RISC-V REFERENCE

RISC-V Instruction Set

Core Instruction Formats

31	27	26	25	24	20	19		15	14	12	11	7	6	0	
	func	:t7		rs	52	1	rs1		fun	ct3		rd	opco	ode	R-type
	ir	nm[:	11:0)]		1	rs1		fun	ct3		rd	opco	ode	I-type
i	mm[1	1:5]		rs	s2	1	rs1		fun	ct3	1	n[4:0]	opco	ode	S-type
im	m[12	10:5	5]	rs	s2	1	rs1		fun	ct3	imm	[4:1 11]	opco	ode	B-type
	imm[31:12]								rd	opco	ode	U-type			
	imm[20 10:1 11 19:12]							rd	opco	ode	J-type				

RV32I Base Integer Instructions

Inst	Name	FMT	Opcode	funct3	funct7	Description (C)	Note
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 = rs1 >> rs2	
sra	Shift Right Arith*	R	0110011	0x5	0x20	rd = rs1 >> rs2	msb-extends
slt	Set Less Than	R	0110011	0x2	0x00	rd = (rs1 < rs2)?1:0	
sltu	Set Less Than (U)	R	0110011	0x3	0x00	rd = (rs1 < rs2)?1:0	zero-extends
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	imm[5:11]=0x00	rd = rs1 << imm[0:4]	
srli	Shift Right Logical Imm	I	0010011	0x5	imm[5:11]=0x00	rd = rs1 >> imm[0:4]	
srai	Shift Right Arith Imm	I	0010011	0x5	imm[5:11]=0x20	rd = rs1 >> imm[0:4]	msb-extends
slti	Set Less Than Imm	I	0010011	0x2		rd = (rs1 < imm)?1:0	
sltiu	Set Less Than Imm (U)	I	0010011	0x3		rd = (rs1 < imm)?1:0	zero-extends
lb	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
lh	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
lbu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
lhu	Load Half (U)	I	0000011	0x5		rd = M[rs1+imm][0:15]	zero-extends
sb	Store Byte	S	0100011	0x0		M[rs1+imm][0:7] = rs2[0:7]	
sh	Store Half	S	0100011	0x1		M[rs1+imm][0:15] = rs2[0:15]	
SW	Store Word	S	0100011	0x2		M[rs1+imm][0:31] = rs2[0:31]	
beq	Branch ==	В	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	В	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	В	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch ≥	В	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	В	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch \geq (U)	В	1100011	0x7		if(rs1 >= rs2) PC += imm	zero-extends
jal	Jump And Link	J	1101111			rd = PC+4; PC += imm	
jalr	Jump And Link Reg	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	imm=0x0	Transfer control to OS	
ebreak	Environment Break	I	1110011	0x0	imm=0x1	Transfer control to debugger	

Standard Extensions

RV32M Multiply Extension

Inst	Name	FMT	Opcode	funct3	funct7	Description (C)
mul	MUL	R	0110011	0x0	0x01	rd = (rs1 * rs2)[31:0]
mulh	MUL High	R	0110011	0x1	0x01	rd = (rs1 * rs2)[63:32]
mulsu	MUL High (S) (U)	R	0110011	0x2	0x01	rd = (rs1 * rs2)[63:32]
mulu	MUL High (U)	R	0110011	0x3	0x01	rd = (rs1 * rs2)[63:32]
div	DIV	R	0110011	0x4	0x01	rd = rs1 / rs2
divu	DIV (U)	R	0110011	0x5	0x01	rd = rs1 / rs2
rem	Remainder	R	0110011	0x6	0x01	rd = rs1 % rs2
remu	Remainder (U)	R	0110011	0x7	0x01	rd = rs1 % rs2

RV32A Atomic Extension

31 27 26 25		25	24	4 20 19		15 14	12	2 11	7 6 0		
func	t5	aq	rl	r	s2	rs1	funct3		rd	opcode	
5		1	1		5	5		3	5	7	
Inst	Name			FMT	Opcode	funct3	funct5	Description (C)			
lr.w	Load	Reserv	ed	R	0101111	0x2	0x02	rd	rd = M[rs1], reserve M[rs1]		
SC.W	Store	Condit	tional	R	0101111	0x2	0x03	if	<pre>(reserved) { M</pre>	[rs1] = rs2; rd = 0 }	
								els	else { rd = 1 }		
amoswap.w	Atomi	c Swaj	p	R	0101111	0x2	0x01	rd	= M[rs1]; swap	(rd, rs2); M[rs1] = rd	
amoadd.w	Atomi	c ADD		R	0101111	0x2	0x00	rd	= M[rs1] + rs2	; M[rs1] = rd	
amoand.w	Atomi	c AND		R	0101111	0x2	0x0C	rd	= M[rs1] & rs2	; M[rs1] = rd	
amoor.w	Atomi	c OR		R	0101111	0x2	0x0A	rd	= M[rs1] rs2	; M[rs1] = rd	
amoxor.w	Atomi	x XOR		R	0101111	0x2	0x04	rd	rd = M[rs1] ^ rs2; M[rs1] = rd		
amomax.w	Atomi	c MAX		R	0101111	0x2	0x14	rd	rd = max(M[rs1], rs2); M[rs1] = rd		
amomin.w	Atomi	c MIN		R	0101111	0x2	0x10	rd	rd = min(M[rs1], rs2); M[rs1] = rd		

RV32F / D Floating-Point Extensions

Inst	Name	FMT	Opcode	funct3	funct5	Description (C)
flw	Flt Load Word	*				rd = M[rs1 + imm]
fsw	Flt Store Word	*				M[rs1 + imm] = rs2
fmadd.s	Flt Fused Mul-Add	*				rd = rs1 * rs2 + rs3
fmsub.s	Flt Fused Mul-Sub	*				rd = rs1 * rs2 - rs3
fnmadd.s	Flt Neg Fused Mul-Add	*				rd = -rs1 * rs2 + rs3
fnmsub.s	Flt Neg Fused Mul-Sub	*				rd = -rs1 * rs2 - rs3
fadd.s	Flt Add	*				rd = rs1 + rs2
fsub.s	Flt Sub	*				rd = rs1 - rs2
fmul.s	Flt Mul	*				rd = rs1 * rs2
fdiv.s	Flt Div	*				rd = rs1 / rs2
fsqrt.s	Flt Square Root	*				rd = sqrt(rs1)
fsgnj.s	Flt Sign Injection	*				rd = abs(rs1) * sgn(rs2)
fsgnjn.s	Flt Sign Neg Injection	*				rd = abs(rs1) * -sgn(rs2)
fsgnjx.s	Flt Sign Xor Injection	*				rd = rs1 * sgn(rs2)
fmin.s	Flt Minimum	*				rd = min(rs1, rs2)
fmax.s	Flt Maximum	*				rd = max(rs1, rs2)
fcvt.s.w	Flt Conv from Sign Int	*				rd = (float) rs1
fcvt.s.wu	Flt Conv from Uns Int	*				rd = (float) rs1
fcvt.w.s	Flt Convert to Int	*				rd = (int32_t) rs1
fcvt.wu.s	Flt Convert to Int	*				rd = (uint32_t) rs1
fmv.x.w	Move Float to Int	*				rd = *((int*) &rs1)
fmv.w.x	Move Int to Float	*				rd = *((float*) &rs1)
feq.s	Float Equality	*				rd = (rs1 == rs2) ? 1 : 0
flt.s	Float Less Than	*				rd = (rs1 < rs2) ? 1 : 0
fle.s	Float Less / Equal	*				rd = (rs1 <= rs2) ? 1 : 0
fclass.s	Float Classify	*				rd = 09

RV32C Compressed Extension

15 14 13	12	11 10	9 8	7	6 5	4 3 2	2 1 0	
funct	4	r	d/rs1			rs2	op	CR-type
funct3	imm	r	d/rs1		j	mm	op	CI-type
funct3	imm				rs2		op	CSS-type
funct3			imm			rd'	op	CIW-type
funct3	imm		rs1'		imm	rd'	op	CL-type
funct3	imm		rd'/rs1'		imm rs2'		op	CS-type
funct3	imm		rs1'		imm		op	CB-type
funct3	offset						op	CJ-type

Inst	Name	FMT	OP	Funct	Description
c.lwsp	Load Word from SP	CI	10	010	lw rd, (4*imm)(sp)
c.swsp	Store Word to SP	CSS	10	110	sw rs2, (4*imm)(sp)
c.lw	Load Word	CL	00	010	lw rd', (4*imm)(rs1')
C.SW	Store Word	CS	00	110	sw rs1', (4*imm)(rs2')
c.j	Jump	CJ	01	101	jal x0, 2*offset
c.jal	Jump And Link	CJ	01	001	jal ra, 2*offset
c.jr	Jump Reg	CR	10	1000	jalr x0, rs1, 0
c.jalr	Jump And Link Reg	CR	10	1001	jalr ra, rs1, 0
c.beqz	Branch $== 0$	CB	01	110	beq rs', x0, 2*imm
c.bnez	Branch!= 0	СВ	01	111	bne rs', x0, 2*imm
c.li	Load Immediate	CI	01	010	addi rd, x0, imm
c.lui	Load Upper Imm	CI	01	011	lui rd, imm
c.addi	ADD Immediate	CI	01	000	addi rd, rd, imm
c.addi16sp	ADD Imm * 16 to SP	CI	01	011	addi sp, sp, 16*imm
c.addi4spn	ADD Imm * 4 + SP	CIW	00	000	addi rd', sp, 4*imm
c.slli	Shift Left Logical Imm	CI	10	000	slli rd, rd, imm
c.srli	Shift Right Logical Imm	CB	01	100x00	srli rd', rd', imm
c.srai	Shift Right Arith Imm	CB	01	100x01	srai rd', rd', imm
c.andi	AND Imm	CB	01	100x10	andi rd', rd', imm
c.mv	MoVe	CR	10	1000	add rd, x0, rs2
c.add	ADD	CR	10	1001	add rd, rd, rs2
c.and	AND	CS	01	10001111	and rd', rd', rs2'
c.or	OR	CS	01	10001110	or rd', rd', rs2'
c.xor	XOR	CS	01	10001101	xor rd', rd', rs2'
c.sub	SUB	CS	01	10001100	sub rd', rd', rs2'
c.nop	No OPeration	CI	01	000	addi x0, x0, 0
c.ebreak	Environment BREAK	CR	10	1001	ebreak

Pseudo Instructions

la rd, symbol	Pseudoinstruction	Base Instruction(s)	Meaning
auipc rd, symbol[31:12] L(b h w d) rd, symbol[11:0](rd) auipc rt, symbol[31:12]	la rd, symbol		Load address
Stoke Member Stoke St	l{b h w d} rd, symbol	auipc rd, symbol[31:12]	Load global
Sign y Till Symbol Till Sign Symbol Till	s{b h w d} rd. svmbol. rt	auipc rt, symbol[31:12]	Store global
fl(w d) rd, symbol, rt fl(w d) rd, symbol[11:0](rt) auipc rt, symbol[31:12] fs(w d) rd, symbol, rt fs(w d) rd, symbol[11:0](rt) nop addi x0, x0, 0 ir d, immediate mv rd, rs not rd, rs not rd, rs neg rd, rs neg rd, rs sub rd, x0, rs seqz rd, rs sut rd, rs, 1 seqz rd, rs sltu rd, x0, rs seqz rd, rs sltu rd, x0, rs set if = zero setz rd, rs sltr rd, x0, rs set if < zero setz rd, rs sltr rd, x0, rs set if < zero setz rd, rs sltr rd, x0, rs set if < zero setz rd, rs sltr rd, x0, rs set if < zero setz rd, rs sltr rd, x0, rs set if < zero setz rd, rs sltr rd, x0, rs set if < zero setz rd, rs sltr rd, x0, rs set if < zero setz rd, rs sltr rd, x0, rs set if < zero setz rd, rs sltr rd, x0, rs set if < zero setz rd, rs sltr rd, x0, rs set if < zero setz rd, rs sltr rd, x0, rs set if < zero setz rd, rs sltr rd, x0, rs set if < zero setz rd, rs sltr rd, x0, rs set if < zero setz rd, rs sltr rd, x0, rs set if < zero setz rd, rs sltr rd, x0, rs set if < zero setz rd, rs sltr rd, x0, rs set if < zero setz rd, rs sltr rd, x0, rs set if < zero setz rd, rs sltr rd, x0, rs set if < zero set if < zero setz re set if < zero			
nop addi x0, x0, 0 No operation Ii rd, immediate mv rd, rs not rd, rs neg rd, rs subw rd, x0, rs sext.w rd, rs sext.if ≥ zero sett.if ≥ zero single-precision register Single-precision negate Copy double-precision negate Copy double-precision negate beq.x rs, offset bex rs, x0, offset bex rs, offset bex rs, x0, offset bex rs, offset bex rs, x0, offset bex rs, x0, offset bex rs, x0, offset by x0, rs, offset by x0,	fl{w d} rd, symbol, rt	fl{w d} rd, symbol[11:0](rt)	Floating-point load global
No operation No operation	fs{w d} rd, symbol, rt		Floating-point store global
mv rd, rs not rd, rs not rd, rs not rd, rs neg rd, rs sub rd, x0, rs negw rd, rs sext.w rd, rs sext.f = zero sextif > zero Double-precision register Double-precision register Branch if = zero Branch if ≤ zero Sext.w rd, rs, rs sextif > zero Sextif > zer	nop		No operation
not rd, rs neg rd, rs neg wrd, rs sub rd, x0, rs negw rd, rs sub wrd, x0, rs subw rd, x0, rs rwo's complement rwo's complemete set if zero site for zero site for zero site for zero site for zero singer fareo copy single-precision register single-precision register single-precision register single-precision register single-precision register specison set if zero specison specis	li rd, immediate	Myriad sequences	Load immediate
not rd, rs neg rd, rs neg wrd, rs sub rd, x0, rs negw rd, rs sub wrd, x0, rs subw rd, x0, rs rwo's complement rwo's complemete set if zero site for zero site for zero site for zero site for zero singer fareo copy single-precision register single-precision register single-precision register single-precision register single-precision register specison set if zero specison specis		addi rd, rs, 0	Copy register
neg rd, rs sub rd, x0, rs Two's complement negw rd, rs subw rd, x0, rs Two's complement word sext.w rd, rs addiw rd, rs, 0 Sign extend word seqz rd, rs sltiu rd, rs, 1 Set if = zero sltz rd, rs slt urd, x0, rs Set if \neq zero sltz rd, rs slt rd, x0, rs Set if \neq zero sgtz rd, rs slt rd, x0, rs Set if \neq zero sgtz rd, rs slt rd, x0, rs Set if \neq zero sgtz rd, rs slt rd, x0, rs Set if \neq zero sgtz rd, rs slt rd, x0, rs Set if \neq zero sgtz rd, rs slt rd, x0, rs Set if \neq zero sgtz rd, rs fsgnj.s.rd, rs, rs Copy single-precision register fabs.s rd, rs fsgnj.s.rd, rs, rs Single-precision absolute value fneg.s rd, rs fsgnj.s.rd, rs, rs Single-precision absolute value fneg.s rd, rs fsgnj.d rd, rs, rs Copy double-precision absolute value fneg.d rd, rs fsgnj.d rd, rs, rs Double-precision absolute value fneg.d rd, rs fsgnj.d rd, rs, rs Double-precision negate beqz rs, offset beq rs, x0, offset Branch if \neq zero blez rs, offset beq rs, x0, offset Branch if \neq zero blez rs, offset bg x0, rs, offset Branch if \neq zero bgz rs, offset bg x0, rs, offset Branch if \neq zero bgtz rs, offset blt rs, x0, offset Branch if \neq zero bgt rs, rt, offset blt rs, x0, offset Branch if \neq zero bgt rs, rt, offset blt rt, rs, offset Branch if \neq zero bgt rs, rt, offset blt rt, rs, offset Branch if \neq zero bgt rs, rt, offset bge rt, rs, offset Branch if \neq Branch if \neq zero bgt rs, rt, offset bgu rt, rs, offset Branch if \neq Dump and link jr rs jalr x0, rs, 0 Jump and link register ret jalr x0, x1, 0 Return from subroutine call offset jalr x1, x1, offset[131:12] jalr x1, x1, offset[131:12] jalr x1, x1, offset[131:12] jalr x1, x1, offset[131:12] jalr x0, x6, offset[131:12]	not rd, rs		
negw rd, rs addiw rd, x0, rs Sign extend word Sign extend space of Sign extend solution register Branch if < zero Single-precision negate Signle-precision absolute value Signle-precision negate Copy double-precision absolute value Signle-precision absolute value Signle-precis	neg rd, rs		
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$			
snez rd, rs sltu rd, x0, rs Set if \neq zero sltz rd, rs slt rd, rs, x0 Set if $<$ zero sgtz rd, rs slt rd, x0, rs Set if $<$ zero sgtz rd, rs slt rd, x0, rs Set if $<$ zero fmv.s rd, rs fsgnj.s rd, rs, rs Copy single-precision register fabs.s rd, rs fsgnjx.s rd, rs, rs Single-precision absolute value fneg.s rd, rs fsgnjx.s rd, rs, rs Single-precision negate fmv.d rd, rs fsgnjx.d rd, rs, rs Copy double-precision register fabs.d rd, rs fsgnjx.d rd, rs, rs Copy double-precision register fabs.d rd, rs fsgnjx.d rd, rs, rs Double-precision absolute value fneg.d rd, rs fsgnjx.d rd, rs, rs Double-precision absolute value fneg.d rd, rs fsgnjx.d rd, rs, rs Double-precision negate fabs.d rd, rs fsgnjx.d rd, rs, rs Double-precision negate fabeq rs, offset beq rs, x0, offset Branch if \neq zero fabeq rs, offset beq rs, x0, offset Branch if \neq zero fabeq rs, offset beq rs, x0, offset Branch if \neq zero fabeq rs, offset beg rs, x0, offset Branch if \neq zero fabeq rs, offset blt rs, x0, offset Branch if \neq zero fabeq rs, rt, offset blt rs, x0, offset Branch if \neq zero fabeq rs, rt, offset blt rs, x0, offset Branch if \neq zero fabeq rs, rt, offset blt rs, x0, offset Branch if \neq zero fabeq rs, rt, offset blt rt, rs, offset Branch if \neq zero fabeq rs, rt, offset blt rt, rs, offset Branch if \neq zero fabeq rs, rt, offset blt rt, rs, offset Branch if \neq zero fabeq rs, rt, offset blt rt, rs, offset Branch if \neq zero fabeq rs, rt, offset blt rt, rs, offset Branch if \neq zero fabeq rs, rt, offset blt rt, rs, offset Branch if \neq zero fabeq rs, rt, offset blt rt, rs, offset Branch if \neq granch if \neq zero fabeq rs, refere fabeq rs, rs, refere fabeq rs,			
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$			
sgtz rd, rs slt rd, x0, rs fmv.s rd, rs fsgnj.s rd, rs, rs fsgnjs.s rd, rs, rs fsgnjs.s rd, rs, rs fsgnje-precision absolute value fneg.s rd, rs fsgnj.s rd, rs, rs fsgnje-precision negate fmv.d rd, rs fsgnj.s rd, rs, rs fsgnje-precision negate for, rs fsgnj.d rd, rs, rs Copy double-precision register fabs.d rd, rs fsgnj.d rd, rs, rs Double-precision absolute value fneg.d rd, rs fsgnjn.d rd, rs, rs Double-precision absolute value fneg.d rd, rs fsgnjn.d rd, rs, rs Double-precision negate beqz rs, offset beq rs, x0, offset Branch if = zero brez rs, offset bge x0, rs, offset bge x0, rs, offset bge rs, x0, offset bgt rs, offset bgt rs, offset blt rs, x0, offset blt rs, x0, offset blt rs, x0, offset blt rs, rs, offset blt rs, rs, offset blt rr, rs, offset blt rr, rs, offset ble rs, rt, offset blu rt, rs, offset blu rs, rt, offset blu rt, rs, offset blu rs, rt, offset blu rt, rs, offset blu rs, rt, offset blu rs, rt, offset blu rt, rs, offset branch if ≤ unsigned bleu rs, rt, offset jal x0, offset Jump jal offset jal x0, offset Jump and link jr rs jalr x0, x1, 0 Return from subroutine call offset jalr x1, x1, offset[11:0] auipc x6, offset[31:12] jalr x0, x6, offset[11:0] Tail call far-away subroutine			
finv.s rd, rs fsgnj.s rd, rs, rs Single-precision register fabs.s rd, rs fsgnjx.s rd, rs, rs Single-precision absolute value fneg.s rd, rs fsgnjn.s rd, rs, rs Single-precision absolute value fneg.s rd, rs fsgnj.d rd, rs, rs Copy double-precision negate fmv.d rd, rs fsgnjx.d rd, rs, rs Copy double-precision register fabs.d rd, rs fsgnjx.d rd, rs, rs Double-precision absolute value fneg.d rd, rs fsgnjn.d rd, rs, rs Double-precision negate beqz rs, offset beq rs, x0, offset Branch if = zero bright brigh			
fabs.s rd, rs fsgnjx.s rd, rs, rs fsgnjn.s rd, rs, rs fsgnjn.s rd, rs, rs fsgnjn.s rd, rs, rs fsgnje-precision absolute value fneg.s rd, rs fsgnjn.s rd, rs, rs fsgnje-precision negate fmv.d rd, rs fsgnjx.d rd, rs, rs Gopy double-precision register fabs.d rd, rs fsgnjx.d rd, rs, rs Double-precision absolute value fneg.d rd, rs fsgnjn.d rd, rs, rs Double-precision negate beqz rs, offset beq rs, x0, offset beqz rs, offset benez rs, offset bez rs, offset bltz rs, offset bltz rs, offset bltz rs, offset bltz rs, offset blt rs, x0, offset blt rs, x0, offset blt rs, x0, offset blt rs, rt, offset blt rs, rt, offset blt rt, rs, offset ble rs, rt, offset blu rt, rs, offset blu rs, rt, offset blu rt, rs, offset ble urs, rt, offset ble urs, rt, offset jal x0, offset jal x1, offset jal x1, offset jal x1, offset jal x1, x1, x2 Jump and link jr rs jalr x2, x1, 0 Jump register jalr rs jalr x3, x1, 0 Return from subroutine call offset auipc x1, offset[31:12] jalr x1, x1, offset[11:0] Tail call far-away subroutine			
fneg.s rd, rs fsgnjn.s rd, rs, rs Copy double-precision negate fmv.d rd, rs fsgnj.d rd, rs, rs Copy double-precision register fabs.d rd, rs fsgnjx.d rd, rs, rs Double-precision absolute value fneg.d rd, rs fsgnjn.d rd, rs, rs Double-precision negate beqz rs, offset beq rs, x0, offset Branch if = zero blez rs, offset bge x0, rs, offset Branch if \neq zero blez rs, offset bge rs, x0, offset Branch if \leq zero bgez rs, offset blt rs, x0, offset Branch if \leq zero blt rs, x0, offset Branch if \leq zero bt rs, x0, offset Branch if \leq zero bgt rs, offset blt ro, rs, offset Branch if \leq zero bgt rs, rt, offset blt rt, rs, offset Branch if \leq zero bgt rs, rt, offset blt rt, rs, offset Branch if \leq zero bgt rs, rt, offset bge rt, rs, offset Branch if \leq bgu rs, rt, offset bge rt, rs, offset Branch if \leq bgu rs, rt, offset bgu rt, rs, offset Branch if \leq bgu rs, rt, offset bgu rt, rs, offset Branch if \leq unsigned bleu rs, rt, offset bgu rt, rs, offset Branch if \leq unsigned j offset jal x0, offset Jump jal offset jal x1, offset Jump and link jr rs jalr x0, rs, 0 Jump register jalr rs jalr x1, rs, 0 Jump and link register ret jalr x0, x1, 0 Return from subroutine call offset jalr x1, x1, offset[31:12] jalr x1, x1, offset[31:12] jalr x1, x1, offset[11:0] Tail call far-away subroutine	′		
$\begin{array}{cccccccccccccccccccccccccccccccccccc$			
fabs.d rd, rs fsgnjx.d rd, rs, rs Double-precision absolute value fneg.d rd, rs fsgnjn.d rd, rs, rs Double-precision negate beqz rs, offset beq rs, x0, offset Branch if = zero bnez rs, offset bge x0, rs, offset Branch if ≤ zero bgez rs, offset bge x0, rs, offset Branch if ≤ zero bgez rs, offset blt rs, x0, offset Branch if ≤ zero bltz rs, offset blt rs, x0, offset Branch if < zero bltz rs, offset blt x0, rs, offset Branch if > zero bgtz rs, offset blt rt, rs, offset Branch if > zero btr rs, rt, offset blt rt, rs, offset Branch if > bge rt, rs, offset Branch if ≤ bgu rs, rt, offset bgu rs, rt, offset bgu rs, rt, offset bltu rt, rs, offset Branch if >, unsigned bleu rs, rt, offset bgu rs, rt, offs			
fneg.d rd, rsfsgnjn.d rd, rs, rsDouble-precision negatebeqz rs, offsetbeq rs, x0, offsetBranch if = zerobnez rs, offsetbne rs, x0, offsetBranch if ≠ zeroblez rs, offsetbge x0, rs, offsetBranch if ≤ zerobgez rs, offsetbge rs, x0, offsetBranch if ≥ zerobltz rs, offsetblt rs, x0, offsetBranch if < zero			
beqz rs, offset beq rs, x0, offset Branch if = zero bnez rs, offset bne rs, x0, offset Branch if \neq zero blez rs, offset bge x0, rs, offset Branch if \leq zero bgez rs, offset bge rs, x0, offset Branch if \leq zero bltz rs, offset blt rs, x0, offset Branch if \leq zero bgtz rs, offset blt x0, rs, offset Branch if \leq zero bgtz rs, offset blt x0, rs, offset Branch if $>$ zero bgt rs, rt, offset blt rt, rs, offset Branch if $>$ ble rs, rt, offset bge rt, rs, offset Branch if \leq bgtu rs, rt, offset bge rt, rs, offset Branch if \leq bgtu rs, rt, offset blu rt, rs, offset Branch if \leq , unsigned bleu rs, rt, offset bgeu rt, rs, offset Branch if \leq , unsigned j offset jal x0, offset Jump jal offset jal x1, offset Jump and link jr rs jalr x0, rs, 0 Jump register jalr rs ret jalr x0, x1, 0 Return from subroutine call offset auipc x1, offset[31:12] jalr x1, x1, offset[11:0] auipc x6, offset[31:12] jalr x0, x6, offset[11:0] Tail call far-away subroutine			
bnez rs, offset bne rs, x0, offset Branch if \neq zero blez rs, offset bge x0, rs, offset Branch if \leq zero bgez rs, offset bge rs, x0, offset Branch if \geq zero bltz rs, offset blt rs, x0, offset Branch if \leq zero bgtz rs, offset blt x0, rs, offset Branch if \leq zero bgtz rs, offset blt x0, rs, offset Branch if \geq zero bgt rs, rt, offset blt rt, rs, offset Branch if \geq blt rt, rs, offset Branch if \geq blt rt, rs, offset Branch if \leq bgtu rs, rt, offset bltu rt, rs, offset Branch if \leq bgtu rs, rt, offset bltu rt, rs, offset Branch if \leq , unsigned bleu rs, rt, offset bgeu rt, rs, offset Branch if \leq , unsigned bleu rs, rt, offset bgeu rt, rs, offset Branch if \leq , unsigned jal offset jal x0, offset Jump and link jr rs jalr x0, rs, 0 Jump and link register ret jalr x0, x1, 0 Return from subroutine call offset auipc x1, offset[31:12] jalr x1, x1, offset[11:0] auipc x6, offset[31:12] Tail call far-away subroutine			
blez rs, offset bge x0, rs, offset Branch if \leq zero bgez rs, offset bge rs, x0, offset Branch if \geq zero bltz rs, offset blt rs, x0, offset Branch if $<$ zero bgtz rs, offset blt x0, rs, offset Branch if $>$ zero bgt rs, rt, offset blt rt, rs, offset Branch if $>$ bge rt, rs, offset Branch if $>$ bgu rs, rt, offset bltu rt, rs, offset Branch if $>$ unsigned bleu rs, rt, offset bgeu rt, rs, offset Branch if $>$, unsigned bleu rs, rt, offset bgeu rt, rs, offset Branch if $>$, unsigned bleu rs, rt, offset bgeu rt, rs, offset Branch if $>$ unsigned bleu rs, rt, offset bgeu rt, rs, offset Branch if $>$ unsigned bleu rs, rt, offset bgeu rt, rs, offset Branch if $>$ unsigned bleu rs, rt, offset bgeu rt, rs, offset Branch if $>$ unsigned bleu rs, rt, offset bgeu rt, rs, offset Branch if $>$ unsigned Branch if $>$ unsigned bleu rs, rt, offset Branch if $>$ unsigned Branch if $>$ un			
bgez rs, offset bge rs, x0, offset Branch if \geq zero bltz rs, offset blt rs, x0, offset Branch if $<$ zero bgtz rs, offset blt x0, rs, offset Branch if $>$ zero bgt rs, rt, offset blt rt, rs, offset Branch if $>$ blt rt, rs, offset Branch if $>$ blt rt, rs, offset Branch if \leq bgtu rs, rt, offset bltu rt, rs, offset Branch if $>$, unsigned bleu rs, rt, offset bgeu rt, rs, offset Branch if $>$, unsigned bleu rs, rt, offset bgeu rt, rs, offset Branch if \leq , unsigned \neq j offset jal x0, offset Jump jal offset jal x1, offset Jump and link jr rs jalr x0, rs, 0 Jump and link register ret jalr x0, x1, 0 Return from subroutine call offset auipc x1, offset[31:12] jalr x1, x1, offset[11:0] auipc x6, offset[31:12] Tail call far-away subroutine			
bltz rs, offset blt rs, x0, offset Branch if < zero bgtz rs, offset blt x0, rs, offset Branch if > zero bgt rs, rt, offset blt rt, rs, offset Branch if > ble rs, rt, offset bge rt, rs, offset Branch if ≤ bgtu rs, rt, offset bltu rt, rs, offset Branch if >, unsigned bleu rs, rt, offset bgeu rt, rs, offset Branch if ≤, unsigned j offset jal x0, offset Jump jal offset jal x1, offset Jump and link jr rs jalr x0, rs, 0 Jump register jalr rs jalr x1, rs, 0 Jump and link register ret jalr x0, x1, 0 Return from subroutine call offset jalr x1, x1, offset[31:12] jalr x1, x1, offset[31:12] jalr x0, x6, offset[11:0] Tail call far-away subroutine			
$\begin{array}{cccccccccccccccccccccccccccccccccccc$			
bgt rs, rt, offset blt rt, rs, offset Branch if > ble rs, rt, offset bge rt, rs, offset Branch if \leq bgtu rs, rt, offset bltu rt, rs, offset Branch if >, unsigned bleu rs, rt, offset bgeu rt, rs, offset Branch if \leq , unsigned \leq j offset jal x0, offset Jump jal offset jal x1, offset Jump and link jr rs jalr x0, rs, 0 Jump register jalr rs jalr x1, rs, 0 Jump and link register ret jalr x0, x1, 0 Return from subroutine call offset jalr x1, x1, offset[31:12] jalr x1, x1, offset[31:12] jalr x0, x6, offset[31:12] Tail call far-away subroutine			
ble rs, rt, offset bge rt, rs, offset Branch if \leq bgtu rs, rt, offset blu rt, rs, offset Branch if $>$, unsigned bleu rs, rt, offset bgeu rt, rs, offset Branch if \leq , unsigned \leq j offset jal x0, offset Jump jal offset jal x1, offset Jump and link jr rs jalr x0, rs, 0 Jump register jalr rs jalr x1, rs, 0 Jump and link register ret jalr x0, x1, 0 Return from subroutine call offset jalr x1, x1, offset[31:12] jalr x1, x1, offset[31:12] jalr x0, x6, offset[31:12] Tail call far-away subroutine			
bgtu rs, rt, offset bltu rt, rs, offset Branch if >, unsigned bleu rs, rt, offset bgeu rt, rs, offset Branch if ≤, unsigned j offset jal x0, offset Jump jal offset jal x1, offset Jump and link jr rs jalr x0, rs, 0 Jump register jalr rs jalr x1, rs, 0 Jump and link register ret jalr x0, x1, 0 Return from subroutine call offset auipc x1, offset[31:12] jalr x1, x1, offset[11:0] tail offset jalr x0, x6, offset[11:0] Tail call far-away subroutine			
bleu rs, rt, offset bgeu rt, rs, offset Branch if ≤, unsigned j offset jal x0, offset Jump jal offset jal x1, offset Jump and link jr rs jalr x0, rs, 0 Jump register jalr rs jalr x1, rs, 0 Jump and link register ret jalr x0, x1, 0 Return from subroutine call offset auipc x1, offset[31:12] jalr x1, x1, offset[11:0] tail offset jalr x0, x6, offset[11:0] Tail call far-away subroutine			-
j offsetjal x0, offsetJumpjal offsetjal x1, offsetJump and linkjr rsjalr x0, rs, 0Jump registerjalr rsjalr x1, rs, 0Jump and link registerretjalr x0, x1, 0Return from subroutinecall offsetauipc x1, offset[31:12] jalr x1, x1, offset[11:0]Call far-away subroutinetail offsetauipc x6, offset[31:12] jalr x0, x6, offset[11:0]Tail call far-away subroutine			
jal offsetjal x1, offsetJump and linkjr rsjalr x0, rs, 0Jump registerjalr rsjalr x1, rs, 0Jump and link registerretjalr x0, x1, 0Return from subroutinecall offsetauipc x1, offset[31:12] jalr x1, x1, offset[11:0]Call far-away subroutinetail offsetauipc x6, offset[31:12] jalr x0, x6, offset[11:0]Tail call far-away subroutine			
jr rsjalr x0, rs, 0Jump registerjalr rsjalr x1, rs, 0Jump and link registerretjalr x0, x1, 0Return from subroutinecall offsetauipc x1, offset[31:12] jalr x1, x1, offset[11:0]Call far-away subroutinetail offsetauipc x6, offset[31:12] jalr x0, x6, offset[11:0]Tail call far-away subroutine	•		
jalr rsjalr x1, rs, 0Jump and link registerretjalr x0, x1, 0Return from subroutinecall offsetauipc x1, offset[31:12] jalr x1, x1, offset[11:0]Call far-away subroutinetail offsetauipc x6, offset[31:12] jalr x0, x6, offset[11:0]Tail call far-away subroutine		=	=
ret jalr x0, x1, 0 Return from subroutine call offset auipc x1, offset[31:12] Call far-away subroutine tail offset jalr x0, x6, offset[11:0] Tail call far-away subroutine	-		
call offset auipc x1, offset[31:12] call far-away subroutine tail offset auipc x6, offset[31:12] auipc x6, offset[31:12] jalr x0, x6, offset[11:0] Tail call far-away subroutine	· ·		
tail offset jalr x1, x1, offset[11:0] tail offset jalr x0, x6, offset[11:0] Tail call far-away subroutine			
jalr x0, x6, offset[11:0]	call offset	jalr x1, x1, offset[11:0]	Call far-away subroutine
	tail offset	•	Tail call far-away subroutine
	fence		Fence on all memory and I/O

Registers

Register	ABI Name	Description	Saver
x0	zero	Zero constant	_
x1	ra	Return address	Callee
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	
x4	tp	Thread pointer	
x5-x7	t0-t2	Temporaries	Caller
x8	s0 / fp	Saved / frame pointer	Callee
x9	s1	Saved register	Callee
x10-x11	a0-a1	Fn args/return values	Caller
x12-x17	a2-a7	Fn args	Caller
x18-x27	s2-s11	Saved registers	Callee
x28-x31	t3-t6	Temporaries	Caller
f0-7	ft0-7	FP temporaries	Caller
f8-9	fs0-1	FP saved registers	Callee
f10-11	fa0-1	FP args/return values	Caller
f12-17	fa2-7	FP args	Caller
f18-27	fs2-11	FP saved registers	Callee
f28-31	ft8-11	FP temporaries	Caller