LAPU-128
Instruction Set Reference

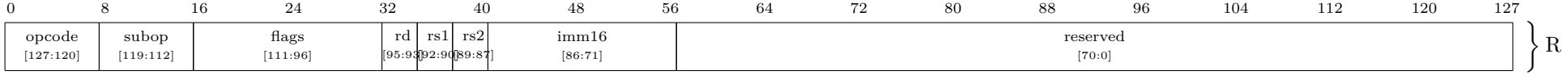
v0.6 (Draft) — September 13, 2025

Abstract

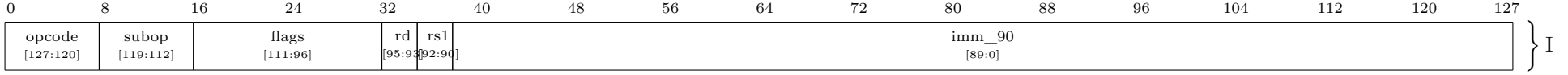
This document outline the 128-bit instruction formats for LAPU-128, focused on complex arithmetic and vector descriptors. LAPU-128 is small focused ISA designed perform complex tensor operations in an embedded enviroment. The following page shows the canonical XL, XC, XV, and XM encodings with 8-bit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

Core Instruction Formats (128-bit)

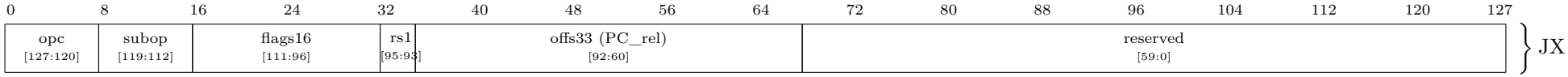
R-Type: Register-to-Register operations of either complex scalar or complex vector types



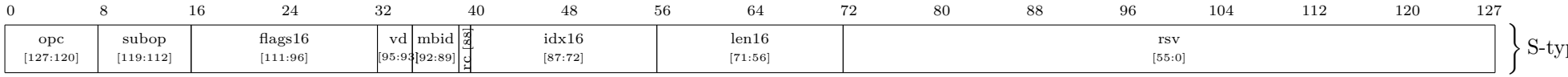
I-type: Immediate operations of just complex scalars



J (conditional jump, 128-bit descriptor)



S-type: matrix-bank load/store, 128-bit



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 (Re, 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)

Scalar

Table 2: Operate on scalar registers (`s*`)

Mnemonic	subop	Operands	Effect	Notes
`cadd.c`	0x00	d, a, b	d ← a + b	Truncating Q32.32+Q32.32.
`csub.c`	0x01	d, a, b	d ← a − b	Truncating.
`cmul.c`	0x02	d, a, b	d ← a × b	Widen internally, truncate to Q32.32+Q32.32.
`cdiv.c`	0x03	d, a, b	d ← a ÷ b	$(a\,\bar{b})/\lvert b\rvert^2$; if $\lvert b\rvert = 0$ then d:= 0.
`cneg.c`	0x10	d, a	d ← $-a$	Two's-complement both halves.
`conj.c`	0x20	d, a	d ← conj(a)	Negate imaginary half.
`csqrt.c`	0x26	d, a	d ← $\sqrt{a}$	Principal complex root; widen internally, truncate to Q32.32+Q32.32.
`cabs2.c`	0x21	d, a	$d_{re} \leftarrow \Re(a)^2+\Im(a)^2$, $d_{im} \leftarrow 0$	Magnitude^2; widen then truncate.
`cabs.c`	0x22	d, a	$d_{re} \leftarrow \sqrt{\Re(a)^2 + \Im(a)^2}$, $d_{im} \leftarrow 0$	Fixed-point $\sqrt{\cdot}$; truncating.
`creal.c`	0x50	d, a	$d_{re} \leftarrow \Re(a)$, $d_{im} \leftarrow 0$	Extract real.
`cimag.c`	0x51	d, a	$d_{re} \leftarrow \Im(a)$, $d_{im} \leftarrow 0$	Extract imaginary to real half.
`crecip.c`	0x25	d, a	d ← $1 \div a$	$(\overline{a})/\lvert a\rvert^2$; if a=0 then d:= 0.
`cscale.c`	0x30	d, a, t	d ← $a \times t$	Real scale t (Q32.32); widen then truncate.
`cmadd.c`	0x33	d, a, b, c	d ← $a \times b + c$	Fused complex MAD; one rounding.
`cmaxabs.c`	0x62	d, a, b	d ← $\arg\max_{x\in\{a,b\}} \lvert x\rvert$	Ties pick `a`.
`cminabs.c`	0x63	d, a, b	d ← $\arg\min_{x\in\{a,b\}} \lvert x\rvert$	Ties pick `a`.

Vector

Table 3: Operate on vector registers (`v*`); always over all VLEN elements

Mnemonic	subop	Operands	Effect	Notes
`cadd.v`	0x10	vD, vA, vB	vD[i] ← vA[i] + vB[i]	Saturating per lane.
`csub.v`	0x11	vD, vA, vB	vD[i] ← vA[i] − vB[i]	Saturating per lane.
`cmul.v`	0x12	vD, vA, vB	vD[i] ← vA[i] × vB[i]	Complex lane-wise multiply.
`cmac.v`	0x13	vD, vA, vB	vD[i] ← vD[i] + vA[i] × vB[i]	Fused complex MAC per lane.
`cdiv.v`	0x16	vD, vA, vB	vD[i] ← vA[i] ÷ vB[i]	$(a\,\bar{b})/\lvert b\rvert^2$ per lane; if $\lvert b\rvert$=0 then lane:=0.
`conj.v`	0x21	vD, vA	vD[i] ← conj(vA[i])	Lane-wise conjugate.
`dotc`	0x30	sD, vA, vB	sD ← $\sum_{i=0}^{\mathrm{VLEN}-1} \mathrm{conj}(vA[i]) \cdot vB[i]$	Reduce to scalar `sD`.
`dotu`	0x36	sD, $\bar{\text{A}}$, $\bar{\text{B}}$	$sD \leftarrow \sum_{i=0}^{\mathrm{VLEN}-1} \bar{A}[i]\,\bar{B}[i]$	Complex dot (no conjugation).
`iamax.v`	0x33	sD, vA	sD ← $\arg\max_i \lvert vA[i]\rvert$	Index (0..VLEN−1) in `sD` real half; imag:=0.
`sum.v`	0x34	sD, $\bar{\text{A}}$	$sD \leftarrow \sum_{i=0}^{\mathrm{VLEN}-1} \bar{A}[i]$	Complex sum; reduces to scalar.

Mnemonic	subop	Operands	Effect	Notes
`asum.v`	0x35	sD, $\bar{\text{A}}$	$sD_{\text{re}} \leftarrow \sum_{i=0}^{\text{VLEN}-1} \lvert\bar{A}[i]\rvert,\ sD_{\text{im}} \leftarrow 0$	Sum of magnitudes (real result).

I-type — Immediate (scalars only)

Table 4: 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).
`cadd_i`	0x01	sD, sA, cIMM	sD ← sA + cIMM	Saturating.
`cmul_i`	0x02	sD, sA, cIMM	sD ← sA × cIMM	Widen then clamp to Q32.32.

J-type — Conditional Jump

Table 5: 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)

Table 6: 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 wi L=VLEN.