RISC-V REFERENCE CARD

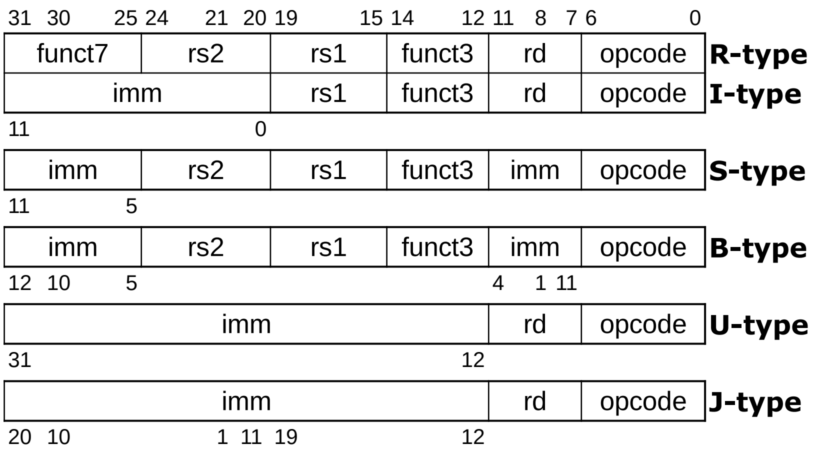
Version 20240411-rev.1

https://github.com/GnicoJP/RISC-V-Reference-Card

RV32I Standard Instruction Set

	Name	T	Opcode	F3	F7	Imm	Description
add	Add	R	0110011	0x0	0x00		rd ← rs1 + rs2
sub	Sub	R	0110011	0x0	0x20		rd ← rs1 − rs2
xor	Xor	R	0110011	0x4	0x00		rd ← rs1 ^ rs2
or	Or	R	0110011	0x6	0x00		rd ← rs1 | rs2
and	And	R	0110011	0x7	0x00		rd ← rs1 & rs2
sll	Shift Left Logical	R	0110011	0x1	0x00		rd ← rs1 << rs2
srl	Shift Right Logical	R	0110011	0x5	0x00		rd ← (unsigned)rs1 >> rs2
sra	Shift Right Arith	R	0110011	0x5	0x20		rd ← (signed)rs1 >> rs2
slt	Set Less Than	R	0110011	0x2	0x00		rd ← (singed)rs1 < (signed)rs2
sltu	Set Less Than Unsigned	R	0110011	0x3	0x00		rd ← (unsigned)rs1 < (singed)rs2
addi	Add Immediate	I	0010011	0x0			rd ← rs1 + #imm
xori	Xor Immediate	I	0010011	0x4			rd ← rs1 ^ #imm
ori	Or Immediate	I	0010011	0x6			rd ← rs1 | #imm
andi	And Immediate	I	0010011	0x7			rd ← rs1 & #imm
slli	Shift Left Logical Imm	I	0010011	0x1		[6:11]=0x00	rd ← rs1 << #imm[0:4]
srli	Shift Right Logical Imm	I	0010011	0x5		[6:11]=0x00	rd ← (unsigned)rs1 >> #imm[0:4]
srai	Shift Right Arith Imm	I	0010011	0x5		[6:11]=0x10	rd ← (signed)rs1 >> #imm[0:4]
slti	Set Less Than Imm	I	0010011	0x2			rd ← (signed)rs1 < #imm[0:4]
sltiu	Set Less Than Imm U	I	0010011	0x3			rd ← (unsigned)rs1 < #imm[0:4]
lb	Load Byte	I	0000011	0x0			rd ← Mem[rs1+#imm][0:7]
lh	Load Half	I	0000011	0x1			rd ← (signed)Mem[rs1+#imm][0:15]
lw	Load Word	I	0000011	0x2			rd ← (signed)Mem[rs1+#imm][0:31]
lbu	Load Byte Unsigned	I	0000011	0x4			rd ← (unsigned)Mem[rs1+#imm][0:7]
lhu	Load Half Unsigned	I	0000011	0x5			rd ← (unsigned)Mem[rs1+#imm][0:15]
sb	Store Byte	S	0100011	0x0			Mem[rs1+#imm][0:7] = rs2[0:7]
sh	Store Half	S	0100011	0x1			Mem[rs1+#imm][0:15] = rs2[0:15]
sw	Store Word	S	0100011	0x2			Mem[rs1+#imm][0:31] = rs2[0:31]
beq	Branch ==	B	1100011	0x0			if(rs1 == rs2) PC += #imm
bneq	Branch !=	B	1100011	0x1			if(rs1 != rs2) PC += #imm
blt	Branch <	B	1100011	0x4			if((signed)rs1 < rs2) PC += #imm
bge	Branch ≥	B	1100011	0x5			if((signed)rs1 >= rs2) PC += #imm
bltu	Branch < Unsigned	B	1100011	0x6			if((unsigned)rs1 < rs2) PC += #imm
bgeu	Branch ≥ Unsigned	B	1100011	0x7			if((unsinged) rs1 >= rs2) PC += #imm
jal	Jump And Link	J	1101111				rd = PC+4; PC += #imm
jalr	Jump And Link Register	I	1100111	0x0			rd = PC+4; PC = rs1 + #imm
lui	Load Upper Imm	U	0110111				rd = #imm << 12
auipc	Add Upper Imm to PC	U	0010111				rd = PC + (#imm << 12)
ecall	Environment Call	I	1110011	0x0		0x0	Transfer control to the OS
ebreak	Environment Break	I	1110011	0x0		0x1	Transfer control to the debugger

Format



Registers

(x0 – x15 are available on RV32E)

Name	ABI	Description	Saver
x0	zero	Zero	-
x1	ra	Return address	Caller
x2	sp	Stack pointer	Cellee
x3	gp	Global pointer	-
x4	tp	Thread pointer	-
x5 - x7	t0 - t2	Temporary Values	Caller
x8	s0 / fp	(Frame pointer)	Callee
x9	s1	Callee-saved	Callee
x10, x11	a0, a1	Args/Return Values	Caller
x12 - x17	a0 - a7	Arguments	Caller
x18 - x27	s2 - s11	Callee-saved	Callee
x28 - x31	t3 - t6	Temporary Values	Caller

FENCE Instruction

N/As are reserved.

31 28	27	26	25	24	23	22	21	20	19 15	14 12	11 7	6 0
fm	PI	PO	PR	PW	SI	SO	SR	SW	rs1	funct3	rd	opcode
x000									N/A	111	N/A	0001111

P : Predecessor S : Successor
R : Read W : Write
I : Input O : Output
FENCE.TSO is {fm=1000, RW, RW}
PAUSE is {fm=0000, W, 0}

RV64I Standard Instruction Set

	Name	T	Opcode	F3	F7	Imm	Description
addw	Add Word	R	0011011	0x0	0x00		rd ← (int32_t)(rs1 + rs2)
subw	Sub Word	R	0011011	0x0	0x20		rd ← (int32_t)(rs1 − rs2)
sllw	Shift Left Logical Word	R	0011011	0x1	0x00		rd ← (int32_t)(rs1 << rs2)
srlw	Shift Right Logical Word	R	0011011	0x5	0x00		rd ← (int32_t)((unsigned)rs1 >> rs2)
sraw	Shift Right Arith Word	R	0011011	0x5	0x20		rd ← (int32_t)((signed)rs1 >> rs2)
addiw	Add Immediate Word	I	0011011	0x0			rd ← (int32_t)rs1[31:0] + (int32_t)#imm
slliw	Shift Left Logical Imm	I	0011011	0x1	[6:11]=0x00		rd ← (int32_t)rs1[31:0] << #imm[0:5]
srliw	Shift Right Logical Imm	I	0011011	0x5	[6:11]=0x00		rd ← (uint32_t) ((int32_t)rs1[31:0] >> #imm[0:5])
sraiw	Shift Right Arith Imm	I	0011011	0x5	[6:11]=0x10		rd ← (int32_t)rs1[31:0] >> #imm[0:5]
ld	Load Double Word	I	0000011	0x3			rd ← Mem[rs1+#imm][0:63]
lwu	Load Word Unsigned	I	0000011	0x6			rd ← (unsigned)Mem[rs1+#imm][0:31]
sd	Store Double Word	S	0100011	0x3			Mem[rs1+#imm][0:63] = rs2[0:63]

A – Atomic Extension

31 27	26	25	24 20	19 15	14 12	11 7	6 0
funct5	AQ	RL	rs2	rs1	funct3	rd	opcode

.w : 2 → 32-bit
.d : 3 → 64-bit
.q : 4 → 128-bit

AQ Acquire : No following memory operations on this RISC-V thread can be observed to take place before the acquire memory operation.
RL Release : The release memory operation cannot be observed to take place before any earlier memory operations on this RISC-V thread.

Zalrsc Extension – Load Reserved/Store Conditional

	Name	Opcode	F5	Description
lr.w/d	Load Reserved	0101111	00010	rd ← Mem[rs1]; Reserve rs1
sc.w/d	Store Conditional	0101111	00011	if(Reserved) Mem[rs1] ← rs2; rd ← Reserved ? 0 : nonzero

Zaamo Extension – Atomic Memory Operations

	Name	Opcode	F5	Description
amoswap.w/d	Atomic Swap	0101111	00001	rd ← Mem[rs1] ← rs2
amoadd.w/d	Atomic Add	0101111	00000	rd ← Mem[rs1] + rs2
amoand.w/d	Atomic And	0101111	01100	rd ← Mem[rs1] & rs2
amoor.w/d	Atomic Or	0101111	01000	rd ← Mem[rs1] | rs2
amoxor.w/d	Atomic Xor	0101111	00100	rd ← Mem[rs1] ^ rs2
amomax.w/d	Atomic Max	0101111	10100	rd ← Max((signed)Mem[rs1], rs2)
amomaxu.w/d	Atomic Max U.	0101111	11100	rd ← Max((unsigned)Mem[rs1], rs2)
amomin.w/d	Atomic Min	0101111	10000	rd ← Min((signed)Mem[rs1], rs2)
amominu.w/d	Atomic Min U.	0101111	11000	rd ← Min((unsigned)Mem[rs1], rs2)

Zacas Extension – Atomic Compare-And-Swap (CAS) (Not included in A Extension)

	Name	Opcode	F5	Description

amocas.w/d/q	Atomic Compare-And-Swap	0101111	00101	temp = Mem[rs1]
				if (temp == rd)
				Mem[rs1] = rs2
				rd = temp

In RV32/64, amocas.d/q reads Mem[rs1] and Mem[rs1+1], and compares with rs2 and rs2+1, and results into rd and rd+1.

Zawrs Extension – Wait-on-Reservation-Set (Not included in A Extension)

31 20	19 15	14 12	11 7	6 0
funct12	rs1	funct3	rd	opcode
	0	0	0	1110011

	Name	F12	Description
wrs.nto	WRS with No TimeOut	0x0d	while(!Reserved);
wrs.sto	WRS with Short TimeOut	0x1d	while(!Reserved&&i++<N);

B – Bit Manipulation Extension

Zba Extension – Address Generation .u and .uw suffixed instructions are RV64 only.

	Name	T	Opcode	F3	F7	Imm	Description
add.uw	Add Unsigned Word	R	0111011	0x0	0x00		rd ← (rs2 + (uint32_t)rs1)
sh1add	Shift Left By 1 And Add	R	0110011	0x2	0x10		rd ← rs1 + (rs2 << 1)
sh1add.uw	Shift Unsigned Word Left By 1 And Add	R	0111011	0x2	0x10		rd ← (uint32_t)rs1 + (rs2 << 1)
sh2add	Shift Left By 2 And Add	R	0110011	0x4	0x10		rd ← rs1 + (rs2 << 2)
sh2add.uw	Shift Unsigned Word Left By 2 And Add	R	0111011	0x4	0x10		rd ← (uint32_t)rs1 + (rs2 << 2)
sh3add	Shift Left By 3 And Add	R	0110011	0x6	0x10		rd ← rs1 + (rs2 << 2)
sh3add.uw	Shift Unsigned Word Left By 3 And Add	R	0111011	0x6	0x10		rd ← (uint32_t)rs1 + (rs2 << 3)
zext.w	Add Unsigned Word	R	0111011	0x0	0x04		rd ← rs2 + (uint32_t)rs1[31:0]
slli.uw	Shift Left Unsigned Word Immediate	I	0011011	0x1		[6:11]=0x02	rd ← ((uint32_t)rs1[31:0] << #imm[0:5])

Zbb Extension – Basic bit-manipulation w suffixed instructions are RV64 only.

	Name	T	Opcode	F3	F7	Imm	Description
andn	And With Inverted Operand	R	0110011	0x7	0x20		rd ← rs1 & ~rs2
orn	Or With Inverted Operand	R	0110011	0x6	0x20		rd ← rs1 | ~rs2
xnor	Exclusive Nor	R	0110011	0x4	0x20		rd ← ~(rs1 ^ rs2)
clz	Count Leading Zero Bits	I	0010011	0x1		011000000000	rd ← CLZ(rs1)
clzw	clz In Word	I	0011011	0x1		011000000000	rd ← CLZ((uint32_t)rs1)
ctz	Count Trailing Zero Bits	I	0010011	0x1		011000000001	rd ← CTZ(rs1)
ctzw	ctz In Word	I	0011011	0x1		011000000001	rd ← CTZ((uint32_t)rs1)
cpop	Count Set Bits	I	0010011	0x1		011000000010	rd ← POPCNT(rs1)
cpopw	Count Set Bits In Word	I	0011011	0x1		011000000010	rd ← POPCNT((uint32_t)rs1)
max	Maximum	R	0110011	0x6	0x05		rd ← MAX((signed)rs1, rs2)
maxu	Unsigned Maximum	R	0110011	0x7	0x05		rd ← MAX((unsigned)rs1, rs2)
min	Minimum	R	0110011	0x4	0x05		rd ← MIN((signed)rs1, rs2)
minu	Unsigned Minimum	R	0110011	0x5	0x05		rd ← MIN((unsigned)rs1, rs2)
rol	Rotate Left	R	0110011	0x1	0x30		rd ← rs1 `rot<<` rs2
rolw	Rotate Left Word	R	0111011	0x1	0x30		rd ← (int32) ((uint32_t)rs1 `rot<<` rs2)
ror	Rotate Right	R	0110011	0x5	0x30		rd ← rs1 `rot>>` rs2
rori	Rotate Right Immediate	I	0010011	0x5		[6:11] = 011000	rd ← rs1 `rot>>` #imm
rorw	Rotate Right Word	R	0111011	0x5	0x30		rd ← (int32) ((uint32_t)rs1 `rot>>` rs2)
roriw	Rotate Right Word Imm	I	0011011	0x5		[6:11] = 011000	rd ← (int32) ((uint32_t)rs1 `rot>>` #imm)

orc.b	Bitwise Or-Combine	I	0010011	0x5		001010000111	rd[0:7] ← rs[0:7]>0 ? 255 : 0;
							rd[8:15] ← rs[8:15]>0 ? 255 : 0;
							...

rev8	Byte-reverse	I	0010011	0x7		011010011000 (RV32) 011010111000 (RV64)	REV: ABCD → DCBA (in RV32) rd ← REV(rs)

Zbc Extension – Carry-less multiplication xlen is the register width.

	Name	T	Opcode	F3	F7	Description

clmul	Carry-less multiply (low-part)	R	0110011	0x1	0x05	foreach (i = 0 to (xlen - 1) by 1)
						if((rs2 >> i) & 1) rd ← rd ^ (rs1 << i);

clmulh	Carry-less multiply (high-part)	R	0110011	0x3	0x05	foreach (i = 0 to (xlen - 1) by 1)
						if((rs2 >> i) & 1) rd ← rd ^ (rs1 >> (xlen - i));

clmulr	Carry-less multiply (reversed)	R	0110011	0x2	0x05	foreach (i = 0 to (xlen - 1) by 1)
						if((rs2 >> i) & 1) rd←rd^ (rs1>>(xlen-i-1));

Zbkc Extension – Carry-less multiplication for Cryptography

clmul and clmulh

Zbs Extension – Single-bit instructions

xlen is the register width. bits is log(2, xlen).

	Name	T	Opcode	F3	F7	Imm	Description
bclr	Single-Bit Clear	R	0110011	0x1	0x24		rd ← rs1 & ~(1 << (rs2 & (xlen-1)))
bext	Single-Bit Extract	R	0110011	0x5	0x24		rd ← (rs1 >> (rs2 & (xlen-1))) & 1
binv	Single-Bit Invert	R	0110011	0x1	0x34		rd ← rs1 ^ (1 << (rs2 & (xlen-1)))
bset	Single-Bit Set	R	0110011	0x1	0x14		rd ← rs1 | (1 << (rs2 & (xlen-1)))
bclri	Single-Bit Clear Imm	I	0010011	0x1	[6:11]=0x12		rd ← rs1 & ~(1 << #imm[bits-1:0])
bexti	Single-Bit Extract Imm	I	0010011	0x5	[6:11]=0x12		rd ← (rs1 >> #imm[bits-1:0]) & 1
binvi	Single-Bit Invert Imm	I	0010011	0x1	[6:11]=0x1A		rd ← rs1 ^ (1 << #imm[bits-1:0])
bseti	Single-Bit Set Imm	I	0010011	0x5	[6:11]=0x0A		rd ← rs1 | (1 << #imm[bits-1:0])

Zbkb Extension – Bit-manipulation for Cryptography

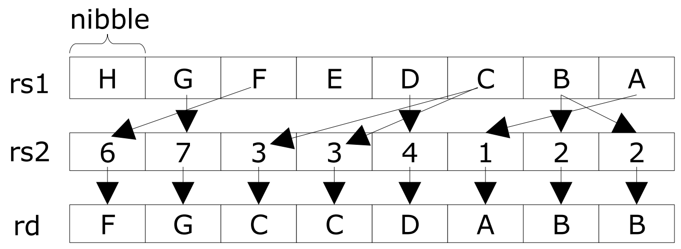
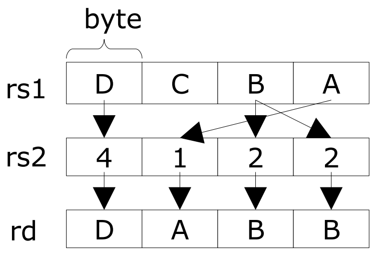
w suffixed instructions are RV64 only.

	Name	T	Opcode	F3	F7	Imm	Description
andn	And With Inverted Operand	R	0110011	0x7	0x20		rd ← rs1 & ~rs2
orn	Or With Inverted Operand	R	0110011	0x6	0x20		rd ← rs1 | ~rs2
xnor	Exclusive Nor	R	0110011	0x4	0x20		rd ← ~(rs1 ^ rs2)
pack	Pack Low Halves	R	0110011	0x4	0x04		rd ← (rs2 << xlen / 2) | (rs2 & (xlen/2 - 1))
packh	Pack Low Bytes	R	0110011	0x7	0x04		rd ← (unsigned)((rs2[0:7] << 8) | rs2[0:7])
packw	Pack Low 16-bits	R	0110011	0x4	0x04		rd ← (unsigned)((rs2[0:15]<<16)|rs2[0:15])
zip	Bit Interleave	I	0010011	0x1		000010011110	foreach(i = 0 to xlen/2-1) rd[2*i] ← rs[i] rd[2*i+1] ← rs[i+xlen/2]
unzip	Bit Deinterleave	I	0010011	0x5		000010011111	foreach(i = 0 to xlen/2-1) rd[i] ← rs[2*i] rd[i+xlen/2] ← rs[2*i+1]
rol	Rotate Left	R	0110011	0x1	0x30		rd ← rs1 \`rot<<\` rs2
rolw	Rotate Left Word	R	0111011	0x1	0x30		rd ← (int32) ((uint32_t)rs1 \`rot<<\` rs2)
ror	Rotate Right	R	0110011	0x5	0x30		rd ← rs1 \`rot>>\` rs2
rori	Rotate Right Immediate	I	0010011	0x5		[6:11] = 011000	rd ← rs1 \`rot>>\` #imm
rorw	Rotate Right Word	R	0111011	0x5	0x30		rd ← (int32) ((uint32_t)rs1 \`rot>>\` rs2)
roriw	Rotate Right Word Imm	I	0011011	0x5		[6:11] = 011000	rd ← (int32) ((uint32_t)rs1 \`rot>>\` #imm)

orc.b	Bitwise Or-Combine	I	0010011	0x5		001010000111	rd[0:7] ← rs[0:7]>0 ? 255 : 0;
							rd[8:15] ← rs[8:15]>0 ? 255 : 0;
							...

rev8	Byte-Reverse	I	0010011	0x7		011010011000 (RV32) 011010111000 (RV64)	REV: ABCD → DCBA (in RV32) rd ← REV(rs)
rev.b	Reverse Bits In Bytes	I	0010011	0x5		011010000111	RBB: ABCD → Rev(A)Rev(B)... rd ← RBB(rs)

Zbkx Extension – Crossbar permutations

	Name	T	Opcode	F3	F7	Description
xperm.n	Crossbar Permutation Nibbles	R	0110011	0x2	0x14	 nibble rs1 H G F E D C B A rs2 6 7 3 3 4 1 2 2 rd F G C C D A B B
xperm.b	Crossbar Permutation Bytes	R	0110011	0x4	0x14	

C – Compressed Extension

15 13	12	11 10	9 7	6 5	4 2	1 0	
funct4		rd/rs1		rs2		op	**CR-type**
funct3	imm[5]	rd/rs1		imm[4:0]		op	**CI-type**
funct3	imm[5:3|8:6]			rs2		op	**CSS-type**
funct3	imm[5:4|9:6|2|3]				rd′	op	**CIW-type**
funct3	imm[5:3]		rs1′	imm[2|6]	rd′	op	**CL-type**
funct3	imm[5:3]		rs1′	imm[7:6]	rd′	op	**CL*-type**
funct3	imm[5:3]		rs1′	imm[2|6]	rs2′	op	**CS-type**
funct3	imm[5:3]		rs1′	imm[7:6]	rs2′	op	**CS*-type**
funct6			rd′/rs1′	funct2	rs2′	op	**CA-type**
funct3	offset[8|4:3]		rd′/rs1′	offset[7:6|2:1|5]		op	**CB-type**
funct3	imm[5]	funct2	rd′/rs1′	imm[4:0]		op	**CB*-type**
funct3	target[11|4|9:8|10|6|7|3:1|5]					op	**CJ-type**

- The lower bits of the immediate values are filled by zero.
- Only the x8-x15 and f8-f15 registers are used in rd′, rs1′ and rs2′. i.e., only s0, s1, a0-a5 registers can be used.
- The least significant 2 bits of the instruction are not "11". If they are, it represents a 32-bit instruction.
- Some opcodes/functs are overlapped (c.ldsp&c.flwsp, c.sdsp&c.fswsp, c.jal&c.addiw).

	T	op	Funct	Misc.	Description	Extension
c.lwsp	CI	10	010	imm[5] and imm[4:2|7:6]	lw rd, offset(sp)	
c.ldsp	CI	10	011	imm[5] and imm[4:3|8:6]	ld rd, offset(sp)	RV64 only
c.flwsp	CI	10	011	imm[5] and imm[4:2|7:6]	flw rd, offset(sp)	RV32F only
c.fldsp	CI	10	001	imm[5] and imm[4:3|8:6]	fld rd, offset(sp)	D only
c.swsp	CSS	10	110		sw rs2, offset(sp)	
c.sdsp	CSS	10	111	imm[5:4|9:6]	sd rs2, offset(sp)	RV64 only
c.fswsp	CSS	10	111		fsw rs2, offset(sp)	RV32F only
c.fsdsp	CSS	10	101	imm[5:4|9:6]	fsd rs2, offset(sp)	D only
c.lw	CL	00	010		lw rd′, offset(rs1′)	
c.ld	CL*	00	011		ld rd′, offset(rs1′)	RV64 only
c.flw	CL	00	011		flw rd′, offset(rs1′)	RV32F only
c.fld	CL*	00	001		fld rd′, offset(rs1′)	D only
c.sw	CS	00	110		sw rs2′, offset(rs1′)	
c.sd	CS*	00	111		sd rs2′, offset(rs1′)	RV64 only
c.fsw	CS	00	111		fsw rs2′, offset(rs1′)	RV32F only
c.fsd	CS*	00	101		fsd rs2′, offset(rs1′)	D only
c.j	CJ	01	101		jal zero, offset	
c.jal	CJ	01	001		jal ra, offset	RV32 only
c.jr	CR	10	1000	rs2=0	jalr x0, 0(rs1)	
c.jalr	CR	10	1001	rs1 != 0, rs2 = 0	jalr ra, 0(rs1)	
c.beqz	CB	01	110		beq rs1′, zero, #offset	
c.bnez	CB	01	111		bne rs1′, zero, #offset	
c.li	CI	01	010		addi rd, zero, #imm	
c.lui	CI	01	011	imm[17] and imm[16:12]	lui rd, #imm	
c.addi	CI	01	000		addi rd, rd, #imm	
c.addiw	CI	01	001		addiw rd, rd, #imm	RV64 only
c.addi16sp	CI	01	011	rd=2, imm[9] and imm[4|6|8:7|5]	addi x2, x2, #imm	
c.addi4spn	CIW	00	000		addi rd′, x2, #imm	
c.slli	CI	10	000		slli rd, rd, #imm	
c.srli	CB*	01	funct3 = 100, funct2 = 00		srli rd′, rd′, #imm	
c.srai	CB*	01	funct3 = 100, funct2 = 01		srai rd′, rd′, #imm	
c.andi	CB*	01	funct3 = 100, funct2 = 10		ndi rd′, rd′, #imm	
c.mv	CR	10	1000	rs2 != 0	add rd, zero, rs2	
c.add	CR	10	1001	rs2 != 0	add rd, rd, rs2	
c.and	CA	01	funct6 = 100011, funct2 = 11		and rd′, rd′, rs2′	
c.or	CA	01	funct6 = 100011, funct2 = 10		or rd′, rd′, rs2′	
c.xor	CA	01	funct6 = 100011, funct2 = 01		xor rd′, rd′, rs2′	
c.sub	CA	01	funct6 = 100011, funct2 = 00		sub rd′, rd′, rs2′	
c.addw	CA	01	funct6 = 100111, funct2 = 01		andw rd′, rd′, rs2′	
c.subw	CA	01	funct6 = 100111, funct2 = 00		subw rd′, rd′, rs2′	
illegal	CR	00	0	rs1=rs1=0		
c.nop	CR	01	000	rs1=imm=0		
c.ebreak	CR	10	1001	rs1=rs2=0	ebreak	

F/D/Zfh/Q – Floating-Point Extension

Status Register

31	8 7	5 4	3	2	1	0	
reserved	Rounding Mode	NV	DZ	OF	UF	NX	**fcsr**

Registers
f0-f31
All of them are general registers.

NV: Invalid Operation
DZ: Divide by Zero
OF: Overflow
UF: Undeflow
NX: Inexact

Rounding mode encoding:

000	RNE	Round to Nearest, ties to Even
001	RTZ	Round towards Zero
010	RDN	Round Down (towards -∞)
011	RUP	Round Up (towards -∞)
100	RMM	Round to Nearest, ties to Max Magnitude
101		RESERVED
110		RESERVED
111	DYN	RESERVED: In instruction's rm field, selects dynamic rounding mode; In Rounding Mode register

Load/Store Instructions   fld and fsd are D extension only.

31 25	24 20	19 15	14 12	11 8 7	6 0	
imm	rs1	width	rd		opcode	**I-type**
11 0						

imm	rs2	rs1	width	imm	opcode	**S-type**
11 5				4 0		

Width = {w(32-bit) : 00, d(64-bit) : 01,
h(16-bit) : 10, q(128-bit) : 11}

	Name	T	Opcode	F7	Description
flw/d	Load Float/Double	I	0110011	0x01	rd ← Mem[rs1]
fsw/d	Store Float/Double	S	0110011	0x01	Mem[rs1] ← rs2

Register-Register Instructions

31 27	26 25	24 20	19 15	14 12	11 7	6 0
funct5	fmt	rs2	rs1	funct3	rd	1010011

Fmt = {S(32-bit) : 00, D(64-bit) : 01,
H(16-bit) : 10, Q(128-bit) : 11}

	Name	F5	F3	Misc.	Description
fadd.s/d	Floating Add	00000	rm		rd ← rs1+rs2
fsub.s/d	Floating Sub	00001	rm		rd ← rs1-rs2
fmul.s/d	Floating Multiplication	00010	rm		rd ← rs1*rs2
fdiv.s/d	Floating Division	00011	rm		rd ← rs1/rs2
fsqrt.s/d	Floating Sqrt	01011	rm	rs2 = 0	rd ← $\sqrt{rs1}$
fmin.s/d	Floating Min	00101	0x0		rd ← fmin(rs1, rs2)
fmax.s/d	Floating Max	00101	0x1		rd ← fmax(rs1, rs2)
fsgnj.s/d	Floating Sign-injection	00100	0x0		rd ← sign(rs2) ? -abs(rs1) : abs(rs1)
fsgnjn.s/d	Floating Sign-injection Not	00100	0x1		rd ← sign(rs2) ? abs(rs1) : -abs(rs1)
fsgnjx.s/d	Floating Sign-injection Xor	00100	0x2		rd ← rs1 * sign(rs2)
fmv.x.w	x Register ← f Register	10100	0x0	rs2 = 0	rd(x) ← rs1(f)
fmv.w.x	f Register ← x Register	10100	0x0	rs2 = 0	rd(f) ← rs1(x)
fmv.x.d	x Register ← f Register	11100	0x0	rs2 = 0	rd(x) ← rs1(f) (RV64 only)
fmv.d.x	f Register ← x Register	11100	0x0	rs2 = 0	rd(f) ← rs1(x) (RV64 only)
feq.s/d	Compare Floatings (EQ)	10100	0x2		rd ← rs1 == rs2
flt.s/d	Compare Floatings (LT)	10100	0x1		rd ← rs1 < rs2
fle.s/d	Compare Floatings (LE)	10100	0x0		rd ← rs1 <= rs2
fclass	Floating Classify	11110	0x0	rs2 = 0	rd ← classify(rs1)

rd bit	Meaning
0	-∞
1	≦ 0 normal number
2	≦ subnormal number
3	-0
4	+0
5	≧ 0 normal number
6	≧ 0 subnormal number
7	+∞
8	Signaling NaN
9	Quiet NaN

3-Operands Instructions

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
rs3	fmt	rs2	rs1	rm	rd	opcode	R4-type

	Name	Opcode	Description
fmadd	Fused-Multiply-Add	1000011	rd ← (rs1*rs2) + rs3
fmsub	Fused-Multiply-Sub	1000111	rd ← (rs1*rs2) - rs3
fnmadd	Fused-Multiply-Add Negated	1001111	rd ← -(rs1*rs2)+ rs3
fnmsub	Fused-Multiply-Sub Negated	1001011	rd ← -(rs1*rs2) - rs3

Convert Instructions

31 27	26 25	24 20	19 15	14 12	11 7	6 0
funct5	fmt	fmt2	rs1	rm	rd	1010011

funct5	Name	funct5
FCVT.int.float	Floating to Integer	11000
FCVT.float.int	Integer to Floating	11010
FCVT.float.float	Floating to Floating	01000

float format	Name	fmt/fmt2
.S	Single (32-bit)	00000
.D	Double (64-bit)	00001
.H	Half (16-bit)	00010
.Q	Quad (128-bit)	00011

int format	Name	fmt2
.W	Integer (32-bit)	00000
.WU	Unsigned Integer	00001
.L	Long (64-bit)	00010
.LU	Unsigned Long	00011

out-of-range neg. or -∞ will converted to INT_MIN.
out-of-range pos. or +∞ or NaN will converted to INT_MAX.

Zfa Extension – Additional Floating-Point Instructions

	Name	T	Opcode	F5	F3	Misc.	Description
fli.s/d	Load Immediate	R	1010011	10100	0x0	rs2 = 1	rd ← LUT(#rs1)
fround.s/d	Round without set status	R	1010011	01000	rm	rs2 = 0x4	rd ← round(rs1)
froundnx.s/d	Round with set NX	R	1010011	01000	rm	rs2 = 0x5	rd ← round(rs1)
fcvtmod.w.d	fcvt.w.d rounding towards zero	R	1010011	11100	0x0	rs2=1	rd(x) ← floor(rs1(f))
fmvh.x.d (D) fmvh.x.q (Q)	Move Floating to Integer (High)	R	1010011	11100	0x0	RV32D only RV64Q only rs2 = 1	rd(x) ← rs1(f)[32:63]
fmvp.d.x (D) fmvp.q.x (Q)	Move Integer to Floating (High)	R	1011001	11100	0x0	RV32D only RV64Q only	rd(x) ← rs2(f) << 32 | rs1(f)
fminm.s/d	Floating Min	R	1010011	00101	0x2	rd ← fmin(rs1, rs2)	If either input is NaN, it results in NaN.
fmaxm.s/d	Floating Max	R	1010011	00101	0x3	rd ← fmax(rs1, rs2)	
fltq.s/d	flt without set status	R	1010011	10100	0x2	rs2 = 0	rd ← classify(rs1)
fleq.s/d	fle without set status	R	1010011	10100	0x4	rs2 = 0	rd ← classify(rs1)

Zfinx Zdinx Zhinx Zinxmin Extension – Floating-Points in x registers

Available with Zicsr extension

List of Profiles

RVI20 optional:
- C
- M
- A
- D
- F
- Zihpm
- Zicntr
- Zifencei

RVA20U64 mandatory:
- RVI20 options
- Zicsr
- Ziccamoa
- Ziccrse
- Zicclsm
- Ziccif
- Za128rs

optional:
- Zihpm

RVA20S64 mandatory:
- RVA20U64 mandatory
- Ss1p11
- Svbare
- Sv39
- Svade
- Ssccpts
- Sstvecd
- Sstvala

optional:
- RVA20U64 options
- Zihpm
- Sv48
- Ssu64xl

RVA22U64 mandatory:
- RVA20U64 mandatory
- Zihintpause
- Zba
- Zbb
- Zbs
- Zihpm
- Zic64b
- Zicbom
- Zicbop
- Zicboz
- Zfhmin
- Zkt

optional:
- Zfh
- V
- Zkn
- Zks

RVA22U64 mandatory:
- RVA22U64 mandatory
- RVA20S64 mandatory
- Sscounterenw
- Svpbmt
- Svinval

optional:
- RVA22U64 options
- Sv48, Sv57
- Svnapot
- Ssu64xl
- Sstc
- Sscofpmf
- Zkr
- H (followings are mandatory if H is implemented.)
  - Ssstateen
  - Shcounterenw
  - Shvstvala
  - Shtvala
  - Shvstvecd
  - Shvsatpa
  - Shgatpa

M – Integer Multiplication and Division Extension

w suffixed instructions are RV64 only. **Zmmul** extension supports multiplications only.

	Name	T	Opcode	F3	F7	Description
mul	Multiplication	R	0110011	0x0	0x01	rd ← (rs1 * rs2)[0:xlen-1]
mulh	Signed × Signed High	R	0110011	0x1	0x01	rd ← (rs1 * rs2)[xlen:xlen*2-1]
mulhsu	Signed × Unsigned High	R	0110011	0x2	0x01	rd ← (rs1*(unsigned)rs2)[xlen:xlen*2-1]
mulhu	Unsigned × Unsigned High	R	0110011	0x3	0x01	rd← ((unsigned)rs1 * (unsigned)rs2)[xlen:xlen*2-1]
mulw	Multiplication Word	R	0111011	0x0	0x01	rd ← ((in32_t)rs1 * (int32_t)rs2)[0:31]
div	Division	R	0110011	0x4	0x01	rd ← rs1/rs2
divu	Division Unsigned	R	0110011	0x5	0x01	rd ← (unsigned)rs1/(unsigned)rs2
divw	Division Word	R	0111011	0x4	0x01	rd ← (int32_t)rs1/(int32_t)rs2
divuw	Division Unsigned Word	R	0111011	0x5	0x01	rd ← (uint32_t)rs1/(uint32_t)rs2
rem	Remainder	R	0110011	0x6	0x01	rd ← rs1%rs2
remu	Remainder Unsigned	R	0110011	0x7	0x01	rd ← (unsigned)rs1%(unsigned)rs2
remw	Remainder Word	R	0111011	0x6	0x01	rd ← (int32_t)rs1%(int32_t)rs2
remuw	Remainder Unsigned Word	R	0111011	0x7	0x01	rd ← (uint32_t)rs1%(uint32_t)rs2