Cranelift IR and XiaoXuan Native Instructions Map

No.	Cranelift IR Instruction	Function	Description	Acc Vec	Acc FP	SIM D	XiaoXuan Native Instruction	Class
1	jump	fn jump(self, block_call_label: Block, block_call_args: &[Value]) -> Inst	Unconditionally jump to a basic block, passing the specified block arguments.					
2	brif	fn brif(self, c: Value, block_then_label: Block, block_then_args: &[Value], block_else_label: Block, block_else_args: &[Value]) -> Inst	Conditional branch when cond is non-zero. Take the then branch when c != 0, and the else branch otherwise.				if	Control Flow
3	br_table	fn br_table(self, x: Value, JT: JumpTable) -> Inst	Indirect branch via jump table.					
4	debugtrap	fn debugtrap(self) -> Inst	Encodes an assembly debug trap.					
5	trap	fn trap<T1: Into<TrapCode>>(self, code: T1) -> Inst	Terminate execution unconditionally.				panic debug unreachable	Machine
6	trapz	fn trapz<T1: Into<TrapCode>>(self, c: Value, code: T1) -> Inst	Trap when zero.					
7	resumable_trap	fn resumable_trap<T1: Into<TrapCode>>(self, code: T1) -> Inst	A resumable trap.					
8	trapnz	n trapnz<T1: Into<TrapCode>>(self, c: Value, code: T1) -> Inst	Trap when non-zero.					
9	resumable_trap	fn resumable_trapnz<T1: Into<TrapCode>>(self, c: Value, code: T1) -> Inst	A resumable trap to be called when the passed condition is non-zero.					
10	return_	fn return_(self, rvals: &[Value]) -> Inst	Return from the function.				return	Control Flow

11	call	fn call(self, FN: FuncRef, args: &[Value]) -> Inst	Direct function call.		call	Function Call
12	call_indirect	fn call_indirect(self, SIG: SigRef, callee: Value, args: &[Value]) -> Inst	Indirect function call.		dyncall	Function Call
13	return_call	fn return_call(self, FN: FuncRef, args: &[Value]) -> Inst	Direct tail call.		rerun	Control Flow
14	return_call_indirect	fn return_call_indirect(self,	Indirect tail call.			
15	func_addr	fn func_addr(self, iAddr: Type, FN: FuncRef) -> Value	Get the address of a function.		addr.function	Address
16	splat	fn splat(self, TxN: Type, x: Value) -> Value	Vector splat.		y	
17	swizzle	fn swizzle(self, x: Value, y: Value) -> Value	Vector swizzle.		y	
18	x86_pshufb	fn x86_pshufb(self, x: Value, y: Value) -> Value	A vector swizzle lookalike which has the semantics of pshufb on x64.		y	
19	insertlane	fn insertlane<T1: Into<Uimm8>>(self, x: Value, y: Value, Idx: T1) -> Value	Insert y as lane Idx in x.		y	
20	extractlane	fn extractlane<T1: Into<Uimm8>>(self, x: Value, Idx: T1) -> Value	Extract lane Idx from x.		y	
21	smin	fn smin(self, x: Value, y: Value) -> Value	Signed integer minimum.	y		
22	umin	fn umin(self, x: Value, y: Value) -> Value	Unsigned integer minimum.	y		
23	smax	fn smax(self, x: Value, y: Value) -> Value	Signed integer maximum.	y		
24	umax	fn umax(self, x: Value, y: Value) -> Value	Unsigned integer maximum.	y		
25	avg_round	fn avg_round(self, x: Value, y: Value) -> Value	Unsigned average with rounding		y	

26	uadd_sat	fn uadd_sat(self, x: Value, y: Value) -> Value	Add with unsigned saturation.	y		
27	sadd_sat	fn sadd_sat(self, x: Value, y: Value) -> Value	Add with signed saturation.	y		
28	usub_sat	fn usub_sat(self, x: Value, y: Value) -> Value	Subtract with unsigned saturation.	y		
29	ssub_sat	fn ssub_sat(self, x: Value, y: Value) -> Value	Subtract with signed saturation.	y		
30	load	fn load<T1: Into<MemFlags>, T2: Into<Offset32>>(self, Mem: Type, MemFlags: T1, p: Value, Offset: T2) -> Value	Load from memory at p + Offset.		{data,heap}.load64_i64 {data,heap}.load32_i32 {data,heap}.load64_i64 {data,heap}.load32_f32	Data Load Heap Load
31	store	fn store<T1: Into<MemFlags>, T2: Into<Offset32>>(self, MemFlags: T1, x: Value, p: Value, Offset: T2) -> Inst	Store x to memory at p + Offset.		{data,heap}.store64_i64 {data,heap}.store32_i32 {data,heap}.store64_f64 {data,heap}.store32_f32	Data Store Heap Store
32	uload8	fn uload8<T1: Into<MemFlags>, T2: Into<Offset32>>(self, iExt8: Type, MemFlags: T1, p: Value, Offset: T2) -> Value	Load 8 bits from memory at p + Offset and zero-extend. This is equivalent to load.i8 followed by uextend.		{data,heap}.load64_i8_u {data,heap}.load32_i8_u	Data Load Heap Load

33	sload8	fn sload8<T1: Into<MemFlags>, T2: Into<Offset32>>(self, iExt8: Type, MemFlags: T1, p: Value, Offset: T2) -> Value	Load 8 bits from memory at p + Offset and sign-extend. This is equivalent to load.i8 followed by sextend.	{data,heap}.load64_i8_s {data,heap}.load32_i8_s	Data Load Heap Load
34	istore8	fn istore8<T1: Into<MemFlags>, T2: Into<Offset32>>(self, MemFlags: T1, x: Value, p: Value, Offset: T2) -> Inst	Store the low 8 bits of x to memory at p + Offset.	{data,heap}.store8_i32	Data Store Heap Store
35	uload16	fn uload16<T1: Into<MemFlags>, T2: Into<Offset32>>(self, iExt16: Type, MemFlags: T1, p: Value, Offset: T2) -> Value	Load 16 bits from memory at p + Offset and zero-extend.	{data,heap}.load64_i16_u {data,heap}.load32_i16_u	Data Load Heap Load
36	sload16	fn sload16<T1: Into<MemFlags>, T2: Into<Offset32>>(self, iExt16: Type, MemFlags: T1, p: Value, Offset: T2) -> Value	Load 16 bits from memory at p + Offset and sign-extend.	{data,heap}.load64_i16_s {data,heap}.load32_i16_s	Data Load Heap Load

37	istore16	fn istore16<T1: Into<MemFlags>, T2: Into<Offset32>>(self, MemFlags: T1, x: Value, p: Value, Offset: T2) -> Inst	Store the low 16 bits of x to memory at p + Offset.	{data,heap}.store16_i32	Data Store Heap Store
38	uload32	fn uload32<T1: Into<MemFlags>, T2: Into<Offset32>>(self, MemFlags: T1, p: Value, Offset: T2) -> Value	Load 32 bits from memory at p + Offset and zero-extend.	{data,heap}.load64_i32_u	
39	sload32	fn sload32<T1: Into<MemFlags>, T2: Into<Offset32>>(self, MemFlags: T1, p: Value, Offset: T2) -> Value	Load 32 bits from memory at p + Offset and sign-extend.	{data,heap}.load64_i32_s	Data Load Heap Load
40	istore32	fn istore32<T1: Into<MemFlags>, T2: Into<Offset32>>(self, MemFlags: T1, x: Value, p: Value, Offset: T2) -> Inst	Store the low 32 bits of x to memory at p + Offset.	{data,heap}.store32_i64	Data Store Heap Store

41	uload8x8	fn uload8x8<T1: Into<MemFlags>, T2: Into<Offset32>>(self, MemFlags: T1, p: Value, Offset: T2) -> Value	Load an 8x8 vector (64 bits) from memory at p + Offset and zero-extend into an i16x8 vector.	y
42	sload8x8	fn sload8x8<T1: Into<MemFlags>, T2: Into<Offset32>>(self, MemFlags: T1, p: Value, Offset: T2) -> Value	Load an 8x8 vector (64 bits) from memory at p + Offset and sign-extend into an i16x8 vector.	y
43	uload16x4	fn uload16x4<T1: Into<MemFlags>, T2: Into<Offset32>>(self, MemFlags: T1, p: Value, Offset: T2) -> Value	Load a 16x4 vector (64 bits) from memory at p + Offset and zero-extend into an i32x4 vector.	y
44	sload16x4	fn sload16x4<T1: Into<MemFlags>, T2: Into<Offset32>>(self, MemFlags: T1, p: Value, Offset: T2) -> Value	Load a 16x4 vector (64 bits) from memory at p + Offset and sign-extend into an i32x4 vector.	y

45	uload32x2	fn uload32x2<T1: Into<MemFlags>, T2: Into<Offset32>>(self, MemFlags: T1, p: Value, Offset: T2) -> Value	Load an 32x2 vector (64 bits) from memory at p + Offset and zero-extend into an i64x2 vector.	y	
46	sload32x2	fn sload32x2<T1: Into<MemFlags>, T2: Into<Offset32>>(self, MemFlags: T1, p: Value, Offset: T2) -> Value	Load a 32x2 vector (64 bits) from memory at p + Offset and sign-extend into an i64x2 vector.	y	
47	stack_load	fn stack_load<T1: Into<Offset32>>(self, Mem: Type, SS: StackSlot, Offset: T1) -> Value	Load a value from a stack slot at the constant offset.	local.load*	Local Load
48	stack_store	fn stack_store<T1: Into<Offset32>>(self, x: Value, SS: StackSlot, Offset: T1) -> Inst	Store a value to a stack slot at a constant offset.	local.store*	Local Store
49	stack_addr	fn stack_addr<T1: Into<Offset32>>(self,	Get the address of a stack slot.	addr.local	Address
50	dynamic_stack_load	fn dynamic_stack_load(self, Mem: Type, DSS: DynamicStackSlot) -> Value	Load a value from a dynamic stack slot.		
51	dynamic_stack_store	fn dynamic_stack_store(self, x: Value, DSS: DynamicStackSlot) -> Inst	Store a value to a dynamic stack slot.		

52	dynamic_stack_addr	fn dynamic_stack_addr(self, iAddr: Type, DSS: DynamicStackSlot) -> Value	Get the address of a dynamic stack slot.			
53	global_value	fn global_value(self, Mem: Type, GV: GlobalValue) -> Value	Compute the value of global GV.			
54	symbol_value	fn symbol_value(self, Mem: Type, GV: GlobalValue) -> Value	Compute the value of global GV, which is a symbolic value.		addr.data	Address
55	tls_value	fn tls_value(self, Mem: Type, GV: GlobalValue) -> Value	Compute the value of global GV, which is a TLS (thread local storage) value.		addr.thread_local_data	Address
56	get_pinned_reg	fn get_pinned_reg(self, iAddr: Type) -> Value	Gets the content of the pinned register, when it’s enabled.			
57	set_pinned_reg	fn set_pinned_reg(self, addr: Value) -> Inst	Sets the content of the pinned register, when it’s enabled.			
58	get_frame_pointer	fn get_frame_pointer(self, iAddr: Type) -> Value	Get the address in the frame pointer register.		addr.fp	Address
59	get_stack_pointer	fn get_stack_pointer(self, iAddr: Type) -> Value	Get the address in the stack pointer register.		addr.sp	Address
60	get_return_address	fn get_return_address(self, iAddr: Type) -> Value	Get the PC where this function will transfer control to when it returns.		addr.return	Address
61	table_addr	fn table_addr<T1: Into<Offset32>>(self,	Bounds check and compute absolute address of a table entry.			
62	iconst	fn iconst<T1: Into<Imm64>>(self, NarrowInt: Type, N: T1) -> Value	Integer constant.		i32.imm i64.imm	Fundamental
63	f32const	fn f32const<T1: Into<Ieee32>>(self, N: T1) -> Value	Floating point constant.	y	f32.imm	Fundamental
64	f64const	fn f64const<T1: Into<Ieee64>>(self, N: T1) -> Value	Floating point constant.	y	f64.imm	Fundamental

65	vconst	fn vconst<T1: Into<Constant>>(self, TxN: Type, N: T1) -> Value	SIMD vector constant.	y		
66	shuffle	fn shuffle<T1: Into<Immediate>>(self, a: Value, b: Value, mask: T1) -> Value	SIMD vector shuffle.	y		
67	null	fn null(self, Ref: Type) -> Value	Null constant value for reference types.			
68	nop	fn nop(self) -> Inst	Just a dummy instruction.		nop	Fundamental
69	select	fn select(self, c: Value, x: Value, y: Value) -> Value	Conditional select.	y		
70	select_spectre_guarc	fn select_spectre_guard(self, c: Value, x: Value, y: Value) -> Value	Conditional select intended for Spectre guards.	y		
71	bitselect	fn bitselect(self, c: Value, x: Value, y: Value) -> Value	Conditional select of bits.	y		
72	x86_blendv	fn x86_blendv(self, c: Value, x: Value, y: Value) -> Value	except with the semantics of blendv-related instructions on x86.	y		
73	vany_true	fn vany_true(self, a: Value) -> Value	Reduce a vector to a scalar boolean.		y	
74	vall_true	fn vall_true(self, a: Value) -> Value	Reduce a vector to a scalar boolean.		y	
75	vhigh_bits	fn vhigh_bits(self, NarrowInt: Type, a: Value) -> Value	Reduce a vector to a scalar integer.		y	
76	icmp	n icmp<T1: Into<IntCC>>(self, Cond: T1, x: Value, y: Value) -> Value	Integer comparison.	y	{i32,i64}. {eq,ne,lt_{u,s},gt_{u,s},le_{ u,s},gt_{u,s}}	Compare
77	icmp_imm	fn icmp_imm<T1: Into<IntCC>, T2: Into<Imm64>>(	Compare scalar integer to a constant.		{i32,i64}.{eqz,nez}	Compare
78	iadd	fn iadd(self, x: Value, y: Value) -> Value	Wrapping integer addition	y	{i32,i64}.add	Arithmetic
79	isub	fn isub(self, x: Value, y: Value) -> Value	Wrapping integer subtraction	y	{i32,i64}.sub	Arithmetic

80	ineg	fn ineg(self, x: Value) -> Value	Integer negation	y	{i32,i64}.neg	Math
81	iabs	fn iabs(self, x: Value) -> Value	Integer absolute value with wrapping	y	{i32,i64}.abs	Math
82	imul	fn imul(self, x: Value, y: Value) -> Value	Wrapping integer multiplication	y	{i32,i64}.mul	Arithmetic
83	umulhi	fn umulhi(self, x: Value, y: Value) -> Value	Unsigned integer multiplication, producing the high half of a double-length result.	y	{i32,i64}.mul_hi_u	Arithmetic
84	smulhi	fn smulhi(self, x: Value, y: Value) -> Value	Signed integer multiplication, producing the high half of a double-length result.	y	{i32,i64}.mul_hi_s	Arithmetic
85	sqmul_round_sat	fn sqmul_round_sat(self, x: Value, y: Value) -> Value	numbers in the QN format, where N + 1 is the number bitwidth. A similar instruction to	y		
86	x86_pmulhrsw	fn x86_pmulhrsw(self, x: Value, y: Value) -> Value	sqmul_round_sat except with the semantics of x86's pmulhrsw instruction.	y		
87	udiv	fn udiv(self, x: Value, y: Value) -> Value	Unsigned integer division		{i32,i64}.div_u	Arithmetic
88	sdiv	fn sdiv(self, x: Value, y: Value) -> Value	Signed integer division rounded toward zero		{i32,i64}.div_s	Arithmetic
89	urem	fn urem(self, x: Value, y: Value) -> Value	Unsigned integer remainder.		{i32,i64}.rem_u	Arithmetic
90	srem	fn srem(self, x: Value, y: Value) -> Value	Signed integer remainder. The result has the sign of the dividend.		{i32,i64}.rem_s	Arithmetic
91	iadd_imm	fn iadd_imm<T1: Into<Imm64>>(self, x: Value, Y: T1) -> Value	Add immediate integer.		{i32,i64}.inc {i32,i64}.dec	Arithmetic
92	imul_imm	fn imul_imm<T1: Into<Imm64>>(self, x: Value, Y: T1) -> Value	Integer multiplication by immediate constant.			

93	udiv_imm	fn udiv_imm<T1: Into<Imm64>>(self, x: Value, Y: T1) -> Value	Unsigned integer division by an immediate constant.
94	sdiv_imm	fn sdiv_imm<T1: Into<Imm64>>(self, x: Value, Y: T1) -> Value	Signed integer division by an immediate constant.
95	urem_imm	fn urem_imm<T1: Into<Imm64>>(self, x: Value, Y: T1) -> Value	Unsigned integer remainder with immediate divisor.
96	srem_imm	fn srem_imm<T1: Into<Imm64>>(self, x: Value, Y: T1) -> Value	Signed integer remainder with immediate divisor.
97	irsub_imm	fn irsub_imm<T1: Into<Imm64>>(self, x: Value, Y: T1) -> Value	Immediate reverse wrapping subtraction
98	iadd_cin	fn iadd_cin(self, x: Value, y: Value, c_in: Value) -> Value	Add integers with carry in.
99	iadd_carry	fn iadd_carry(self, x: Value, y: Value, c_in: Value) -> (Value, Value)	Add integers with carry in and out.
100	uadd_overflow	fn uadd_overflow(self, x: Value, y: Value) -> (Value, Value)	Add integers unsigned with overflow out. of is set when the addition overflowed.
101	sadd_overflow	fn sadd_overflow(self, x: Value, y: Value) -> (Value, Value)	Add integers signed with overflow out. of is set when the addition over- or underflowed
102	usub_overflow	fn usub_overflow(self, x: Value, y: Value) -> (Value, Value)	Subtract integers unsigned with overflow out. of is set when the subtraction underflowed.
103	ssub_overflow	fn ssub_overflow(self, x: Value, y: Value) -> (Value, Value)	Subtract integers signed with overflow out. of is set when the subtraction over- or underflowed.

104	umul_overflow	fn umul_overflow(self, x: Value, y: Value) -> (Value, Value)	Multiply integers unsigned with overflow out. of is set when the multiplication overflowed.				
105	smul_overflow	fn smul_overflow(self, x: Value, y: Value) -> (Value, Value)	Overflow out. of is set when the multiplication over- or underflowed.				
106	uadd_overflow_trap	fn uadd_overflow_trap<T1: Into<TrapCode>>(self,	Unsigned addition of x and y, trapping if the result overflows.				
107	isub_bin	fn isub_bin(self, x: Value, y: Value, b_in: Value) -> Value	Subtract integers with borrow in.				
108	isub_borrow	fn isub_borrow(self, x: Value, y: Value, b_in: Value) -> (Value, Value)	Subtract integers with borrow in and out.				
109	band	fn band(self, x: Value, y: Value) -> Value	Bitwise and.	y	y	{i32,i64}.and	Bitwise
110	bor	fn bor(self, x: Value, y: Value) -> Value	Bitwise or.	y	y	{i32,i64}.or	Bitwise
111	bxor	fn bxor(self, x: Value, y: Value) -> Value	Bitwise xor.	y	y	{i32,i64}.xor	Bitwise
112	bnot	fn bnot(self, x: Value) -> Value	Bitwise not.	y	y	{i32,i64}.not	Bitwise
113	band_not	fn band_not(self, x: Value, y: Value) -> Value	Bitwise and not.	y	y		
114	bor_not	fn bor_not(self, x: Value, y: Value) -> Value	Bitwise or not.	y	y		
115	bxor_not	fn bxor_not(self, x: Value, y: Value) -> Value	Bitwise xor not.	y	y		
116	band_imm	fn band_imm<T1: Into<Imm64>>(self, x: Value, Y: T1) -> Value	Bitwise and with immediate.				
117	bor_imm	fn bor_imm<T1: Into<Imm64>>(self, x: Value, Y: T1) -> Value	Bitwise or with immediate.				

118	bxor_imm	fn bxor_imm<T1: Into<Imm64>>(self, x: Value, Y: T1) -> Value	Bitwise xor with immediate.			
119	rotl	fn rotl(self, x: Value, y: Value) -> Value	Rotate left.	y	{i32,i64}.rotate_left	Bitwise
120	rotr	fn rotr(self, x: Value, y: Value) -> Value	Rotate right.	y	{i32,i64}.rotate_right	Bitwise
121	rotl_imm	fn rotl_imm<T1: Into<Imm64>>(self, x: Value, Y: T1) -> Value	Rotate left by immediate.	y		
122	rotr_imm	fn rotr_imm<T1: Into<Imm64>>(self, x: Value, Y: T1) -> Value	Rotate right by immediate.	y		
123	ishl	fn ishl(self, x: Value, y: Value) -> Value	Integer shift left. Shift the bits in x towards the MSB by y places. Shift in zero bits to the LSB.	y	{i32,i64}.shift_left	Bitwise
124	ushr	fn ushr(self, x: Value, y: Value) -> Value	Unsigned shift right. Shift bits in x towards the LSB by y places, shifting in zero bits to the MSB. Also called a logical shift.	y	{i32,i64}.shift_right_u	Bitwise
125	sshr	fn sshr(self, x: Value, y: Value) -> Value	Signed shift right. Shift bits in x towards the LSB by y places, shifting in sign bits to the MSB. Also called an arithmetic shift.	y	{i32,i64}.shift_right_s	Bitwise
126	ishl_imm	fn ishl_imm<T1: Into<Imm64>>(self, x: Value, Y: T1) -> Value	Integer shift left by immediate.	y		
127	ushr_imm	fn ushr_imm<T1: Into<Imm64>>(self, x: Value, Y: T1) -> Value	Unsigned shift right by immediate.	y		
128	sshr_imm	fn sshr_imm<T1: Into<Imm64>>(self, x: Value, Y: T1) -> Value	Signed shift right by immediate.	y		
129	bitrev	fn bitrev(self, x: Value) -> Value	Reverse the bits of a integer.			
130	clz	fn clz(self, x: Value) -> Value	Count leading zero bits.		{i32,i64}.leading_zeros	Bitwise
131	cls	fn cls(self, x: Value) -> Value	Count leading sign bits.		{i32,i64}.leading_ones	Bitwise
132	ctz	fn ctz(self, x: Value) -> Value	Count trailing zeros.		{i32,i64}.trailing_zeros	Bitwise

133	bswap	fn bswap(self, x: Value) -> Value	Reverse the byte order of an integer.				
134	popcnt	fn popcnt(self, x: Value) -> Value	Population count			{i32,i64}.count_ones	Bitwise
135	fcmp	fn fcmp<T1: Into<FloatCC>>(self, Cond: T1, x: Value, y: Value) -> Value	Floating point comparison.	y	y	{f32,f64}.{eq,ne,lt,gt,le,ge}	Compare
136	fadd	fn fadd(self, x: Value, y: Value) -> Value	Floating point addition.	y	y	{f32,f64}.add	Arithmetic
137	fsub	fn fsub(self, x: Value, y: Value) -> Value	Floating point subtraction.	y	y	{f32,f64}.sub	Arithmetic
138	fmul	fn fmul(self, x: Value, y: Value) -> Value	Floating point multiplication.	y	y	{f32,f64}.mul	Arithmetic
139	fdiv	fn fdiv(self, x: Value, y: Value) -> Value	Floating point division.	y	y	{f32,f64}.div	Arithmetic
140	sqrt	fn sqrt(self, x: Value) -> Value	Floating point square root.	y	y	{f32,f64}.sqrt	Math
141	fma	fn fma(self, x: Value, y: Value, z: Value) -> Value	Floating point fused multiply-and-add.	y	y		
142	fneg	fn fneg(self, x: Value) -> Value	Floating point negation.	y	y	{f32,f64}.neg	Math
143	fabs	fn fabs(self, x: Value) -> Value	Floating point absolute value.	y	y	{f32,f64}.abs	Math
144	fcopysign	fn fcopysign(self, x: Value, y: Value) -> Value	Floating point copy sign.	y	y	{f32,f64}.copysign	Math
145	fmin	fn fmin(self, x: Value, y: Value) -> Value	Floating point minimum	y	y	{f32,f64}.min	Math
146	fmax	fn fmax(self, x: Value, y: Value) -> Value	Floating point maximum	y	y	{f32,f64}.max	Math
147	ceil	fn ceil(self, x: Value) -> Value	Round floating point round to integral, towards positive infinity.	y	y	{f32,f64}.ceil	Math
148	floor	fn floor(self, x: Value) -> Value	Round floating point round to integral, towards negative infinity.	y	y	{f32,f64}.floor	Math
149	trunc	fn trunc(self, x: Value) -> Value	Round floating point round to integral, towards zero.	y	y	{f32,f64}.trunc	Math

150	nearest	fn nearest(self, x: Value) -> Value	Round floating point round to integral, towards nearest with ties to even.	y	y	{f32,f64}.round_half_to_even	Math
151	is_null	fn is_null(self, x: Value) -> Value	Reference verification. The condition code determines if the reference type in question is null or not.				
152	is_invalid	fn is_invalid(self, x: Value) -> Value	Reference verification.				
153	bitcast	fn bitcast<T1: Into<MemFlags>>(self, MemTo: Type,	Reinterpret the bits in x as a different type.			[i32/i64/f32/f64].reinterpret_[f32/f64/i32/i64]	Convert
154	scalar_to_vector	fn scalar_to_vector(self, TxN: Type, s: Value) -> Value	Copies a scalar value to a vector value.		y		
155	bmask	fn bmask(self, IntTo: Type, x: Value) -> Value	Convert x to an integer mask. Non-zero maps to all 1s and zero maps to all 0s.				
156	ireduce	fn ireduce(self, Int: Type, x: Value) -> Value	Convert x to a smaller integer type by discarding the most significant bits.			i32.truncate_i64	Convert
157	snarrow	fn snarrow(self, x: Value, y: Value) -> Value	Combine x and y into a vector with twice the lanes but half the integer width while saturating overflowing values to the signed maximum and minimum.		y		
158	unarrow	fn unarrow(self, x: Value, y: Value) -> Value	Combine x and y into a vector with twice the lanes but half the integer width while saturating overflowing values to the		y		
159	uunarrow	fn uunarrow(self, x: Value, y: Value) -> Value	Combine x and y into a vector with twice the lanes but half the integer width while saturating overflowing values to the		y		

160	swiden_low	fn swiden_low(self, x: Value) -> Value	Widen the low lanes of x using signed extension.		y		
161	swiden_high	fn swiden_high(self, x: Value) -> Value	Widen the high lanes of x using signed extension.		y		
162	uwiden_low	fn uwiden_low(self, x: Value) -> Value	Widen the low lanes of x using unsigned extension.		y		
163	uwiden_high	fn uwiden_high(self, x: Value) -> Value	Widen the high lanes of x using unsigned extension.		y		
164	iadd_pairwise	fn iadd_pairwise(self, x: Value, y: Value) -> Value	addition on two operands, putting the combined results into a single vector result.		y		
165	x86_pmaddubsw	fn x86_pmaddubsw(self, x: Value, y: Value) -> Value	An instruction with equivalent semantics to pmaddubsw on x86.		y		
166	uextend	fn uextend(self, Int: Type, x: Value) -> Value	Convert x to a larger integer type by zero-extending.			i64.extend_i32_u	Convert
167	sextend	fn sextend(self, Int: Type, x: Value) -> Value	Convert x to a larger integer type by sign-extending.			i64.extend_i32_s	Convert
168	fpromote	fn fpromote(self, FloatScalar: Type, x: Value) -> Value	Convert x to a larger floating point format.	y		f64.promote_f32	Convert
169	fdemote	fn fdemote(self, FloatScalar: Type, x: Value) -> Value	Convert x to a smaller floating point format.	y		f32.demote_f64	Convert
170	fvdemote	fn fvdemote(self, x: Value) -> Value	Convert x to a smaller floating point format.	y	y		
171	fvpromote_low	fn fvpromote_low(self, a: Value) -> Value	Converts packed single precision floating point to packed double precision floating point.	y	y		
172	fcvt_to_uint	fn fcvt_to_uint(self, IntTo: Type, x: Value) -> Value	Converts floating point scalars to unsigned integer.	y		i32.convert_f32_u i32.convert_f64_u i64.convert_f32_u i64.convert_f64_u	Convert

173	fcvt_to_sint	fn fcvt_to_sint(self, IntTo: Type, x: Value) -> Value	Converts floating point scalars to signed integer.		y	i32.convert_f32_s i32.convert_f64_s i64.convert_f32_s i64.convert_f64_s	Convert
174	fcvt_to_uint_sat	fn fcvt_to_uint_sat(self, IntTo: Type, x: Value) -> Value	Convert floating point to unsigned integer as fcvt_to_uint does, but saturates the input instead of trapping.	y	y	i32.sat_convert_f32_u i32.sat_convert_f64_u i64.sat_convert_f32_u i64.sat_convert_f64_u	Convert
175	fcvt_to_sint_sat	fn fcvt_to_sint_sat(self, IntTo: Type, x: Value) -> Value	Convert floating point to signed integer as fcvt_to_sint does, but saturates the input instead of trapping.	y	y	i32.sat_convert_f32_s i32.sat_convert_f64_s i64.sat_convert_f32_s i64.sat_convert_f64_s	Convert
176	x86_cvtt2dq	fn x86_cvtt2dq(self, IntTo: Type, x: Value) -> Value	A float-to-integer conversion instruction for vectors-of-floats which has the same semantics as cvttp{s,d}2dq on x86. This	y	y		
177	fcvt_from_uint	fn fcvt_from_uint(self, FloatTo: Type, x: Value) -> Value	Convert unsigned integer to floating point.	y		f32.convert_i32_u f32.convert_i64_u	Convert
178	fcvt_from_sint	fn fcvt_from_sint(self, FloatTo: Type, x: Value) -> Value	Convert signed integer to floating point.	y		f32.convert_i32_s f32.convert_i64_s f64.convert_i32_s f64.convert_i64_s	Convert
179	isplit	fn isplit(self, x: Value) -> (Value, Value)	Split an integer into low and high parts.				
180	iconcat	fn iconcat(self, lo: Value, hi: Value) -> Value	Concatenate low and high bits to form a larger integer type.				

181	atomic_rmw	fn atomic_rmw<T1: Into<MemFlags>, T2: Into<AtomicRmwOp>>(self, AtomicMem: Type, MemFlags: T1, AtomicRmwOp: T2, p: Value,	Atomically read-modify-write memory at p, with second operand x. The old value is returned.	{i32,i64}.atomic_rmw_{add, sub,and,or,xor,exchage}	Atomic
182	atomic_cas	fn atomic_cas<T1: Into<MemFlags>>(self, MemFlags: T1, p: Value, e: Value, x: Value) -> Value	Perform an atomic compare-and-swap operation on memory at p, with expected value e, storing x if the value at p equals e. The old value at p is returned, regardless of whether the operation succeeds or fails.	{i32,i64}.atomic_cas	Atomic
183	atomic_load	fn atomic_load<T1: Into<MemFlags>>(self, AtomicMem: Type, MemFlags: T1, p: Value) -> Value	Atomically load from memory at p.		
184	atomic_store	fn atomic_store<T1: Into<MemFlags>>(self,	Atomically store x to memory at p.		
185	fence	fn fence(self) -> Inst	A memory fence. This must provide ordering to ensure that, at a minimum, neither loads nor stores of any kind may move		
186	extract_vector	fn extract_vector<T1: Into<Uimm8>>(self, x: Value, y: T1) -> Value	Return a fixed length sub vector, extracted from a dynamic vector.		

Cranelift IR Instructions: https://docs.rs/cranelift-codegen/latest/cranelift_codegen/ir/trait.InstBuilder.html
Linux Standard Base: https://refspecs.linuxbase.org/lsb.shtml
ELF Special Sections: https://refspecs.linuxbase.org/LSB_5.0.0/LSB-Core-generic/LSB-Core-generic/specialsectio
System V ABI Update: http://www.sco.com/developers/gabi/2003-12-17/contents.html

note

icmp

Signed	Unsigned	Condition
eq	eq	Equal
ne	ne	Not equal
slt	ult	Less than
sge	uge	Greater than or equal
sgt	ugt	Greater than
sle	ule	Less than or equal

fcmp

C	`Cond`	Subset
`==`	eq	EQ
`!=`	ne	UN | LT | GT
`<`	lt	LT
`<=`	le	LT | EQ
`>`	gt	GT
`>=`	ge	GT | EQ

ɔns.html