CARTÃO DE REFERÊNCIA ABERTO

①

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`	+RV64I
	Load Half Unsigned	I	`LHU rd,rs1,imm`	`LWU rd,rs1,imm`
	Load Word	I	`LW 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 14 13	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.

CARTÃO DE REFERÊNCIA ABERTO RISC-V ②

Extensão de Instrução Multiplicação-Divisão: RVM

Categoria	Nome	Fmt	RV32M (Multiplicação-Divisão)	+RV64M
Multiplicação	MULtiply	R	`MUL rd,rs1,rs2`	`MULW rd,rs1,rs2`
	MULtiply High	R	`MULH rd,rs1,rs2`	
	MULtiply High Sign/Uns	R	`MULHSU rd,rs1,rs2`	
	MULtiply High Uns	R	`MULHU rd,rs1,rs2`	
Divisão	DIVide	R	`DIV rd,rs1,rs2`	`DIVW rd,rs1,rs2`
	DIVide Unsigned	R	`DIVU rd,rs1,rs2`	
Restante	REMainder	R	`REM rd,rs1,rs2`	`REMW rd,rs1,rs2`
	REMainder Unsigned	R	`REMU rd,rs1,rs2`	`REMUW rd,rs1,rs2`

Extensão de Instruções Atômicas: RVA

Categoria	Nome	Fmt	RV32A (atômica)	+RV64A
Load	Load Reserved	R	`LR.W rd,rs1`	`LR.D rd,rs1`
Store	Store Conditional	R	`SC.W rd,rs1,rs2`	`SC.D rd,rs1,rs2`
Swap	SWAP	R	`AMOSWAP.W rd,rs1,rs2`	`AMOSWAP.D rd,rs1,rs2`
Soma	ADD	R	`AMOADD.W rd,rs1,rs2`	`AMOADD.D rd,rs1,rs2`
Lógica	XOR	R	`AMOXOR.W rd,rs1,rs2`	`AMOXOR.D rd,rs1,rs2`
	AND	R	`AMOAND.W rd,rs1,rs2`	`AMOAND.D rd,rs1,rs2`
	OR	R	`AMOOR.W rd,rs1,rs2`	`AMOOR.D rd,rs1,rs2`
Min/Max	MINimum	R	`AMOMIN.W rd,rs1,rs2`	`AMOMIN.D rd,rs1,rs2`
	MAXimum	R	`AMOMAX.W rd,rs1,rs2`	`AMOMAX.D rd,rs1,rs2`
	MINimum Unsigned	R	`AMOMINU.W rd,rs1,rs2`	`AMOMINU.D rd,rs1,rs2`
	MAXimum Unsigned	R	`AMOMAXU.W rd,rs1,rs2`	`AMOMAXU.D rd,rs1,rs2`

Duas extensões de instrução de ponto flutuante: RVF e RVD

Categoria	Nome	Fmt	RV32{F|D} (SP,DP Fl. Pt.)	+RV64{F|D}
Move	Move from Integer	R	`FMV.W.X rd,rs1`	`FMV.D.X rd,rs1`
	Move to Integer	R	`FMV.X.W rd,rs1`	`FMV.X.D rd,rs1`
Conversão	ConVerT from Int	R	`FCVT.{S|D}.W rd,rs1`	`FCVT.{S|D}.L rd,rs1`
	ConVerT from Int Unsigned	R	`FCVT.{S|D}.WU rd,rs1`	`FCVT.{S|D}.LU rd,rs1`
	ConVerT to Int	R	`FCVT.W.{S|D} rd,rs1`	`FCVT.L.{S|D} rd,rs1`
	ConVerT to Int Unsigned	R	`FCVT.WU.{S|D} rd,rs1`	`FCVT.LU.{S|D} rd,rs1`
Load	Load	I	`FL{W,D} rd,rs1,imm`	
Store	Store	S	`FS{W,D} rs1,rs2,imm`	
Aritmética	ADD	R	`FADD.{S|D} rd,rs1,rs2`	
	SUBtract	R	`FSUB.{S|D} rd,rs1,rs2`	
	MULtiply	R	`FMUL.{S|D} rd,rs1,rs2`	
	DIVide	R	`FDIV.{S|D} rd,rs1,rs2`	
	SQuare RooT	R	`FSQRT.{S|D} rd,rs1`	
Mul-Soma	Multiply-ADD	R	`FMADD.{S|D} rd,rs1,rs2,rs3`	
	Multiply-SUBtract	R	`FMSUB.{S|D} rd,rs1,rs2,rs3`	
	Negative Multiply-SUBtract	R	`FNMSUB.{S|D} rd,rs1,rs2,rs3`	
	Negative Multiply-ADD	R	`FNMADD.{S|D} rd,rs1,rs2,rs3`	
Sign Inject	SiGN source	R	`FSGNJ.{S|D} rd,rs1,rs2`	
	Negative SiGN source	R	`FSGNJN.{S|D} rd,rs1,rs2`	
	Xor SiGN source	R	`FSGNJX.{S|D} rd,rs1,rs2`	
Min/Max	MINimum	R	`FMIN.{S|D} rd,rs1,rs2`	
	MAXimum	R	`FMAX.{S|D} rd,rs1,rs2`	
Comparação	compare Float =	R	`FEQ.{S|D} rd,rs1,rs2`	
	compare Float <	R	`FLT.{S|D} rd,rs1,rs2`	
	compare Float ≤	R	`FLE.{S|D} rd,rs1,rs2`	
Categorizar	CLASSify type	R	`FCLASS.{S|D} rd,rs1`	
Configurar	Read Status	R	`FRCSR rd`	
	Read Rounding Mode	R	`FRRM rd`	
	Read Flags	R	`FRFLAGS rd`	
	Swap Status Reg	R	`FSCSR rd,rs1`	
	Swap Rounding Mode	R	`FSRM rd,rs1`	
	Swap Flags	R	`FSFLAGS rd,rs1`	
	Swap Rounding Mode Imm	I	`FSRMI rd,imm`	
	Swap Flags Imm	I	`FSFLAGSI rd,imm`	

Convenção de chamada

Registrador	Nome ABI	Saver
x0	zero	---
x1	ra	Caller
x2	sp	Callee
x3	gp	---
x4	tp	---
x5-7	t0-2	Caller
x8	s0/fp	Callee
x9	s1	Callee
x10-11	a0-1	Caller
x12-17	a2-7	Caller
x18-27	s2-11	Callee
x28-31	t3-t6	Caller
f0-7	ft0-7	Caller
f8-9	fs0-1	Callee
f10-11	fa0-1	Caller
f12-17	fa2-7	Caller
f18-27	fs2-11	Callee
f28-31	ft8-11	Caller
zero	Zero hardwired	
ra	Endereço de ret.	
sp	Ponteiro p. pilha	
gp	Ponteiro global	
tp	Thread pointer	
t0-6,ft0-6	Temporários	
s0-11,fs0-11	Registradores salvos	
a0-7,fa0-7	Args. de função	

Extensão Vetorial: RVV

Nome	Fmt	RV32V/R64V
SET Vector Len.	R	`SETVL rd,rs1`
MULtiply High	R	`VMULH rd,rs1,rs2`
REMainder	R	`VREM rd,rs1,rs2`
Shift Left Log.	R	`VSLL rd,rs1,rs2`
Shift Right Log.	R	`VSRL rd,rs1,rs2`
Shift R. Arith.	R	`VSRA rd,rs1,rs2`
LoaD	I	`VLD rd,rs1,imm`
LoaD Strided	R	`VLDS rd,rs1,rs2`
LoaD indeXed	R	`VLDX rd,rs1,rs2`
STore	S	`VST rd,rs1,imm`
STore Strided	R	`VSTS rd,rs1,rs2`
STore indeXed	R	`VSTX rd,rs1,rs2`
AMO SWAP	R	`AMOSWAP rd,rs1,rs2`
AMO ADD	R	`AMOADD rd,rs1,rs2`
AMO XOR	R	`AMOXOR rd,rs1,rs2`
AMO AND	R	`AMOAND rd,rs1,rs2`
AMO OR	R	`AMOOR rd,rs1,rs2`
AMO MINimum	R	`AMOMIN rd,rs1,rs2`
AMO MAXimum	R	`AMOMAX rd,rs1,rs2`
Predicate =	R	`VPEQ rd,rs1,rs2`
Predicate ≠	R	`VPNE rd,rs1,rs2`
Predicate <	R	`VPLT rd,rs1,rs2`
Predicate ≥	R	`VPGE rd,rs1,rs2`
Predicate AND	R	`VPAND rd,rs1,rs2`
Pred. AND NOT	R	`VPANDN rd,rs1,rs2`
Predicate OR	R	`VPOR rd,rs1,rs2`
Predicate XOR	R	`VPXOR rd,rs1,rs2`
Predicate NOT	R	`VPNOT rd,rs1`
Pred. SWAP	R	`VPSWAP rd,rs1`
MOVe	R	`VMOV rd,rs1`
ConVerT	R	`VCVT rd,rs1`
ADD	R	`VADD rd,rs1,rs2`
SUBtract	R	`VSUB rd,rs1,rs2`
MULtiply	R	`VMUL rd,rs1,rs2`
DIVide	R	`VDIV rd,rs1,rs2`
SQuare RooT	R	`VSQRT rd,rs1`
Multiply-ADD	R	`VFMADD rd,rs1,rs2,rs3`
Multiply-SUB	R	`VFMSUB rd,rs1,rs2,rs3`
Neg. Mul.-SUB	R	`VFNMSUB rd,rs1,rs2,rs3`
Neg. Mul.-ADD	R	`VFNMADD rd,rs1,rs2,rs3`
SiGN inJect	R	`VSGNJ rd,rs1,rs2`
Neg SiGN inJect	R	`VSGNJN rd,rs1,rs2`
Xor SiGN inJect	R	`VSGNJX rd,rs1,rs2`
MINimum	R	`VMIN rd,rs1,rs2`
MAXimum	R	`VMAX rd,rs1,rs2`
XOR	R	`VXOR rd,rs1,rs2`
OR	R	`VOR rd,rs1,rs2`
AND	R	`VAND rd,rs1,rs2`
CLASS	R	`VCLASS rd,rs1`
SET Data Conf.	R	`VSETDCFG rd,rs1`
EXTRACT	R	`VEXTRACT rd,rs1,rs2`
MERGE	R	`VMERGE rd,rs1,rs2`
SELECT	R	`VSELECT rd,rs1,rs2`

Convenção de chamada RISC-V e cinco extensões opcionais: 8 RV32M; 11 RV32A; 34 instruções de ponto flutuante para dados de 32 e 64 bits (RV32F, RV32D); e 53 RV32V. Usando a notação regex, {} signica set, então FADD. {F | D} é tanto FADD.F quanto FADD.D. RV32 {F | D} adiciona registradores f0-f31, cuja largura corresponde à maior precisão, e comprimento v1. O RV64 adiciona instruções: o RVM obtém 4 , RVA 11, RVF 6, RVD 6 e RVV 0.