

Base Integer Instructions: RV32I and RV64I					RV Privileged Instructions						
Category	Name	Fmt	RV32I Base		+RV64I		Category	Name	Fmt	RV mnemonic	
Shifts	Shift Left Logical	R	SLL	rd,rs1,rs2	SLLW	rd,rs1,rs2	Trap	Mach-mode trap return	R	MRET	
	Shift Left Log. Imm.	I	SLLI	rd,rs1,shamt	SLLIW	rd,rs1,shamt		Supervisor-mode trap return	R	SRET	
	Shift Right Logical	R	SRL	rd,rs1,rs2	SRLW	rd,rs1,rs2	Interrupt	Wait for Interrupt	R	WFI	
	Shift Right Log. Imm.	I	SRLI	rd,rs1,shamt	SRLIW	rd,rs1,shamt		MMU Virtual Memory FENCE	R	SFENCE.VMA rs1,rs2	
	Shift Right Arithmetic	R	SRA	rd,rs1,rs2	SRAW	rd,rs1,rs2	Examples of the 60 RV Pseudoinstructions				
	Shift Right Arith. Imm.	I	SRAI	rd,rs1,shamt	SRAIW	rd,rs1,shamt					
Arithmetic	ADD	R	ADD	rd,rs1,rs2	ADDW	rd,rs1,rs2	Optional Compressed (16-bit) Instruction Extension: RV32C				
	ADD Immediate	I	ADDI	rd,rs1,imm	ADDIW	rd,rs1,imm					
	SUBtract	R	SUB	rd,rs1,rs2	SUBW	rd,rs1,rs2	Category Name Fmt RVC RISC-V equivalent				
	Load Upper Imm	U	LUI	rd,imm							
Add Upper Imm to PC	U	AUIPC	rd,imm								
Logical	XOR	R	XOR	rd,rs1,rs2	Loads	Load Word	CL	C.LW	rd',rs1',imm	LW	rd',rs1',imm*4
	XOR Immediate	I	XORI	rd,rs1,imm		Load Word SP	CI	C.LWSP	rd,imm	LW	rd,sp,imm*4
	OR	R	OR	rd,rs1,rs2		Float Load Word SP	CL	C.FLW	rd',rs1',imm	FLW	rd',rs1',imm*8
	OR Immediate	I	ORI	rd,rs1,imm		Float Load Word	CI	C.FLWSP	rd,imm	FLW	rd,sp,imm*8
	AND	R	AND	rd,rs1,rs2		Float Load Double	CL	C.FLD	rd',rs1',imm	FLD	rd',rs1',imm*16
	AND Immediate	I	ANDI	rd,rs1,imm		Float Load Double SP	CI	C.FLDSP	rd,imm	FLD	rd,sp,imm*16
Compare	Set <	R	SLT	rd,rs1,rs2	Stores	Store Word	CS	C.SW	rs1',rs2',imm	SW	rs1',rs2',imm*4
	Set < Immediate	I	SLTI	rd,rs1,imm		Store Word SP	CSS	C.SWSP	rs2,imm	SW	rs2,sp,imm*4
	Set < Unsigned	R	SLTU	rd,rs1,rs2		Float Store Word	CS	C.FSW	rs1',rs2',imm	FSW	rs1',rs2',imm*8
	Set < Imm Unsigned	I	SLTIU	rd,rs1,imm		Float Store Word SP	CSS	C.FSWSP	rs2,imm	FSW	rs2,sp,imm*8
Branches	Branch =	B	BEQ	rs1,rs2,imm	Float Store Double	CS	C.FSD	rs1',rs2',imm	FSD	rs1',rs2',imm*16	
	Branch ≠	B	BNE	rs1,rs2,imm	Float Store Double SP	CSS	C.FSDSP	rs2,imm	FSD	rs2,sp,imm*16	
	Branch <	B	BLT	rs1,rs2,imm	Arithmetic	ADD	CR	C.ADD	rd,rs1	ADD	rd,rd,rs1
	Branch ≥	B	BGE	rs1,rs2,imm		ADD Immediate	CI	C.ADDI	rd,imm	ADDI	rd,rd,imm
	Branch < Unsigned	B	BLTU	rs1,rs2,imm		ADD SP Imm * 16	CI	C.ADDI16SP	x0,imm	ADDI	sp,sp,imm*16
	Branch ≥ Unsigned	B	BGEU	rs1,rs2,imm		ADD SP Imm * 4	CIW	C.ADDI4SPN	rd',imm	ADDI	rd',sp,imm*4
Jump & Link	J&L	J	JAL	rd,imm		SUB	CR	C.SUB	rd,rs1	SUB	rd,rd,rs1
	Jump & Link Register	I	JALR	rd,rs1,imm		AND	CR	C.AND	rd,rd,rs1	AND	rd,rd,rs1
Synch	Synch thread	I	FENCE		AND Immediate	CI	C.ANDI	rd,imm	ANDI	rd,rd,imm	
	Synch Instr & Data	I	FENCE.I		OR	CR	C.OR	rd,rs1	OR	rd,rd,rs1	
Environment	CALL	I	ECALL		eXclusive OR	CR	C.XOR	rd,rs1	AND	rd,rd,rs1	
	BREAK	I	EBREAK		MoVe	CR	C.MV	rd,rs1	ADD	rd,rs1,x0	
Control Status Register (CSR)					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						
	+RV64I					Optional Compressed Extention: RV64C					
Loads	Load Byte	I	LB	rd,rs1,imm	LWU	rd,rs1,imm	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)				
	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	SD	rs1,rs2,imm					
	Store Halfword	S	SH	rs1,rs2,imm							
	Store Word	S	SW	rs1,rs2,imm							

Optional Multiply-Divide Instruction Extension: RVM					Optional Vector Extension: RVV				
Category	Name	Fmt	RV32M (Multiply-Divide)	+RV64M	Name	Fmt	RV32V/R64V		
Multiply	MULTIPLY	R	MUL rd,rs1,rs2	MULW rd,rs1,rs2	SET Vector Len.	R	SETVL rd,rs1		
	MULTIPLY High	R	MULH rd,rs1,rs2		MULTIPLY High	R	VMULH rd,rs1,rs2		
	MULTIPLY High Sign/Uns	R	MULHSU rd,rs1,rs2		REMAinder	R	VREM rd,rs1,rs2		
	MULTIPLY High Uns	R	MULHU rd,rs1,rs2		Shift Left Log.	R	VSLL rd,rs1,rs2		
Divide	DIVide	R	DIV rd,rs1,rs2	DIWV rd,rs1,rs2	Shift Right Log.	R	VSRL rd,rs1,rs2		
	DIVide Unsigned	R	DIVU rd,rs1,rs2		Shift R. Arith.	R	VSRA rd,rs1,rs2		
Remainder	REMAinder	R	REM rd,rs1,rs2	REMW rd,rs1,rs2	LoaD	I	VLD rd,rs1,imm		
	REMAinder Unsigned	R	REMU rd,rs1,rs2	REMUW rd,rs1,rs2	LoaD Strided	R	VLDS rd,rs1,rs2		
					LoaD indeXed	R	VLDX rd,rs1,rs2		
Optional Atomic Instruction Extension: RVA									
Category	Name	Fmt	RV32A (Atomic)	+RV64A	STore	S	VST rd,rs1,imm		
Load	Load Reserved	R	LR.W rd,rs1	LR.D rd,rs1	STore Strided	R	VSTS rd,rs1,rs2		
Store	Store Conditional	R	SC.W rd,rs1,rs2	SC.D rd,rs1,rs2	STore indeXed	R	VSTX rd,rs1,rs2		
Swap	SWAP	R	AMOSWAP.W rd,rs1,rs2	AMOSWAP.D rd,rs1,rs2	AMO SWAP	R	AMOSWAP rd,rs1,rs2		
Add	ADD	R	AMOADD.W rd,rs1,rs2	AMOADD.D rd,rs1,rs2	AMO ADD	R	AMOADD rd,rs1,rs2		
Logical	XOR	R	AMOXOR.W rd,rs1,rs2	AMOXOR.D rd,rs1,rs2	AMO XOR	R	AMOXOR rd,rs1,rs2		
	AND	R	AMOAND.W rd,rs1,rs2	AMOAND.D rd,rs1,rs2	AMO AND	R	AMOAND rd,rs1,rs2		
	OR	R	AMOOR.W rd,rs1,rs2	AMOOR.D rd,rs1,rs2	AMO OR	R	AMOOR rd,rs1,rs2		
					AMO MINimum	R	AMOMIN rd,rs1,rs2		
Min/Max	MINimum	R	AMOMIN.W rd,rs1,rs2	AMOMIN.D rd,rs1,rs2	AMO MAXimum	R	AMOMAX 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	Predicate =	R	VPEQ rd,rs1,rs2		
	MAXimum Unsigned	R	AMOMAXU.W rd,rs1,rs2	AMOMAXU.D 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		
Two Optional Floating-Point Instruction Extensions: RVF & RVD					MOVE	R	VMOV rd,rs1		
Category	Name	Fmt	RV32{F D} (SP,DP Fl. Pt.)	+RV64{F D}	ConVerT	R	VCVT rd,rs1		
Move	Move from Integer	R	FMV.W.X rd,rs1	FMV.D.X rd,rs1	ADD	R	VADD rd,rs1,rs2		
	Move to Integer	R	FMV.X.W rd,rs1	FMV.X.D rd,rs1	SUBtract	R	VSUB rd,rs1,rs2		
Convert	ConVerT from Int	R	FCVT.{S D}.W rd,rs1	FCVT.{S D}.L rd,rs1	MULTIPLY	R	VMUL rd,rs1,rs2		
	ConVerT from Int Unsigned	R	FCVT.{S D}.WU rd,rs1	FCVT.{S D}.LU rd,rs1	DIVide	R	VDIV rd,rs1,rs2		
	ConVerT to Int	R	FCVT.W.{S D} rd,rs1	FCVT.L.{S D} rd,rs1	SQuare RooT	R	VSQRT rd,rs1,rs2		
	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	Calling Convention					
Store	Store	S	FS{W,D} rs1,rs2,imm	Register	ABI Name	Saver			
Arithmetic	ADD	R	FADD.{S D} rd,rs1,rs2	x0	zero	---			
	SUBtract	R	FSUB.{S D} rd,rs1,rs2	x1	ra	Caller			
	MULTIPLY	R	FMUL.{S D} rd,rs1,rs2	x2	sp	Callee			
	DIVide	R	FDIV.{S D} rd,rs1,rs2	x3	gp	---			
	SQuare RooT	R	FSQRT.{S D} rd,rs1	x4	tp	---			
Mul-Add	Multiply-ADD	R	FMADD.{S D} rd,rs1,rs2,rs3	x5-7	t0-2	Caller	Multiply-ADD	R	VFMADD rd,rs1,rs2,rs3
	Multiply-SUBtract	R	FMSUB.{S D} rd,rs1,rs2,rs3	x8	s0/fp	Callee	Multiply-SUB	R	VFMSUB rd,rs1,rs2,rs3
	Negative Multiply-SUBtract	R	FNMSUB.{S D} rd,rs1,rs2,rs3	x9	s1	Callee	Neg. Mul.-SUB	R	VFNMSUB rd,rs1,rs2,rs3
Sign Inject	Negative Multiply-ADD	R	FNMADD.{S D} rd,rs1,rs2,rs3	x10-11	a0-1	Caller	Neg. Mul.-ADD	R	VFNMADD rd,rs1,rs2,rs3
	SiGN source	R	FSGNJ.{S D} rd,rs1,rs2	x12-17	a2-7	Caller	SiGN inJect	R	VSGNJ rd,rs1,rs2
	Negative SiGN source	R	FSGNJN.{S D} rd,rs1,rs2	x18-27	s2-11	Callee	Neg SiGN inJect	R	VSGNJN rd,rs1,rs2
Min/Max	Xor SiGN source	R	FSGNJX.{S D} rd,rs1,rs2	x28-31	t3-t6	Caller	Xor SiGN inJect	R	VSGNJX rd,rs1,rs2
	MINimum	R	FMIN.{S D} rd,rs1,rs2	f0-7	ft0-7	Caller	MINimum	R	VMIN rd,rs1,rs2
	MAXimum	R	FMAX.{S D} rd,rs1,rs2	f8-9	fs0-1	Callee	MAXimum	R	VMAX rd,rs1,rs2
Compare	compare Float =	R	FEQ.{S D} rd,rs1,rs2	f10-11	fa0-1	Caller	XOR	R	VXOR rd,rs1,rs2
	compare Float <	R	FLT.{S D} rd,rs1,rs2	f12-17	fa2-7	Caller	OR	R	VOR rd,rs1,rs2
	compare Float ≤	R	FLE.{S D} rd,rs1,rs2	f18-27	fs2-11	Callee	AND	R	VAND rd,rs1,rs2
Categorize	CLASSify type	R	FCLASS.{S D} rd,rs1	f28-31	ft8-11	Caller	CLASS	R	VCLASS rd,rs1
Configure	Read Status	R	FRCSR rd	zero	Hardwired zero		SET Data Conf.	R	VSETDCFG rd,rs1
	Read Rounding Mode	R	FRRM rd	ra	Return address		EXTRACT	R	VEXTRACT rd,rs1,rs2
	Read Flags	R	FRFLAGS rd	sp	Stack pointer		MERGE	R	VMERGE rd,rs1,rs2
	Swap Status Reg	R	FSCSR rd,rs1	gp	Global pointer		SELECT	R	VSELECT rd,rs1,rs2
	Swap Rounding Mode	R	FSRM rd,rs1	tp	Thread pointer				
	Swap Flags	R	FSFLAGS rd,rs1	t0-6,ft0-11	Temporaries				
	Swap Rounding Mode Imm	I	FSRMI rd,imm	s0-11,fs0-11	Saved registers				
	Swap Flags Imm	I	FSFLAGSI rd,imm	a0-7,fa0-7	Function args				

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.