vs2[i], x[rs1])	
V			vssrl.vi vd, vs2, uimm, vm	101010	vm	vs2	zimm5	011	vd	1010111	vd[i] = roundoff_unsigned(vs2[i], uimm)	
V			vssra.vv vd, vs2, vs1, vm	101011	vm	vs2	vs1	000	vd	1010111	vd[i] = roundoff_signed(vs2[i],vs1[i])	Scaling shift right arithmetic
V			vssra.vx vd, vs2, rs1, vm	101011	vm	vs2	rs1	100	vd	1010111	vd[i] = roundoff_signed(vs2[i], x[rs1])	
V			vssra.vi vd, vs2, uimm, vm	101011	vm	vs2	zimm5	011	vd	1010111	vd[i] = roundoff_signed(vs2[i], uimm)	
V		Vector Narrowing Fixed-Point Clip Instructions	vnclipu.wv vd, vs2, vs1, vm	101110	vm	vs2	vs1	000	vd	1010111	vd[i] = clip(roundoff_unsigned(vs2[i], vs1[i]))	vs2: 2*SEW vd: SEW 另一个源也是SEW
V			vnclipu.wx vd, vs2, rs1, vm	101110	vm	vs2	rs1	100	vd	1010111	vd[i] = clip(roundoff_unsigned(vs2[i], x[rs1]))	
V			vnclipu.wi vd, vs2, uimm, vm	101110	vm	vs2	zimm5	011	vd	1010111	vd[i] = clip(roundoff_unsigned(vs2[i], uimm))	
V			vnclip.wv vd, vs2, vs1, vm	101111	vm	vs2	vs1	000	vd	1010111	vd[i] = clip(roundoff_signed(vs2[i], vs1[i]))	
V			vnclip.wx vd, vs2, rs1, vm	101111	vm	vs2	rs1	100	vd	1010111	vd[i] = clip(roundoff_signed(vs2[i], x[rs1]))	
V			vnclip.wi vd, vs2, uimm, vm	101111	vm	vs2	zimm5	011	vd	1010111	vd[i] = clip(roundoff_signed(vs2[i], uimm))	
V	Vector Floating-Point Instructions	Vector Single-Width Floating-Point Add/Subtract Instructions	vfadd.vv vd, vs2, vs1, vm	000000	vm	vs2	vs1	001	vd	1010111	vd[i] = vs2[i] + vs1[i]	浮点加
V			vfadd.vf vd, vs2, rs1, vm	000000	vm	vs2	rs1	101	vd	1010111	vd[i] = vs2[i] + f[rs1]	
V			vfsub.vv vd, vs2, vs1, vm	000010	vm	vs2	vs1	001	vd	1010111	vd[i] = vs2[i] - vs1[i]	浮点减
V			vfsub.vf vd, vs2, rs1, vm	000010	vm	vs2	rs1	101	vd	1010111	Vector-scalar vd[i] = vs2[i] - f[rs1]	
V			vfrsub.vf vd, vs2, rs1, vm	100111	vm	vs2	rs1	101	vd	1010111	Scalar-vector vd[i] = f[rs1] - vs2[i]	
V		Vector Widening Floating-Point Add/Subtract Instructions	vfwadd.vv vd, vs2, vs1, vm	110000	vm	vs2	vs1	001	vd	1010111	vd[i] = vs2[i] + vs1[i],2*SEW = SEW +/- SEW	Widening FP add/subtract, 2*SEW = SEW +/- SEW
V			vfwadd.vf vd, vs2, rs1, vm	110000	vm	vs2	rs1	101	vd	1010111	vd[i] = vs2[i] + f[rs1],2*SEW = SEW +/- SEW	
V			vfwsub.vv vd, vs2, vs1, vm	110010	vm	vs2	vs1	001	vd	1010111	vd[i] = vs2[i] - vs1[i],2*SEW = SEW +/- SEW	
V			vfwsub.vf vd, vs2, rs1, vm	110010	vm	vs2	rs1	101	vd	1010111	vd[i] = vs2[i] - f[rs1],2*SEW = SEW +/- SEW	
V			vfwadd.wv vd, vs2, vs1, vm	110100	vm	vs2	vs1	001	vd	1010111	vd[i] = vs2[i] + vs1[i],2*SEW = 2*SEW +/- SEW	Widening FP add/subtract, 2*SEW = 2*SEW +/- SEW
V			vfwadd.wf vd, vs2, rs1, vm	110100	vm	vs2	rs1	101	vd	1010111	vd[i] = vs2[i] + f[rs1],2*SEW = 2*SEW +/- SEW	
V			vfwsub.wv vd, vs2, vs1, vm	110110	vm	vs2	vs1	001	vd	1010111	vd[i] = vs2[i] - vs1[i],2*SEW = 2*SEW +/- SEW	
V			vfwsub.wf vd, vs2, rs1, vm	110110	vm	vs2	rs1	101	vd	1010111	vd[i] = vs2[i] - f[rs1],2*SEW = 2*SEW +/- SEW	
V		Vector Single-Width Floating-Point Multiply/Divide Instructions	vfmul.vv vd, vs2, vs1, vm	100100	vm	vs2	vs1	001	vd	1010111	vd[i] = vs2[i] * vs1[i]	Floating-point multiply
V			vfmul.vf vd, vs2, rs1, vm	100100	vm	vs2	rs1	101	vd	1010111	vd[i] = vs2[i] * f[rs1]	
V			vfdiv.vv vd, vs2, vs1, vm	100000	vm	vs2	vs1	001	vd	1010111	vd[i] = vs2[i] / vs1[i]	Floating-point divide
V			vfdiv.vf vd, vs2, rs1, vm	100000	vm	vs2	rs1	101	vd	1010111	vd[i] = vs2[i] / f[rs1]	
V			vfrdiv.vf vd, vs2, rs1, vm	100001	vm	vs2	rs1	101	vd	1010111	scalar-vector, vd[i] = f[rs1]/vs2[i]	

V		Vector Widening Floating-Point Multiply	vfwmul.vv vd, vs2, vs1, vm	111000	vm	vs2	vs1	001	vd	1010111	vd[i] = vs2[i] * vs[1],2*SEW= SEW*SEW	
V			vfwmul.vf vd, vs2, rs1, vm	111000	vm	vs2	rs1	101	vd	1010111	vd[i] = vs2[i] * f[rs1],2*SEW= SEW*SEW	
V		Vector Single-Width Floating-Point Fused Multiply-Add Instructions	vfmacc.vv vd, vs1, vs2, vm	101100	vm	vs2	vs1	001	vd	1010111	vd[i] = +(vs1[i] * vs2[i]) + vd[i]	FP multiply-accumulate, overwrites addend
V			vfmacc.vf vd, rs1, vs2, vm	101100	vm	vs2	rs1	101	vd	1010111	vd[i] = +(f[rs1] * vs2[i]) + vd[i]	
V			vfnmacc.vv vd, vs1, vs2, vm	101101	vm	vs2	vs1	001	vd	1010111	vd[i] = -(vs1[i] * vs2[i]) - vd[i]	FP negate-(multiply-accumulate), overwrites subtrahend
V			vfnmacc.vf vd, rs1, vs2, vm	101101	vm	vs2	rs1	101	vd	1010111	vd[i] = -(f[rs1] * vs2[i]) - vd[i]	
V			vfmsac.vv vd, vs1, vs2, vm	101110	vm	vs2	vs1	001	vd	1010111	vd[i] = +(vs1[i] * vs2[i]) - vd[i]	FP multiply-subtract-accumulator, overwrites subtrahend
V			vfmsac.vf vd, rs1, vs2, vm	101110	vm	vs2	rs1	101	vd	1010111	vd[i] = +(f[rs1] * vs2[i]) - vd[i]	
V			vfnmsac.vv vd, vs1, vs2, vm	101111	vm	vs2	vs1	001	vd	1010111	vd[i] = -(vs1[i] * vs2[i]) + vd[i]	FP negate-(multiply-subtract-accumulator), overwrites minuend
V			vfnmsac.vf vd, rs1, vs2, vm	101111	vm	vs2	rs1	101	vd	1010111	vd[i] = -(f[rs1] * vs2[i]) + vd[i]	
V			vfmadd.vv vd, vs1, vs2, vm	101000	vm	vs2	vs1	001	vd	1010111	vd[i] = +(vs1[i] * vd[i]) + vs2[i]	FP multiply-add, overwrites multiplicand
V			vfmadd.vf vd, rs1, vs2, vm	101000	vm	vs2	rs1	101	vd	1010111	vd[i] = +(f[rs1] * vd[i]) + vs2[i]	
V			vfnmadd.vv vd, vs1, vs2, vm	101001	vm	vs2	vs1	001	vd	1010111	vd[i] = -(vs1[i] * vd[i]) - vs2[i]	FP negate-(multiply-add), overwrites multiplicand
V			vfnmadd.vf vd, rs1, vs2, vm	101001	vm	vs2	rs1	101	vd	1010111	vd[i] = -(f[rs1] * vd[i]) - vs2[i]	
V			vfmsub.vv vd, vs1, vs2, vm	101010	vm	vs2	vs1	001	vd	1010111	vd[i] = +(vs1[i] * vd[i]) - vs2[i]	FP multiply-sub, overwrites multiplicand
V			vfmsub.vf vd, rs1, vs2, vm	101010	vm	vs2	rs1	101	vd	1010111	vd[i] = +(f[rs1] * vd[i]) - vs2[i]	
V			vfnmsub.vv vd, vs1, vs2, vm	101011	vm	vs2	vs1	001	vd	1010111	vd[i] = -(vs1[i] * vd[i]) + vs2[i]	FP negate-(multiply-sub), overwrites multiplicand
V			vfnmsub.vf vd, rs1, vs2, vm	101011	vm	vs2	rs1	101	vd	1010111	vd[i] = -(f[rs1] * vd[i]) + vs2[i]	
V		Vector Widening Floating-Point Fused Multiply-Add Instructions	vfwmacc.vv vd, vs1, vs2, vm	111100	vm	vs2	vs1	001	vd	1010111	vd[i] = +(vs1[i] * vs2[i]) + vd[i],FP widening multiply-accumulate, overwrites addend	乘号的两端都是SEW，乘积是2*SEW 加法另一端也是2*SEW 结果也是2*SEW，存入vd
V			vfwmacc.vf vd, rs1, vs2, vm	111100	vm	vs2	rs1	101	vd	1010111	vd[i] = +(f[rs1] * vs2[i]) + vd[i],FP widening multiply-accumulate, overwrites addend	
V			vfwnmacc.vv vd, vs1, vs2, vm	111101	vm	vs2	vs1	001	vd	1010111	vd[i] = -(vs1[i] * vs2[i]) - vd[i],FP widening negate-(multiply-accumulate), overwrites addend	
V			vfwnmacc.vf vd, rs1, vs2, vm	111101	vm	vs2	rs1	101	vd	1010111	vd[i] = -(f[rs1] * vs2[i]) - vd[i],FP widening negate-(multiply-accumulate), overwrites addend	
V			vfwmsac.vv vd, vs1, vs2, vm	111110	vm	vs2	vs1	001	vd	1010111	vd[i] = +(vs1[i] * vs2[i]) - vd[i],FP widening multiply-subtract-accumulator, overwrites addend	
V			vfwmsac.vf vd, rs1, vs2, vm	111110	vm	vs2	rs1	101	vd	1010111	vd[i] = +(f[rs1] * vs2[i]) - vd[i],FP widening multiply-subtract-accumulator, overwrites addend	
V			vfwnmsac.vv vd, vs1, vs2, vm	111111	vm	vs2	vs1	001	vd	1010111	vd[i] = -(vs1[i] * vs2[i]) + vd[i],FP widening negate-(multiply-subtract-accumulator), overwrites addend	
V			vfwnmsac.vf vd, rs1, vs2, vm	111111	vm	vs2	rs1	101	vd	1010111	vd[i] = -(f[rs1] * vs2[i]) + vd[i],FP widening negate-(multiply-subtract-accumulator), overwrites addend	
V												
V		Vector Floating-Point Square-Root Instruction	vfsqrt.v vd, vs2, vm	010011	vm	vs2	00000	001	vd	1010111	向量浮点平方根	
V		Vector Floating-Point Reciprocal Square-Root Estimate Instruction	vfrsqrt7.v vd, vs2, vm	010011	vm	vs2	00100	001	vd	1010111	一元向量-向量指令，用于返回 1/√x 的估计值，精度约为 7 位	
V		Vector Floating-Point Reciprocal Estimate Instruction	vfrec7.v vd, vs2, vm	010011	vm	vs2	00101	001	vd	1010111	一元向量-向量指令，返回 1/x的估计值，精度为7位	
V		Vector Floating-Point MIN/MAX Instructions	vfmin.vv vd, vs2, vs1, vm	000100	vm	vs2	vs1	001	vd	1010111	vd[i]是vs1[i],vs2[i]中的最小的	
V			vfmin.vf vd, vs2, rs1, vm	000100	vm	vs2	rs1	101	vd	1010111	vd[i]是f[rs1],vs2[i]中的最小的	
V			vfmax.vv vd, vs2, vs1, vm	000100	vm	vs2	vs1	001	vd	1010111	vd[i]是vs1[i],vs2[i]中的最大的	
V			vfmax.vf vd, vs2, rs1, vm	000100	vm	vs2	rs1	101	vd	1010111	vd[i]是f[rs1],vs2[i]中的最大的	
V												
V		Vector Floating-Point Sign-Injection Instructions	vfsgnj.vv vd, vs2, vs1, vm	001000	vm	vs2	vs1	001	vd/rd	1010111	把 vs1 的符号给 vs2，把vs2存储在vd	
V			vfsgnj.vf vd, vs2, rs1, vm	001000	vm	vs2	rs1	101	vd	1010111	把 rs1 的符号给 vs2，把vs2存储在vd	
V			vfsgnjn.vv vd, vs2, vs1, vm	001001	vm	vs2	vs1	001	vd/rd	1010111	把 vs1 的符号取反后给 vs2，把vs2存储在vd	
V			vfsgnjn.vf vd, vs2, rs1, vm	001001	vm	vs2	rs1	101	vd	1010111	把 rs1 的符号取反后给 vs2，把vs2存储在vd	
V			vfsgnjx.vv vd, vs2, vs1, vm	001010	vm	vs2	vs1	001	vd/rd	1010111	vs2的符号异或vs1的符号作为vs2的新符号，存储在vd	
V			vfsgnjx.vf vd, vs2, rs1, vm	001010	vm	vs2	rs1	101	vd	1010111	vs2的符号异或rs1的符号作为vs2的新符号，存储在vd	
V												
V		Vector Floating-Point Compare Instructions	vmfeq.vv vd, vs2, vs1, vm	011000	vm	vs2	vs1	001	vd	1010111	vd.mask[i] = vs2[i] == vs1[i]	
V			vmfeq.vf vd, vs2, rs1, vm	011000	vm	vs2	rs1	101	vd	1010111	vd.mask[i] = vs2[i] == f[rs1]	
V			vmfne.vv vd, vs2, vs1, vm	011100	vm	vs2	vs1	001	vd	1010111	vd.mask[i] = vs2[i] ! = vs1[i]	
V			vmfne.vf vd, vs2, rs1, vm	011100	vm	vs2	rs1	101	vd	1010111	vd.mask[i] = vs2[i] ! = f[rs1]	
V			vmflt.vv vd, vs2, vs1, vm	011011	vm	vs2	vs1	001	vd	1010111	vd.mask[i] = vs2[i] < vs1[i]	
V			vmflt.vf vd, vs2, rs1, vm	011011	vm	vs2	rs1	101	vd	1010111	vd.mask[i] = vs2[i] < f[rs1]	
V			vmfle.vv vd, vs2, vs1, vm	011001	vm	vs2	vs1	001	vd	1010111	vd.mask[i] = vs2[i] <= vs1[i]	
V			vmfle.vf vd, vs2, rs1, vm	011001	vm	vs2	rs1	101	vd	1010111	vd.mask[i] = vs2[i] <= f[rs1]	
V			vmfgt.vf vd, vs2, rs1, vm	011101	vm	vs2	rs1	101	vd	1010111	vd.mask[i] = vs2[i] > f[rs1]	
V			vmfge.vf vd, vs2, rs1, vm	011111	vm	vs2	rs1	101	vd	1010111	vd.mask[i] = vs2[i] >= f[rs1]	
V		Vector Floating-Point Classify Instruction	vfclass.v vd, vs2, vm	010011	vm	vs2	10000	001	vd	1010111	对每个向量浮点元素做分类，输出对应的 10-bit 类型编码，其他位填充0	
V		Vector Floating-Point Merge Instruction	vfmerge.vfm vd, vs2, rs1, v0	010111	0	vs2	rs1	101	vd	1010111	vd[i] = v0.mask[i] ? f[rs1] : vs2[i]	
V		Vector Floating-Point Move Instruction	vfmv.v.f vd, rs1	010111	1	0	rs1	101	vd	1010111	vd[i] = f[rs1]	

V		Single-Width Floating-Point/Integer Type-Convert Instructions	vfcvt.xu.f.v vd, vs2, vm	010010	vm	vs2	00000	001	vd	1010111	Convert float to unsigned integer.	
V			vfcvt.x.f.v vd, vs2, vm	010010	vm	vs2	00001	001	vd	1010111	Convert float to signed integer.	
V			vfcvt.rtz.xu.f.v vd, vs2, vm	010010	vm	vs2	00110	001	vd	1010111	Convert float to unsigned integer, truncating	
V			vfcvt.rtz.x.f.v vd, vs2, vm	010010	vm	vs2	00111	001	vd	1010111	Convert float to signed integer, truncating.	
V			vfcvt.f.xu.v vd, vs2, vm	010010	vm	vs2	00010	001	vd	1010111	Convert unsigned integer to float.	
V			vfcvt.f.x.v vd, vs2, vm	010010	vm	vs2	00011	001	vd	1010111	Convert signed integer to float.	
V		Widening Floating-Point/Integer Type-Convert Instructions	vfwcvt.xu.f.v vd, vs2, vm	010010	vm	vs2	01000	001	vd	1010111	Convert float to double-width unsigned integer.	
V			vfwcvt.x.f.v vd, vs2, vm	010010	vm	vs2	01001	001	vd	1010111	Convert float to double-width signed integer.	
V			vfwcvt.rtz.xu.f.v vd, vs2, vm	010010	vm	vs2	01110	001	vd	1010111	Convert float to double-width unsigned integer, truncating.	
V			vfwcvt.rtz.x.f.v vd, vs2, vm	010010	vm	vs2	01111	001	vd	1010111	Convert float to double-width signed integer, truncating.	
V			vfwcvt.f.xu.v vd, vs2, vm	010010	vm	vs2	01010	001	vd	1010111	Convert unsigned integer to double-width float.	
V			vfwcvt.f.x.v vd, vs2, vm	010010	vm	vs2	01011	001	vd	1010111	Convert signed integer to double-width float.	
V			vfwcvt.f.f.v vd, vs2, vm	010010	vm	vs2	01100	001	vd	1010111	Convert single-width float to double-width float.	
V		Narrowing Floating-Point/Integer Type-Convert Instructions	vfncvt.xu.f.w vd, vs2, vm	010010	vm	vs2	10000	001	vd	1010111	Convert double-width float to unsigned integer.	
V			vfncvt.x.f.w vd, vs2, vm	010010	vm	vs2	10001	001	vd	1010111	Convert double-width float to signed integer.	
V			vfncvt.rtz.xu.f.w vd, vs2, vm	010010	vm	vs2	10110	001	vd	1010111	Convert double-width float to unsigned integer, truncating.	
V			vfncvt.rtz.x.f.w vd, vs2, vm	010010	vm	vs2	10111	001	vd	1010111	Convert double-width float to signed integer, truncating.	
V			vfncvt.f.xu.w vd, vs2, vm	010010	vm	vs2	10010	001	vd	1010111	Convert double-width unsigned integer to float.	
V			vfncvt.f.x.w vd, vs2, vm	010010	vm	vs2	10011	001	vd	1010111	Convert double-width signed integer to float.	
V			vfncvt.f.f.w vd, vs2, vm	010010	vm	vs2	10100	001	vd	1010111	Convert double-width float to single-width float.	
V			vfncvt.rod.f.f.w vd, vs2, vm	010010	vm	vs2	10101	001	vd	1010111	Convert double-width float to single-width float,rounding towards odd.	
V												
V												
V	Vector Reduction Operations	Vector Single-Width Integer Reduction Instructions	vredsum.vs vd, vs2, vs1, vm	000000	vm	vs2	vs1	010	vd/rd	1010111	vd[0] = sum(vs1[0] , vs2[*])	
V			vredmaxu.vs vd, vs2, vs1, vm	000110	vm	vs2	vs1	010	vd/rd	1010111	vd[0] = maxu(vs1[0] , vs2[*])	
V			vredmax.vs vd, vs2, vs1, vm	000111	vm	vs2	vs1	010	vd/rd	1010111	vd[0] = max(vs1[0] , vs2[*])	
V			vredminu.vs vd, vs2, vs1, vm	000100	vm	vs2	vs1	010	vd/rd	1010111	vd[0] = minu(vs1[0] , vs2[*])	
V			vredmin.vs vd, vs2, vs1, vm	000101	vm	vs2	vs1	010	vd/rd	1010111	vd[0] = min(vs1[0] , vs2[*])	
V			vredand.vs vd, vs2, vs1, vm	000001	vm	vs2	vs1	010	vd/rd	1010111	vd[0] = and(vs1[0] , vs2[*])	
V			vredor.vs vd, vs2, vs1, vm	000010	vm	vs2	vs1	010	vd/rd	1010111	vd[0] = or(vs1[0] , vs2[*])	
V			vredxor.vs vd, vs2, vs1, vm	000011	vm	vs2	vs1	010	vd/rd	1010111	vd[0] = xor(vs1[0] , vs2[*])	
V												
V		Vector Widening Integer Reduction Instructions	vwredsumu.vs vd, vs2, vs1, vm	110000	vm	vs2	vs1	000	vd	1010111	2*SEW = 2*SEW + sum(zero-extend(SEW)), Unsigned sum reduction into double-width accumulator	
V			vwredsum.vs vd, vs2, vs1, vm	110001	vm	vs2	vs1	000	vd	1010111	2*SEW = 2*SEW + sum(sign-extend(SEW)), Signed sum reduction into double-width accumulator	
V		Vector Single-Width Floating-Point Reduction Instructions	vfredosum.vs vd, vs2, vs1, vm	000011	vm	vs2	vs1	001	vd	1010111	vd[0] = `(((vs1[0] + vs2[0]) + vs2[1]) + ...) + vs2[vl-1]`，按照元素顺序求和	
V			vfredusum.vs vd, vs2, vs1, vm	000001	vm	vs2	vs1	001	vd	1010111	浮点无序求和归约（允许实现自由决定加法顺序，保证结果符合要求）	
V			vfredmax.vs vd, vs2, vs1, vm	000111	vm	vs2	vs1	001	vd	1010111	找出最大的浮点数	
V			vfredmin.vs vd, vs2, vs1, vm	000101	vm	vs2	vs1	001	vd	1010111	找出最小的浮点数	
V		Vector Widening Floating-Point Reduction	vfwredosum.vs vd, vs2, vs1, vm	110011	vm	vs2	vs1	001	vd	1010111	对浮点数有序求和，结果是两倍位宽	
V			vfwredusum.vs vd, vs2, vs1, vm	110001	vm	vs2	vs1	001	vd	1010111	对浮点数无序求和，结果是两倍位宽	
V												
V												
V	Vector Mask Instructions	Vector Mask-Register Logical Instructions	vmand.mm vd, vs2, vs1	011001	1	vs2	vs1	010	vd	1010111	vd.mask[i] = vs2.mask[i] && vs1.mask[i]	操作的寄存器存放的是掩码，对vstart之后的元素生效，无视vlmul
V			vmnand.mm vd, vs2, vs1	011101	1	vs2	vs1	010	vd	1010111	vd.mask[i] = !(vs2.mask[i] && vs1.mask[i])	
V			vmandn.mm vd, vs2, vs1	011000	1	vs2	vs1	010	vd	1010111	vd.mask[i] = vs2.mask[i] && !vs1.mask[i]	
V			vmxor.mm vd, vs2, vs1	011011	1	vs2	vs1	010	vd	1010111	vd.mask[i] = vs2.mask[i] ^^ vs1.mask[i]	
V			vmor.mm vd, vs2, vs1	011010	1	vs2	vs1	010	vd	1010111	vd.mask[i] = vs2.mask[i] || vs1.mask[i]	
V			vmnor.mm vd, vs2, vs1	011110	1	vs2	vs1	010	vd	1010111	vd.mask[i] = !(vs2.mask[i] || vs1.mask[i])	
V			vmorn.mm vd, vs2, vs1	011100	1	vs2	vs1	010	vd	1010111	vd.mask[i] = vs2.mask[i] || !vs1.mask[i]	
V			vmxnor.mm vd, vs2, vs1	011111	1	vs2	vs1	010	vd	1010111	vd.mask[i] = !(vs2.mask[i] ^^ vs1.mask[i])	
V		Vector count population in mask vcpop.m	vcpop.m rd, vs2, vm	010000	vm	vs2	0x10	010	rd	1010111	统计存放掩码的寄存器中1的个数，并写入到x[rd]，使用vm时，只统计mask=1的元素	
V		vfirst find-first-set mask bit	vfirst.m rd, vs2, vm	010000	vm	vs2	0x11	010	rd	1010111	向x[rd]写入最小的，vs2为1的index，如果vm=0，只在掩码为1的元素中	
V		vmsbf.m set-before-first mask bit	vmsbf.m vd, vs2, vm	010100	vm	vs2	0x01	010	vd	1010111	以vs2第一个为1的元素为界限，vd中index在它之前的置1，之后的置0，这个元素也置0	
V		vmsif.m set-including-first mask bit	vmsif.m vd, vs2, vm	010100	vm	vs2	0x03	010	vd	1010111	与set-before-first，vmsbf.m类似，只是第一个为1的对应的位置置1	
V		vmsof.m set-only-first mask bit	vmsof.m vd, vs2, vm	010100	vm	vs2	0x02	010	vd	1010111	与vmsbf.m类似，但是只设置第一个满足条件的元素	
V		Vector Iota Instruction	viota.m vd, vs2, vm	010100	vm	vs2	0x10	010	vd	1010111	计算“vs2中某一元素之前有多少个元素是1”，并将这个前缀计数写入向量寄存器	
V		Vector Element Index Instruction	vid.v vd, vm	010100	vm	0	0x11	010	vd	1010111	writes each element’s index to the destination vector register group, from 0 to vl-1.	使用vm时不改变mask=0的元素在vd中的值
V	Vector Permutation Instructions	Integer Scalar Move Instructions	vmv.x.s rd, vs2	010000	1	vs2	0	010	rd	1010111	x[rd] = vs2[0] (vs1=0)	
V			vmv.s.x vd, rs1	010000	1	0	rs1	110	vd	1010111	vd[0] = x[rs1] (vs2=0)	
V		Floating-Point Scalar Move Instructions	vfmv.f.s rd, vs2	010000	1	vs2	0	001	rd	1010111	f[rd] = vs2[0] (rs1=0)	
V			vfmv.s.f vd, rs1	010000	1	0	rs1	101	vd	1010111	vd[0] = f[rs1] (vs2=0)	
V		Vector Slide Instructions	vslideup.vx vd, vs2, rs1, vm	001110	vm	vs2	rs1	100	vd	1010111	vd[i+x[rs1]] = vs2[i]	
V			vslideup.vi vd, vs2, uimm, vm	001110	vm	vs2	zimm5	011	vd	1010111	vd[i+uimm] = vs2[i]	
V			vslidedown.vx vd, vs2, rs1, vm	001111	vm	vs2	rs1	100	vd	1010111	vd[i] = vs2[i+x[rs1]]	
V			vslidedown.vi vd, vs2, uimm, vm	001111	vm	vs2	zimm5	011	vd	1010111	vd[i] = vs2[i+uimm]	
V			vslide1up.vx vd, vs2, rs1, vm	001110	vm	vs2	rs1	110	vd	1010111	vd[0]=x[rs1], vd[i+1] = vs2[i]	

V			vfslide1up.vf vd, vs2, rs1, vm	001110	vm	vs2	rs1	101	vd	1010111	vd[0]=f[rs1], vd[i+1] = vs2[i]	
V			vslide1down.vx vd, vs2, rs1, vm	001111	vm	vs2	rs1	110	vd	1010111	vd[i] = vs2[i+1], vd[vl-1]=x[rs1]	
V			vfslide1down.vf vd, vs2, rs1, vm	001111	vm	vs2	rs1	101	vd	1010111	vd[i] = vs2[i+1], vd[vl-1]=f[rs1]	
V		Vector Register Gather Instructions	vrgather.vv vd, vs2, vs1, vm	001100	vm	vs2	vs1	000	vd	1010111	vd[i] = (vs1[i] >= VLMAX) ? 0 : vs2[vs1[i]];	
V			vrgatherei16.vv vd, vs2, vs1, vm	001110	vm	vs2	vs1	000	vd	1010111	vd[i] = (vs1[i] >= VLMAX) ? 0 : vs2[vs1[i]];	
V			vrgather.vx vd, vs2, rs1, vm	001100	vm	vs2	rs1	100	vd	1010111	vd[i] = (x[rs1] >= VLMAX) ? 0 : vs2[x[rs1]]	
V			vrgather.vi vd, vs2, uimm, vm	001100	vm	vs2	zimm5	011	vd	1010111	vd[i] = (uimm >= VLMAX) ? 0 : vs2[uimm]	
V		Vector Compress Instruction	vcompress.vm vd, vs2, vs1	010111	1	vs2	vs1	010	vd	1010111	The vector compress instruction allows elements selected by a vector mask register from a source vector register group to be packed into contiguous elements at the start of the destination vector register group.	
V			vmv<nr>r.v vd, vs2									
V		Whole Vector Register Move	vmv1r.v v1, v2	100111	1	vs2	0	011	vd	1010111	Copy v1=v2	根据指令模板vmv<nr>r.v vd, vs2 只允许nr=1,2,4,8派生出
V			vmv2r.v v10, v12	100111	1	vs2	1	011	vd	1010111	Copy v10=v12; v11=v13	
V			vmv4r.v v4, v8	100111	1	vs2	3	011	vd	1010111	Copy v4=v8; v5=v9; v6=v10; v7=v11	
V			vmv8r.v v0, v8	100111	1	vs2	7	011	vd	1010111	Copy v0=v8; v1=v9; ...; v7=v15	

RVA23中额外的向量相关扩展

extension	class	group	inst	func6	vm	op2_bit	op1_bit	func3_seg	dst_seg	opcode	function	note
Zvbb		Bitwise And-Not	vandn.vv vd, vs2, vs1, vm	000001	vm	vs2	vs1	000	vd	1010111	把vs1按位取反然后跟vs2按位AND	
Zvbb			vandn.vx vd, vs2, rs1, vm	000001	vm	vs2	rs1	100	vd	1010111	把x[rs1]符号扩展或者截断，然后再取反并与vs2按位AND	
Zvbb		Vector Reverse Bits in Elements	vbrev.v vd, vs2, vm	010010	vm	vs2	01010	010	vd	1010111	把SEW位宽的每个数据的每一位翻转出现的顺序，LSB变MSB	
Zvbb		Vector Reverse Bits in Bytes	vbrev8.v vd, vs2, vm	010010	vm	vs2	01000	010	vd	1010111	以8bits为单位，单位内反转每个Bit出现的顺序	
Zvbb		Vector Reverse Bytes	vrev8.v vd, vs2, vm	010010	vm	vs2	01001	010	vd	1010111	翻转每个byte出现的顺序	
Zvbb		Vector Count Leading Zeros	vclz.v vd, vs2, vm	010010	vm	vs2	01100	010	vd	1010111	计算每个元素先导零的个数	
Zvbb		Vector Count Trailing Zeros	vctz.v vd, vs2, vm	010010	vm	vs2	01101	010	vd	1010111	计算每个元素从最低位开始零的个数	
Zvbb		Count the number of bits set in each element	vcpop.v vd, vs2, vm	010010	vm	vs2	01110	010	vd	1010111	计算每个元素1的个数	
Zvbb		Vector rotate left by vector/scalar.	vrol.vv vd, vs2, vs1, vm	010101	vm	vs2	vs1	000	vd	1010111	把vs2中的每个元素循环左移vs1指定的位数	
Zvbb			vrol.vx vd, vs2, rs1, vm	010101	vm	vs2	rs1	100	vd	1010111	把vs2中的每个元素循环左移x[rs1]指定的位数	
Zvbb		Vector rotate right by vector/scalar/immediate.	vror.vv vd, vs2, vs1, vm	010100	vm	vs2	vs1	000	vd	1010111	把vs2中的元素旋转右移vs1中指定的位数	
Zvbb			vror.vx vd, vs2, rs1, vm	010100	vm	vs2	rs1	100	vd	1010111	把vs2中的元素旋转右移x[rs1]中指定的位数	
Zvbb			vror.vi vd, vs2, uimm, vm	01010|i5	vm	vs2	uimm[4:0]	011	vd	1010111	使用i5以及uimm[4:0]拼凑出6位的无符号数，然后vs2中的元素旋转右移相应的位	
Zvbb		Vector widening shift left logical by vector/scalar/immediate.	vwsll.vv vd, vs2, vs1, vm	110101	vm	vs2	vs1	000	vd	1010111	vs2先使用零填充由SEW填充到2*SEW，然后根据vs1相应的值左移，偏移量只有低log2(2*SEW)有效	
Zvbb			vwsll.vx vd, vs2, rs1, vm	110101	vm	vs2	rs1	100	vd	1010111	vs2先使用零填充由SEW填充到2*SEW，然后根据x[rs1]相应的值左移，偏移量只有低log2(2*SEW)有效	
Zvbb			vwsll.vi vd, vs2, uimm, vm	110101	vm	vs2	zimm5	011	vd	1010111	vs2先使用零填充由SEW填充到2*SEW，然后根据立即数相应的值左移，偏移量只有低log2(2*SEW)有效	
Zvbc		Vector Carry-less Multiply by vector or	vclmul.vv vd, vs2, vs1, vm	001100	vm	vs2	vs1	010	vd	1010111	对vs1和vs2中相应的两个64位数进行无进位乘法，结果取乘积的最低有效64位	只对SEW等于64有效
			vclmul.vx vd, vs2, rs1, vm	001100	vm	vs2	rs1	110	vd	1010111	对vs2和x[rs1]中相应的两个64位数进行无进位乘法，结果取乘积的最低有效64位	
		Vector Carry-less Multiply by vector or	vclmulh.vv vd, vs2, vs1, vm	001101	vm	vs2	vs1	010	vd	1010111	将vs2与vs1中的对应元素进行无进位乘法，结果取乘积的最高有效64位	
			vclmulh.vx vd, vs2, rs1, vm	001101	vm	vs2	rs1	110	vd	1010111	将vs2与x[rs1]中的对应元素进行无进位乘法，结果取乘积的最高有效64位	
Zvknhb		Vector SHA-2 message schedule.	vsha2ms.vv vd, vs2, vs1	101101	1	vs2	vs1	010	vd	1110111	执行SHA-256(SEW=32)/SHA-512(SEW=64)信息调度的两个步骤，一次性计算4个新的信息调度值	SEW只允许是32或者64,vd不能跟vs1或vs2重叠
		Vector SHA-2 two rounds of compression.	vsha2ch.vv vd, vs2, vs1	101110	1	vs2	vs1	010	vd	1110111	更新前4个状态	来自 vs1 的两个字，会与当前状态中保存在 vd 和 vs2 中的八个字一起参与计算，执行两轮哈希运算，生成下一
			vsha2cl.vv vd, vs2, vs1	101111	1	vs2	vs1	010	vd	1110111	更新后4个状态	
Zvkned	NIST Suite: Vector AES Block Cipher	Vector AES final-round encryption	vaesef.vv vd, vs2	101000	1	vs2	00011	010	vd	1110111	对 AES 分组密码执行最终一轮的加密，轮密钥来自vs2中对应的元素组	SEW必须是32
			vaesef.vs vd, vs2	101001	1	vs2	00011	010	vd	1110111	对 AES 分组密码执行最终一轮的加密，轮密钥来自vs2的第0个元素	
		Vector AES middle-round encryption	vaesem.vv vd, vs2	101000	1	vs2	00010	010	vd	1110111	执行 AES 分组密码的一般（中间）轮加密，轮密钥来自vs2	SEW必须是32
			vaesem.vs vd, vs2	101001	1	vs2	00010	010	vd	1110111	执行 AES 分组密码的一般（中间）轮加密，轮密钥来自vs2的第0个元素	
		Vector AES final-round decryption	vaesdf.vv vd, vs2	101000	1	vs2	00001	010	vd	1110111	执行 AES 块密码的最终轮解密，轮密钥来自vs2	SEW必须是32
			vaesdf.vs vd, vs2	101001	1	vs2	00001	010	vd	1110111	执行 AES 块密码的最终轮解密，轮密钥来自vs2的第0个元素	
		Vector AES middle-round decryption	vaesdm.vv vd, vs2	101000	1	vs2	00000	010	vd	1110111	A middle-round AES block cipher decryption is performed.This is then XORed with the round key in either the corresponding element group in vs2 (vector-vector form)	SEW必须是32
			vaesdm.vs vd, vs2	101001	1	vs2	00000	010	vd	1110111	A middle-round AES block cipher decryption is performed.This is then XORed with the round key in either the corresponding element group in vs2 (vector-vector form) the scalar element group in vs2 (vector-scalar form)	
		Vector AES-128 Forward KeySchedule generation	vaeskf1.vi vd, vs2, uimm	100010	1	vs2	zimm5	010	vd	1110111	执行一次前向 AES-128 密钥扩展（KeySchedule）的单轮操作。轮数（round number）范围为 1 到 10，由 uimm[3:0] 提供；uimm[4] 被忽略。超出范围的 uimm[3:0] 值（即 0 和 11-15）通过取 uimm[3] 的反码映射到有效范围。因此，0 映射为 8，11-15 映射为 3-7。轮数用于指定轮常量，从而生成第一个轮密钥字。	SEW必须是32
		Vector AES-256 Forward KeySchedule generation	vaeskf2.vi vd, vs2, uimm	101010	1	vs2	zimm5	010	vd	1110111	A single round of the forward AES-256 KeySchedule is performed.The round number, which ranges from 2 to 14, comes from uimm[3:0]; uimm[4] is ignored. The out-of-range uimm[3:0] values of 0-1 and 15 are mapped to in-range values by inverting uimm[3]. Thus, 0-1 maps to 8-9, and 15 maps to 7.	SEW必须是32
		Vector AES round zero encryption/decryption	vaesz.vs vd, vs2	101001	1	vs2	00111	010	vd	1110111	A round-0 AES block cipher operation is performed.Vs2 holds a scalar element group that is used as the round key for all of the round state element groups	SEW必须是32

Zvksed	ShangMi Suite: SM4 Block Cipher	Vector SM4 KeyExpansion	vsm4k.vi vd, vs2, uimm	100001	1	vs2	zimm5	010	vd	1110111	Four rounds of the SM4 Key Expansion are performed. The Round group number (rnd) comes from uimm[2:0]; the bits in uimm[4:3] are ignored. Round group numbers range from 0 to 7 and indicate which group of four round keys are being generated. Round Keys range from 0-31.	
		Vector SM4 Rounds	vsm4r.vv vd, vs2	101000	1	vs2	10000	010	vd	1110111	Four rounds of SM4 Encryption/Decryption are performed. round keys are read from corresponding 4-element group in vs2	
			vsm4r.vs vd, vs2	101001	1	vs2	10000	010	vd	1110111	Four rounds of SM4 Encryption/Decryption are performed. round keys are read from scalar element group in vs2	
Zvkg		Vector Add-Multiply over GHASH Galois-Field	vghsh.vv vd, vs2, vs1	101100	1	vs2	vs1	010	vd	1110111	A single "iteration" of the GHASH$_H$ algorithm is performed.	
		Vector Multiply over GHASH Galois-Field	vgmul.vv vd, vs2	101000	1	vs2	vs1	010	vd	1110111	A GHASH$_H$ multiply is performed.	
Zvksh	ShangMi Suite: SM3 Secure Hash	Vector SM3 Message Expansion	vsm3me.vv vd, vs2, vs1	100000	1	vs1	vs1	010	vd	1110111	Eight rounds of SM3 message expansion are performed.	SEW要等于32
		Vector SM3 Compression	vsm3c.vi vd, vs2, uimm	101011	1	vs1	zimm5	010	vd	1110111	Two rounds of SM3 compression are performed.	