RISC-V REFERENCE

James Zhu <jameszhu@berkeley.edu>, extended by Tim Hotfilter <hotfilter@kit.edu>

RISC-V Instruction Set

Core Instruction Formats

31	27	26	25	24	20	19		15	14	12	11	7	6		0	
	func	:t7		rs	32	1	s1		fun	ct3		rd	op	code		R-type
	ir	nm[:	11:0)]		1	s1		fun	ct3		rd	op	code		I-type
ir	nm[1	1:5]		rs	2	1	s1		fun	ct3	imr	n[4:0]	op	code		S-type
imı	n[12	10:5	5]	rs	32	1	s1		fun	ct3	imm	[4:1 11]	op	code		B-type
	imm[31:12]								rd	op	code		U-type			
imm[20 10:1			11 19	:12]					rd	op	code		J-type			

RV32I Base Integer Instructions

slt Set Less Than R 0110011 0x2 0x00 rd = (rs1 < rs2)?1:0	msb-extends zero-extends	
xor XOR R 0110011 0x4 0x00 rd = rs1 ^ rs2 rd = rs1 rs2 or OR R 0110011 0x6 0x00 rd = rs1 rs2 rd = rs1 rs2 and AND R 0110011 0x7 0x00 rd = rs1 rs2 rd = rs1 rs2 sll Shift Right Logical R 0110011 0x1 0x00 rd = rs1 rs2 rd = rs1 rs2 sra Shift Right Logical R 0110011 0x5 0x00 rd = rs1 rs2 rd = rs1 rs2 sra Shift Right Arith* R 0110011 0x5 0x00 rd = rs1 rs2 rd = rs1 rs2 slt Set Less Than R 0110011 0x2 0x00 rd = rs1 rs2 rd = rs1 rs2 slt Set Less Than (U) R 0110011 0x3 0x00 rd = (rs1 rs2)?1:0 ze addi ADD Immediate I 0010011 0x4 rd = rs1 rimm rd = rs1 rimm rd = rs1 rimm rd = rs1 rimm rd = rs1 rim		
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 m sra Shift Right Arith* R 0110011 0x5 0x20 rd = rs1 >> rs2 m slt Set Less Than R 0110011 0x2 0x00 rd = rs1 rs2)?1:0 ze addi ADD Immediate I 0010011 0x0 rd = rs1 + imm rd = rs1 imm[5:11]=0x00 rd = rs1 imm[0:4] rd = rs1 imm[0:4] rd = rs1 imm[0:4] rd = rs1		
and AND R 0110011 0x7 0x00 rd = rs1 & rs2 sl1 Shift Left Logical R 0110011 0x1 0x00 rd = rs1 rs2 sr1 Shift Right Logical R 0110011 0x5 0x00 rd = rs1 >> rs2 m sra Shift Right Arith* R 0110011 0x5 0x20 rd = rs1 >> rs2 m slt Set Less Than R 0110011 0x2 0x00 rd = (rs1 < rs2)?1:0		
sl1 Shift Left Logical R 0110011 0x1 0x00 rd = rs1 << rs2 sr1 Shift Right Logical R 0110011 0x5 0x00 rd = rs1 >> rs2 m sra Shift Right Arith* R 0110011 0x5 0x20 rd = rs1 >> rs2 m slt Set Less Than R 0110011 0x2 0x00 rd = rs1 <> rs2)?1:0 rd = rs1 >> rs2 m slt Set Less Than (U) R 0110011 0x2 0x00 rd = (rs1 < rs2)?1:0 ze addi ADD Immediate I 0010011 0x0 rd = rs1 + imm rd = rs1 + imm vori XOR Immediate I 0010011 0x4 rd = rs1 ^ imm rd = rs1 ^ imm ori OR Immediate I 0010011 0x7 rd = rs1 & imm rd = rs1 & imm slli Shift Left Logical Imm I 0010011 0x1 imm[5:11]=0x00 rd = rs1 >> imm[0:4] m srai Shift Right Arith Imm I 0010011		
sr1 Shift Right Logical R 0110011 0x5 0x00 rd = rs1 >> rs2 m sra Shift Right Arith* R 0110011 0x5 0x20 rd = rs1 >> rs2 m slt Set Less Than R 0110011 0x2 0x00 rd = (rs1 < rs2)?1:0		
sra Shift Right Arith* R 0110011 0x5 0x20 rd = rs1 >> rs2 m slt Set Less Than R 0110011 0x2 0x00 rd = (rs1 < rs2)?1:0		
slt Set Less Than R 0110011 0x2 0x00 rd = (rs1 < rs2)?1:0 zet sltu Set Less Than (U) R 0110011 0x3 0x00 rd = (rs1 < rs2)?1:0		
sltu Set Less Than (U) R 0110011 0x3 0x00 rd = (rs1 < rs2)?1:0 zet addi ADD Immediate I 0010011 0x0 rd = rs1 + imm rd = rs1 + imm rd = rs1 ^ imm rd = rs1 ^ imm rd = rs1 imm	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 sli Shift Left Logical Imm I 0010011 0x1 imm[5:11]=0x00 rd = rs1 < <td>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] m slti Set Less Than Imm I 0010011 0x2 rd = (rs1 < imm)?1:0</td> ze lb Load Byte I 0000011 0x0 rd = M[rs1+imm][0:7]	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] m slti Set Less Than Imm I 0010011 0x2 rd = (rs1 < imm)?1:0	zero-extends
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 <		
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 <		
andi AND Immediate I 0010011 0x7 rd = rs1 & imm slli Shift Left Logical Imm I 0010011 0x1 imm[5:11]=0x00 rd = rs1 & imm 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] m slti Set Less Than Imm I 0010011 0x2 rd = (rs1 < imm)?1:0		
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] m slti Set Less Than Imm I 0010011 0x2 rd = (rs1 < imm)?1:0		
srli Shift Right Logical Imm I 0010011 0x5 imm[5:11]=0x00 rd = rs1 >> imm[0:4] rd = rs1 >> imm[0:4] m srai Shift Right Arith Imm I 0010011 0x5 imm[5:11]=0x20 rd = rs1 >> imm[0:4] m slti Set Less Than Imm I 0010011 0x2 rd = (rs1 < imm)?1:0		
srai Shift Right Arith Imm I 0010011 0x5 imm[5:11]=0x20 rd = rs1 >> imm[0:4] m slti Set Less Than Imm I 0010011 0x2 rd = (rs1 < imm)?1:0		
slti Set Less Than Imm I 0010011 0x2 rd = (rs1 < imm)?1:0 rd = (rs1 < imm)?1:0 ze sltiu Set Less Than Imm (U) I 0010011 0x3 rd = (rs1 < imm)?1:0		
sltiu Set Less Than Imm (U) I 0010011 0x3 rd = (rs1 < imm)?1:0 ze 1b Load Byte I 0000011 0x0 rd = M[rs1+imm][0:7]	msb-extends	
1b Load Byte I 0000011 0x0 rd = M[rs1+imm][0:7]		
	zero-extends	
16 Lood Half I 0000011 0.1		
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] ze	zero-extends	
lhu Load Half (U) I 0000011 0x5 rd = M[rs1+imm][0:15] ze	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 == B 1100011 0x0 if(rs1 == rs2) PC += imm		
bne Branch!= B 1100011 0x1 if(rs1 != rs2) PC += imm		
blt Branch < B 1100011 0x4 if(rs1 < rs2) PC += imm		
bge $ Branch \ge B 1100011 0x5 if(rs1 >= rs2) PC += imm$		
bltu Branch < (U) B 1100011 0x6 if(rs1 < rs2) PC += imm ze	zero-extends	
bgeu $ Branch \ge (U) $ $ B 1100011 0x7 $ $ if(rs1 >= rs2) PC += imm zero ze$	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	24	20	19	15	14	12 11	6 0	
funct5 aq rl			rs2		rs1	funct		3 rd	opcode		
5 1 1				5	5		3 5		7		
Inst	Name			FMT	Opcode	funct3	func	ct5 I	Description (C)		
lr.w	Load	Reserv	ed	R	0101111	0x2	0x02	2 r	rd = M[rs1], reserve M[rs1]		
SC.W	Store	Condit	ional	R	0101111	0x2	0x0	3 i	if (reserved) { M[rs1] = rs2; rd = 0 }		
								e	lse { rd = 1 }		
amoswap.w	Atomi	ic Swaj		R	0101111	0x2	0x0	1 r	rd = M[rs1]; swap(rd, rs2); M[rs1] = rd		
amoadd.w	Atomi	c ADD		R	0101111	0x2	0x00	0 r	d = M[rs1] + rs2	; M[rs1] = rd	
amoand.w	Atomi	c AND		R	0101111	0x2	0x00	C r	d = M[rs1] & rs2	; M[rs1] = rd	
amoor.w	Atomi	c OR		R	0101111	0x2	0x0/	A r	rd = M[rs1] rs2; M[rs1] = rd		
amoxor.w	Atomi	x XOR		R	0101111	0x2	0x04	4 r	rd = M[rs1] ^ rs2; M[rs1] = rd		
amomax.w	Atomi	c MAX		R	0101111	0x2	0x14	4 r	rd = max(M[rs1], rs2); M[rs1] = rd		
amomin.w	Atomi	c MIN		R	0101111	0x2	0x10	0 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	
funct	rd/rs1 5			rs2 5				О	p	CR-type				
funct3	imm rd/rs1 5			imm 5			0	p	CI-type					
funct3	imm 6			rs2 5				0	p	CSS-type				
funct3	imm 8				•	rď 3				О	p	CIW-type		
funct3	imm 3 rs1' 3			im	m		rd' 3		О	p	CL-type			
funct3	im 3 rd'/rs1'		1'	im	m	rs2' 3			0	p	CS-type			
funct3	im	imm 3 rs1'			imm 5			0	p	CB-type				
funct3	offset 1			et 11				0	p	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')
For eac	h load and store: Double v	vord wit	h Func	t = x11, Qu	ad with Funct = x01
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.addiw	ADD Immediate Word	CI	01	001	addiw 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.addw	ADD Word	CS	01	10011101	addw rd', rd', rs2'
c.subw	SUB Word	CS	01	10011100	subw rd', rd', rs2'
c.nop	No OPeration	CI	01	000	addi x0, x0, 0
c.ebreak	Environment BREAK	CR	10	1001	ebreak

RVC Register Number Integer Register Number Integer Register ABI Name Floating-Point Register Number Floating-Point Register ABI Name

000	001	010	011	100	101	110	111
x8	x9	x10	x11	x12	x13	x14	x15
s0	s1	a0	a1	a2	a3	a4	a5
f8	f9	f10	f11	f12	f13	f14	f15
fs0	fs1	fa0	fa1	fa2	fa3	fa4	fa5

Table 1: Registers shortcuts for rs1', rs2' and rd'

Pseudo Instructions

la rd, symbol	Pseudoinstruction	Base Instruction(s)	Meaning
	la rd, symbol		Load address
s{b h w d} rd, symbol, rt fl(w d} rd, symbol[31:12] fl(w d} rd, symbol[11:0](rt) auipc rt, symbol[11:0](rt) auipc rt, symbol[11:0](rt) fs(w d} rd, symbol, rt fs(w d} rd, symbol, rt fl(w d} rd, symbol[11:0](rt) nop addi x0, x0, 0 No operation Load immediate mv rd, rs addi rd, rs, 0 Copy register Two's complement Set if ≠ zero Single-precision register Two's complement Sex ver if ver if ver if ver if ver if ver if ve	l{b h w d} rd, symbol		Load 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) auipc rt, symbol[31:12] fs{w d} rd, symbol[11:0](rt) nop addi x0, x0, 0 No operation Load immediate Myriad sequences addi rd, rs, 0 Copy register One's complement reg rd, rs neg rd, rs sub rd, x0, rs regw rd, rs subw rd, x0, rs seqz rd, rs sext.w rd, rs addiw rd, rs, 0 Set if = zero Set if ≥ zero set if <	s{b h w d} rd, symbol, rt		Store global
nop addi x0, x0, 0 No operation li rd, immediate mv rd, rs addi rd, rs, 0 Copy register not rd, rs xori rd, rs xori rd, rs Two's complement neg rd, rs xori rd, rs Two's complement xord xord xori rd, rs Two's complement xord xord xord xord xord xord xord xord	<pre>fl{w d} rd, symbol, rt</pre>	fl{w d} rd, symbol[11:0](rt)	Floating-point load 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 word sext.w rd, rs addiw rd, rs, 0 Sign extend word sext.w rd, rs addiw rd, rs, 0 Set if = zero sext rd, rs sltu rd, x0, rs Set if ≤ zero sltz rd, rs slt rd, x0, rs Set if ≤ zero sltz rd, rs slt rd, x0, rs Set if ≤ zero sltz rd, rs slt rd, x0, rs Set if ≤ zero set rd, rs slt rd, x0, rs Set if ≤ zero styr, rs slt rd, x0, rs Set if ≤ zero set rd, rs fsgnj.s.rd, rs, rs Copy single-precision register fabs.s.d rd, rs fsgnj.s. rd, rs, rs Single-precision absolute value fneg.s rd, rs fsgnj.d. rd, rs, rs Copy double-precision register fabs.d. rd, rs fsgnj.d. rd, rs, rs Double-precision negate beq.rs, offset	fs{w d} rd, symbol, rt	· · · · · · · · · · · · · · · · · · ·	Floating-point store global
mv 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 set if > zero single-precision register Single-precision regi	nop	addi x0, x0, 0	No operation
not rd, rs neg rd, rs negw rd, rs sub rd, x0, rs regw rd, rs sub rd, x0, rs regw rd, rs sub rd, x0, rs regw rd, rs subw rd, x0, rs sext.w rd, rs addiw rd, rs, 0 Set if = zero Sign extend word Set if = zero set if ≠ zero set if ≠ zero set if ≠ zero set if ≠ zero set if > zero	li rd, immediate	Myriad sequences	Load immediate
neg rd, rs negw rd, rs subw rd, x0, rs sylbw rd, x0, rs sext.w rd, rs sext.w rd, rs sext.w rd, rs sext.w rd, rs seqz rd, rs sltu rd, rs, 1 seqz rd, rs sltu rd, x0, rs set if = zero snez rd, rs sltu rd, x0, rs set if ≠ zero sltz rd, rs slt rd, rs, x0 set if ≠ zero sltz rd, rs slt rd, rs, x0 set if ≠ zero set if ≥ zero set	mv rd, rs	addi rd, rs, 0	Copy register
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 snez rd, rs sltu rd, x0, rs Set if ≠ 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 fsgnjs.s rd, rs, rs Copy single-precision register fabs.s rd, rs fsgnjs.s rd, rs, rs Single-precision absolute value fneg.s rd, rs fsgnjs.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 fsgnjs.s rd, rs, rs Single-precision absolute value fneg.s rd, rs fsgnj.s rd, rs, rs Copy double-precision negate bed rd, rs fsgnjn.d rd, rs, rs Double-precision absolute value fneg.d rd, rs fsgnjs.d rd, rs, rs Copy double-precision negate bed rd, rs fsgnjd.d rd, rs, rs Double-precision absolute value fneg.d rd, rs fsgnjs.d rd, rs, rs Double-precision negate bed	not rd, rs	xori rd, rs, −1	One'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 snez rd, rs sltu rd, x0, rs Set if ≠ 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 fsgnjs.s rd, rs, rs Copy single-precision register fabs.s rd, rs fsgnjs.s rd, rs, rs Single-precision absolute value fneg.s rd, rs fsgnjs.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 fsgnjs.s rd, rs, rs Single-precision absolute value fneg.s rd, rs fsgnj.s rd, rs, rs Copy double-precision negate bed rd, rs fsgnjn.d rd, rs, rs Double-precision absolute value fneg.d rd, rs fsgnjs.d rd, rs, rs Copy double-precision negate bed rd, rs fsgnjd.d rd, rs, rs Double-precision absolute value fneg.d rd, rs fsgnjs.d rd, rs, rs Double-precision negate bed	neg rd, rs	sub rd, x0, rs	Two's complement
$ \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 fmv.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 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 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 beq. rs, offset branch if \neq zero beq. rs, offset beq. rs, x0, offset Branch if \neq zero beg. rs, offset beg. x0, rs, offset Branch if \leq zero beg. rs, offset beg. rs, x0, offset Branch if \leq zero beg. rs, offset blt. rs, x0, offset Branch if \leq beg. rs, rt, offset blt. rt, rs, offset Branch if \leq beg. rt, rs, offset Branch if \leq beg. rt, rs, offset Branch if \leq beg. rt, rs, offset Branch if \leq unsigned bleu rs, rt, offset blu rt, rs, offset Branch if \leq unisgned jal offset jal x0, offset Jump and link jr rs jalr x0, rs, 0 Jump and link register ret 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] Tail call far-away subroutine			
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$			Set if ≠ zero
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 fsgnjs.s rd, rs, rs fsgnjs.s rd, rs, rs fsgnje-precision absolute value fneg.s rd, rs fsgnj.d rd, rs, rs fsgnjx.d rd, rs, rs fouble-precision absolute value fneg.d rd, rs fsgnjn.d rd, rs, rs beqz rs, offset beq rs, x0, offset blez rs, offset bge x0, rs, offset bge rs, x0, offset bgt rs, offset blt rs, x0, offset blt rs, x0, offset blt rs, rt, offset blt rt, rs, offset bgt rs, rt, offset branch if ≤ zero branch if ≤ zero branch if ≤ ze			
finv.s rd, rs fsgnj.s rd, rs, rs fsgnjs.s rd, rs, rs fsgnjs.s rd, rs, rs fsgnjx.s rd, rs, 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 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 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 Copy double-precision register Copy double-precision absolute value fneg.d rd, rs fsgnjn.d rd, rs, rs Double-precision absolute value beqz rs, offset beq rs, x0, offset beq rs, x0, offset bnez rs, offset bbez rs, offset bbez rs, offset bge x0, rs, offset bgez rs, offset bge rs, x0, offset bltz rs, offset bltz rs, offset bltz rs, offset blt rs, x0, offset bltz rs, offset blt rs, x0, offset blt rs, rt, offset blt rs, rt, offset ble rs, rt, offset blu rt, rs, offset blu rt, rs, offset blu rt, rs, offset blu rt, rs, offset blu rs, rt, offset blu rs, rt, offset blu rt, rs, offset blu rs, rt, offset blu rt, rs, offset blu rs, rt, offset blu rt, rs, offset Branch if ≤ unsigned Jump Jump Jump and link jr rs jalr x0, x1, 0 Return from subroutine Call far-away subroutine Call far-away subroutine			
fneg.s rd, rs fsgnjn.s rd, rs, rs fsgnj.d rd, rs, rs fsgnjx.d rd, rs, 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 blez rs, offset blez rs, offset beg x0, rs, offset beg rs, x0, offset blez rs, offset bltz rs, offset bltz rs, offset bltz rs, offset blt rs, x0, offset blt rs, rt, offset blt rt, rs, offset ble rs, rt, offset blt rt, rs, offset ble rs, rt, offset blu rt, rs, offset blu rt, rs, offset blu rt, rs, offset bleu rs, rt, offset bleu rs, r			
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	•		
fabs.d rd, rs fneg.d rd			
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 ≤ zerobgz 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 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 bge rt, rs, offset Branch if \leq bgtu rs, rt, offset bltu rt, rs, offset Branch if \leq nusigned 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 \leq 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 $>$ ble rs, rt, offset bge rt, rs, offset Branch if $>$ blu rt, rs, offset Branch if $>$ blu rs, rt, offset bltu rt, rs, offset Branch if $>$ 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 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] 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 \leq bgtu rs, rt, offset bger rt, rs, offset Branch if \leq bgtu rs, rt, offset bltu rt, rs, offset Branch if \leq nunsigned bleu 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 bgeu 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 bleu rs, rt, offset bgeu rt, rs, offset Branch if \leq nunsigned bleu rs, rt, offset bgeu rt, rs, offset Branch if \leq btune panch if \leq derivative by an and if \leq nunsigned bleu rs, rt, offset jal 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			
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 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			*
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 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[31:12] jalr x0, x6, offset[31:12] Tail call far-away subroutine			_
$\begin{array}{cccccccccccccccccccccccccccccccccccc$			_
bgt rs, rt, offset ble rs, rt, offset bge rt, rs, offset bgtu rs, rt, offset bgtu rs, rt, offset bleu rs, rt, offset bleu rs, rt, offset bgeu rt, rs, offset bgranch if ≤, unsigned branch if ≤ branch if ≥ branch if ≥ branch if ≤ branch if ≥ branch if ≥ branch if ≤ branch if ≥ branch			
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[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] Call far-away subroutine 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] auipc x6, offset[31:12] 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[31:12] Tail call far-away subroutine			
call offset auipc x1, offset[31:12] auipc x1, x1, offset[11:0] auipc x6, offset[31:12] auipc x6, offset[31:12] jalr x0, x6, offset[11:0] Tail call far-away subroutine	jalr rs		
jalr x1, x1, offset[11:0] tail offset jalr x1, x1, offset[31:12] jalr x0, x6, offset[11:0] Tail call far-away subroutine	ret	jalr x0, x1, 0	Return from subroutine
jalr x0, x6, offset[11:0]	call offset	jalr x1, x1, offset[11:0]	Call far-away subroutine
fence fence iorw, iorw Fence on all memory and I/O	tail offset		•
	fence	fence iorw, iorw	Fence on all memory and I/O

Registers

- D	ADINI		.	
Register	ABI Name	Encoding	Description	Saver
x0	zero	00000	Zero constant	_
x1	ra	00001	Return address	Caller
x2	sp	00010	Stack pointer	Callee
x3	gp	00011	Global pointer	_
x4	tp	00100	Thread pointer	_
x5-x7	t0-t2	00101 - 00111	Temporaries	Caller
x8	s0 / fp	01000	Saved / frame pointer	Callee
x9	s1	01001	Saved register	Callee
x10-x11	a0-a1	01010 - 01011	Fn args/return values	Caller
x12-x17	a2-a7	01100 - 10001	Fn args	Caller
x18-x27	s2-s11	10010 - 11011	Saved registers	Callee
x28-x31	t3-t6	11100 - 11111	Temporaries	Caller
f0-7	ft0-7	00000 - 00111	FP temporaries	Caller
f8-9	fs0-1	01000 - 01001	FP saved registers	Callee
f10-11	fa0-1	01010 - 01011	FP args/return values	Caller
f12-17	fa2-7	01100 - 10001	FP args	Caller
f18-27	fs2-11	10010 - 11011	FP saved registers	Callee
f28-31	ft8-11	11100 - 11111	FP temporaries	Caller