#instructions (assumed to be single-cycle) per operation	x86 SSE4, AVX2										ppc (POWER8/9) w/ VSX										arm64 NEON										intersection										Number of instructions by type and platform
Operations ⬇ Types ➡	i8	u8	i16	u16	i32	u32	i64	u64	f32	f64	i8	u8	i16	u16	i32	u32	i64	u64	f32	f64	i8	u8	i16	u16	i32	u32	i64	u64	f32	f64	i8	u8	i16	u16	i32	u32	i64	u64	f32	f64	Instructions
ARITH																																									**ARITH**
add	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	add
sub	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	sub
pairwise add	9	9	1	1	1	1	9	9	1	1	9	9	9	9	9	9	9	9	9	9	1	1	1	1	1	1	1	1	1	1	9	9	9	9	9	9	9	9	9	9	pairwise add
pairwise sub	9	9	1	1	1	1	9	9	1	1	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	pairwise sub
saturated_add	1	1	1	1	9	9	9	9			1	1	1	1	1	1	9	9			1	1	1	1	1	1	1	1			1	1	1	1	9	9	9	9			saturated_add
saturated_sub	1	1	1	1	9	9	9	9			1	1	1	1	1	1	9	9			1	1	1	1	1	1	1	1			1	1	1	1	9	9	9	9			saturated_sub
average	2	1	2	1	2	2	2	2			1	1	1	1	1	1	8	8			1	1	1	1	1	1	1	1			2	1	2	1	2	2	8	8			average
shift left/right by constant bits	2	2	1	1	1	1	1	1			2	2	2	2	2	2	2	2			1	1	1	1	1	1	1	1			2	2	2	2	2	2	2	2			shift left/right by constant bits
bit-shift-right-var (independent lanes)	9	9	9	9	1	1	9	1			1	1	1	1	1	1	1	1			2	2	2	2	2	2	2	2			9	9	9	9	2	2	9	2			bit-shift-right-var
bit-shift-left-var (independent lanes)	9	9	9	9	1	1	1	1			1	1	1	1	1	1	1	1			1	1	1	1	1	1	1	1			9	9	9	9	1	1	1	1			bit-shift-left-var
abs	1		1		1		9		1	1	3		3		3		3		1	1	1		1		1		1		1	1	3		3		3		9		1	1	abs
negate	2		2		2		2		1	1	2		2		2		2		1	1	1		1		1		1		1	1	2		2		2		2		1	1	neg
min/max	1	1	1	1	1	1	2	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	3	1	1	min/max
mul_truncate (x86 mullo)	9	9	1	1	1	1	9	9			1	1	1	1	1	1	1	1			1	1	1	1	1	1	9	9			9	9	1	1	1	1	9	9			mul_truncate
mul_even (PPC mule)	9	9	9	9	1	1					1	1	1	1	1	1					2	2	2	2	2	2					9	9	9	9	2	2					mul_even
mul_fp									1	1									1	1									1	1									1	1	mul_fp
muladd_fp (3 variants with +/-)									1	1									1	1									1	1									1	1	muladd_fp (all 4 variants with +/-)
div_fp									1	1									1	1									1	1									1	1	div_fp
reciprocal_approx									1	9									1	1									1	1									1	9	reciprocal_approx
sqrt									1	1									1	1									1	1									1	1	sqrt
recip_sqrt_approx									1	9									1	1									1	1									1	9	recip_sqrt_approx
floor									1	1									1	1									1	1									1	1	floor
ceil									1	1									1	1									1	1									1	1	ceil
round									1	1									1	1									1	1									1	1	round
COMPARE																																									**COMPARE**
compare LT/GT	1	3	1	3	1	3	1	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3	1	3	1	3	1	3	1	1	compare LT/GT
compare LE/GE	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	1	1	compare LE/GE
compare ==	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	compare ==
compare !=	2	2	2	2	2	2	2	2	1	1	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	compare !=
compare entire register to 0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	compare entire register to 0
LOGICAL																																									**LOGICAL**
and	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	and
and(not a, b)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	and(not a, b)
or	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	or
xor	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	xor
bitwise NOT	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	bitwise NOT
movmskb (concat high bit of each byte)	1	1	2	2	1	1	1	1	1	1	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	movmskb
LOAD/STORE											Note: may be big-endian																														**LOAD/STORE**
load_aligned	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	load_all_aligned
load_unaligned	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	load_all_unaligned
load_64_unaligned	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	load_lower_half
load1_and_broadcast	2	2	2	2	2	2	2	2	1	1	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	1	1	load1_and_broadcast
store_aligned	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	store_all_aligned
store_unaligned	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	store_all_unaligned
store64_unaligned	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	store64_unaligned_lo
store_32	3	3	3	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3	3	3	3	1	1	1	1	1	1	store_lowest_aligned
stream (non-temporal write)	9	9	9	9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	9	9	9	9	1	1	1	1	1	1	stream (non-temporal write)
SWIZZLE																																									**SWIZZLE**
shift128 left/right by constant bytes	1	1	1	1	1	1	1	1			1	1	1	1	1	1	1	1			1	1	1	1	1	1	1	1			1	1	1	1	1	1	1	1			shift128 left/right by constant bytes
Shift 2x128 bit right in byte increments	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Shift 2x128 bit right in byte increments
broadcast any lane	9	9	9	9	1	1	1	1	1	1	9	9	9	9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	9	9	9	9	1	1	1	1	1	1	broadcast any lane
16-byte shuffle (var indices, >127 to zero)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	16-byte shuffle (var indices, >15 to zero)
Shuffle1032, 0321, 2103	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Shuffle1032, 0321, 2103
Interleave/zip = unpack	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Merge/zip = unpack
BlendV with full bit mask, not just MSB	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	BlendV with full bit mask, not just MSB
CONVERSION																																									**CONVERSION**
Expand to 2x width (u8->u16, f32->f64)	1	1	1	1	1	1			1		1	2	1	2	1	2			1		1	1	1	1	1	1			1		1	2	1	2	1	2			1		Expand to 2x width (u8->u16, f32->f64)
Reducing to half width (e.g. u16->u8)			1	1	1	1	9	9		1			1	1	1	1	1	1		1			1	1	1	1	1	1		1			1	1	1	1	9	9		1	Reducing to half width (e.g. u16->u8)
Convert integer -> same size real					1	9	9	9							1	1	1	1					1	1	1	1	1	1							1	9	9	9			Convert integer -> same size real
Convert real -> same size integer									1	9									1	1									1	1									1	9	Convert real -> same size integer
Extract lane 0 to reg/aligned mem	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Extract lane 0 to reg/aligned mem
Insert reg/aligned mem into lane 0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Insert reg/aligned mem into lane 0
CRYPTO/HASH																																									**CRYPTO/HASH**
SHA1						1										1										1										1					SHA1
SHA256						1										1										1										1					SHA256
AES		1										1										1										1									AES
CRC32C		1		1		1		1				3		3		3		3				1		1		1		1				3		3		3		3			CRC32C
CLMUL								1										1																							CLMUL
EMULATED																																									slow/emulate:
mulhi16			1										1	3									2																		mulhi16
horz_sum		1										5										3																			horz_sum