PTO ISA Cheat Sheet

Tile Instructions

Opcode	PTO-AS form	Description

GETVAL	`%dst = getval %src, %offset : !pto.tile<...> -> T`
	`%dst = getval %src, offset:uimm : !pto.tile<...> -> T`
		Read a single tile element into a scalar value.

MGATHER	`%dst = mgather %mem, %idx : !pto.memref<...>, !pto.tile<...> -> !pto.tile<...>`	Gather-load elements from global memory into a tile using per-element indices.
MSCATTER	`mscatter %src, %mem, %idx : !pto.memref<...>, !pto.tile<...>, !pto.tile<...>`	Scatter-store elements from a tile into global memory using per-element indices.

SETVAL	`setval %dst, %offset, %val : !pto.tile<...>`
	`setval %dst, offset:uimm, %val : !pto.tile<...>`
		Write a scalar value into a single tile element.

TABS	`%dst = tabs %src : !pto.tile<...> -> !pto.tile<...>`	Elementwise absolute value of a tile.
TADD	`%dst = tadd %src0, %src1 : !pto.tile<...>`	Elementwise add of two tiles.
TADDC	`%dst = taddc %src0, %src1, %src2 : !pto.tile<...>`	Elementwise ternary add: `src0 + src1 + src2`.
TADDS	`%dst = tadds %src, %scalar : !pto.tile<...>, f32`	Elementwise add a scalar to a tile.
TADDSC	`%dst = taddsc %src0, %scalar, %src1 : !pto.tile<...>, f32, !pto.tile<...>`	Elementwise fused add with scalar and a second tile: `src0 + scalar + src1`.
TAND	`%dst = tand %src0, %src1 : !pto.tile<...>`	Elementwise bitwise AND of two tiles.
TANDS	`%dst = tands %src, %scalar : !pto.tile<...>, i32`	Elementwise bitwise AND of a tile and a scalar.
TASSIGN	`tassign %tile, %addr : !pto.tile<...>, index`	Bind a Tile object to an implementation-defined on-chip address (manual placement).
TCI	`%dst = tci %S {descending = false} : !pto.tile<...>`	Generate a contiguous integer sequence into a destination tile.
TCMP	`%dst = tcmp %src0, %src1 {cmpMode = #pto.cmp<EQ>} : !pto.tile<...> -> !pto.tile<...>`	Compare two tiles and write a packed predicate mask.
TCMPS	`%dst = tcmps %src, %scalar {cmpMode = #pto.cmp<EQ>} : !pto.tile<...> -> !pto.tile<...>`	Compare a tile against a scalar and write per-element comparison results.
TCOLEXPAND	`%dst = tcolexpand %src : !pto.tile<...> -> !pto.tile<...>`	Broadcast the first element of each source column across the destination column.
TCOLMAX	`%dst = tcolmax %src : !pto.tile<...> -> !pto.tile<...>`	Reduce each column by taking the maximum across rows.
TCOLMIN	`%dst = tcolmin %src : !pto.tile<...> -> !pto.tile<...>`	Reduce each column by taking the minimum across rows.
TCOLSUM	`%dst = tcolsum %src {isBinary = false} : !pto.tile<...> -> !pto.tile<...>`	Reduce each column by summing across rows.
TCVT	`%dst = tcvt %src {rmode = #pto.round_mode<CAST_RINT>} : !pto.tile<...> -> !pto.tile<...>`	Elementwise type conversion with a specified rounding mode.

Opcode	PTO-AS form	Description
TDIV	%dst = tdiv %src0, %src1 : !pto.tile<...>	Elementwise division of two tiles.
TDIVS	%dst = tdivs %src, %scalar : !pto.tile<...>, f32	Elementwise division with a scalar (tile/scalar or scalar/tile).
TEXP	%dst = texp %src : !pto.tile<...>	Elementwise exponential.
TEXPANDS	%dst = texpands %scalar : f32, !pto.tile<...>	Broadcast a scalar into a destination tile.
TEXTRACT	%dst = textract %src[%r0, %r1] : !pto.tile<...> -> !pto.tile<...>	Extract a sub-tile from a source tile.
TGATHER	%dst = tgather %src0, %indices : !pto.tile<...> -> !pto.tile<...>	Gather/select elements using either an index tile or a compile-time mask pattern.
TGATHERB	%dst = tgatherb %src, %offsets : !pto.tile<...> -> !pto.tile<...>	Gather elements using byte offsets.
TLOAD	%t0 = tload %sv[%c0, %c0] : (!pto.memref<...>, index, index) -> !pto.tile<...>	Load data from a GlobalTensor (GM) into a Tile.
TLOG	%dst = tlog %src : !pto.tile<...>	Elementwise natural logarithm of a tile.
TLRELU	%dst = tlrelu %src, %slope : !pto.tile<...>, f32	Leaky ReLU with a scalar slope.
TMATMUL	%acc = tmatmul %a, %b : (!pto.tile<...>, !pto.tile<...>) -> !pto.tile<...>	Matrix multiply (GEMM) producing an accumulator/output tile.
TMATMUL_ACC	%acc1 = tmatmul.acc %acc0, %a, %b : (!pto.tile<...>, !pto.tile<...>, !pto.tile<...>) -> !pto.tile<...>	Matrix multiply with accumulator input (fused accumulate).
TMATMUL_BIAS	%acc = tmatmul.bias %a, %b, %bias : (!pto.tile<...>, !pto.tile<...>, !pto.tile<...>) -> !pto.tile<...>	Matrix multiply with bias add.
TMATMUL_MX	%c = tmatmul.mx %a, %a_scale, %b, %b_scale : (!pto.tile<...>, !pto.tile<...>, !pto.tile<...>, !pto.tile<...>) -> !pto.tile<...> %c_out = tmatmul.mx.acc %c_in, %a, %a_scale, %b, %b_scale : (!pto.tile<...>, !pto.tile<...>, !pto.tile<...>, !pto.tile<...>, !pto.tile<...>) -> !pto.tile<...> %c = tmatmul.mx.bias %a, %a_scale, %b, %b_scale, %bias : (!pto.tile<...>, !pto.tile<...>, !pto.tile<...>, !pto.tile<...>, !pto.tile<...>) -> !pto.tile<...>	Matrix multiply (GEMM) with additional scaling tiles for mixed-precision / quantized matmul on supported targets.
TMAX	%dst = tmax %src0, %src1 : !pto.tile<...>	Elementwise maximum of two tiles.
TMAXS	%dst = tmaxs %src, %scalar : !pto.tile<...>, f32	Elementwise max of a tile and a scalar: `max(src, scalar)`.
TMIN	%dst = tmin %src0, %src1 : !pto.tile<...>	Elementwise minimum of two tiles.
TMINS	%dst = tmins %src, %scalar : !pto.tile<...>, f32	Elementwise minimum of a tile and a scalar.
TMOV	%left = tmov.m2l %mat : !pto.tile<...> -> !pto.tile<...> %right = tmov.m2r %mat : !pto.tile<...> -> !pto.tile<...> %bias = tmov.m2b %mat : !pto.tile<...> -> !pto.tile<...> %scale = tmov.m2s %mat : !pto.tile<...> -> !pto.tile<...> %vec = tmov.a2v %acc : !pto.tile<...> -> !pto.tile<...> %v1 = tmov.v2v %v0 : !pto.tile<...> -> !pto.tile<...>	Move/copy between tiles, optionally applying implementation-defined conversion modes.
TMOV_FP	%dst = tmov.fp %src, %fp : !pto.tile<...>, !pto.tile<...> -> !pto.tile<...>	Move/convert from an accumulator tile into a destination tile, using a scaling (`fp`) tile for vector quantization parameters.
TMRGSORT	%dst, %executed = tmrgsort %src0, %src1 {exhausted = false} : !pto.tile<...>, !pto.tile<...> -> (!pto.tile<...>, vector<4xi16>)	Merge sort for multiple sorted lists (implementation-defined element format and layout).
TMUL	%dst = tmul %src0, %src1 : !pto.tile<...>	Elementwise multiply of two tiles.
TMULS	%dst = tmuls %src, %scalar : !pto.tile<...>, f32	Elementwise multiply a tile by a scalar.
TNEG	%dst = tneg %src : !pto.tile<...>	Elementwise negation of a tile.
TNOT	%dst = tnot %src : !pto.tile<...>	Elementwise bitwise NOT of a tile.
TOR	%dst = tor %src0, %src1 : !pto.tile<...>	Elementwise bitwise OR of two tiles.
TORS	%dst = tors %src, %scalar : !pto.tile<...>, i32	Elementwise bitwise OR of a tile and a scalar.

Opcode	PTO-AS form	Description
TPARTADD	%dst = tpartadd %src0, %src1 : !pto.tile<...> -> !pto.tile<...>	Partial elementwise add with implementation-defined handling of mismatched valid regions.
TPARTMAX	%dst = tpartmax %src0, %src1 : !pto.tile<...> -> !pto.tile<...>	Partial elementwise max with implementation-defined handling of mismatched valid regions.
TPARTMIN	%dst = tpartmin %src0, %src1 : !pto.tile<...> -> !pto.tile<...>	Partial elementwise min with implementation-defined handling of mismatched valid regions.
TPRELU	%dst = tprelu %src0, %src1 : !pto.tile<...>	Elementwise PReLU (parametric ReLU) with a per-element slope tile.
TRECIP	%dst = trecip %src : !pto.tile<...>	Elementwise reciprocal of a tile.
TRELU	%dst = trelu %src : !pto.tile<...>	Elementwise ReLU of a tile.
TREM	%dst = trem %src0, %src1 : !pto.tile<...>	Elementwise remainder of two tiles.
TREMS	%dst = trems %src, %scalar : !pto.tile<...>, f32	Elementwise remainder with a scalar: `fmod(src, scalar)` (or `%` for integers).
TRESHAPE	%dst = treshape %src : !pto.tile<...>	Reinterpret a tile as another tile type/shape while preserving the underlying bytes.
TROWEXPAND	%dst = trowexpand %src : !pto.tile<...> -> !pto.tile<...>	Broadcast the first element of each source row across the destination row.
TROWEXPANDDIV	%dst = trowexpanddiv %src0, %src1 : !pto.tile<...>, !pto.tile<...> -> !pto.tile<...>	Row-wise broadcast divide: divide each row of `src0` by a per-row scalar vector `src1`.
TROWEXPANDMUL	%dst = trowexpandmul %src0, %src1 : !pto.tile<...>, !pto.tile<...> -> !pto.tile<...>	Row-wise broadcast multiply: multiply each row of `src0` by a per-row scalar vector `src1`.
TROWEXPANDSUB	%dst = trowexpandsub %src0, %src1 : !pto.tile<...>, !pto.tile<...> -> !pto.tile<...>	Row-wise broadcast subtract: subtract a per-row scalar vector `src1` from each row of `src0`.
TROWMAX	%dst = trowmax %src : !pto.tile<...> -> !pto.tile<...>	Reduce each row by taking the maximum across columns.
TROWMIN	%dst = trowmin %src : !pto.tile<...> -> !pto.tile<...>	Reduce each row by taking the minimum across columns.
TROWSUM	%dst = trowsum %src : !pto.tile<...> -> !pto.tile<...>	Reduce each row by summing across columns.
TRSQRT	%dst = trsqrt %src : !pto.tile<...>	Elementwise reciprocal square root.
TSCATTER	%dst = tscatter %src, %idx : !pto.tile<...>, !pto.tile<...> -> !pto.tile<...>	Scatter rows of a source tile into a destination tile using per-element row indices.
TSEL	%dst = tsel %mask, %src0, %src1 : !pto.tile<...>	Select between two tiles using a mask tile (per-element selection).
TSELS	%dst = tsels %src0, %src1, %selectMode : !pto.tile<...>	Select one of two source tiles using a scalar `selectMode` (global select).
TSHL	%dst = tshl %src0, %src1 : !pto.tile<...>	Elementwise shift-left of two tiles.
TSHR	%dst = tshr %src0, %src1 : !pto.tile<...>	Elementwise shift-right of two tiles.
TSORT32	%dst, %idx = tsort32 %src : !pto.tile<...> -> (!pto.tile<...>, !pto.tile<...>)	Sort a fixed-size 32-element block and produce an index mapping.
TSQRT	%dst = tsqrt %src : !pto.tile<...>	Elementwise square root.
TSTORE	tstore %t1, %sv_out[%c0, %c0]	Store data from a Tile into a GlobalTensor (GM), optionally using atomic write or quantization parameters.
TSTORE_FP	tstore.fp %src, %fp, %sv_out[%c0, %c0]	Store an accumulator tile into global memory using a scaling (`fp`) tile for vector quantization parameters.
TSUB	%dst = tsub %src0, %src1 : !pto.tile<...>	Elementwise subtract of two tiles.
TSUBC	%dst = tsubc %src0, %src1, %src2 : !pto.tile<...>	Elementwise ternary op: `src0 - src1 + src2`.
TSUBS	%dst = tsubs %src, %scalar : !pto.tile<...>, f32	Elementwise subtract a scalar from a tile.

Opcode	PTO-AS form	Description
TSUBSC	`%dst = tsubsc %src0, %scalar, %src1 : !pto.tile<...>, f32, !pto.tile<...>`	Elementwise fused op: `src0 - scalar + src1`.
TSYNC	`tsync %e0, %e1 : !pto.event<...>, !pto.event<...>`	Synchronize PTO execution:
TTRANS	`%dst = ttrans %src : !pto.tile<...> -> !pto.tile<...>`	Transpose with an implementation-defined temporary tile.
TXOR	`%dst = txor %src0, %src1 : !pto.tile<...>`	Elementwise bitwise XOR of two tiles.
TXORS	`%dst = txors %src, %scalar : !pto.tile<...>, i32`	Elementwise bitwise XOR of a tile and a scalar.

Scalar Instructions

Opcode	Forms	Description
ABS	`ABS %dst:T, %src0:T`	Compute absolute value.

ADD	`ADD %dst:T, %src0:T, %src1:T`
	`ADD %dst:T, %src0:T, simm`
		Add two scalar values (or a scalar and a signed immediate).

AND	`AND %dst:T, %src0:T, %src1:T`
	`AND %dst:T, %src0:T, uimm`
		Bitwise AND.

ATOMIC_CAS	`ATOMIC_CAS %old:MemT, %m:memref<gm,MemT>, %idx:idx, off:simm, %exp:MemT, %new:MemT, mo:uimm`	Atomic compare-and-swap (CAS) on global memory.
ATOMIC_RMW	`ATOMIC_RMW %old:MemT, %m:memref<gm,MemT>, %idx:idx, off:simm, %val:MemT, op:uimm, mo:uimm`	Unified atomic read-modify-write (RMW) on global memory.
BCOND	`BCOND %cond:u1, t_label:uimm, f_label:uimm`	Conditional branch based on a predicate.
BITEXTR	`BITEXTR %dst:T, %src0:T, lsb:uimm, width:uimm`	Extract a bitfield from a scalar value and extend it to the destination type.
BR	`BR label:uimm`	Unconditional branch.
BREAK	`BREAK`	Break out of the nearest enclosing structured construct.

CALL	`CALL %target:idx`
	`CALL addr:uimm`
		Call a function target (unstructured control flow).

CASE	`CASE imm`	Introduce a case label within a `SWITCH`.
CLZ	`CLZ %dst:u32/u64, %src0:T`	Count leading zeros in the binary representation of an integer.
CMP	`CMP %dst:u1, %src0:T, %src1:T, cc`	Compare two scalar values and produce a predicate (`u1`).
CONTINUE	`CONTINUE`	Continue the nearest enclosing loop.
CTZ	`CTZ %dst:u32/u64, %src0:T`	Count trailing zeros in the binary representation of an integer.
CVT	`CVT %dst:DstT, %src0:SrcT`	Convert a scalar value between types.
DEFAULT	`DEFAULT`	Introduce the default label within a `SWITCH`.
DIV	`DIV %dst:T, %src0:T, %src1:T`	Divide two scalar values.
DO	`DO`	Separate condition-region from body-region in a `WHILE`.
ELSE	`ELSE`	Begin the else-region of an `IF`.
ENDFOR	`ENDFOR`	End a structured for-loop.
ENDIF	`ENDIF`	End an `IF` construct.
ENDSWITCH	`ENDSWITCH`	End a `SWITCH` construct.

Opcode	Forms	Description

ENDWHILE	`ENDWHILE`
	`ENDWHILE %cond:u1`
		End a structured while-loop.

FENCE	`FENCE scope:uimm, mode:uimm`	Memory fence/barrier.
FOR	`FOR %iv:idx, %lb:idx, %ub:idx, %step:idx`	Begin a structured for-loop.

IF	`IF %cond:u1`
	`IF (%out0:T0, ...), %cond:u1, (%in0:T0, ...)`
		Begin a structured if-region.

LI	`LI %dst:T, imm`	Materialize an immediate constant into a typed scalar register.
LOAD	`LOAD %dst:DstT, %m:memref<S,MemT>, %idx:idx, off:simm`	Load a scalar value from a `memref` at `idx + off`.
MAD	`MAD %dst:T, %a:T, %b:T, %c:T`	Compute multiply-add (`a*b + c`).
MAX	`MAX %dst:T, %a:T, %b:T`	Compute the maximum of two scalar values.
MIN	`MIN %dst:T, %a:T, %b:T`	Compute the minimum of two scalar values.
MOV	`MOV %dst:T, %src0:T`	Move a scalar value between registers.
MUL	`MUL %dst:T, %src0:T, %src1:T`	Multiply two scalar values.
NEG	`NEG %dst:T, %src0:T`	Negate a scalar value.
NOT	`NOT %dst:T, %src0:T`	Bitwise NOT.

OR	`OR %dst:T, %src0:T, %src1:T`
	`OR %dst:T, %src0:T, uimm`
		Bitwise OR.

POPCNT	`POPCNT %dst:u32/u64, %src0:T`	Count set bits (population count).
PREFETCH	`PREFETCH %m:memref<S,MemT>, %idx:idx, off:simm, hint:uimm`	Prefetch a memory location (no architectural side effects).
REINTERPRET	`REINTERPRET %dst:memref<S,T2>, %src0:memref<S,T1>`	Reinterpret a `memref` view with a different element type.
REM	`REM %dst:T, %src0:T, %src1:T`	Compute the remainder of integer division.

RET	`RET`
	`RET %ret0:T`
		Return from a call.

SAR	`SAR %dst:iT, %src0:iT, %sh:uimm`
	`SAR %dst:iT, %src0:iT, %sh:idx`
		Arithmetic right shift (sign-extending).

SEL	`SEL %dst:T, %cond:u1, %t:T, %f:T`	Select between two scalar values using a predicate.

SHL	`SHL %dst:T, %src0:T, %sh:uimm`
	`SHL %dst:T, %src0:T, %sh:idx`
		Logical left shift.

SHR	`SHR %dst:uT, %src0:uT, %sh:uimm`
	`SHR %dst:uT, %src0:uT, %sh:idx`
		Logical right shift (zero-extending).

STORE	`STORE %m:memref<S,MemT>, %idx:idx, off:simm, %val:MemT`	Store a scalar value to a `memref` at `idx + off`.

SUB	`SUB %dst:T, %src0:T, %src1:T`
	`SUB %dst:T, %src0:T, simm`
		Subtract two scalar values (or a signed immediate from a scalar).

SUBVIEW	`SUBVIEW %dst:memref<S,T>, %src0:memref<S,T>, %off:idx`
	`SUBVIEW %dst:memref<S,T>, %src0:memref<S,T>, simm`
		Create a zero-copy `memref` view with an element offset.

SWITCH	`SWITCH %key:T`	Begin a structured switch construct.
TRAP	`TRAP code:uimm`	Unconditionally trigger a synchronous trap/exception.
WHILE	`WHILE (%out...), (%in...)`	Begin a structured while-loop.

XOR	`XOR %dst:T, %src0:T, %src1:T`
	`XOR %dst:T, %src0:T, uimm`
		Bitwise XOR.

Opcode	Forms	Description
YIELD	`YIELD (%v0:T0, %v1:T1, ...)`	Yield values from a structured control-flow region.