	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	func5	func2	rs2	rs1	func3	rd	opcode
R4	rs3	func2	rs2	rs1	func3	rd	opcode
I	imm[11:0]			rs1	func3	rd	opcode
S	imm[11:5]		rs2	rs1	func3	imm[4:0]	opcode
SB	rel[12 | 10:5]		rs2	rs1	func3	rel[4:1 | 11]	opcode
U	imm[31:12]					rd	opcode
UJ	rel[20 | 10:1 | 11 | 19:12]					rd	opcode
CSR	CSR			rs1	func3	rd	opcode

RV32I & RV64I (40 + 12) *RV32E & RV64E only x0-x15 reg (x16-x31 remove)*

	Load Byte	I	LB	rd, offs(rs1)	*rd = sext(byte[rs1 + offs])*
	Load HWord	I	LH	rd, offs(rs1)	*rd = sext(hword[rs1 + offs*2])*
	Load Word	I	LW	rd, offs(rs1)	*rd = sext(word[rs1 + offs*4])*
RV64	*Load DWord*	I	LD	rd, offs(rs1)	*rd = sext(dword[rs1 + offs*8])*
	Load Byte Unsigned	I	LBU	rd, offs(rs1)	*rd = zext(byte[rs1 + offs])*
	Load HWord Unsigned	I	LHU	rd, offs(rs1)	*rd = zext(hword[rs1 + offs*2])*
RV64	*Load Word Unsigned*	I	LWU	rd, offs(rs1)	*rd = zext(word[rs1 + offs*4])*
	Store Byte	S	SB	rs2, offs(rs1)	*byte[rs1 + offs] = rs2[7:0]*
	Store HWord	S	SH	rs2, offs(rs1)	*hword[rs1 + offs*2] = rs2[15:0]*
	Store Word	S	SW	rs2, offs(rs1)	*word[rs1 + offs*4] = rs2[31:0]*
RV64	*Store DWord*	S	SD	rs2, offs(rs1)	*dword[rs1 + offs*8] = rs2[63:0]*
	Load Upper Imm	U	LUI	rd, imm	*rd = (imm << 12)*
	Add Upper Imm to PC	U	AUIPC	rd, imm	*rd = (imm << 12) + PC*
	ADD,SUB,XOR,OR,AND	R	<op>	rd, rs1, rs2	*rd = rs1 <op> rs2*
RV64	ADDW, SUBW	R	<op>	rd, rs1, rs2	*rd = sext(rs1[31:0] <op> rs2[31:0])*
	ADDI	I	<op>	rd, rs1, imm	*rd = rs1 <op> sext(imm)*
RV64	ADDIW	I	<op>	rd, rs1, imm	*rd = sext(rs1[31:0] <op> sext(imm))*
	XORI, ORI, ANDI	I	<op>	rd, rs1, imm	*rd = rs1 <op> imm*
	SLL, SRL,SRA	R	<op>	rd, rs1, rs2	*rd = rs1 <shift> on rs2 count*
RV64	SLLW, SRLW, SRAW	R	<op>	rd, rs1, rs2	*rd = sext(rs1[31:0] <shift> on rs2 count)*
	SLLI, SRLI, SRAI	I	<op>	rd, rs1, imm	*rd = rs1 <shift> on imm count*
RV64	SLLIW,SRLIW,SRAIW	I	<op>	rd, rs1, imm	*rd = sext(rs1[31:0] <shift> on imm count)*
	Set if rs1 < rs2	R	SLT{U}	rd, rs1, rs2	*rd = 1 (if rs1 < rs2) / U -unsigned*
	Set if rs1 < imm	I	SLTI{U}	rd, rs1, imm	*rd = 1 (if rs1 < imm) / U -unsigned*
	Branch rs1 = rs2	SB	BEQ	rs1, rs2, rel	*if (rs1 = rs2) PC += rel / ±4K*
	Branch rs1 ≠ rs2	SB	BNE	rs1, rs2, rel	*if (rs1 ≠ rs2) PC += rel / ±4K*
	Branch rs1 < rs2	SB	BLT{U}	rs1, rs2, rel	*if (rs1 < rs2) PC += rel / ±4K*
	Branch rs1 ≥ rs2	SB	BGE{U}	rs1, rs2, rel	*if (rs1 ≥ rs2) PC += rel / ±4K*
	Jump & Link	UJ	JAL	rd, rel	*rd = PC + 4; PC = PC + rel / ±1M*
	Jump & Link Register	I	JALR	rd, imm(rs1)	*rd = PC + 4; PC = rs1 + sext(imm)*
	Synch thread	IF	FENCE		
	Environment Call	SYS	ECALL		
	Environment Break	SYS	EBREAK		*for debug*

Priveleged instruction

PRIV	*Wait For Interrupt*	SYS	WFI		

Zifencei (1)

PRIV		SYS	FENCE.I		

Zicsr - work with control registers (6)

	Atomic Read/Write	CSR	CSRRW	rd, rs1, csr	*rd = zext(csr); csr = rs1*
	Atomic Read and Set Bits	CSR	CSRRS	rd, rs1, csr	*rd = zext(csr); csr OR rs1*
	Atomic Read and Clear Bits	CSR	CSRRC	rd, rs1, csr	*rd = zext(csr); csr AND NOT rs1*
	Atomic Read/Write	CSR	CSRRWI	rd, rs1, uimm5	*rd = zext(csr); csr = uimm5*
	Atomic Read and Set Bits	CSR	CSRRSI	rd, rs1, uimm5	*rd = zext(csr); csr OR uimm5*
	Atomic Read and Clear Bits	CSR	CSRRCI	rd, rs1, uimm5	*rd = zext(csr); csr AND NOT uimm5*

RV32I & RV64I		Pseudoinstruction		Equivalent	
No operation		NOP		*ADDI*	*x0, x0, 0*
Load Immediate (Univ)		LI	rd, imm	*ADDI or LUI+ADDI*	
Load Address		LA	rd, label	*LUI+ADDI or AUIPC+ADDI*	
Load from Memory		L[BHWD]{U}	rd, label	*LUI+Load or AUIPC+Load*	
Store to Memory		S[BHWD]	rs1, label	*LUI+Store or AUIPC+Store*	
Move register		MV	rd, rs1	*ADDI*	*rd, rs1, 0*
Zero extend Byte		ZEXT.B	rd,rs1	*ANDI*	*rd, rs1, 0xFF*
Negate		NEG	rd, rs1	*SUB*	*rd, x0, rs1*
Logical NOT		NOT	rd, rs1	*XORI*	*rd, rs1, -1*
Set if rs1 = 0		SEQZ	rd, rs1	*SLTIU*	*rd, rs1, 1*
Set if rs1 ≠ 0		SNEZ	rd, rs1	*SLTU*	*rd, x0, rs1*
Set if rs1 < 0		SLTZ	rd, rs1	*SLT*	*rd, rs1, x0*
Set if rs1 > 0		SGTZ	rd, rs1	*SLT*	*rd, x0, rs1*
Branch if rs1 = 0		BEQZ	rs1, rel	*BEQ*	*rs1, x0, rel*
Branch if rs1 ≠ 0		BNEZ	rs1, rel	*BNE*	*rs1, x0, rel*
Branch if rs1 ≤ 0		BLEZ	rs1, rel	*BGE*	*x0, rs1, rel*
Branch if rs1 ≥ 0		BGEZ	rs1, rel	*BGE*	*rs1, x0, rel*
Branch if rs1 < 0		BLTZ	rs1, rel	*BLT*	*rs1, x0, rel*
Branch if rs1 > 0		BGTZ	rs1, rel	*BLT*	*x0, rs1, rel*
Branch if rs1 > rs2		BGT	rs1, rs2, rel	*BLT*	*rs2, rs1, rel*
Branch if rs1 ≤ rs2		BLE	rs1, rs2, rel	*BGE*	*rs2, rs1, rel*
Branch if rs1 > rs2 Unsign		BGTU	rs1, rs2, rel	*BLTU*	*rs2, rs1, rel*
Branch if rs1 ≤ rs2 Unsign		BLEU	rs1, rs2, rel	*BGEU*	*rs2, rs1, rel*
Return from proc		RET		*JALR*	*x0, 0(ra)*
Jump to		J	rel	*JAL*	*x0, rel*
Jump to proc		JR	rs1	*JALR*	*x0, 0(rs1)*
Jump to proc (univ)		CALL	label	*JAL or LUI+JALR or AUIPC+JALR*	
Jump to		TAIL	label	*JAL or LUI+JALR or AUIPC+JALR*	
RV64 *Sign Ext lowWord*		SEXT.W	rd, rs1	*ADDW*	*rd, rs1, x0*
RV64 *Negate lowWord*		NEGW	rd, rs1	*SUBW*	*rd, x0, rs1*
Synch thread	IF	FENCE.TSO		*FENCE*	*RW,RW*
Pause	IF	PAUSE		*FENCE*	*0*

Zicsr Pseudoinstructions

CSR only read		CSRR	rd, csr	*CSRRS*	*rd, csr, x0*
CSR only write		CSRW	csr, rs1	*CSRRW*	*x0, csr, rs1*
CSR write Imm		CSRWI	csr, uimm	*CSRRWI*	*x0, csr, uimm*
CSR set bits		CSRS	csr, rs1	*CSRRS*	*x0, csr, rs1*
CSR clear bits		CSRC	csr, rs1	*CSRRC*	*x0, csr, rs1*
CSR set bits Imm		CSRSI	csr, uimm	*CSRRSI*	*x0, csr, uimm*
CSR clear bits Imm		CSRCI	csr, uimm	*CSRRCI*	*x0, csr, uimm*

M - extensions (MUL / DIV / REM) include Zmmul extensions - all only MUL instructions

Multiplay		MUL	rd, rs1, rs2	*rd = low(rs1 * rs2)*	
RV64 *Multiplay Word*		MULW	rd, rs1, rs2	*rd = rs1 * rs2 (low words)*	
High from Mul		MULH{U}	rd, rs1, rs2	*rd = high(rs1 * rs2) (sign/unsign)*	
High from Mul		MULHSU	rd, rs1, rs2	*rd = high(sign(rs1) * unsign(rs2))*	
Divide		DIV{U}	rd, rs1, rs2	*rd = rs1* ***div*** *rs2 (sign/unsign)*	
RV64 *Divide Word*		DIV{U}W	rd, rs1, rs2	*rd = rs1* ***div*** *rs2 (sign/unsign) low words*	
Remainder		REM{U}	rd, rs1, rs2	*rd = rs1* ***mod*** *rs2 (sign/unsign)*	
RV64 *Remainder Word*		REM{U}W	rd, rs1, rs2	*rd = rs1 mod rs2 (sign/unsign) low words*	

15 14 13	12	11 10 9 8 7	6 5 4 3 2	1 0	
func4		rsd	rs2	op	CR
func3	imm	rsd	imm_5	op	CI
func3	imm_6		rs2	op	CSS
func3	imm_8		rd`	op	CIW
func3	imm_3	rs1`	imm_2 rd`	op	CL
func3	imm_3	rs1`	imm_2 rs2`	op	CS
func6		rsd`	func2 rs2`	op	CA
func3	offset_3	rsd`	offset_5	op	CB
func3	offset_11			op	CJ

C extension (base) Zca - new name group instruction (27 + 7)

	Load Word	CL	C.LW	rd`, offs( rs1`)	*LW*	*rd`, offs*4(rs1`)*
RV64	*Load DoubleWord*	CL	C.LD	rd`, offs( rs1`)	*LD*	*rd`, offs*8(rs1`)*
	Load Word SP	CI	C.LWSP	rd, offs(sp)	*LW*	*rd, offs*4(sp)*
RV64	*Load DoubleWord SP*	CI	C.LDSP	rd, offs(sp)	*LD*	*rd, offs*8(sp)*
	Store Word	CS	C.SW	rs2`, offs( rs1`)	*SW*	*rs1`, offs*4(rs2`)*
RV64	*Store Word*	CS	C.SD	rs2`, offs( rs1`)	*SD*	*rs1`, offs*8(rs2`)*
	Store Word SP	CSS	C.SWSP	rs2, offs(sp)	*SW*	*rs2, offs*4(sp)*
RV64	*Store Word SP*	CSS	C.SDSP	rs2, offs(sp)	*SD*	*rs2, offs*8(sp)*
	Load Immediate	CI	C.LI	rd, imm	*ADDI*	*rd, x0, imm*
	Load Upper Immediate	CI	C.LUI	rd, imm	*LUI*	*rd, imm*
	Move	CR	C.MV	rd, rs2	*ADD*	*rd, rs2, x0*
	ADD	CR	C.ADD	rsd, rs2	*ADD*	*rsd, rsd, rs2*
	ADD Immediate	CI	C.ADDI	rsd, imm	*ADDI*	*rsd, rsd, imm*
RV64	*ADD Immediate Word*	CI	C.ADDIW	rsd, imm	*ADDIW*	*rsd, rsd, imm*
	*ADD SP Imm*16*	CI	C.ADDI16SP	imm	*ADDI*	*sp, sp, imm*16*
	*ADD SP Imm*4*	CIW	C.ADDI4SPN	rd`, imm	*ADDI*	*rd`, sp, imm*4*
	Shift Left Immediate	CI	C.SLLI	rsd, imm	*SLLI*	*rd, rd, imm*
	SUB, OR, XOR, AND	CA	C.<op>	rsd`, rs1`	<op>	*rsd`, rsd`, rs1`*
RV64	ADDW, SUBW	CA	C.<op>	rsd`, rs1`	<op>	*rsd`, rsd`, rs1`*
	ANDI, SRAI, SRLI	CIW	C.<op>	rsd`, imm	<op>	*rsd`, imm*
	Branch = 0	CB	C.BEQZ	rs1`, rel	*BEQ*	*rs1`, x0, rel / ±256*
	Branch != 0	CB	C.BNEZ	rs1`, rel	*BNE*	*rs1`, x0, rel / ±256*
	Jump	CJ	C.J	rel	*JAL*	*x0, rel / ±2K*
	Jump Register	CR	C.JR	rs2	*JALR*	*x0, 0(rs2)*
	Jump & Link	CJ	C.JAL	rel	*JAL*	*ra, rel / ±2K*
	Jump & Link Register	CR	C.JALR	rs2	*JALR*	*ra, 0(rs2)*
	No Operation	CI	C.NOP		*ADDI*	*x0, x0, 0*
	Env. Break	CI	C.EBREAK		*EBREAK*	

Zcf include if F extension (not avaible RV64 - mapped C.LD, C.LDSP, C.SD, C.SDSP)

	Load Float	CL	C.FLW	rd`, offs( rs1`)	*FLD.S*	*rd`, offs*4(rs1`)*
	Load Float SP	CI	C.FLWSP	rd, offs(sp)	*FLD.S*	*rd, offs*4(sp)*
	Store Float	CS	C.FSW	rs2`, offs( rs1`)	*FST.S*	*rs1`, offs*4(rs2`)*
	Store Float SP	CSS	C.FSWSP	rs2, offs(sp)	*FST.S*	*rs2, offs*4(sp)*

Zcd include if D extension

	Load Float	CL	C.FLD	rd`, offs( rs1`)	*FLD.D*	*rd`, offs*8(rs1`)*
	Load Float SP	CI	C.FLDSP	rd, offs(sp)	*FLD.D*	*rd, offs*8(sp)*
	Store Float	CS	C.FSD	rs2`, offs( rs1`)	*FST.D*	*rs1`, offs*8(rs2`)*
	Store Float SP	CSS	C.FSDSP	rs2, offs(sp)	*FST.D*	*rs2, offs*8(sp)*

mark blue line *rd`, rs1`, rs2` - only x8 - x15 registers (s0,s1,a0 - a5)*