Open RISC-V Reference Card ①

Base Integer Instructions: RV32I and RV64I

Category	Name	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`
Arithmetic	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`	
Logical	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`	
Compare	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`	
Branches	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`	
Jump & 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`	

Control Status Register (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`

Category	Name	Fmt	RV32I Base	+RV64I
Loads	Load Byte	I	`LB rd,rs1,imm`	`LWU rd,rs1,imm`
	Load Halfword	I	`LH rd,rs1,imm`	`LD 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`	
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`

RV Privileged Instructions

Category	Name	Fmt	RV mnemonic
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`

Examples of the 60 RV Pseudoinstructions

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`

Optional Compressed (16-bit) Instruction Extension: RV32C

Category	Name	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`
Arithmetic	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`
Shifts	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`
Branches	Branch=0	CB	`C.BEQZ rs1′,imm`	`BEQ rs1',x0,imm`
	Branch≠0	CB	`C.BNEZ rs1′,imm`	`BNE rs1',x0,imm`
Jump	Jump	CJ	`C.J imm`	`JAL x0,imm`
	Jump Register	CR	`C.JR rd,rs1`	`JALR x0,rs1,0`
Jump & Link	J&L	CJ	`C.JAL imm`	`JAL ra,imm`
	Jump & Link Register	CR	`C.JALR rs1`	`JALR ra,rs1,0`
System	Env. BREAK	CI	`C.EBREAK`	`EBREAK`

Optional Compressed Extention: RV64C

All RV32C (except `C.JAL`, 4 word loads, 4 word strores) 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`)

32-bit Instruction Formats

	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

16-bit (RVC) Instruction Formats

	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

RISC-V Integer Base (RV32I/64I), privileged, and optional RV32/64C. Registers `x1`-`x31` and the `PC` are 32 bits wide in RV32I and 64 in RV64I (`x0`=0). RV64I adds 12 instructions for the wider data. Every 16-bit RVC instruction maps to an existing 32-bit RISC-V instruction.

Open RISC-V Reference Card ②

Optional Multiply-Divide Instruction Extension: RVM

Category	Name	Fmt	RV32M (Multiply-Divide)		+RV64M	
Multiply	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		
Divide	DIVide	R	DIV	rd,rs1,rs2	DIVW	rd,rs1,rs2
	DIVide Unsigned	R	DIVU	rd,rs1,rs2		
Remainder	REMainder	R	REM	rd,rs1,rs2	REMW	rd,rs1,rs2
	REMainder Unsigned	R	REMU	rd,rs1,rs2	REMUW	rd,rs1,rs2

Optional Atomic Instruction Extension: RVA

Category	Name	Fmt	RV32A (Atomic)		+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
Add	ADD	R	AMOADD.W	rd,rs1,rs2	AMOADD.D	rd,rs1,rs2
Logical	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

Two Optional Floating-Point Instruction Extensions: RVF & RVD

Category	Name	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
Convert	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		
Arithmetic	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-Add	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		
Compare	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		
Categorize	CLASSify type	R	FCLASS.{S|D}	rd,rs1		
Configure	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		

Calling Convention

Register	ABI Name	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	Hardwired zero
ra	Return address
sp	Stack pointer
gp	Global pointer
tp	Thread pointer
t0-6,ft0-11	Temporaries
s0-11,fs0-11	Saved registers
a0-7,fa0-7	Function args

Optional Vector Extension: RVV

Name	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,rs2
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

RISC-V calling convention and five optional extensions: 8 RV32M; 11 RV32A; 34 floating-point instructions each for 32- and 64-bit data (RV32F, RV32D); and 53 RV32V. Using regex notation, {} means set, so `FADD.{F|D}` is both `FADD.F` and `FADD.D`. RV32{F|D} adds registers `f0-f31`, whose width matches the widest precision, and a floating-point control and status register `fcsr`. RV32V adds vector registers `v0-v31`, vector predicate registers `vp0-vp7`, and vector length register `vl`. RV64 adds a few instructions: RVM gets 4, RVA 11, RVF 6, RVD 6, and RVV 0.