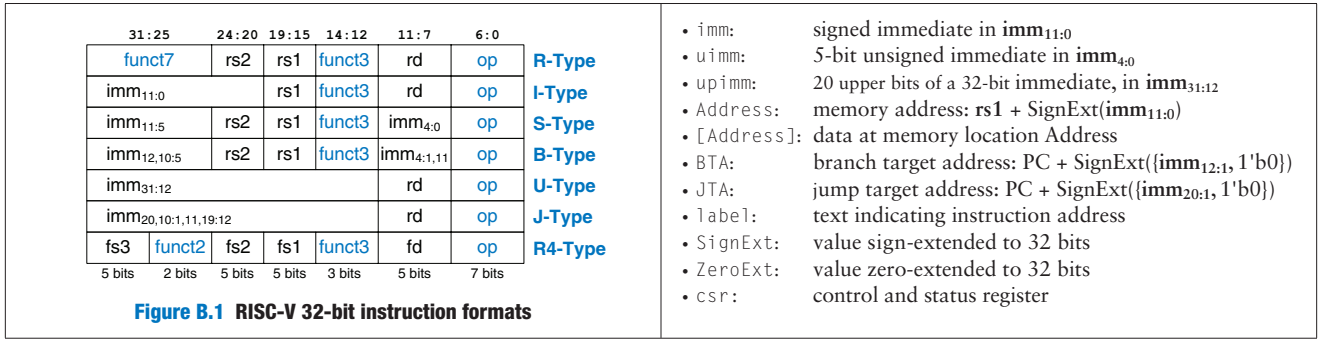


# RISC-V Instruction Set Summary



**Table B.1 RV32I: RISC-V integer instructions**

op	funct3	funct7	Type	Instruction	Description	Operation
0000011 (3)	000	–	I	lb rd, imm(rs1)	load byte	rd = SignExt([Address] <sub>7:0</sub> )
0000011 (3)	001	–	I	lh rd, imm(rs1)	load half	rd = SignExt([Address] <sub>15:0</sub> )
0000011 (3)	010	–	I	lw rd, imm(rs1)	load word	rd = [Address] <sub>31:0</sub>
0000011 (3)	100	–	I	lbu rd, imm(rs1)	load byte unsigned	rd = ZeroExt([Address] <sub>7:0</sub> )
0000011 (3)	101	–	I	lhu rd, imm(rs1)	load half unsigned	rd = ZeroExt([Address] <sub>15:0</sub> )
0010011 (19)	000	–	I	addi rd, rs1, imm	add immediate	rd = rs1 + SignExt(imm)
0010011 (19)	001	0000000*	I	slli rd, rs1, uimm	shift left logical immediate	rd = rs1 << uimm
0010011 (19)	010	–	I	slti rd, rs1, imm	set less than immediate	rd = (rs1 < SignExt(imm))
0010011 (19)	011	–	I	sltiu rd, rs1, imm	set less than imm. unsigned	rd = (rs1 < SignExt(imm))
0010011 (19)	100	–	I	xori rd, rs1, imm	xor immediate	rd = rs1 ^ SignExt(imm)
0010011 (19)	101	0000000*	I	srli rd, rs1, uimm	shift right logical immediate	rd = rs1 >> uimm
0010011 (19)	101	0100000*	I	srai rd, rs1, uimm	shift right arithmetic imm.	rd = rs1 >>> uimm
0010011 (19)	110	–	I	ori rd, rs1, imm	or immediate	rd = rs1   SignExt(imm)
0010011 (19)	111	–	I	andi rd, rs1, imm	and immediate	rd = rs1 & SignExt(imm)
0010111 (23)	–	–	U	auipc rd, upimm	add upper immediate to PC	rd = {upimm, 12'b0} + PC
0100011 (35)	000	–	S	sb rs2, imm(rs1)	store byte	[Address] <sub>7:0</sub> = rs2 <sub>7:0</sub>
0100011 (35)	001	–	S	sh rs2, imm(rs1)	store half	[Address] <sub>15:0</sub> = rs2 <sub>15:0</sub>
0100011 (35)	010	–	S	sw rs2, imm(rs1)	store word	[Address] <sub>31:0</sub> = rs2
0110011 (51)	000	0000000	R	add rd, rs1, rs2	add	rd = rs1 + rs2
0110011 (51)	000	0100000	R	sub rd, rs1, rs2	sub	rd = rs1 – rs2
0110011 (51)	001	0000000	R	sll rd, rs1, rs2	shift left logical	rd = rs1 << rs2 <sub>4:0</sub>
0110011 (51)	010	0000000	R	slt rd, rs1, rs2	set less than	rd = (rs1 < rs2)
0110011 (51)	011	0000000	R	sltu rd, rs1, rs2	set less than unsigned	rd = (rs1 < rs2)
0110011 (51)	100	0000000	R	xor rd, rs1, rs2	xor	rd = rs1 ^ rs2
0110011 (51)	101	0000000	R	srl rd, rs1, rs2	shift right logical	rd = rs1 >> rs2 <sub>4:0</sub>
0110011 (51)	101	0100000	R	sra rd, rs1, rs2	shift right arithmetic	rd = rs1 >>> rs2 <sub>4:0</sub>
0110011 (51)	110	0000000	R	or rd, rs1, rs2	or	rd = rs1   rs2
0110011 (51)	111	0000000	R	and rd, rs1, rs2	and	rd = rs1 & rs2
0110111 (55)	–	–	U	lui rd, upimm	load upper immediate	rd = {upimm, 12'b0}
1100011 (99)	000	–	B	beq rs1, rs2, label	branch if =	if (rs1 == rs2) PC = BTA
1100011 (99)	001	–	B	bne rs1, rs2, label	branch if ≠	if (rs1 ≠ rs2) PC = BTA
1100011 (99)	100	–	B	blt rs1, rs2, label	branch if <	if (rs1 < rs2) PC = BTA
1100011 (99)	101	–	B	bge rs1, rs2, label	branch if ≥	if (rs1 ≥ rs2) PC = BTA
1100011 (99)	110	–	B	bltu rs1, rs2, label	branch if < unsigned	if (rs1 < rs2) PC = BTA
1100011 (99)	111	–	B	bgeu rs1, rs2, label	branch if ≥ unsigned	if (rs1 ≥ rs2) PC = BTA
1100111 (103)	000	–	I	jalr rd, rs1, imm	jump and link register	PC = rs1 + SignExt(imm), rd = PC + 4
1101111 (111)	–	–	J	jal rd, label	jump and link	PC = JTA, rd = PC + 4

\*Encoded in instr<sub>31:25</sub>, the upper seven bits of the immediate field

**Table B.2 RV64I: Extra integer instructions**

op	funct3	funct7	Type	Instruction	Description	Operation
0000011 (3)	011	–	I	ld rd, imm(rs1)	load double word	rd = [Address] <sub>63:0</sub>
0000011 (3)	110	–	I	lwu rd, imm(rs1)	load word unsigned	rd = ZeroExt([Address] <sub>31:0</sub> )
0011011 (27)	000	–	I	addiw rd, rs1, imm	add immediate word	rd = SignExt((rs1 + SignExt(imm)) <sub>31:0</sub> )
0011011 (27)	001	0000000	I	slliw rd, rs1, uimm	shift left logical immediate word	rd = SignExt((rs1 <sub>31:0</sub> << uimm) <sub>31:0</sub> )
0011011 (27)	101	0000000	I	srliw rd, rs1, uimm	shift right logical immediate word	rd = SignExt((rs1 <sub>31:0</sub> >> uimm) <sub>31:0</sub> )
0011011 (27)	101	0100000	I	sraiw rd, rs1, uimm	shift right arith. immediate word	rd = SignExt((rs1 <sub>31:0</sub> >>> uimm) <sub>31:0</sub> )
0100011 (35)	011	–	S	sd rs2, imm(rs1)	store double word	[Address] <sub>63:0</sub> = rs2
0111011 (59)	000	0000000	R	addw rd, rs1, rs2	add word	rd = SignExt((rs1 + rs2) <sub>31:0</sub> )
0111011 (59)	000	0100000	R	subw rd, rs1, rs2	subtract word	rd = SignExt((rs1 – rs2) <sub>31:0</sub> )
0111011 (59)	001	0000000	R	sllw rd, rs1, rs2	shift left logical word	rd = SignExt((rs1 <sub>31:0</sub> << rs2 <sub>4:0</sub> ) <sub>31:0</sub> )
0111011 (59)	101	0000000	R	srlw rd, rs1, rs2	shift right logical word	rd = SignExt((rs1 <sub>31:0</sub> >> rs2 <sub>4:0</sub> ) <sub>31:0</sub> )
0111011 (59)	101	0100000	R	sraw rd, rs1, rs2	shift right arithmetic word	rd = SignExt((rs1 <sub>31:0</sub> >>> rs2 <sub>4:0</sub> ) <sub>31:0</sub> )

In RV64I, registers are 64 bits, but instructions are still 32 bits. The term “word” generally refers to a 32-bit value. In RV64I, immediate shift instructions use 6-bit immediates: uimm<sub>5:0</sub>; but for word shifts, the most significant bit of the shift amount (uimm<sub>5</sub>) must be 0. Instructions ending in “w” (for “word”) operate on the lower half of the 64-bit registers. Sign- or zero-extension produces a 64-bit result.

**Table B.3 RVF/D: RISC-V single- and double-precision floating-point instructions**

op	funct3	funct7	rs2	Type	Instruction	Description	Operation
1000011 (67)	rm	fs3, fmt	–	R4	fmadd fd, fs1, fs2, fs3	multiply-add	fd = fs1 * fs2 + fs3
1000111 (71)	rm	fs3, fmt	–	R4	fmsub fd, fs1, fs2, fs3	multiply-subtract	fd = fs1 * fs2 – fs3
1001011 (75)	rm	fs3, fmt	–	R4	fnmsub fd, fs1, fs2, fs3	negate multiply-add	fd = –(fs1 * fs2 + fs3)
1001111 (79)	rm	fs3, fmt	–	R4	fnmadd fd, fs1, fs2, fs3	negate multiply-subtract	fd = –(fs1 * fs2 – fs3)
1010011 (83)	rm	00000, fmt	–	R	fadd fd, fs1, fs2	add	fd = fs1 + fs2
1010011 (83)	rm	00001, fmt	–	R	fsub fd, fs1, fs2	subtract	fd = fs1 – fs2
1010011 (83)	rm	00010, fmt	–	R	fmul fd, fs1, fs2	multiply	fd = fs1 * fs2
1010011 (83)	rm	00011, fmt	–	R	fdiv fd, fs1, fs2	divide	fd = fs1 / fs2
1010011 (83)	rm	01011, fmt	00000	R	fsqrt fd, fs1	square root	fd = sqrt(fs1)
1010011 (83)	000	00100, fmt	–	R	fsgnj fd, fs1, fs2	sign injection	fd = fs1, sign = sign(fs2)
1010011 (83)	001	00100, fmt	–	R	fsgnjn fd, fs1, fs2	negate sign injection	fd = fs1, sign = –sign(fs2)
1010011 (83)	010	00100, fmt	–	R	fsgnjx fd, fs1, fs2	xor sign injection	fd = fs1, sign = sign(fs2) ^ sign(fs1)
1010011 (83)	000	00101, fmt	–	R	fmin fd, fs1, fs2	min	fd = min(fs1, fs2)
1010011 (83)	001	00101, fmt	–	R	fmax fd, fs1, fs2	max	fd = max(fs1, fs2)
1010011 (83)	010	10100, fmt	–	R	feq rd, fs1, fs2	compare =	rd = (fs1 == fs2)
1010011 (83)	001	10100, fmt	–	R	flt rd, fs1, fs2	compare <	rd = (fs1 < fs2)
1010011 (83)	000	10100, fmt	–	R	fle rd, fs1, fs2	compare ≤	rd = (fs1 ≤ fs2)
1010011 (83)	001	11100, fmt	00000	R	fclass rd, fs1	classify	rd = classification of fs1
RVF only							
0000111 (7)	010	–	–	I	flw fd, imm(rs1)	load float	fd = [Address] <sub>31:0</sub>
0100111 (39)	010	–	–	S	fsw fs2, imm(rs1)	store float	[Address] <sub>31:0</sub> = fd
1010011 (83)	rm	1100000	00000	R	fcvt.w.s rd, fs1	convert to integer	rd = integer(fs1)
1010011 (83)	rm	1100000	00001	R	fcvt.wu.s rd, fs1	convert to unsigned integer	rd = unsigned(fs1)
1010011 (83)	rm	1101000	00000	R	fcvt.s.w fd, rs1	convert int to float	fd = float(rs1)
1010011 (83)	rm	1101000	00001	R	fcvt.s.wu fd, rs1	convert unsigned to float	fd = float(rs1)
1010011 (83)	000	1110000	00000	R	fmv.x.w rd, fs1	move to integer register	rd = fs1
1010011 (83)	000	1111000	00000	R	fmv.w.x fd, rs1	move to f.p. register	fd = rs1
RVD only							
0000111 (7)	011	–	–	I	fld fd, imm(rs1)	load double	fd = [Address] <sub>63:0</sub>
0100111 (39)	011	–	–	S	fsd fs2, imm(rs1)	store double	[Address] <sub>63:0</sub> = fd
1010011 (83)	rm	1100001	00000	R	fcvt.w.d rd, fs1	convert to integer	rd = integer(fs1)
1010011 (83)	rm	1100001	00001	R	fcvt.wu.d rd, fs1	convert to unsigned integer	rd = unsigned(fs1)
1010011 (83)	rm	1101001	00000	R	fcvt.d.w fd, rs1	convert int to double	fd = double(rs1)
1010011 (83)	rm	1101001	00001	R	fcvt.d.wu fd, rs1	convert unsigned to double	fd = double(rs1)
1010011 (83)	rm	0100000	00001	R	fcvt.s.d fd, fs1	convert double to float	fd = float(fs1)
1010011 (83)	rm	0100001	00000	R	fcvt.d.s fd, fs1	convert float to double	fd = double(fs1)

fs1, fs2, fs3, fd: floating-point registers. fs1, fs2, and fd are encoded in fields rs1, rs2, and rd; only R4-type also encodes fs3. ffmt: precision of computational instruction (single=00<sub>2</sub>, double=01<sub>2</sub>, quad=11<sub>2</sub>). rm: rounding mode (0=to nearest, 1=toward zero, 2=down, 3=up, 4=to nearest (max magnitude), 7=dynamic). sign(fs1): the sign of fs1.