CARTÃO DE REFERÊNCIA ABERTO

Instruções da Base de Números Inteiros: RV32I e RV64

Categoria	Nome	Fmt	RV32I Base	+RV64I
Shifts	Shift Left Logical	R	`SLL rd,rs1,rs2`	`SLLW rd,rs1,rs2`
	Shift Left Log. Imm.	I	`SLLI rd,rs1,shamt`	`SLLIW rd,rs1,shamt`
	Shift Right Logical	R	`SRL rd,rs1,rs2`	`SRLW rd,rs1,rs2`
	Shift Right Log. Imm.	I	`SRLI rd,rs1,shamt`	`SRLIW rd,rs1,shamt`
	Shift Right Arithmetic	R	`SRA rd,rs1,rs2`	`SRAW rd,rs1,rs2`
	Shift Right Arith. Imm.	I	`SRAI rd,rs1,shamt`	`SRAIW rd,rs1,shamt`
Aritmética	ADD	R	`ADD rd,rs1,rs2`	`ADDW rd,rs1,rs2`
	ADD Immediate	I	`ADDI rd,rs1,imm`	`ADDIW rd,rs1,imm`
	SUBtract	R	`SUB rd,rs1,rs2`	`SUBW rd,rs1,rs2`
	Load Upper Imm	U	`LUI rd,imm`	
	Add Upper Imm to PC	U	`AUIPC rd,imm`	
Lógica	XOR	R	`XOR rd,rs1,rs2`	
	XOR Immediate	I	`XORI rd,rs1,imm`	
	OR	R	`OR rd,rs1,rs2`	
	OR Immediate	I	`ORI rd,rs1,imm`	
	AND	R	`AND rd,rs1,rs2`	
	AND Immediate	I	`ANDI rd,rs1,imm`	
Comparação	Set <	R	`SLT rd,rs1,rs2`	
	Set < Immediate	I	`SLTI rd,rs1,imm`	
	Set < Unsigned	R	`SLTU rd,rs1,rs2`	
	Set < Imm Unsigned	I	`SLTIU rd,rs1,imm`	
Desvios	Branch =	B	`BEQ rs1,rs2,imm`	
	Branch ≠	B	`BNE rs1,rs2,imm`	
	Branch <	B	`BLT rs1,rs2,imm`	
	Branch ≥	B	`BGE rs1,rs2,imm`	
	Branch < Unsigned	B	`BLTU rs1,rs2,imm`	
	Branch ≥ Unsigned	B	`BGEU rs1,rs2,imm`	
Salto & Link	J&L	J	`JAL rd,imm`	
	Jump & Link Register	I	`JALR rd,rs1,imm`	
Synch	Synch thread	I	`FENCE`	
	Synch Instr & Data	I	`FENCE.I`	
Environment	CALL	I	`ECALL`	
	BREAK	I	`EBREAK`	

Registrador de controle de Status (CSR)

Read/Write	I	`CSRRW rd,csr,rs1`	
Read & Set Bit	I	`CSRRS rd,csr,rs1`	
Read & Clear Bit	I	`CSRRC rd,csr,rs1`	
Read/Write Imm	I	`CSRRWI rd,csr,imm`	
Read & Set Bit Imm	I	`CSRRSI rd,csr,imm`	
Read & Clear Bit Imm	I	`CSRRCI rd,csr,imm`	

Loads	Load Byte	I	`LB rd,rs1,imm`	
	Load Halfword	I	`LH rd,rs1,imm`	
	Load Byte Unsigned	I	`LBU rd,rs1,imm`	
	Load Half Unsigned	I	`LHU rd,rs1,imm`	
	Load Word	I	`LW rd,rs1,imm`	`LWU rd,rs1,imm`
				`LD rd,rs1,imm`
Stores	Store Byte	S	`SB rs1,rs2,imm`	
	Store Halfword	S	`SH rs1,rs2,imm`	
	Store Word	S	`SW rs1,rs2,imm`	`SD rs1,rs2,imm`

Instruções Privilegiadas para RV

Categoria	Nome	Fmt	Mnemônica do RV
Trap	Mach-mode trap return	R	`MRET`
	Supervisor-mode trap return	R	`SRET`
Interrupt	Wait for Interrupt	R	`WFI`
MMU	Virtual Memory FENCE	R	`SFENCE.VMA rs1,rs2`

Exemplos das 60 pseudo-instruções do RV

Branch = 0 (`BEQ rs,x0,imm`)	J	`BEQZ rs,imm`
Jump (uses `JAL x0,imm`)	J	`J imm`
MoVe (uses `ADDI rd,rs,0`)	R	`MV rd,rs`
RETurn (uses `JALR x0,0,ra`)	I	`RET`

Extensão de Instrução Compactada (16 bits): RV32C

Categoria	Nome	Fmt	RVC	RISC-V equivalent
Loads	Load Word	CL	`C.LW rd',rs1',imm`	`LW rd',rs1',imm*4`
	Load Word SP	CI	`C.LWSP rd,imm`	`LW rd,sp,imm*4`
	Float Load Word SP	CL	`C.FLW rd',rs1',imm`	`FLW rd',rs1',imm*8`
	Float Load Word	CI	`C.FLWSP rd,imm`	`FLW rd,sp,imm*8`
	Float Load Double	CL	`C.FLD rd',rs1',imm`	`FLD rd',rs1',imm*16`
	Float Load Double SP	CI	`C.FLDSP rd,imm`	`FLD rd,sp,imm*16`
Stores	Store Word	CS	`C.SW rs1',rs2',imm`	`SW rs1',rs2',imm*4`
	Store Word SP	CSS	`C.SWSP rs2,imm`	`SW rs2,sp,imm*4`
	Float Store Word	CS	`C.FSW rs1',rs2',imm`	`FSW rs1',rs2',imm*8`
	Float Store Word SP	CSS	`C.FSWSP rs2,imm`	`FSW rs2,sp,imm*8`
	Float Store Double	CS	`C.FSD rs1',rs2',imm`	`FSD rs1',rs2',imm*16`
	Float Store Double SP	CSS	`C.FSDSP rs2,imm`	`FSD rs2,sp,imm*16`
Aritmética	ADD	CR	`C.ADD rd,rs1`	`ADD rd,rd,rs1`
	ADD Immediate	CI	`C.ADDI rd,imm`	`ADDI rd,rd,imm`
	ADD SP Imm * 16	CI	`C.ADDI16SP x0,imm`	`ADDI sp,sp,imm*16`
	ADD SP Imm * 4	CIW	`C.ADDI4SPN rd',imm`	`ADDI rd',sp,imm*4`
	SUB	CR	`C.SUB rd,rs1`	`SUB rd,rd,rs1`
	AND	CR	`C.AND rd,rs1`	`AND rd,rd,rs1`
	AND Immediate	CI	`C.ANDI rd,imm`	`ANDI rd,rd,imm`
	OR	CR	`C.OR rd,rs1`	`OR rd,rd,rs1`
	eXclusive OR	CR	`C.XOR rd,rs1`	`AND rd,rd,rs1`
	MoVe	CR	`C.MV rd,rs1`	`ADD rd,rs1,x0`
	Load Immediate	CI	`C.LI rd,imm`	`ADDI rd,x0,imm`
	Load Upper Imm	CI	`C.LUI rd,imm`	`LUI rd,imm`
Desloc.	Shift Left Imm	CI	`C.SLLI rd,imm`	`SLLI rd,rd,imm`
	Shift Right Ari. Imm.	CI	`C.SRAI rd,imm`	`SRAI rd,rd,imm`
	Shift Right Log. Imm.	CI	`C.SRLI rd,imm`	`SRLI rd,rd,imm`
Desvios	Branch=0	CB	`C.BEQZ rs1',imm`	`BEQ rs1',x0,imm`
	Branch≠0	CB	`C.BNEZ rs1',imm`	`BNE rs1',x0,imm`
Salto	Jump	CJ	`C.J imm`	`JAL x0,imm`
	Jump Register	CR	`C.JR rd,rs1`	`JALR x0,rs1,0`
Salto & Link	J&L	CJ	`C.JAL imm`	`JAL ra,imm`
	Jump & Link Register	CR	`C.JALR rs1`	`JALR ra,rs1,0`
Sistema	Env. BREAK	CI	`C.EBREAK`	`EBREAK`

Extensão Compactada Opcional: RV64C

All RV32C (except `C.JAL`, 4 word loads, 4 word stores) plus:

ADD Word (`C.ADDW`)	Load Doubleword (`C.LD`)
ADD Imm. Word (`C.ADDIW`)	Load Doubleword SP (`C.LDSP`)
SUBtract Word (`C.SUBW`)	Store Doubleword (`C.SD`)
	Store Doubleword SP (`C.SDSP`)

Formatos de instrução de 32 bits

	31–27	26–25	24–20	19–15	14–12	11–7	6–0
R	funct7		rs2	rs1	funct3	rd	opcode
I	imm[11:0]			rs1	funct3	rd	opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
B	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
J	imm[20|10:1|11|19:12]					rd	opcode

Formatos de instrução de 16 bits (RVC)

	15–12	11–10	9–8	7–6	5–4	3–2	1–0
CR	funct4		rd/rs1		rs2		op
CI	funct3	imm	rd/rs1		imm		op
CSS	funct3	imm			rs2		op
CIW	funct3	imm				rd′	op
CL	funct3	imm	rs1′		imm	rd′	op
CS	funct3	imm	rs1′		imm	rs2′	op
CB	funct3	offset	rs1′		offset		op
CJ	funct3	jump target					op

Inteiro base RISC-V (RV32I / 64I), RV32 / 64C privilegiado e opcional. Registradores x1-x3 e o PC têm 32 bits de largura em RV32I e RV64I (x0 = 0).
.RV64I adiciona 12 instruções para os dados mais amplos. Cada instrução RVC de 16 bits é mapeada para uma instrução RISC-V de 32 bits.