LAPU-128
Instruction Set Reference

v0.6 (Draft) — September 29, 2025

Abstract

This document outline the 128-bit instruction formats for LAPU-128, focused on complex arithmetic and vector descriptors. LAPU-128 is small focused ISA designed perform complex tensor operations in an embedded enviroment. The following page shows the canonical XL, XC, XV, and XM encodings with 8-bit tick marks.

Core Instruction Formats (128-bit)

R-Type: Register-to-Register operations of either complex scalar or complex vector types

0 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120 127

opcode	subop	flags	rd	rs1	rs2	imm16	reserved
[127:120]	[119:112]	[111:96]	[95:93]	[92:90]	[89:87]	[86:71]	[70:0]

R

I-type: Immediate operations of just complex scalars

0 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120 127

opcode	subop	flags	rd	rs1	imm_90
[127:120]	[119:112]	[111:96]	[95:93]	[92:90]	[89:0]

I

J (conditional jump, 128-bit descriptor)

0 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120 127

opc	subop	flags16	rs1	offs33 (PC_rel)	reserved
[127:120]	[119:112]	[111:96]	[95:93]	[92:60]	[59:0]

JX

S-type: matrix-bank load/store (scalar & vector), 128-bit

0 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120 127

opc	subop	FLAGS16	reg3	mbid	i16	j16	len16	rsv
[127:120]	[119:112]	[111:96]	[95:93]	[92:89]	[88:73]	[72:57]	[56:41]	[40:0]

S-type (stride

Register Layout

Architectural Registers (Summary)

Class	Names	Width / Elements	Notes
Scalar (complex)	$s0..s7$	128 b each (complex Q32.32 + Q32.32)	$s0$ **is hard-wired to 0**. $s1$ is the conventional branch predicate (0/1).
Vector (complex)	$v0..v7$	VLEN elements; each element 128 b complex	$v0$ **is hard-wired to all-zeros**. Vector ops always operate on **all VLEN elements**.

Vector Length

VLEN is a hardware/HDL parameter fixed at synthesis time. It is constant at runtime. All vector instructions operate over the entire range $[0, \mathrm{VLEN} - 1]$.

Complex Number Format (Q32.32 + Q32.32)

Each scalar register and each vector element encodes a complex value $(\mathrm{Re}, \mathrm{Im})$ in fixed point:

$$\mathrm{Re}, \mathrm{Im} \in \text{Q32.32 two's complement} \quad \Rightarrow \quad x_{\mathrm{real}} = \frac{X_{\mathrm{int}}}{2^{32}}, \;\; x_{\mathrm{imag}} = \frac{Y_{\mathrm{int}}}{2^{32}}.$$

The 128-bit complex is stored little-endian in memory with **Re at the lower address** and **Im at the higher address**. Each half (Re or Im) is a 64-bit two's-complement fixed-point integer with 32 integer bits and 32 fractional bits.

Endianness

Instruction words (128 b) and data are little-endian. For complex numbers in memory: bytes for Re precede bytes for Im.

Zero Registers

The following are architecturally fixed to zero and never written:

$$s0 \equiv 0 \quad \text{(complex zero)}, \qquad v0[i] \equiv 0 \;\; \forall\, i \in [0, \mathrm{VLEN} - 1].$$

Instruction Semantics

R-type — Register-to-Register (complex)

OPCODE: `0x01`

Description

These are register to register operations involve either two vectors, two scalars, or a vector and a scalar. To determine mapping check bit position [97:96] under flags. Additionally each subop code range will be defined per mapping

$$
\begin{aligned}
f(s_1, s_2) &\mapsto s \in \mathrm{S} & \texttt{00} \\
f(\mathbf{v}_1, \mathbf{v}_2) &\mapsto \mathbf{v} \in \mathrm{V} & \texttt{01} \\
f(\mathbf{v}_1, \mathbf{v}_2) &\mapsto s \in \mathrm{S} & \texttt{11} \\
f(\mathbf{v}, s) &\mapsto \mathbf{v} \in \mathrm{V} & \texttt{10}
\end{aligned}
$$

The undefined flag fields are open to future use.

Scalar ops (unary and binary)

Table 2: Scalar register ops (`s*`): $\mathrm{S} \to \mathrm{S}$ and $\mathrm{S} \times \mathrm{S} \to \mathrm{S}$

Mnemonic	subop	Operands	Effect	Notes
Unary: $\mathrm{S} \to \mathrm{S}$				
`cneg.c`	0x00	d, a	$\mathrm{d} \leftarrow -a$	Two's-complement both halves.
`conj.c`	0x01	d, a	$\mathrm{d} \leftarrow \mathrm{conj}(a)$	Negate imaginary half.
`csqrt.c`	0x02	d, a	$\mathrm{d} \leftarrow \sqrt{a}$	Principal root; widen, then truncate to Q32.32+Q32.32.
`cabs2.c`	0x03	d, a	$\mathrm{d}_{\mathrm{re}} \leftarrow \Re(a)^2 + \Im(a)^2$, $\mathrm{d}_{\mathrm{im}} \leftarrow 0$	Magnitude^2; widen then truncate.
`cabs.c`	0x04	d, a	$\mathrm{d}_{\mathrm{re}} \leftarrow \sqrt{\Re(a)^2 + \Im(a)^2}$, $\mathrm{d}_{\mathrm{im}} \leftarrow 0$	Fixed-point $\sqrt{\cdot}$; truncating.
`creal.c`	0x05	d, a	$\mathrm{d}_{\mathrm{re}} \leftarrow \Re(a)$, $\mathrm{d}_{\mathrm{im}} \leftarrow 0$	Extract real.
`cimag.c`	0x06	d, a	$\mathrm{d}_{\mathrm{re}} \leftarrow \Im(a)$, $\mathrm{d}_{\mathrm{im}} \leftarrow 0$	Extract imaginary to real half.
`crecip.c`	0x07	d, a	$\mathrm{d} \leftarrow 1 \div a$	$(\overline{a})/\lvert a\rvert^2$; if a=0 then d:= 0.
Binary: $\mathrm{S} \times \mathrm{S} \to \mathrm{S}$				
`cadd.c`	0x08	d, a, b	$\mathrm{d} \leftarrow \mathrm{a} + \mathrm{b}$	Truncating Q32.32+Q32.32.
`csub.c`	0x09	d, a, b	$\mathrm{d} \leftarrow \mathrm{a} - \mathrm{b}$	Truncating.
`cmul.c`	0x0A	d, a, b	$\mathrm{d} \leftarrow \mathrm{a} \times \mathrm{b}$	Widen internally, truncate to Q32.32+Q32.32.
`cdiv.c`	0x0B	d, a, b	$\mathrm{d} \leftarrow \mathrm{a} \div \mathrm{b}$	$(a\,\overline{b})/\lvert b\rvert^2$; if $\lvert b\rvert = 0$ then d:= 0.
`cmaxabs.c`	0x0C	d, a, b	$\mathrm{d} \leftarrow \arg\max_{x \in \{a,b\}} \lvert x\rvert$	Ties pick `a`.
`cminabs.c`	0x0D	d, a, b	$\mathrm{d} \leftarrow \arg\min_{x \in \{a,b\}} \lvert x\rvert$	Ties pick `a`.

Vector $\to$ Vector

Table 3: Lane-wise vector ops (`v*`): $\mathrm{V} \to \mathrm{V}$ and $\mathrm{V} \times \mathrm{V} \to \mathrm{V}$

Mnemonic	subop	Operands	Effect	Notes
`cadd.v`	0x00	vD, vA, vB	vD[i] $\leftarrow$ vA[i] + vB[i]	Saturating per lane.
`csub.v`	0x01	vD, vA, vB	vD[i] $\leftarrow$ vA[i] $-$ vB[i]	Saturating per lane.

Mnemonic	subop	Operands	Effect	Notes
`cmul.v`	0x02	vD, vA, vB	vD[i] ← vA[i] × vB[i]	Complex lane-wise multiply.
`cmac.v`	0x03	vD, vA, vB	vD[i] ← vD[i] + vA[i] × vB[i]	Fused complex MAC per lane.
`cdiv.v`	0x04	vD, vA, vB	vD[i] ← vA[i] ÷ vB[i]	$(a\,\bar{b})/\lvert b\rvert^2$; if $\lvert b\rvert$=0 then lane:=0.
`conj.v`	0x05	vD, vA	vD[i] ← conj(vA[i])	Lane-wise conjugate.

Vector / Vector → Scalar (reductions)

Table 4: Reductions to scalar: V → S and V × V → S

Mnemonic	subop	Operands	Effect	Notes
`dotc`	0x00	sD, vA, vB	$sD \leftarrow \sum_{i=0}^{\mathrm{VLEN}-1} \mathrm{conj}(vA[i]) \cdot vB[i]$	Reduce to scalar `sD`.
`dotu`	0x01	sD, Ā, B̄	$sD \leftarrow \sum_{i=0}^{\mathrm{VLEN}-1} \bar{A}[i]\ \bar{B}[i]$	Complex dot (no conjugation).
`iamax.v`	0x02	sD, vA	sD ← $\arg\max_i \lvert vA[i]\rvert$	Index in `sD` real half; imag:=0.
`sum.v`	0x03	sD, Ā	$sD \leftarrow \sum_{i=0}^{\mathrm{VLEN}-1} \bar{A}[i]$	Complex sum; reduces to scalar.
`asum.v`	0x04	sD, Ā	$sD_{\mathrm{re}} \leftarrow \sum_{i=0}^{\mathrm{VLEN}-1} \lvert\bar{A}[i]\rvert,\ sD_{\mathrm{im}} \leftarrow 0$	Sum of magnitudes (real result).

Vector × Scalar → Vector (broadcast per lane)

Table 5: Vector–scalar broadcast ops: V × S → V

Mnemonic	subop	Operands	Effect	Notes
`cadd.vs`	0x18	vD, vA, sB	vD[i] ← vA[i] + sB	Broadcast add; saturating per lane.
`csub.vs`	0x19	vD, vA, sB	vD[i] ← vA[i] − sB	Broadcast sub (vector minus scalar); saturating per lane.
`cmul.vs`	0x1A	vD, vA, sB	vD[i] ← vA[i] × sB	Complex lane-wise multiply by complex scalar; widen then truncate.
`cdiv.vs`	0x1B	vD, vA, sB	vD[i] ← vA[i] ÷ sB	$(a\,\bar{b})/\lvert b\rvert^2$ per lane; if $\lvert sB\rvert$=0 then lane:=0.
`cscale.vs`	0x1C	vD, vA, t	vD[i] ← vA[i] × t	Real scale t (Q32.32) broadcast to all lanes; widen then truncate.

I-type — Immediate (scalars only)

OPCODE: `0x02`

Table 6: I-type: Immediate operations (scalar complex)

Mnemonic	subop	Operands	Effect	Notes
`cloadi`	0x00	sD, cIMM	sD ← cIMM	cIMM packed in `imm_90` (Re/Im per spec).

Mnemonic	subop	Operands	Effect	Notes
`cadd_i`	0x01	sD, sA, cIMM	sD ← sA + cIMM	Saturating per scalar; Q32.32 truncation as needed.
`cmul_i`	0x02	sD, sA, cIMM	sD ← sA × cIMM	Widen internally, clamp/truncate to Q32.32.
`csub_i`	0x03	sD, sA, cIMM	sD ← sA − cIMM	Saturating; truncation semantics match `csub.c`.
`cdiv_i`	0x04	sD, sA, cIMM	sD ← sA ÷ cIMM	$(sA\,\overline{\text{cIMM}})/\lvert\text{cIMM}\rvert^2$; if $\lvert\text{cIMM}\rvert=0$ then sD:=0.
`cmaxabs_i`	0x05	sD, sA, cIMM	$\text{sD} \leftarrow \arg\max_{x\in\{sA,\ \text{cIMM}\}} \lvert x\rvert$	Ties pick `sA`.
`cminabs_i`	0x06	sD, sA, cIMM	$\text{sD} \leftarrow \arg\min_{x\in\{sA,\ \text{cIMM}\}} \lvert x\rvert$	Ties pick `sA`.
`cscale_i`	0x10	sD, sA, rIMM	sD ← sA × rIMM	Real scale rIMM (Q32.32 in `imm_90`); widen then truncate.

J-type — Conditional Jump

OPCODE: `0x03`

Table 7: J-type: Conditional jump (single predicate)

Mnemonic	subop	Operands	Effect	Notes
`jrel`	0x00	offs33	If $s1 \neq 0$: PC ← PC + offs33 (instruction-relative).	$s0 \equiv 0$; $s1$ is the conventional branch predicate (0/1).

S-type — Matrix-bank Vector Load/Store (stride implicit)

OPCODE: `0x04`

Table 8: S-type: Matrix-bank vector load/store

Mnemonic	subop	Operands	Effect	Notes
`vld`	0x00	vD, mbid, rc, idx16, len16	Load into vD the sequence: if $rc{=}0$: ($r{=}$idx16, $c{=}0..L{-}1$), if $rc{=}1$: ($r{=}0..L{-}1$, $c{=}$idx16), where L = len16 if nonzero, else L = VLEN.	Row stride = 1; column = VLEN. Elements are 1 complex (Re then Im).
`vst`	0x01	vS, mbid, rc, idx16, len16	Store from vS to the same address pattern as `vld`.	Vectors cover all lanes w L=VLEN.
`sld.xy`	0x02	sD, mbid, x16, y16	Load the single element at coordinates (r=y16, c=x16) from matrix-bank `mbid` into scalar register `sD`.	Coordinates are 0-based unsigned 16-bit. Element is one 128-bit complex (I then Im). Out-of-bounds coordinates trap.
`sst.xy`	0x03	sS, mbid, x16, y16	Store scalar `sS` to the element at (r=y16, c=x16) in matrix-bank `mbid`.	Same addressing and tra rules as `sld.xy`. Element 128-bit complex (Re then