Mnemonic	Size	Operands	Short description	Equivalent C computation (i=0 to 3 for W, i=0 to 7 for B)
paddw	W	<VEA>,b,d	vector add short	d[i] = a[i] + b[i]
psubw	W	<VEA>,b,d	vector subtract short	d[i] = a[i] - b[i]
paddusw	W	<VEA>,b,d	vector add unsigned short with saturation	d[i] = min(0xffff, a[i] + b[i])
psubusw	W	<VEA>,b,d	vector subtract unsigned short with saturation	d[i] = min(0xffff, b[i] - a[i])
paddb	B	<VEA>,b,d	vector add bytes	d[i] = b[i] + a[i]
psubb	B	<VEA>,b,d	vector subtract bytes	d[i] = b[i] - a[i]
paddusb	B	<VEA>,b,d	vector add unsigned bytes with unsigned saturation	d[i] = max(0, (b[i] + a[i]))
psubusb	B	<VEA>,b,d	vector subtract unsigned bytes with unsigned saturation	d[i] = max(0, (b[i] - a[i]))
pmula	B	<VEA>,b,d	vector multiply and add unsigned bytes	d[i] = ((a[i] * b[i]) >> 8) + d[i]
pmull	W	<VEA>,b,d	vector multiply short and keep lower 16 Bit	d[i] = (a[i] * b[i]) & 0xffff
pmulh	W	<VEA>,b,d	vector multiply short and keep upper 16 Bit	d[i] = (a[i] * b[i]) >> 16
pmul88	W	<VEA>,b,d	vector multiply short and shift down 8	d[i] = (a[i] * b[i]) >> 8
packuswb	-	a,b,<VEA>	pack 2x4 signed shorts into 8 unsigned char, saturate to 0..255	d[i+0] = (a[i] < 0) ? 0 : (a[i] > 255) ? 255 : a[i] d[i+4] = (b[i] < 0) ? 0 : (b[i] > 255) ? 255 : b[i]
pack3216	-	a,b,<VEA>	pack 32 Bit ARGB data into 16 Bit RGB565	((a[1 + (i & 1) * 4)] & 0xf8) << 8) | ((a[2 + (i & 1) * 4)] & 0xfc) << 3) | ((a[3 + (i & 1) * 4)] & 0xf8) >> 3)
unpack1632	-	<VEA>,d:d+1	unpack 16 Bit RGB565 data into 32 Bit ARGB	d[i*4+0] = 0xff d[i*4+1] = ((a[i] >> 8) & 0xf8) | ((a[i] >> 13) & 0x7) d[i*4+2] = ((a[i] >> 3) & 0xfc) | ((a[i] >> 9) & 0x3) d[i*4+3] = ((a[i] << 3) & 0xf8) | ((a[i] >> 2) & 0x7)
vperm	W	#N,a,b,d	permute the contents of two registers into destination register	n = N[i >> 1] >> (i & 1) * 4 d[i] = (n < 8) ? a[n] : b[n-8]
transhi	-	a-d,e:f	matrix transposition, upper half	e[i] = a[(4 * i) + 0] f[i] = a[(4 * i) + 1]
translo	-	a-d,e:f	matrix transposition, upper half	e[i] = a[(4 * i) + 2] f[i] = a[(4 * i) + 3]
bflyw	-	<VEA>,b,d:d+1	butterfly operation, single-cycle vector short addition _and_ subtraction	d[i] = b[i] + a[i] f[i] = b[i] - a[i]
c2p	-	<VEA>,d	chunky to planar conversion, bit-wise transpose	for(j=0, t=0; j<8; j++) { t |= ((a[j] >> (7-i)) & 1) << (7-j) } d[i] = t
bsel	-	<VEA>,b,d	bit-by-bit selection of data from two sources into the destination	d = (d & (!b)) | (a & b)
pand	Q	<VEA>,b,d	AND 64 Bit logic operations	d = a & b
por	Q	<VEA>,b,d	OR 64 Bit logic operations	d = a | b
peor	Q	<VEA>,b,d	EOR 64 Bit logic operations	d = a ^ b

pandn	Q	<VEA>,b,d	ANDN 64 Bit logic operations	d = !a & b
pavgb	B	<VEA>,b,d	average 8 unsigned bytes with 8 unsigned bytes	d[i] = (a[i] + b[i] + 1) >> 1
load	Q	<VEA>,d	load 64 bit into destination register	d = src
loadi	Q	<VEA>,d	load 64 bit indirect into destination register	regfile[d] = src
store	Q	a,<VEA>	store 64 bit from source register in memory	dest = a
storei	Q	a,<VEA>	store 64 bit indirect source into memory location	dest = regfile[d]
storem	Q	a,m,<VEA>	store a selection of bytes from a into destination	d[i] = (m & (1<<(7-i))) ? a[i] : d[i]
storeilm	B	a,m,<VEA>	store bytes from a into destination, based on inverted long mask	d[i] = (m[i] & 128) ? d[i] : a[i]
storec	B	a,count,<VEA>	store at most "count" bytes from a into destination	if((count - i) > 0) d[i] = a[i]
pcmpeqb	B	<VEA>,b,d	byte-by-byte vector compare	d[i] = (bu[i] == au[i]) ? 0xff : 0x00 d[i] = (bs[i] == as[i]) ? 0xff : 0x00
pcmpgtb	B	<VEA>,b,d	byte-by-byte vector compare	d[i] = (bs[i] > as[i]) ? 0xff : 0x00
pcmpgeb	B	<VEA>,b,d	byte-by-byte vector compare	d[i] = (bs[i] >= as[i]) ? 0xff : 0x00
pcmphib	B	<VEA>,b,d	byte-by-byte vector compare	d[i] = (bu[i] > au[i]) ? 0xff : 0x00
pcmpeqw	W	<VEA>,b,d	short-by-short vector compare	d[i] = (bu[i] == au[i]) ? 0xffff : 0x0000 d[i] = (bs[i] == as[i]) ? 0xffff : 0x0000
pcmpgtw	W	<VEA>,b,d	short-by-short vector compare	d[i] = (bs[i] > as[i]) ? 0xffff : 0x0000
pcmpgew	W	<VEA>,b,d	short-by-short vector compare	d[i] = (bs[i] >= as[i]) ? 0xffff : 0x0000
pcmphiw	W	<VEA>,b,d	short-by-short vector compare	d[i] = (bu[i] > au[i]) ? 0xffff : 0x0000
pminub	B	<VEA>,b,d	byte-by-byte vector compare and obtain smaller	d[i] = (bu[i] < au[i]) ? bu[i] : au[i]
pminsb	B	<VEA>,b,d	byte-by-byte vector compare and obtain smaller	d[i] = (bs[i] < as[i]) ? bs[i] : as[i]
pmaxub	B	<VEA>,b,d	byte-by-byte vector compare and obtain larger	d[i] = (bu[i] > au[i]) ? bu[i] : au[i]
pmaxsb	B	<VEA>,b,d	byte-by-byte vector compare and obtain larger	d[i] = (bs[i] > as[i]) ? bs[i] : as[i]
pminuw	W	<VEA>,b,d	short-by-short vector compare and obtain smaller	d[i] = (bu[i] < au[i]) ? bu[i] : au[i]
pminsw	W	<VEA>,b,d	short-by-short vector compare and obtain smaller	d[i] = (bs[i] < as[i]) ? bs[i] : as[i]
pmaxuw	W	<VEA>,b,d	short-by-short vector compare and obtain larger	d[i] = (bu[i] > au[i]) ? bu[i] : au[i]
pmaxsw	W	<VEA>,b,d	short-by-short vector compare and obtain larger	d[i] = (bs[i] > as[i]) ? bs[i] : as[i]

register map - index numbers

00 - 07	D0 - D7
08 - 15	A0 - A7
16 - 23	B0 - B7
40 - 47	E0 - E7
48 - 55	E8 - E15
56 - 63	E16 - E23