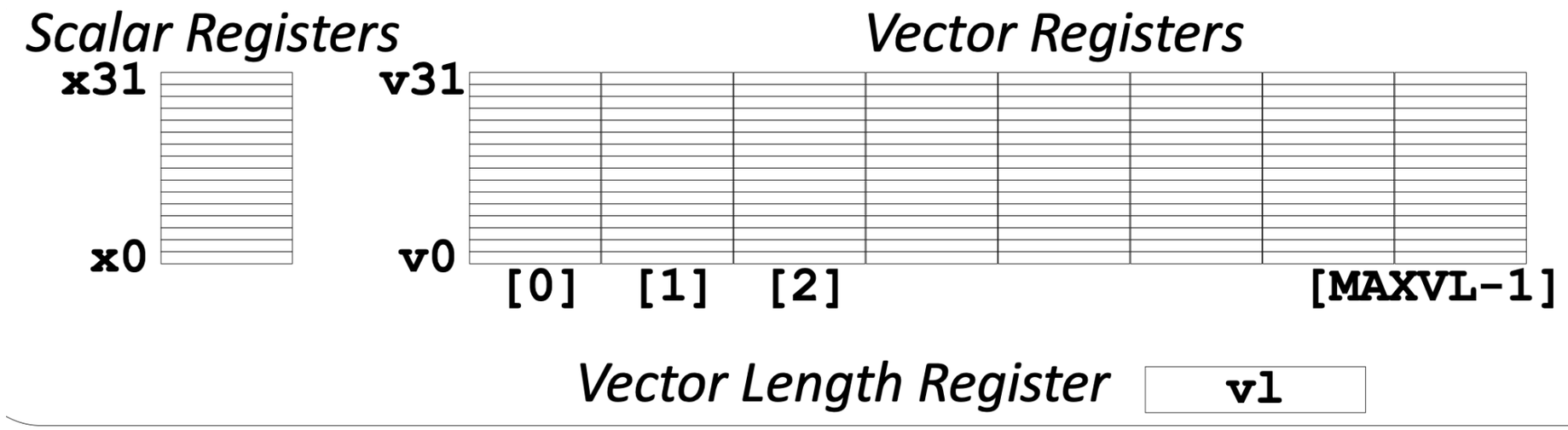
RISC-V V
Vector Extension

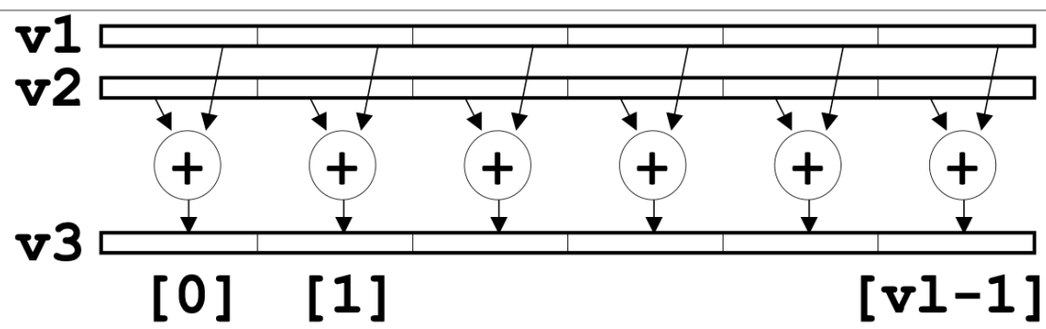
Xiayuan Wen

Vector Programming Model



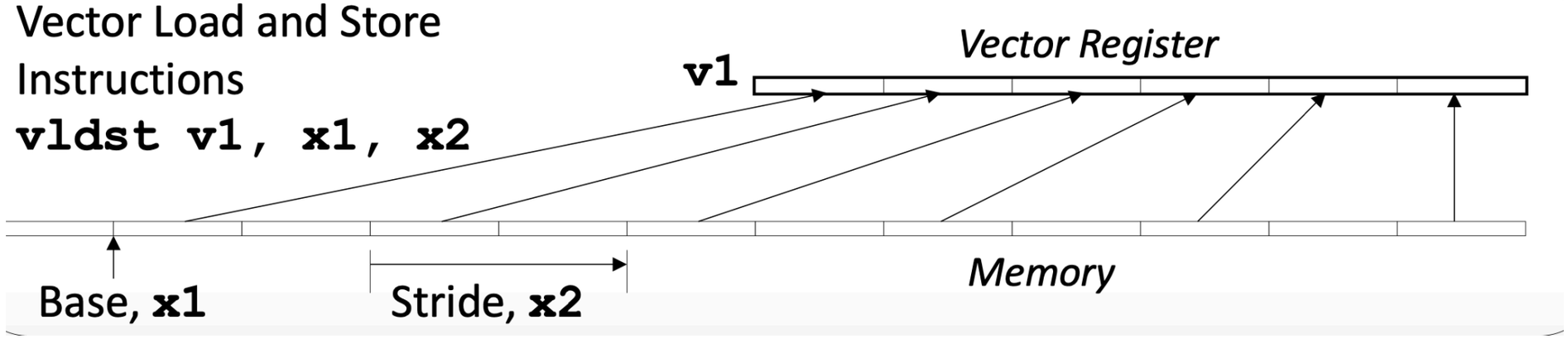
Vector Arithmetic Instructions

`vadd v3, v1, v2`

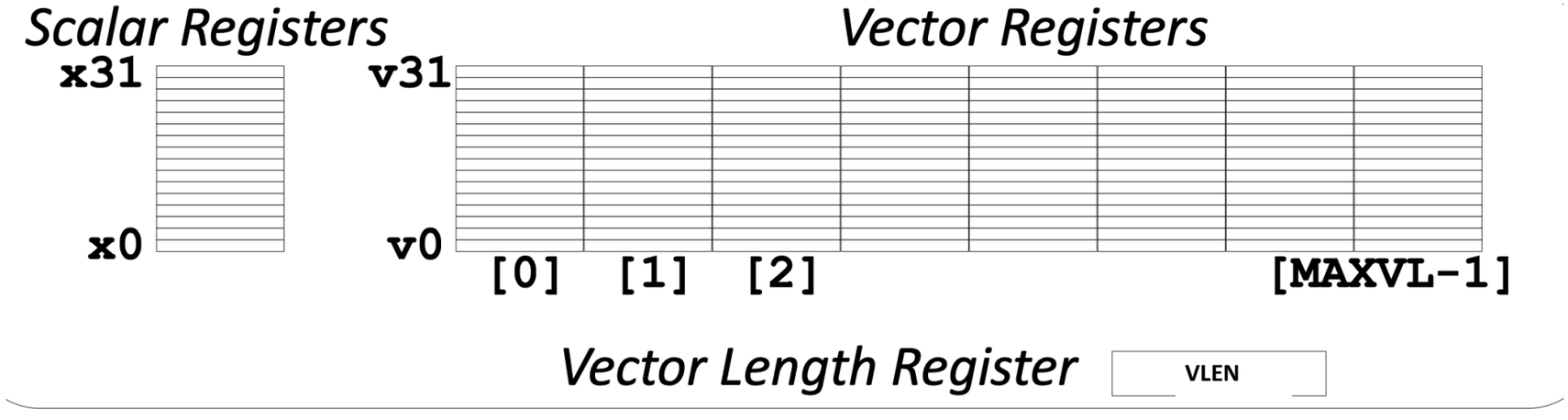


Vector Load and Store Instructions

`vldst v1, x1, x2`



Vector Extension — Registers



- The vector extension adds 32 architectural vector registers, v0-v31 to the base scalar registers, x0-x31.
  - The number of bits in a single vector register, VLEN, which must be a power of 2, and must be no greater than 2^{16}.
- The vector extension adds 7 unprivileged CSRs

Address	Privilege	Name	Description
0x008	URW	vstart	Vector start position
0x009	URW	vxsat	Fixed-Point Saturate Flag
0x00A	URW	vxrm	Fixed-Point Rounding Mode
0x00F	URW	vcsr	Vector control and status register
0xC20	URO	vl	Vector length
0xC21	URO	vtype	Vector data type register
0xC22	URO	vlenb	VLEN/8 (vector register length in bytes)

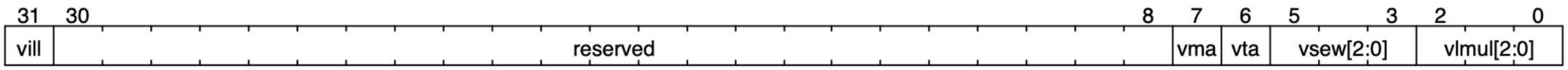
Unprivileged CSRs

- The read-only CSR **vlenb** holds the value VLEN/8, the vector register length in bytes. The value in vlenb is a design-time constant.
- The vector fixed-point rounding-mode register holds a two-bit read-write rounding-mode field in the least-signicant bits (**vxrm**[1:0]).
- The **vxsat** CSR has a single read-write least-signicant bit (vxsat[0]) that indicates if a fixed-point instruction has had to saturate an output value to fit into a destination format.
- The **vxrm** and **vxsat** are given separate CSR addresses to allow independent accesses, but are also reflected in **vcsr**. (Vector Control and Status Register)

VCSR Layout

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

Unprivileged CSRs — vtype



Bits	Name	Description
XLEN-1	vill	Illegal value if set
XLEN-2:8	0	Reserved if non-zero
7	vma	Vector mask agnostic
6	vta	Vector tail agnostic
5:3	vsew[2:0]	Selected element width (SEW) setting
2:0	vlmul[2:0]	Vector register group multiplier (LMUL) setting

- The vector type CSR, **vtype** provides the default type used to interpret the contents of the vector register file, and can only be updated by vset{i}vl{i} instructions.
- The value in vsew sets the dynamic selected element width (SEW). By default, a vector register is viewed as being divided into VLEN/SEW elements.
- Multiple vector registers can be grouped together, so that a single vector instruction can operate on multiple vector registers. The vector length multiplier, LMUL, when greater than 1, represents the default number of vector registers that are combined to form a vector register group.
- VLMAX = VLEN/SEW * LMUL

Unprivileged CSRs

- The read-only **vl** CSR can only be updated by the vset{i}vl{i} instructions, and the fault-only-first vector load instruction variants. In order to distribute work evenly over the last two iterations of a stripmine loop.

1. `vl = AVL if AVL ≤ VLMAX`
2. `ceil(AVL / 2) ≤ vl ≤ VLMAX if AVL < (2 * VLMAX)`
3. `vl = VLMAX if AVL ≥ (2 * VLMAX)`

- The **vstart** CSR species the index of the first element to be executed by a vector instruction.

```
for element index x
prestart(x) = (0 <= x < vstart)
body(x)     = (vstart <= x < vl)
tail(x)     = (vl <= x < max(VLMAX,VLEN/SEW))
mask(x)     = unmasked || v0.mask[x] == 1
active(x)   = body(x) && mask(x)
inactive(x) = body(x) && !mask(x)
```

Mapping of Elements to Vector Register

- Mask Register Layout: A vector mask occupies only one vector register regardless of SEW and LMUL. The mask bit for element i is located in bit i of the mask register, independent of SEW or LMUL.
- Mapping for LMUL = 1

```
VLEN=32b

Byte          3 2 1 0

SEW=8b         3 2 1 0
SEW=16b          1   0
SEW=32b              0

VLEN=64b

Byte          7 6 5 4 3 2 1 0

SEW=8b        7 6 5 4 3 2 1 0
SEW=16b         3   2   1   0
SEW=32b             1       0
SEW=64b                     0
```

Mapping of Elements to Vector Register

- Mapping for LMUL < 1: only the first LMUL*VLEN/SEW elements in the vector register are used. The remaining space in the vector register is treated as part of the tail.

```
Example, VLEN=128b, LMUL=1/4

Byte        F E D C B A 9 8 7 6 5 4 3 2 1 0

SEW=8b      - - - - - - - - - - - - - 3 2 1 0
SEW=16b       -   -   -   -   -   -   1   0
SEW=32b           -       -       -       0
```

- Mapping for LMUL > 1

```
VLEN=32b, SEW=8b, LMUL=2

Byte          3 2 1 0
v2*n          3 2 1 0
v2*n+1        7 6 5 4
```

```
VLEN=32b, SEW=16b, LMUL=4

Byte            3 2 1 0
v4*n              1   0
v4*n+1            3   2
v4*n+2            5   4
v4*n+3            7   6
```

Instruction Formats

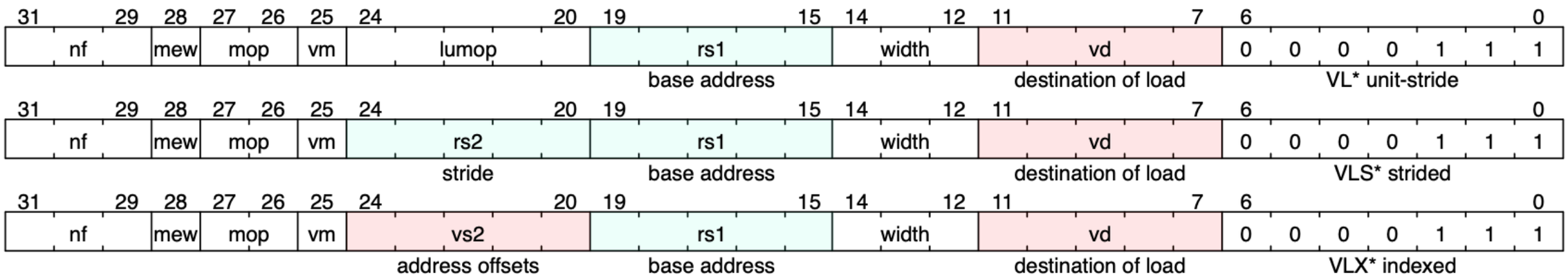
- inst[1:0] = 00, 01, 10: Compressed Instructions
- inst[1:0] = 11:
  - inst[6:2] = 00001: vector load and LOAD-FP
  - inst[6:2] = 01001: vector store and STORE-FP
  - inst[6:2] = 10101: vector arithmetic instructions and vector configuration instructions (reserved)
  - inst[6:2] = 00010: custom-0 (including vector-fetch control thread instructions)
  - inst[6:2] = 01010: custom-1 (including vector-fetch control thread instructions)
  - inst[6:2] = 01111: 64-bit instructions (including vector-fetch worker thread instructions)

inst[4:2] / inst[6:5]	000	001	010	011	100	101	110	111 ($> 32b$)
00	LOAD	LOAD-FP	*custom-0*	MISC-MEM	OP-IMM	AUIPC	OP-IMM-32	$48b$
01	STORE	STORE-FP	*custom-1*	AMO	OP	LUI	OP-32	$64b$
10	MADD	MSUB	NMSUB	NMADD	OP-FP	*reserved*	*custom-2/rv128*	$48b$
11	BRANCH	JALR	*reserved*	JAL	SYSTEM	*reserved*	*custom-3/rv128*	$\geq 80b$

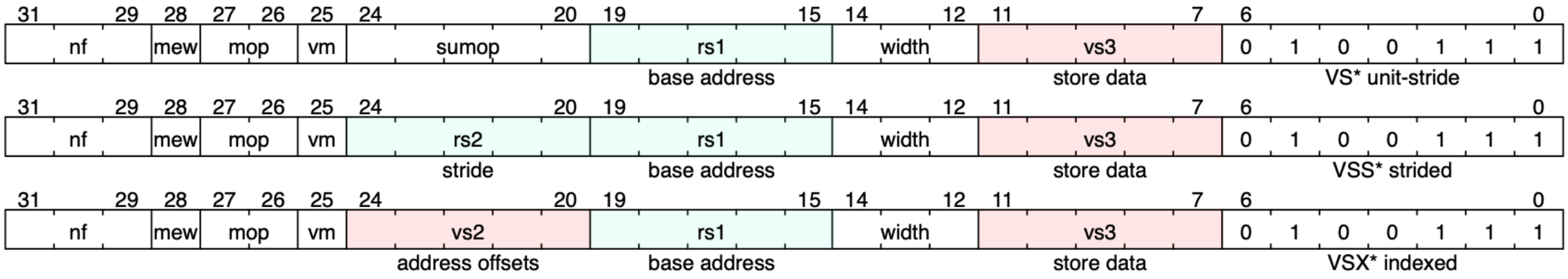
Vector Instruction Formats

- Vector loads and stores are encoded within the scalar floating-point load and store major opcodes (LOAD-FP/STORE-FP).

Format for Vector Load Instructions under LOAD-FP major opcode

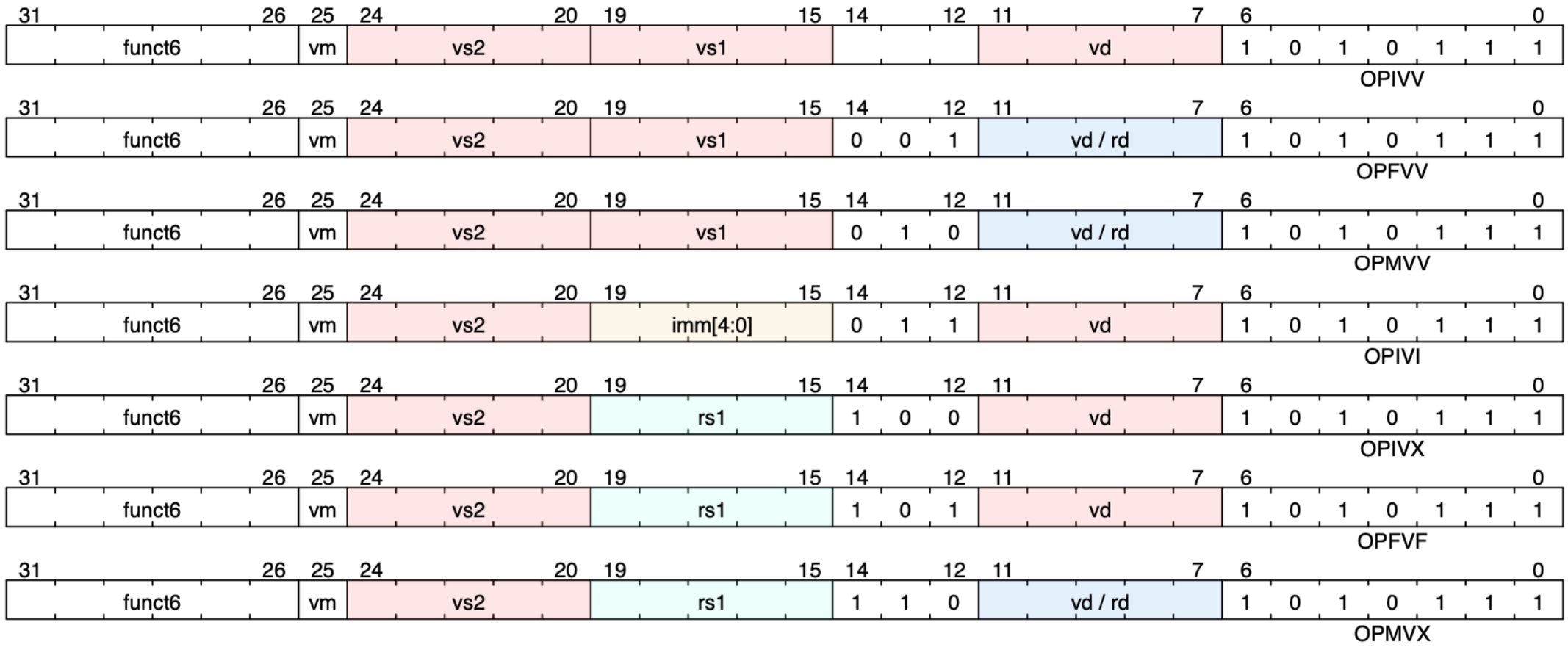


Format for Vector Store Instructions under STORE-FP major opcode

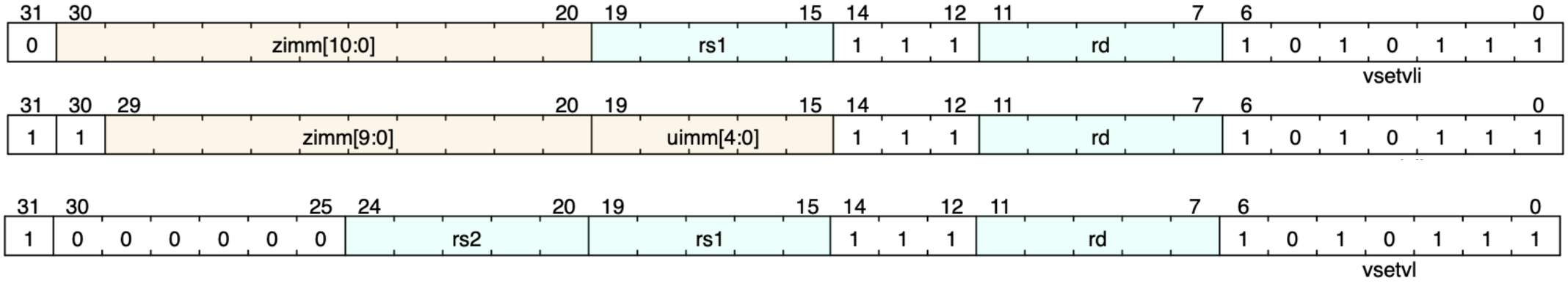


Vector Instruction Formats

Formats for Vector Arithmetic Instructions under OP-V major opcode



Formats for Vector Configuration Instructions under OP-V major opcode



Vector Instructions

- Vector instructions can have scalar or vector source operands and produce scalar or vector results, and most vector instructions can be performed either unconditionally or conditionally under a mask.
- Mask:
  - The mask value used to control execution of a masked vector instruction is always supplied by vector register v0.
  - Where available, masking is encoded in a single-bit vm eld in the instruction (inst[25]).

vm	Description
0	vector result, only where v0.mask[i] = 1
1	unmasked

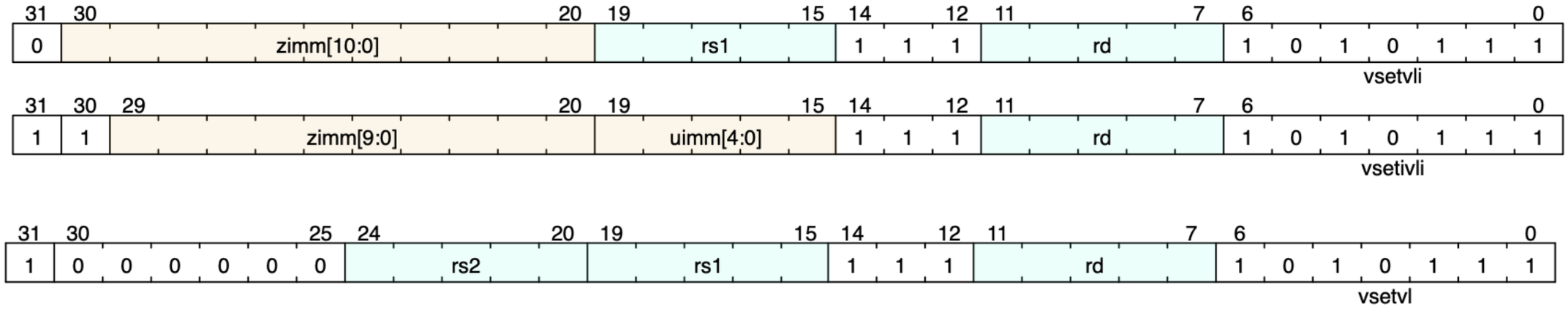
```
vop.v*    v1, v2, v3, v0.t  # enabled where v0.mask[i]=1, vm=0
vop.v*    v1, v2, v3        # unmasked vector operation, vm=1
```

Vector ISA

- Configuration-Setting Instructions
- Vector Loads and Stores:
  - Vector Unit-Stride Instructions
  - Vector Strided Instructions
  - Vector Unordered or Ordered Indexed Instructions
- Vector Arithmetic Instructions:
  - Vector Integer Arithmetic Instructions
  - Vector Fixed-point Arithmetic Instructions
  - Vector Floating-point Arithmetic Instructions
  - Vector Reduction Instructions
  - Vector Mask Instructions
  - Vector Permutation Instructions

Configuration-Setting Instructions

Formats for Vector Configuration Instructions under OP-V major opcode



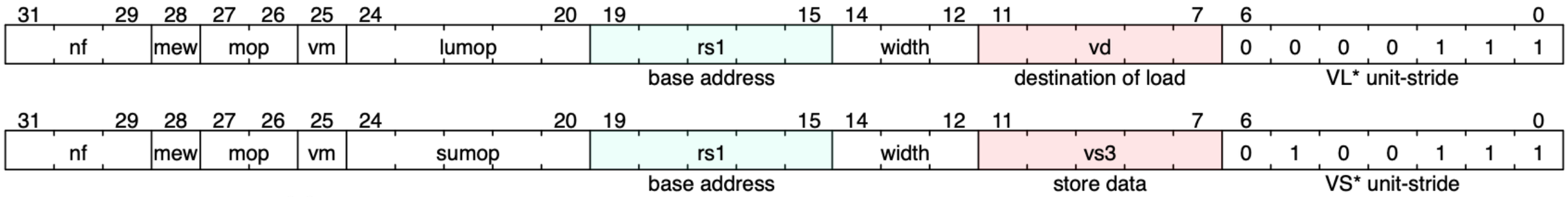

```
vsetvli rd, rs1, vtypei   # rd = new vl, rs1 = AVL, vtypei = new vtype setting
vsetivli rd, uimm, vtypei # rd = new vl, uimm = AVL, vtypei = new vtype setting
vsetvl  rd, rs1, rs2      # rd = new vl, rs1 = AVL, rs2 = new vtype value
```

- Application specifies the total number of elements to be processed (the application vector length or AVL) as a candidate value for vl, and the hardware responds via a general-purpose register with the (frequently smaller) number of elements that the hardware will handle per iteration (stored in vl), based on the microarchitectural implementation and the vtype setting.

Vector ISA

- Configuration-Setting Instructions
- Vector Loads and Stores:
  - Vector Unit-Stride Instructions
  - Vector Strided Instructions
  - Vector Unordered or Ordered Indexed Instructions
- Vector Arithmetic Instructions:
  - Vector Integer Arithmetic Instructions
  - Vector Fixed-point Arithmetic Instructions
  - Vector Floating-point Arithmetic Instructions
  - Vector Reduction Instructions
  - Vector Mask Instructions
  - Vector Permutation Instructions

Vector Unit-Stride Inst.



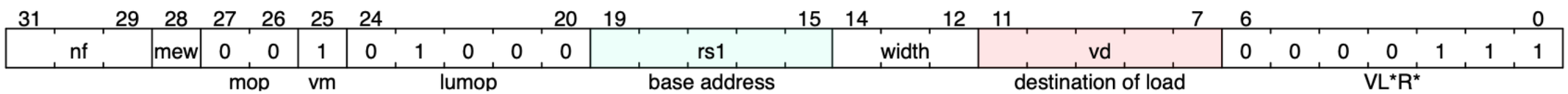
mop = 00

unit-stride: transfers vectors whose elements are held in contiguous locations in memory
unit-stride-segmented: transfers values between consecutive vector registers and memory

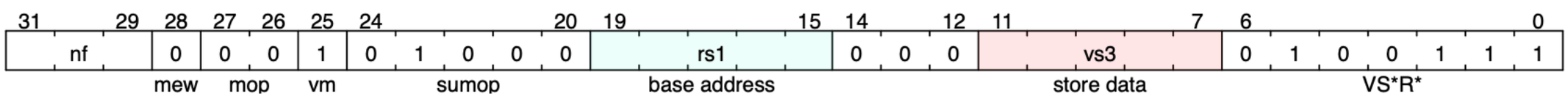
```
    # vd destination, rs1 base address, vm is mask encoding (v0.t or <missing>)
    vle8.v    vd, (rs1), vm  #    8-bit unit-stride load
    vle16.v   vd, (rs1), vm  #   16-bit unit-stride load
    vle32.v   vd, (rs1), vm  #   32-bit unit-stride load
    vle64.v   vd, (rs1), vm  #   64-bit unit-stride load

# Format
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

# Examples
vlseg8e8.v vd, (rs1), vm   # Load eight vector registers with eight byte fields.

vsseg3e32.v vs3, (rs1), vm  # Store packed vector of 3*4-byte segments from vs3,vs3+1,vs3+2 to mem
```

Vector Unit-Stride Inst. — Whole Register

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



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



These instructions load and store whole vector register groups. These instructions are intended to be used to save and restore vector registers when the type or length of the current contents of the vector register is not known, or where modifying vl and vtype would be costly.

```
vl2re8.v    v2, (a0)  # Load v2-v3 with 2*VLEN/8 bytes from address in a0
vl2re16.v   v2, (a0)  # Load v2-v3 with 2*VLEN/16 halfwords held at address in a0
vl2re32.v   v2, (a0)  # Load v2-v3 with 2*VLEN/32 words held at address in a0
vl2re64.v   v2, (a0)  # Load v2-v3 with 2*VLEN/64 doublewords held at address in a0
```

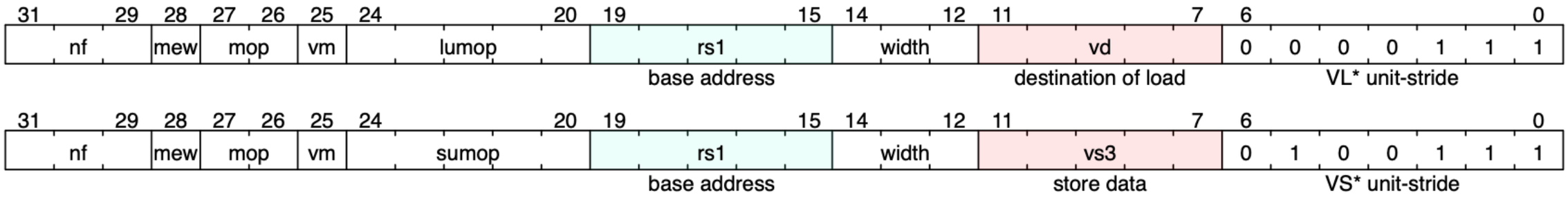
Vector Unit-Stride Inst. — Fault-Only-First Loads

lumop = 10000

These instructions execute as a regular load except that they will only take a trap caused by a synchronous exception on element 0. If element 0 raises an exception, vl is not modied, and the trap is taken. If an element > 0 raises an exception, the corresponding trap is not taken, and the vector length vl is reduced to the index of the element that would have raised an exception.
They used to vectorize loops with data-dependent exit conditions.

```
# vd destination, rs1 base address, vm is mask encoding (v0.t or <missing>)
vle8ff.v    vd, (rs1), vm  #    8-bit unit-stride fault-only-first load
vle16ff.v   vd, (rs1), vm  #   16-bit unit-stride fault-only-first load
vle32ff.v   vd, (rs1), vm  #   32-bit unit-stride fault-only-first load
vle64ff.v   vd, (rs1), vm  #   64-bit unit-stride fault-only-first load
```

Vector Unit-Stride Inst. — Mask

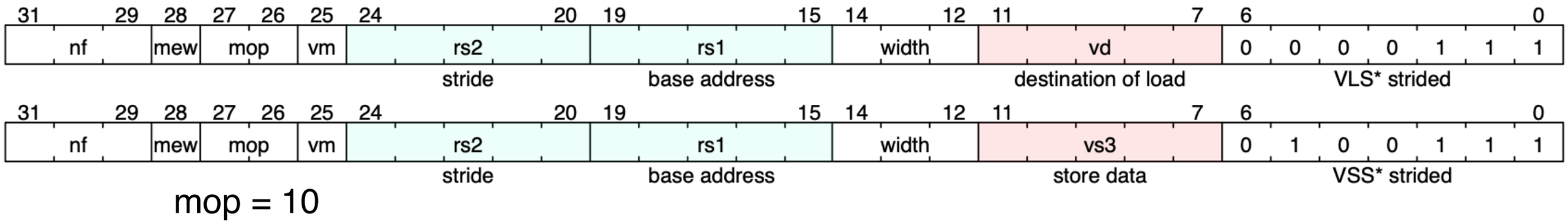


lumop = 01011

These instructions transfer mask values to/from memory.

```
# Vector unit-stride mask load
vlm.v vd, (rs1)   #  Load byte vector of length ceil(vl/8)

# Vector unit-stride mask store
vsm.v vs3, (rs1)  #  Store byte vector of length ceil(vl/8)
```

Vector Strided Inst.

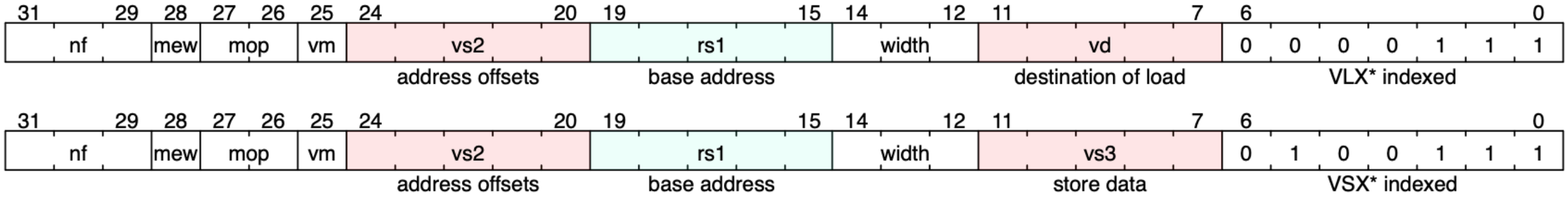


strided: transfers vectors whose elements are held in memory addresses that form an arithmetic progression

strided-segmented: transfers values between consecutive vector registers and memory

```
# vd destination, rs1 base address, rs2 byte stride
vlse8.v    vd, (rs1), rs2, vm  #    8-bit strided load
vlse16.v   vd, (rs1), rs2, vm  #   16-bit strided load
vlse32.v   vd, (rs1), rs2, vm  #   32-bit strided load
vlse64.v   vd, (rs1), rs2, vm  #   64-bit strided load
```

Vector Indexed Inst. —Unordered or Ordered



mop = 01/11

Vector indexed memory instructions transfer vectors whose elements are located at offsets from a base address, with the offsets specified by the values of an index vector.

The indexed-ordered variants preserve element ordering on memory accesses.

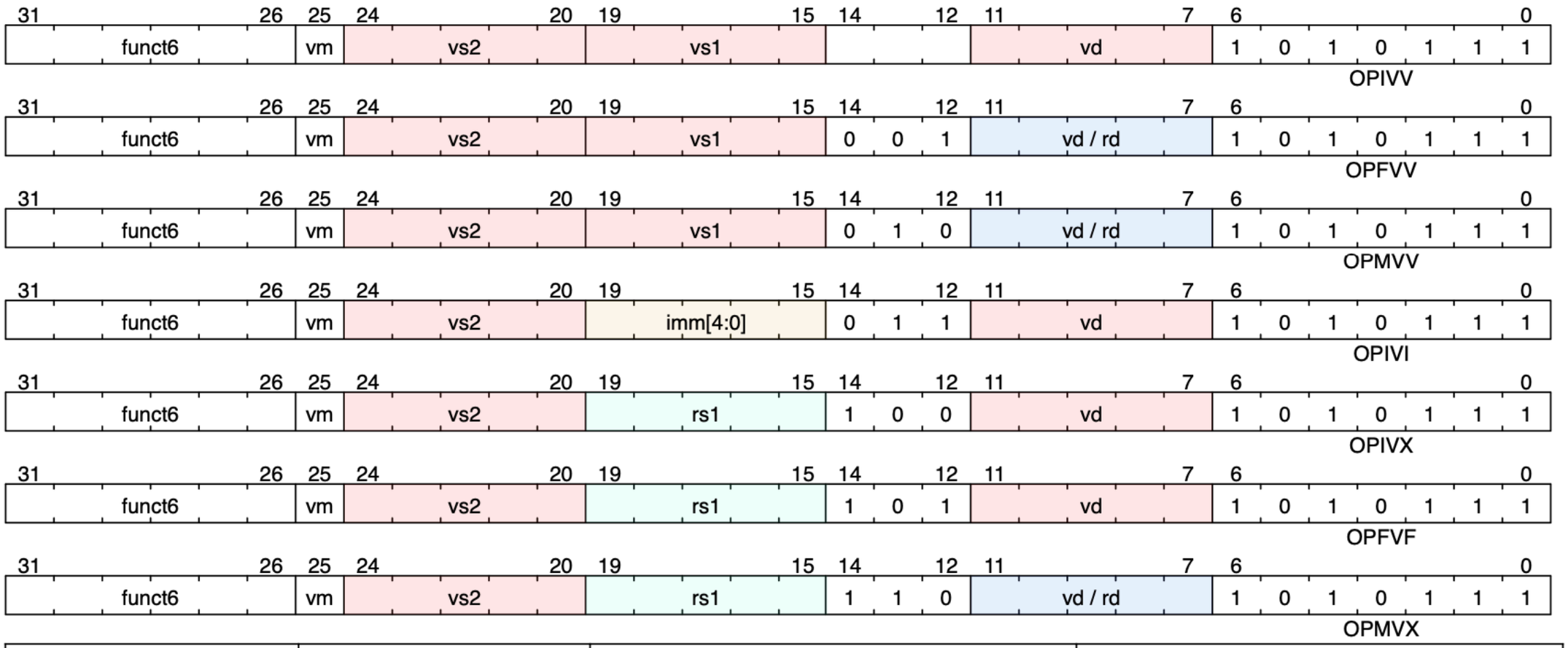
```
# vd destination, rs1 base address, vs2 byte offsets
vluxei8.v    vd, (rs1), vs2, vm  # unordered  8-bit indexed load of SEW data
vluxei16.v   vd, (rs1), vs2, vm  # unordered 16-bit indexed load of SEW data
vluxei32.v   vd, (rs1), vs2, vm  # unordered 32-bit indexed load of SEW data
vluxei64.v   vd, (rs1), vs2, vm  # unordered 64-bit indexed load of SEW data

# Vector indexed-ordered load instructions
# vd destination, rs1 base address, vs2 byte offsets
vloxei8.v    vd, (rs1), vs2, vm  # ordered  8-bit indexed load of SEW data
vloxei16.v   vd, (rs1), vs2, vm  # ordered 16-bit indexed load of SEW data
vloxei32.v   vd, (rs1), vs2, vm  # ordered 32-bit indexed load of SEW data
vloxei64.v   vd, (rs1), vs2, vm  # ordered 64-bit indexed load of SEW data
```

Vector ISA

- Configuration-Setting Instructions
- Vector Loads and Stores:
  - Vector Unit-Stride Instructions
  - Vector Strided Instructions
  - Vector Unordered or Ordered Indexed Instructions
- Vector Arithmetic Instructions:
  - Vector Integer Arithmetic Instructions
  - Vector Fixed-point Arithmetic Instructions
  - Vector Floating-point Arithmetic Instructions
  - Vector Reduction Instructions
  - Vector Mask Instructions
  - Vector Permutation Instructions

Vector Arithmetic Inst.

Formats for Vector Arithmetic Instructions under OP-V major opcode



funct3[2:0]			Category	Operands	Type of scalar operand
0	0	0	OPIVV	vector-vector	N/A
0	0	1	OPFVV	vector-vector	N/A
0	1	0	OPMVV	vector-vector	N/A
0	1	1	OPIVI	vector-immediate	`imm[4:0]`
1	0	0	OPIVX	vector-scalar	GPR x register `rs1`
1	0	1	OPFVF	vector-scalar	FP f register `rs1`
1	1	0	OPMVX	vector-scalar	GPR x register `rs1`
1	1	1	OPCFG	scalars-imms	GPR x register `rs1` & `rs2/imm`

Vector Integer Arithmetic Inst.

- Vector Single-Width Integer Add and Subtract
- Vector Widening Integer Add/Subtract
  The destination vector register group has EEW=2*SEW and EMUL=2*LMUL.
- Vector Integer Extension
  The vector integer extension instructions zero- or sign-extend a source vector integer operand with EEW less than SEW to fill SEW-sized elements in the destination.
- Vector Integer Add-with-Carry / Subtract-with-Borrow Instructions
  The carry inputs and outputs are represented using the mask register
- Vector Bitwise Logical Instructions
- Vector Single-Width Shift Instructions
- Vector Narrowing Integer Right Shift Instructions
  The narrowing right shifts extract a smaller field from a wider operand and have both zero-extending (srl) and sign- extending (sra) forms.
- Vector Integer Compare Instructions
  The following integer compare instructions write results to the destination mask register.
- Vector Integer Min/Max Instructions
  eg. vminu.vv vd, vs2, vs1, vm

Vector Integer Arithmetic Inst.

- Vector Single-Width Integer Multiply Instructions
  The single-width multiply instructions perform a SEW-bit*SEW-bit multiply to generate a 2*SEW-bit product, then return one half of the product in the SEW-bit-wide destination.
- Vector Integer Divide Instructions
- Vector Widening Integer Multiply Instructions
- Vector Single-Width Integer Multiply-Add Instructions

  eg.
  ```
  # Integer multiply-add, overwrite addend
  vmacc.vv vd, vs1, vs2, vm    # vd[i] = +(vs1[i] * vs2[i]) + vd[i]
  vmacc.vx vd, rs1, vs2, vm    # vd[i] = +(x[rs1] * vs2[i]) + vd[i]
  ```
- Vector Widening Integer Multiply-Add Instructions
- Vector Integer Merge Instructions

  eg.
  ```
  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 Integer Move Instructions
  The vector integer move instructions copy a source operand to a vector register group

Vector Fixed-point Arithmetic Inst.

- Vector Single-Width Saturating Add and Subtract
- Vector Single-Width Averaging Add and Subtract
  The averaging add and subtract instructions right shift the result by one bit and round off the result
- Vector Single-Width Fractional Multiply with Rounding and Saturation
  The signed fractional multiply instruction produces a 2*SEW product of the two SEW inputs, then shifts the result right by SEW-1 bits, rounding these bits, then saturates the result to fit into SEW bits
- Vector Single-Width Scaling Shift Instructions
  These instructions shift the input value right, and round off the shifted out bits
- Vector Narrowing Fixed-Point Clip Instruction
  The vnclip instructions are used to pack a fixed-point value into a narrower destination.

Vector Floating-point Arithmetic Inst.

- Vector Single-Width Floating-Point Add/Subtract Instructions
- Vector Widening Floating-Point Add/Subtract Instructions
- Vector Single-Width Floating-Point Multiply/Divide Instructions
- Vector Widening Floating-Point Multiply
- Vector Single-Width Floating-Point Fused Multiply-Add Instructions
- Vector Widening Floating-Point Fused Multiply-Add Instructions
- Vector Floating-Point Square-Root Instruction
- Vector Floating-Point Reciprocal Square-Root Estimate Instruction
  Returns an estimate of 1/sqrt(x) accurate to 7 bits.
- Vector Floating-Point Reciprocal Estimate Instruction
  Returns an estimate of 1/x accurate to 7 bits
- Vector Floating-Point MIN/MAX Instructions
- Vector Floating-Point Sign-Injection Instructions
  The result takes all bits from vs1/rs1 except the sign bit from the vector vs2 operands.
- Vector Floating-Point Compare Instructions
- Vector Floating-Point Classify Instruction
- Vector Floating-Point Merge Instruction
- Vector Floating-Point Move Instruction
- Single-Width Floating-Point/Integer Type-Convert Instructions
- Widening Floating-Point/Integer Type-Convert Instructions
- Narrowing Floating-Point/Integer Type-Convert Instructions

Vector Reduction Operation

Vector reduction operations take a vector register group of elements and a scalar held in element 0 of a vector register, and perform a reduction using some binary operator, to produce a scalar result in element 0 of a vector register.

- Vector Single-Width Integer Reduction Instructions

  ```
  vredsum.vs  vd, vs2, vs1, vm   # vd[0] =  sum( vs1[0] , vs2[*] )
  ```
  eg. `vredmaxu.vs vd, vs2, vs1, vm   # vd[0] = maxu( vs1[0] , vs2[*] )`
- Vector Widening Integer Reduction Instructions
- Vector Single-Width Floating-Point Reduction Instructions
  - Vector Ordered Single-Width Floating-Point Sum Reduction
    vd[0] = (((vs1[0] + vs2[0]) + vs2[1]) + ...) + vs2[vl-1]
  - Vector Unordered Single-Width Floating-Point Sum Reduction
  - Vector Single-Width Floating-Point Max and Min Reductions
- Vector Widening Floating-Point Reduction Instructions

Vector Mask Instructions

Vector Mask instructions are provided to help operate on mask values held in a vector register.

- Vector Mask-Register Logical Instructions
- Vector count population in mask vcpop.m

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

- Vfirst find-first-set mask bit
  The vfirst instruction finds the lowest-numbered active element of the source mask vector that has the value 1 and writes that element's index to a GPR.
- vmsbf.m set-before-first mask bit

  eg.

```
1 0 0 1 0 1 0 0   v3 contents
                  vmsbf.m v2, v3
0 0 0 0 0 0 1 1   v2 contents
```

- vmsif.m set-including-first mask bit
- vmsof.m set-only-first mask bit
- Vector Iota Instruction

  eg.

```
1 0 0 1 0 0 0 1   v2 contents
                  viota.m v4, v2 # Unmasked
2 2 2 1 1 1 1 0   v4 result
```

- Vector Element Index Instruction
  The vid.v instruction writes each element's index to the destination vector register group, from 0 to vl-1.

Vector Permutation Instructions

Vector Permutation instructions are provided to help to move elements around within the vector registers.

- Integer Scalar Move Instructions
  The integer scalar read/write instructions transfer a single value between a scalar x register and element 0 of a vector register.
- Floating-Point Scalar Move Instructions
- Vector Slide Instructions
  The slide instructions move elements up and down a vector register group.
  eg.
```
vslideup.vx vd, vs2, rs1, vm        # vd[i+rs1] = vs2[i]
vslideup.vi vd, vs2, uimm, vm       # vd[i+uimm] = vs2[i]
```
- Vector Register Gather Instructions
```
vrgather.vv vd, vs2, vs1, vm     # vd[i] = (vs1[i] >= VLMAX) ? 0 : vs2[vs1[i]];
vrgatherei16.vv vd, vs2, vs1, vm # vd[i] = (vs1[i] >= VLMAX) ? 0 : vs2[vs1[i]];
```
- Vector Compress Instruction
  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.
- Whole Vector Register Move
  Copy whole vector registers (i.e., all VLEN bits) and can copy whole vector register groups.