The RISC-V Vector ISA Update

Krste Asanovic, krste@berkeley.edu, Vector WG Chair

Roger Espasa, roger.espasa@esperantotech.com , Vector WG Co-Chair

Vector Extension Working Group

8th RISC-V Workshop, May, 2018

No, the spec is not ready

…. But getting really close

Updates & Discussion points

1. Register types moved to an extension
2. Widening multiplies
3. Debating whether reductions should be in base or not
4. Worked on overlaying V-reg and F-reg to save state – won't happen
5. Fixed point vclip instruction (not really new, but reporting out)
6. Mask support for speculative vectorization
7. Possibility to fit integer MADD within encoding

Register types pushed to extension

- Concerns raised on total state needed to hold reg types
- Register Type Extension will provide
  - 2D shapes
  - Associated polymorphic opcodes
- As a consequence
  - Vector instructions back to having classical types (see next slides)
  - Working on encoding everything within existing space
    - Looking good for no major opcodes being taken, but not yet fully closed
  - Scalar support still there, slightly different than before
  - Reducing mask encoding to single bit
  - Configuration information slightly simplified
    - vemaxw = holds the maximum element width across all vector
    - vregmax = holds the maximum vreg available

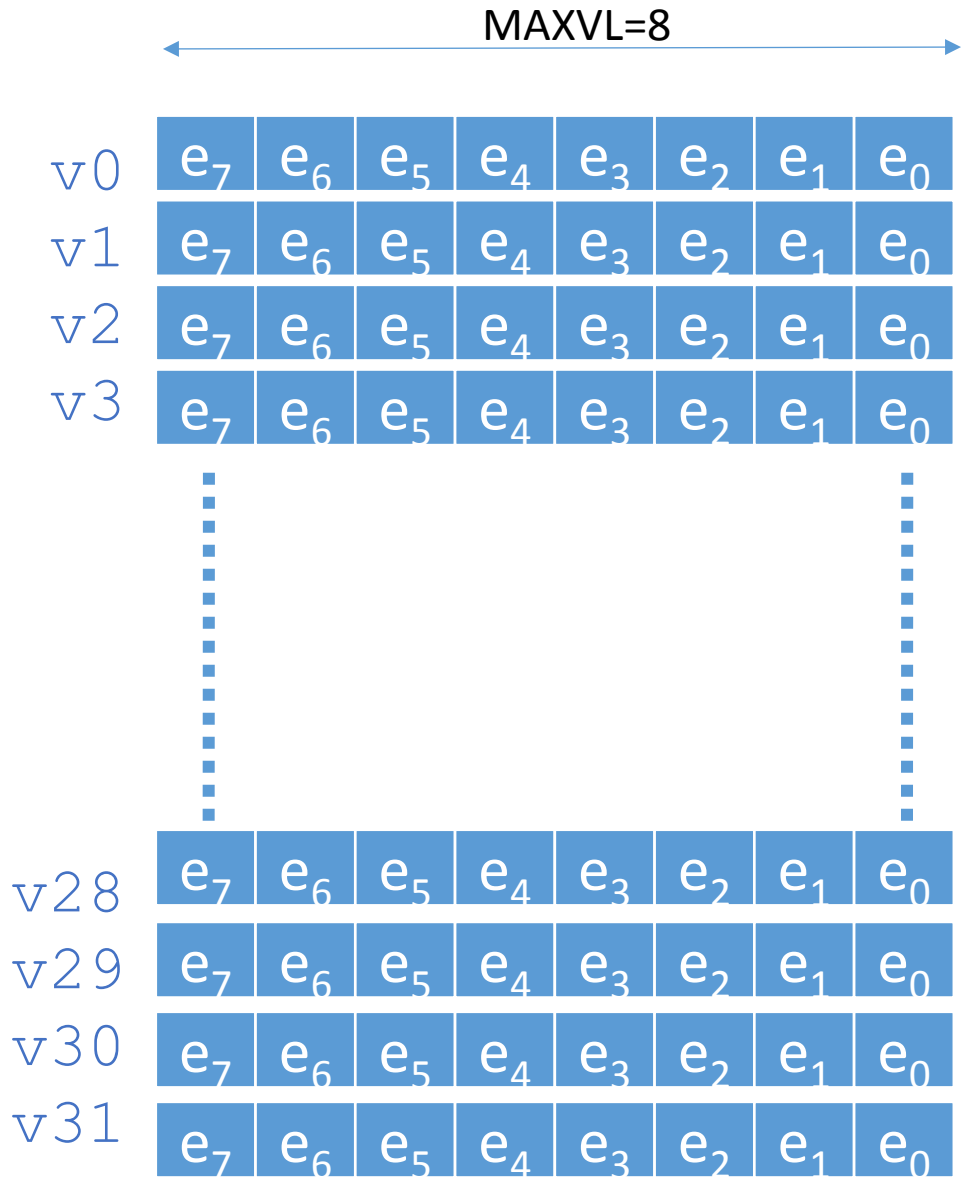
Scalar support

- The data inside a VREG can have 3 possible shapes:
  - A vector (i.e., what you'd expect)
  - A single scalar value (or, equivalently, the same value in all vector elements)
  - Future: A matrix (optional, not in the base spec)
- Base ISA only supports scalar and vector shape
- Encoding allows indicating that DEST vreg is a "scalar shape"
  - vop vd.s, vs1, vs2
  - vd[0..MAXVL] = vs1[0] op vs2[0]
- For vector loads/stores, new VLO and VSO operate on ONE scalar element
  - For VLO*: vd[0..MAXVL] = single value read from memory
  - For VSO*: write vs1[0] to memory (one single value written)

Masked execution

- Masks are stored in regular vector registers
  - The LSB of each element is used as a boolean “0” or “1” value
  - Other bits ignored
- Masks are computed with compare operations (vseq, vslt, vsltu)
  - veq v6, v7 → v1
  - Comparison results are integer “0” or “1”
  - Encoded with as many bits as the destination register element size
  - Other compare operations can be realized using the opposite mask encoding
- Instructions use 1 bit of encoding to select masked execution
  - 0 : operation unmasked
  - 1 : operation masked, execute only for elements where LSB(v0[i]) == 1

Architectural State



vl	(xlen)
vregmax	(8b)
vemaxw	(3b)
vtypeen	(1b)
vxrm	(2b)
vxcm	(1b)
fcsr.vxsat	(1b)

Note: Floating point flags use the existing scalar flags

Vector Memory Instructions

operation	instructions
vector load one	vlob, vlobu, vloh, vlohu, vlow, vlowu, vlod, vfloh, vflow, vflod
vector load	vlb, vlbu, vlh, vlhu, vlw, vlwu, vld, vflh, vflw, vfld
vector load, strided	vlsb, vlsbu, vlsh, vlshu, vlsw, vlswu, vlsd, vflsh, vflsw, vflsd
vector load, indexed (gather)	vlxb, vlxbu, vlxh, vlxhu, vlxw, vlxwu, vlxd, vflxh, vflxw, vflxd
vector store one	vsob, vsoh, vsow, vsod
vector store	vsb, vsh, vsw, vsd
vector store, strided	vssb, vssh, vssw, vssd
vector store, indexed (scatter)	vsxb, vsxh, vsxw, vsxd
vector store, indexed, unordered	vsxub, vsxuh, vsxuw, vsxud

Vector Integer Instructions

operation	instructions
add	vadd, vaddi, vaddw, vaddiw
subtract	vsub, vsubw
multiply	vmul, vmulh, vmulhsu, vmulhu
widening multiply	vmulwdn
divide	vdiv, vdivu, vrem, vremu
shift	vsll, vslli, vsra, vsrai, vsrl, vsrli
logical	vand, vandi, vor, vori, vxor, vxori
compare	vseq, vslt, vsltu
fixed point	vclipb, vclipbu, vcliph, vcliphu, vclipw, vclipwu

Widening multiplies

- vmulwdn vd, vs1, vs2
  - Signed multiply of the bottom (vemaxw/2)+1 bits from vs1 and vs2
  - Puts the low vemxaw bits of the result into vd
  - Including the extra bit, allows for signed/unsigned multiplies of vemaxw/2 bits to be supported without separate instructions or reduced precision.
  - Vector loads and clips can be used to extend narrower values correctly before using them in widening multiplies.

- Depending on encoding, we're considering fused integer mul-add including widening mul-add

Fixed Point Support: vclip

- vxrm (2b) holds rounding mode for vclip
  - trn (00): truncate
  - jam (01): OR bits into LSB
  - rne (10): round to nearest-even
  - rup (11) round-up (+0.5 LSB)
- vxcm (1b) holds clipping mode for vclip
  - Wraparound (0)
  - Saturate (1)
- fsr.vxsat (1b) holds sticky fixed-point saturation flag. Set if any vclip instruction causes saturation
- vclip.{type} vd, vs1, vs2
  - vs1 holds the value to be clipped
  - vs2 holds the shift amount to shift vs1 right (to round off low order bits)
  - type = b, bu, h, hu, w, wu

Vector Floating Point Instructions, w/ FP16

operation	instructions
add	vfadd.h, vfadd.s, vfadd.d
subtract	vfsub.h, vfsub.s, vfsub.d
multiply	vfmul.h, vfmul.s, vfmul.d
divide	vfdiv.h, vfdiv.s, vfdiv.d
sign	vfsgn{j,jn,jx}.h, vfsgn{j,jn,jx}.s, vfsgn{j,jn,jx}.d
max	vfmax.h, vfmax.s, vfmax.d
min	vfmin.h, vfmin.s, vfmin.d
compare	vfeq.h, vfeq.s, vfeq.d, vltq.h, vflt.s, vflt.d, vfle.h, vfle.s, vfle.d
sqrt	vfsqrt.h, vfsqrt.s, vfsqrt.d
class	vfclass.h, vflcass.s, vflcass.d

Vector Floating Point Multiply Add

operation	instructions
add	vfmadd.h, vfmadd.s, vfmadd.d
sub	vfmsub.h, vfmsub.s, vfmsub.d
widening add	vfmaddwdn.h, vfmaddwdn.s, vfmaddwdn.d
widening sub	vfmsubwdn.h, vfmsubwdn.s, vfmsubwdn.d

- WIP: it is likely that we can have the full set of fma, including ‘n’ versions within encoding

Vector Convert

From Integer to Float	
To Half	vfcvt.h.i, vfcvt.h.u
To Single	vfcvt.s.i, vfcvt.s.u
To Double	vfcvt.d.i, vfcvt.d.u
From Float to Vemaxw Integer	
To Signed	vfcvt.i.h, vfcvt.i.s, vfcvt.i.d
To Unsigned	vfcvt.u.h, vfcvt.u.s, vfcvt.u.d
From Float to Float	
To Half	vfcvt.h.s, vfcvt.h.d
To Single	vfcvt.s.h, vfcvt.s.d
To Double	vfcvt.d.h, vfcvt.d.s

From Vemaxw Int to Narrow Int	
To Byte	vcvt.{b,bu}.i
To Half	vcvt.{h,hu}.i
To Word	vcvt.{w,wu}.i

- Final convert list depending on encoding options. Could possibly be exactly the list that we have in base

Vector Data Movement

operation	instructions	action
insert gpr into vector	vins vd, rs1, rs2	vd[rs2] = rs1
insert fp into vector	vins vd, fs1, rs2	vd[rs2] = fs1
extract velem to gpr	vext rd, vs1, rs2	rd = vs1[rs2]
extract velem to fp	vext fd, vs1, rs2	fd = vs1[rs2]
vector-vector merge	vmerge vd, vs1, vs2, vm	mask picks src
vector-gpr merge	vmergex vd, rs1, vs2, vm	mask picks src
vector-fp merge	vmergef vd, fs1, vs2, vm	mask picks src
vector register gather	vrgather vd, vs1, vs2, vm	vd[i] = vs1[vs2[i]]
Gpr splat/bcast	vsplatx vd, rs1	Vd[0..MAXVL] = rs1
fpr splat/bcast	vsplatf vd, fs1	Vd[0..MAXVL] = fs1
vector slide down	vslidedwn vd, vs1, rs2, vm	vd[i] = vs1[rs2+i]
vector slide up	vslideup vd, vs1, rs2, vm	vd[rs2+i] = vs1[i]

Vector Mask Operations

operation	instructions
Find first set bit in mask	vmfirst rd, vs1
Mask pop count	vmpopc rd, vs1
Count preceding mask bits	vmiota vd, vm
Flag before first	vmfbf vd, vs1, vm
Flag including first	vmfif vd, vs1, vm

Vector Reductions

operation	INT	FP
add	vredsum	vfredsum.{h,s,d}
max	vredmax, vredmaxu	vfredmax.{h,s,d}
min	vredmin, vredminu	vfredmin.{h,s,d}
logical	vredand, vredor, vredxor	

- Debate ongoing whether to include in base or not

Encoding (WIP)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
imm4				m	0	0	imm5					rs1					1	s	s	vd						0x7						VFLO
imm4				m	0	1	0					rs1					1	s	s	vd						0x7						VFL
imm4				m	1	0	rs2					rs1					1	s	s	vd						0x7						VFLS
imm4				m	1	1	vs2					rs1					1	s	s	vd						0x7						VFLX
vs3					0	0	0					rs1					1	s	s	m	imm4					0x27						VSO
vs3					0	1	0					rs1					1	s	s	m	imm4					0x27						VS
vs3					1	0	rs2					rs1					1	s	s	m	imm4					0x27						VSS
vs3					1	1	vs2					rs1					1	s	s	m	imm4					0x27						VSX
vs3					1	1	vs2					00000					1	s	m	vd						0x27						VAMO
imm4				m	0	0	imm4				0	rs1					1	s	s	vd						0x2F						VLO
imm4				m	0	1	0					rs1					1	s	s	vd						0x2F						VL
imm4				m	1	0	rs2					rs1					1	s	s	vd						0x2F						VLS
imm4				m	1	1	vs2					rs1					1	s	s	vd						0x2F						VLX
vs3					0	0	vs2					vs1					1	!m	m	rd						0x43						VFMADD.S
vs3					0	0	vs2					vs1					1	!m	m	rd						0x47						VFMSUB.S
vs3					0	0	vs2					vs1					1	!m	m	rd						0x4B						VFNMSUB.S
vs3					0	0	vs2					vs1					1	!m	m	rd						0x4F						VFNMADD.S
vs3					0	1	vs2					vs1					1	!m	m	rd						0x43						VFMADD.D
vs3					0	1	vs2					vs1					1	!m	m	rd						0x47						VFMSUB.D
vs3					0	1	vs2					vs1					1	!m	m	rd						0x4B						VFNMSUB.D
vs3					0	1	vs2					vs1					1	!m	m	rd						0x4F						VFNMADD.D
vs3					1	0	vs2					vs1					1	!m	m	rd						0x43						VFMADD.H
vs3					1	0	vs2					vs1					1	!m	m	rd						0x47						VFMSUB.H
vs3					1	0	vs2					vs1					1	!m	m	rd						0x4B						VFNMSUB.H
vs3					1	0	vs2					vs1					1	!m	m	rd						0x4F						VFNMADD.H
f	f	f	f	m	1	0	rs2					rs1					sk	s	s	rd						0x53						VF-binary
1	1	1	1	m	1	0	ufunc					rs1					sk	s	s	rd						0x53						VF-unary
0	0	0	0	m	1	0	rs2					rs1					0	0	0	rd						0x33						VADD
0	1	0	0	m	1	0	rs2					rs1					0	0	0	rd						0x33						VSUB

Status

- Will be closing base spec very soon
- Want to see some compiler output before bringing for ratification
- Have not started on formal spec for vector isa subset

Thank You!