RISC-V REFERENCE

James Zhu <jameszhu@berkeley.edu>

RISC-V Instruction Set

Core Instruction Formats

31 27 26 25	24 20	19	15	14	12	11	7	6	0	
funct7	rs2	rs1		fun	ct3	1	rd	opcode		R-type
imm[11:	0]	rs1		fun	ct3	1	rd	opcode		I-type
imm[11:5]	rs2	rs1		fun	ct3	imm	1[4:0]	opcode		S-type
imm[12 10:5]	rs2	rs1		fun	ct3	imm[4:1 11]	opcode		B-type
	imm[31	:12]				1	rd	opcode		U-type
im	n[20 10:1	11 19:12]				1	rd	opcode		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	
eni eak	Environment Dreak	1	ווששווו	0.00	TIIIII-AY I	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	24	20	19	15 1	14	12 11	7 6 0
func	funct5 aq rl		rl	r	rs2		rs1 fun		rd rd	opcode
5		1	1	•	5	5	•	3	5	7
Inst	Name			FMT	Opcode	funct3	func	t5 D	Description (C)	
lr.w	Load 1	Reserve	ed	R	0101111	0x2	0x02	2 r	d = M[rs1], rese	erve M[rs1]
SC.W	Store	Condit	ional	R	0101111	0x2	0x03	3 i	f (reserved) { N	1[rs1] = rs2; rd = 0 }
								e.	lse { rd = 1 }	
amoswap.w	Atomi	c Swap	2	R	0101111	0x2	0x01	l r	d = M[rs1];	o(rd, rs2); M[rs1] = rd
amoadd.w	Atomi	c ADD		R	0101111	0x2	0x00) r	d = M[rs1] + rs2	2; M[rs1] = rd
amoand.w	Atomi	c AND		R	0101111	0x2	0x00	r	d = M[rs1] & rs2	2; M[rs1] = rd
amoor.w	Atomi	c OR		R	0101111	0x2	0x0A	\ r	d = M[rs1] rs2	2; M[rs1] = rd
amoxor.w	Atomi	x XOR		R	0101111	0x2	0x04	l r	d = M[rs1] ^ rs2	2; M[rs1] = rd
amomax.w	Atomi	c MAX		R	0101111	0x2	0x14	l r	d = max(M[rs1],	rs2); M[rs1] = rd
amomin.w	Atomi	c MIN		R	0101111	0x2	0x10) r	d = 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	1	0	
f	unct	:4		rc	l/rs	1				rs2			О	p	CR-type
func	t3	imm		rc	l/rs	1				imm	ı		О	p	CI-type
func	t3		i	mm						rs2			0	p	CSS-type
func	t3			i	mm	1					rď		0	p	CIW-type
func	t3	in	ım			rs1'		im	m		rd'		0	p	CL-type
func	t3	in	ım		rc	l'/rs	1'	im	m		rs2'		0	p	CS-type
func	t3	in	ım			rs1'				imm	l		0	p	CB-type
func	t3					off	set						О	р	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	CB	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

1a rd, symbol	Pseudoinstruction	Base Instruction(s)	Meaning
	la rd, symbol		Load address
S(b h w d) rd, symbol, rt S(b h w d) rd, symbol[31:12] S(b h w d) rd, symbol[31:12] fl(w d) rd, symbol[31	l{b h w d} rd, symbol		Load global
### Figure 1. Symbol 1. 1. 1. 1. 1. 1. 1. 1	s{b h w d} rd, symbol, rt	auipc rt, symbol[31:12]	Store global
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$	fl{w d} rd, symbol, rt	auipc rt, symbol[31:12]	Floating-point load global
No operation No operation	fs{w d} rd, symbol, rt	auipc rt, symbol[31:12]	Floating-point store global
li rd, immediate Myriad sequences Load immediate mv rd, rs addi rd, rs, 0 Copy register not rd, rs xori rd, rs, -1 One's complement neg rd, rs sub rd, x0, rs Two's complement negw rd, rs subw rd, x0, rs Two's complement sext.w rd, rs addiw rd, rs, 0 Sign extend word sext.w rd, rs sltu rd, x0, rs Set if = zero sext rd, rs sltu rd, x0, rs Set if < zero	nop		No operation
mv rd, rs not rd, rs not rd, rs not rd, rs neg rd, rs sub rd, x0, rs negw rd, rs subw rd, x0, rs sext.w rd, rs sext.f re.x rs sext.x rd, rs sext.x rd sext.x rd sext.fl.x rero sext.x rd sext.fl.x rero sext.x rd sext.fl.x rero sext.fl.x rero	*		-
not rd, rs neg rd, rs neg rd, rs sub rd, x0, rs negw rd, rs subw rd, x0, rs sext.w rd, rs sext.w rd, rs seqz rd, rs seqz rd, rs sets sltu rd, x0, rs seqz rd, rs seqz rd, rs sltu rd, x0, rs set if = zero seqz rd, rs sltu rd, x0, rs set if ≠ zero seqz rd, rs sltu rd, x0, rs set if ≠ zero seqz rd, rs sltu rd, x0, rs set if ≠ zero seqz rd, rs sltu rd, x0, rs set if ≠ zero sequ rd, rs set if ≥ zero sequiple-precision register sequiple-precision regis	,		
neg rd, rs negw rd, rs subw rd, x0, rs sext.w rd, rs set addiw rd, rs, 0 seqz rd, rs sltu rd, x0, rs sltu rd, x0, rs set if = zero snez rd, rs sltu rd, x0, rs sltu rd, x0, rs set if ≠ zero sltz rd, rs slt rd, rs, x0 Set if ≠ zero set if ≥	•		1. 0
negw rd, rs sext.w rd sext.w rd, rs sext.if ≠ zero Sext.if ≠ zero Sext.if ≠ zero Sext.w rd, rs single-precision absolute value fneg.s rd, rs fsgnjx.s rd, rs, rs Single-precision negate Copy double-precision negate Copy double-precision negate Copy double-precision negate Single-precision absolute value fneg.s rd, rs single-precision register Double-precision negate Single-precision absolute value forps single-precision register Double-precision negate Branch if = zero Double-precision negate Branch if = zero Branch if = zero Branch if = zero Branch if ≤ zero better s, rd, offset blt rs, vd, offset Branch if ≤ zero better s, rt, offset blt rs, rt, offset blt rs, rt, offset blt rs, rt, offset blt rs, rt, offset blt rx, rs, offset Branch if ≤ zero Branch if ≤ zero better sext.w rd branch if ≤ zero better sext	•		-
			-
seqz rd, rs sltiu rd, rs, 1 Set if = zero snez rd, rs sltu rd, x0, rs Set if \neq zero sltz rd, rs slt rd, 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 \Rightarrow zero fmv.s rd, rs fsgnjs. rd, rs, rs Copy single-precision register fabs.s rd, rs fsgnjs.s rd, rs, rs Single-precision absolute value fneg.s rd, rs fsgnjn.s rd, rs, rs Single-precision negate fmv.d rd, rs fsgnjn.d rd, rs, rs Copy double-precision register fabs.d rd, rs fsgnjn.d rd, rs, rs Copy double-precision register fabs.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 branch if \neq zero branch i			
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 fmv.s rd, rs fsgnj.s rd, rs, rs Single-precision absolute value fneg.s rd, rs fsgnjx.s rd, rs, rs Single-precision absolute value fneg.s rd, rs fsgnjx.d 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 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 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 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 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 absolute value fneg.d rd, rs fsgnjx.d rd, rs, rs Double-precision negate fable value fneg.d rd, rs, rs Double-precision negate from fiser for bounds fiser for free from fiser for free free free free free free fre			
sltz rd, rs sgtz rd, rs sgtz rd, rs sgtz rd, rs sgtz rd, rs slt rd, x0, rs slt rd, x0, rs Set if < zero Set if > zero fmv.s rd, rs fsgnj.s rd, rs, rs fsgnjx.s rd, rs, rs fsgnjx.s rd, rs, rs fsgnje-precision register single-precision negate fmv.d rd, rs fsgnj.d rd, rs, rs fsgnj.d rd, rs, rs fsgnj.d rd, rs, rs fsgnj.d rd, rs, rs fopy double-precision negate fopy double-precision negate fmv.d rd, rs fsgnjn.d rd, rs, rs fopy double-precision negate beqz rs, offset beq rs, x0, offset beqz rs, offset beq rs, x0, offset beqz rs, offset beqz rs, offset begz r			
sgtz rd, rs fmv.s rd, rs fsgnj.s rd, rs, rs fsgnjx.s rd, rs, rs fsgnje-precision absolute value fneg.s rd, rs fsgnjx.d rd, rs, rs fsgnjx.d rd, rs, rs fouble-precision absolute value fneg.d rd, rs beqz rs, offset beq rs, x0, offset blez rs, offset bge x0, rs, offset bge x0, rs, offset bgt rs, offset blt rs, x0, offset bgt rs, rt, offset blt rt, rs, offset bgt rs, rt, offset bge rt, rs, offset bgranch if ≤ unsigned bleu rs, rt, offset bge rt, rs, offset bgranch if ≤ unsigned bleu rs, rt, offset bgranch if ≤ unsigned bleu rs, rt, offset bgranch if ≤ unsigned branch if ≤ unsigned branch if ≤ unsigned branch if ≤ unsigned call offset jal x0, offset jal x0, offset jal x0, rs, 0 Gall far-away subroutine Tail call far-away subroutine			•
$\begin{array}{cccccccccccccccccccccccccccccccccccc$			
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 fsgnjn.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 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 begz rs, offset beg x0, rs, offset Branch if \neq zero blez rs, offset bge x0, rs, offset Branch if \neq zero blez rs, offset bge rs, x0, offset Branch if \neq zero bgz rs, offset blt rs, x0, offset Branch if \neq zero bgtz rs, offset blt rs, x0, offset Branch if \neq zero bgtz rs, offset blt rs, x0, offset Branch if \neq zero bgtz rs, offset blt rs, x0, offset Branch if \neq zero bgtz rs, rt, offset blt rt, rs, offset Branch if \neq zero bgtz rs, rt, offset bge rt, rs, offset Branch if \neq zero bgtz rs, rt, offset bge rt, rs, offset Branch if \neq zero bgtz rs, rt, offset bge rt, rs, offset Branch if \neq zero bgtz rs, rt, offset bge rt, rs, offset Branch if \neq zero bgtz rs, rt, offset bgtu rs, rt, offset bgtu rt, rs, offset Branch if \neq zero bgtz rs, rt, offset bgtu rs, rt, offset bgtu rt, rs, offset Branch if \neq zero bgtz rs, rt, offset bgtu rs, rt, offset bgtu rt, rs, offset Branch if \neq zero bgtz rs, rt, offset bgtu rs, rt, offset bgtu rt, rs, offset Branch if \neq zero bgtz rs, rt, offset bgtu rs, rt, offset bgtu rt, rs, offset Branch if \neq zero bgtz rs, rt, offset bgtu rs, rt, offset bgtu rt, rs, offset Branch if \neq zero bgtz rs, rt, offset bgtu rs, rt, o			
fneg.s rd, rs fsgnjn.s rd, rs, rs fsgnj.d rd, rs, rs fsgnj.d rd, rs, rs fsgnjx.d rd, rs, rs fouble-precision negate fmv.d rd, rs fsgnjx.d rd, rs, rs fsgnjx.d rd, rs, rs pouble-precision absolute value fneg.d rd, rs beqz rs, offset beq rs, x0, offset bez rs, offset blez rs, offset beg x0, rs, offset beg x3, x0, offset beg x5, x0, offset beg x6, rs, offset beg x6, rs, offset bltz rs, offset branch if = zero branch if =	*		
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 absolute value beqz rs, offset beq rs, x0, offset Branch if = zero bez rs, offset by sex vs, offset Branch if \neq zero bez rs, offset bge x0, rs, offset Branch if \neq zero by sez rs, offset by sex vs, offset Branch if \neq zero by sez rs, offset blt rs, x0, offset Branch if \neq zero by sez rs, offset blt x0, rs, offset Branch if \neq zero by tx rs, offset blt x0, rs, offset Branch if \neq zero by tx rs, offset blt x0, rs, offset Branch if \neq by tx rs, offset Branch if \neq by tx rs, rt, offset by tx rs, offset Branch if \neq by tx rs, rt, offset by tx rs, offset Branch if \neq by tx rs, rt, offset by tx rs, offset Branch if \neq by tx rs, rt, offset by tx rs, offset Branch if \neq by tx rs, rt, offset by tx rs, offset Branch if \neq by tx rs, rt, offset by tx rs, offset Branch if \neq by tx rs, rt, offset by tx rs, offset Branch if \neq by tx rs, rt, offset by tx rs, rt, offset Branch if \neq by tx rs, rt, offset by tx rs, offset Branch if \neq by tx rs, rt, offset by tx rs, rt, offset Branch if \neq by tx rs, rt, offset B			
fabs.d rd, rs fneg.d rd, rs fneg.d rd, rs fsgnjn.d rd, rs, rs beqz rs, offset beq rs, x0, offset bnez rs, offset bnez rs, offset bge x0, rs, offset bgez rs, x0, offset bltz rs, rdfset bltz rs, rdfset bltz rs, rdfset bltz rs, offset bltz rs, rt, offset bltz rs, offset bltz rs, rt, offset bltz rs, offset bltz rs, rt,			
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 bne rs, x0, offset Branch if ≠ zero blez rs, offset bge x0, rs, offset Branch if ≤ zero bgez rs, offset bge rs, x0, offset Branch if ≤ zero bgtz rs, offset blt rs, x0, offset Branch if < zero bgtz rs, offset blt rs, x0, offset Branch if > zero bgt rs, rt, offset blt rt, rs, offset Branch if > beg rt, rs, offset Branch if > beg rt, rs, offset Branch if ≤ bgtu rs, rt, offset blt rt, rs, offset Branch if ≤ bgtu rs, rt, offset blt rt, rs, offset Branch if > unsigned bleu 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 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 Return from subroutine call offset auipc x1, offset[31:12] jalr x1, x1, offset[11:0] tail offset Tail call far-away subroutine Tail call far-away subroutine			
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 bgt rs, rt, 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 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 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 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[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 $>$ zero bgt rs, rt, offset blt rt, rs, offset Branch if $>$ ble rs, rt, offset bge rt, rs, offset Branch if $>$ bge 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 j offset jal x0, offset Jump jal offset jal x0, offset Jump and link jr rs jalr x0, rs, 0 Jump register jalr rs jalr x0, x1, 0 Return from subroutine call offset jal x1, offset[31:12] jalr x1, x1, offset[11:0] Call far-away subroutine tail offset jalr x0, x6, offset[11:0] 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 $>$ blt rt, rs, offset Branch if $>$ 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 bleu rs, rt, offset Branch if $>$ unsigned bleu rs, rt, offset Branch if $>$ unsigned bleu rs, rt, offset Branch if $>$ unsigned Branch			
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 \leq bge rt, rs, offset Branch if \leq bgu rs, rt, offset bltu rt, rs, offset Branch if \leq bgu rs, rt, offset bgeu rt, rs, offset Branch if \leq , unsigned bleu rs, rt, offset bgeu rt, rs, offset Branch if \leq , unsigned bleu rs, rt, 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 jalr x1, x1, offset[31:12] jalr x1, x1, offset[11:0] Call far-away subroutine tail offset jalr x0, 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 ≤ 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			
bgtz rs, offset blt x0, rs, offset bgt rs, rt, offset blt rt, rs, offset ble rs, rt, offset bge rt, rs, offset bgu rs, rt, offset bgu rs, rt, offset bltu rt, rs, offset bltu rt, rs, offset bleu rs, rt, offset bgeu rt, rs, offset bgeu rt, rs, offset bgeu rt, rs, offset bgu rt, rs, offset Jump and link jr rs jalr x0, rs, 0 Jump and link register palr x0, x1, 0 Return from subroutine call offset auipc x1, offset[31:12] jalr x1, x1, offset[11:0] tail offset Tail call far-away subroutine			-
bgt rs, rt, offset ble rs, rt, offset bge rt, rs, offset bgtu rs, rt, offset bgtu rs, rt, offset ble urs, rt, offset ble urs, rt, offset bge urt, rs, offset branch if ≤, unsigned branch if ≤ branch if ≥ branch if ≤ branch if			
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 \neq 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 call far-away subroutine tail offset jalr x0, x6, offset[11:0] 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 \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 auipc x1, offset[31:12] jalr x1, x1, offset[11:0] Call far-away subroutine tail offset jalr x0, x6, offset[11:0] Tail call far-away subroutine			<u> </u>
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	bgtu rs, rt, offset	bltu rt, rs, offset	
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	bleu rs, rt, offset	bgeu rt, rs, offset	Branch if \leq , unsigned
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	j offset	jal x0, offset	
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	jal offset	jal x1, offset	Jump and link
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	jr rs	jalr x0, rs, 0	Jump register
ret jalr x0, x1, 0 Return from subroutine call offset auipc x1, offset[31:12] jalr x1, x1, offset[11:0] tail offset auipc x6, offset[31:12] jalr x0, x6, offset[11:0] Tail call far-away subroutine	jalr rs		Jump and link register
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	ret		
tail offset jalr x1, x1, offset[11:0] tail offset jalr x0, x6, offset[11:0] Tail call far-away subroutine	11 -664		Call for anyon ask
tail offset auipc x6, offset[31:12] auipc x6, offset[11:0] Tail call far-away subroutine	call offset		Can far-away subroutine
jalr x0, x6, offset[11:0]			m 1 11 0 1 1 1
	tall offset		iali cali far-away subroutine
<u> </u>	fence		Fence on all memory and I/O

Registers

Register	ABI Name	Description	Saver
x0	zero	Zero constant	_
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	l —
x4	tp	Thread pointer	l —
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