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	32	1	s1		fun	ct3		rd	opco	ode		R-type
	in	nm[1	11:0)]		1	s1		fun	ct3		rd	opco	ode		I-type
im	m[1	1:5]		rs	2	1	s1		fun	ct3	imr	n[4:0]	opco	ode		S-type
imn	n[12	10:5	5]	rs	2	1	s1		fun	ct3	imm	[4:1 11]	opco	ode		B-type
imm[31:1				_						rd	opco	ode		U-type		
imm[20 10:1				11 19	:12]					rd	opco	ode		J-type		

RV32I Base Integer Instructions

Sub SUB R 0000011 0x0 0x0 0x20 rd = rs1 + rs2 xor XOR R 0000011 0x4 0x00 rd = rs1 - rs2 xor XOR R 0000011 0x6 0x00 rd = rs1 rs2 xor XOR R 0000011 0x6 0x00 rd = rs1 rs2 xor XOR R 0000011 0x6 0x00 rd = rs1 rs2 xor XOR R 0000011 0x6 0x00 rd = rs1 rs2 xor XOR XDD R 0000011 0x6 0x00 rd = rs1 xor2 xor XOR XDD	Inst	Name	FMT	Opcode	F3	F7	Description (C)	Note
xor XOR R 0000011 0x6 0x00 rd = rs1 rs2 rd = rs1 rs2 and AND R 0000011 0x7 0x00 rd = rs1 rs2 rd = rs1 rs2 s11 Shift Left Logical R 0000011 0x1 0x00 rd = rs1 × rs2 rs1 shift Right Arith* R 0000011 0x2 0x00 rd = rs1 > rs2 msb-extends s1t Set Less Than R 0000011 0x2 0x00 rd = rs1 > rs2 msb-extends s1t Set Less Than R 0110011 0x2 rd = rs1 + rs2 msb-extends s1t Set Less Than R 0110011 0x3 0x20 rd = rs1 + rs2 msb-extends s1t Set Less Than I 0010011 0x3 0x00 rd = rs1 + rs2 msb-extends addi ADD Immediate I 0010011 0x0 0x00 rd = rs1 + rs2 msb-extends s1ii Shift Left Logical Imm I 0010011 0x0 <	add	ADD	R	0000011	0x0	0x00	rd = rs1 + rs2	
or and AND QR R R 0000011 0000011 0x6 0x00 0x00 0x00 0x00 0x00 0x00 0x00	sub	SUB	R	0000011	0x0	0x20	rd = rs1 - rs2	
AND	xor	XOR	R	0000011	0x4	0x00	rd = rs1 ^ rs2	
Shift Left Logical	or	OR	R	0000011	0x6	0x00	rd = rs1 rs2	
sr1 Shift Right Logical sra R 0000011 0000011 0x2 000001 0x3 0x20 rd = rs1 >> rs2 rd = (rs1 < rs2)?1:0 msb-extends slt Set Less Than (U) R 0110011 0x3 0x3 rd = (rs1 < rs2)?1:0	and	AND	R	0000011	0x7	0x00	rd = rs1 & rs2	
sra Shift Right Arith* R 0000011 0x3 0x20 rd = rs1 >> rs2 msb-extends slt Set Less Than R 0110011 0x2 rd = (rs1 < rs2)?1:0	sll	Shift Left Logical	R	0000011	0x1	0x00	rd = rs1 << rs2	
Set Less Than R	srl	Shift Right Logical	R	0000011	0x2	0x00	rd = rs1 >> rs2	
Set Less Than (U)	sra	Shift Right Arith*	R	0000011	0x3	0x20	rd = rs1 >> rs2	msb-extends
Addi	slt	Set Less Than	R	0110011	0x2		rd = (rs1 < rs2)?1:0	
xori XOR Immediate I 0010011 0x0 0x00 rd = rs1 ^ imm nd ori OR Immediate I 0010011 0x0 0x00 rd = rs1 ^ imm nd andi AND Immediate I 0010011 0x0 0x00 rd = rs1 × imm s1li Shift Right Logical Imm I 0010011 0x1 0x00 rd = rs1 ×> imm srai Shift Right Arith Imm I 0010011 0x3 0x20 rd = rs1 ×> imm msb-extends slti Set Less Than Imm I 0010011 0x3 0x20 rd = rs1 ×> imm msb-extends slti Set Less Than Imm I 0010011 0x3 0x20 rd = (rs1 < imm)?1:0	sltu	Set Less Than (U)	R	0110011	0x3		rd = (rs1 < rs2)?1:0	zero-extends
ori OR Immediate andi I 0010011 0x0 0x00 rd = rs1 imm andi slli Shift Left Logical Imm I 0010011 0x1 0x00 rd = rs1 & imm andi slimm	addi	ADD Immediate	I	0010011	0x0	0x00	rd = rs1 + imm	
AND Immediate I 0010011 0x0 0x00 rd = rs1 & imm s1li Shift Left Logical Imm I 0010011 0x1 0x00 rd = rs1 × imm rd = rs1 × imm srai Shift Right Logical Imm I 0010011 0x1 0x00 rd = rs1 × imm rd = rs1 ×	xori	XOR Immediate	I	0010011	0x0	0x00	rd = rs1 ^ imm	
slli Shift Left Logical Imm I 0010011 0x1 0x00 rd = rs1 << imm m srli Shift Right Logical Imm I 0010011 0x1 0x00 rd = rs1 >> imm msb-extends slti Set Less Than Imm I 0010011 0x2 rd = rs1 >> imm msb-extends slti Set Less Than Imm I 0010011 0x2 rd = (rs1 < imm)?1:0	ori	OR Immediate	I	0010011	0x0	0x00	rd = rs1 imm	
srli Shift Right Logical Imm I 0010011 0x1 0x00 rd = rs1 >> imm msb-extends slti Set Less Than Imm I 0010011 0x2 rd = rs1 >> imm msb-extends slti Set Less Than Imm I 0010011 0x2 rd = (rs1 < imm)?1:0	andi	AND Immediate	I	0010011	0x0	0x00	rd = rs1 & imm	
srli Shift Right Logical Imm I 0010011 0x1 0x00 rd = rs1 >> imm msb-extends slti Set Less Than Imm I 0010011 0x2 rd = rs1 >> imm msb-extends slti Set Less Than Imm I 0010011 0x2 rd = (rs1 < imm)?1:0	slli	Shift Left Logical Imm	I	0010011	0x1	0x00	rd = rs1 << imm	
srai Shift Right Arith Imm I 0010011 0x3 0x20 rd = rs1 >> imm msb-extends slti Set Less Than Imm I 0010011 0x2 rd = (rs1 < imm)?1:0 zero-extends lb Load Byte I 0000011 0x0 rd = M[rs1+imm][0:7] zero-extends lb Load Byte I 0000011 0x0 rd = M[rs1+imm][0:7] zero-extends lb Load Half I 0000011 0x2 rd = M[rs1+imm][0:7] zero-extends lbu Load Byte (U) I 0000011 0x4 rd = M[rs1+imm][0:7] zero-extends sb Store Byte S 0100011 0x5 m[rs1+imm][0:7] zero-extends sb Store Byte S 0100011 0x0 M[rs1+imm][0:7] rs2[0:7] msb sb Store Byte S 0100011 0x0 M[rs1+imm][0:7] rs2[0:7] msb store Half S 010011 0x0 M[rs1+imm][0:7] rs2[0:7] msb store Half<	srli		I	0010011	0x1	0x00	rd = rs1 >> imm	
slti Set Less Than Imm I 0010011 0x2 rd = (rs1 < imm)?1:0 zero-extends lb Load Byte I 0010011 0x3 rd = (rs1 < imm)?1:0	srai		I	0010011	0x3	0x20	rd = rs1 >> imm	msb-extends
sltiu Set Less Than Imm (U) I 0010011 0x3 rd = (rs1 < imm)?1:0 zero-extends 1b Load Byte I 0000011 0x0 rd = M[rs1+imm][0:7] rd = M[rs1+imm][0:15] 1h Load Word I 0000011 0x1 rd = M[rs1+imm][0:15] zero-extends 1bu Load Byte (U) I 0000011 0x4 rd = M[rs1+imm][0:7] zero-extends 1bu Load Half (U) I 0000011 0x5 rd = M[rs1+imm][0:7] zero-extends 1bu Load Half (U) I 0000011 0x5 rd = M[rs1+imm][0:7] zero-extends 1bu Load Half (U) I 0000011 0x5 rd = M[rs1+imm][0:7] zero-extends 1bu Load Half (U) I 0000011 0x0 M[rs1+imm][0:7] zero-extends 1bu Sotore Byte S 0100011 0x0 M[rs1+imm][0:7] rs2[0:7] 1bu Branch = B 1100011 0x0 M[rs1+imm][0:31] rs2[0:5] 1	slti		I	0010011	0x2		rd = (rs1 < imm)?1:0	
Decorate Decorate	sltiu	Set Less Than Imm (U)	I	0010011	0x3			zero-extends
1h Load Half I 0000011 0x1 rd = M[rs1+imm][0:15] rd = M[rs1+imm][0:31] 1w Load Word I 0000011 0x2 rd = M[rs1+imm][0:31] zero-extends 1bu Load Byte (U) I 0000011 0x4 rd = M[rs1+imm][0:7] zero-extends sb Store Byte S 0100011 0x0 M[rs1+imm][0:7] = rs2[0:7] sh sb Store Half S 0100011 0x0 M[rs1+imm][0:15] = rs2[0:7] sh sw Store Word S 0100011 0x1 M[rs1+imm][0:31] = rs2[0:15] sh beq Branch == B 1100011 0x2 M[rs1+imm][0:31] = rs2[0:31] sh beq Branch == B 1100011 0x0 if(rs1 = rs2) PC += imm sh beq Branch != B 1100011 0x1 if(rs1 < rs2) PC += imm	-lb	1 1	I	0000011	0x0			
Ibu Load Byte (U) I 0000011 0x4 rd = M[rs1+imm][0:7] zero-extends sh 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 < (U) B 1100011 0x5 if(rs1 <= rs2) PC += imm zero-extends bgeu Branch ≥ (U) B 1100011 0x6 if(rs1 <= rs2) PC += imm zero-extends jal Jump And Link J 1101111 rd = PC+4; PC += imm rd = PC+4; PC += imm jalr Jump And Link Reg I 1100111 0x0 rd = imm << 12 rd =	lh	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
Ihu 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 == B 1100011 0x0 if(rs1 = rs2) PC += imm bne Branch != B 1100011 0x1 if(rs1 != rs2) PC += imm blt Branch <	lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
Ihu 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 == B 1100011 0x0 if(rs1 = rs2) PC += imm bne Branch != B 1100011 0x1 if(rs1 != rs2) PC += imm blt Branch <	1bu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
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 0x5 if(rs1 <= rs2) PC += imm zero-extends bge Branch < (U) B 1100011 0x6 if(rs1 <= rs2) PC += imm zero-extends bgu Branch ≥ (U) B 1100011 0x7 if(rs1 <= rs2) PC += imm zero-extends jal Jump And Link J 1101111 rd = PC+4; PC += imm zero-extends jal Jump And Link Reg I 1100111 0x0 rd = PC+4; PC = rs1 + imm lui Load Upper Imm U 0110111 rd = imm << 12 rd = PC + (imm << 12) ecall Environment Call I 1110011 0x0 0x00 Tran	1hu		I	0000011	0x5		rd = M[rs1+imm][0:15]	
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 0x5 if(rs1 >= rs2) PC += imm zero-extends blt Branch < (U) B 1100011 0x6 if(rs1 >= rs2) PC += imm zero-extends bgeu Branch ≥ (U) B 1100011 0x7 if(rs1 >= rs2) PC += imm zero-extends jal Jump And Link J 1101111 rd = PC+4; PC += imm rd = PC+4; PC = rs1 + imm lui Load Upper Imm U 0110111 rd = imm << 12 ecall Environment Call I 1110011 0x0 0x00 Transfer control to 0S imm: 0x000	sb	Store Byte	S	0100011	0x0		M[rs1+imm][0:7] = rs2[0:7]	
beq Branch == B 1100011 0x0 if(rs1 == rs2) PC += imm bne Branch!= B 1100011 0x1 if(rs1 != rs2) PC += imm blt Branch <	sh		S	0100011	0x1		M[rs1+imm][0:15] = rs2[0:15]	
beq Branch == B 1100011 0x0 if(rs1 == rs2) PC += imm bne Branch!= B 1100011 0x1 if(rs1 != rs2) PC += imm blt Branch <	SW	Store Word	S	0100011	0x2		M[rs1+imm][0:31] = rs2[0:31]	
bne Branch!= B 1100011 0x1 if(rs1 != rs2) PC += imm blt Branch <	beg	Branch ==	В	1100011	0x0			
blt Branch < B 1100011 0x4 if(rs1 < rs2) PC += imm pc lmm pc pc lmm pc pc lmm pc	bne	l .	В	1100011	0x1			
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	blt	Branch <	В	1100011	0x4			
bltu Branch < (U) B 1100011 0x6 if(rs1 < rs2) PC += imm zero-extends bgeu Branch \geq (U) B 1100011 0x7 if(rs1 >= rs2) PC += imm zero-extends jal Jump And Link J 1101111 rd = PC+4; PC += imm rd = PC+4; PC = rs1 + imm lui Load Upper Imm U 0110111 rd = imm << 12 rd = PC + (imm << 12) ecall Environment Call I 1110011 0x0 0x00 Transfer control to 0S imm: 0x000	bge	Branch <	В	1100011	0x5		I	
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	•			1100011	0x6			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	bgeu		В	1100011	0x7			zero-extends
jalr Jump And Link Reg I 1100111 0x0 rd = PC+4; PC = rs1 + imm lui Load Upper Imm U 0110111 rd = imm << 12			J	1101111				
lui Load Upper Imm U 0110111 rd = imm << 12 auipc Add Upper Imm to PC U 0010111 rd = PC + (imm << 12)	_	1 -			0x0		· · · · · · · · · · · · · · · · · · ·	
auipc Add Upper Imm to PC U 0010111 rd = PC + (imm << 12) ecall Environment Call I 1110011 0x0 0x00 Transfer control to 0S imm: 0x000								
ecall Environment Call I 1110011 0x0 0x00 Transfer control to OS imm: 0x000								
					0x0	0x00	-	imm: 0x000
- CDICAN DIMINICIAL DICAN I	ebreak	Environment Break	I	1110011	0x0	0x00	Transfer control to debugger	imm: 0x001

Standard Extensions

RV32M Multiply Extension

Inst	Name	FMT	Opcode	F3	F7	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		7 6 0
funct5		aq	rl		rs2			rs1		funct	3	rd	opcode
5		1	1		5			5		3	•	5	7
Inst	Nar	ne			FMT	Opco	ode	F3	F5	De	scription	n (C)	
lr.w	Loa	d Rese	erved		R	01011	111	0x2	0x02	2 rd	= M[rs	l], reserv	ve M[rs1]
SC.W	Sto	re Con	ditiona	al	R	01011	111	0x2	0x03	3 if	(reserv	ved) { M[r	rs1] = rs2; rd = 0 }
										els	e { rd	= 1 }	
amoswap.w	Ato:	mic Sv	vap		R	01011	111	0x2	0x01	l rd	= M[rs	l]; swap(r	rd, rs2); M[rs1] = rd
amoadd.w	Ato:	mic AI	DD		R	01011	111	0x2	0x00	o rd	= M[rs	1] + rs2;	M[rs1] = rd
amoand.w	Ato:	mic Al	ND		R	01011	111	0x2	0x00	rd	= M[rs	1] & rs2;	M[rs1] = rd
amoor.w	Ato:	mic OI	R		R	01011	111	0x2	0x0	\ rd	= M[rs	1] rs2;	M[rs1] = rd
amoxor.w	Ato:	mix X0	OR		R	01011	111	0x2	0x04	1 rd	= M[rs	1] ^ rs2;	M[rs1] = rd
amomax.w	Ato	mic M	AX		R	01011	111	0x2	0x14	1 rd	= max(N	1[rs1], rs	s2); M[rs1] = rd
amomin.w	Ato:	mic M	IN		R	01011	111	0x2	0x16	rd	= min(N	1[rs1], rs	s2); M[rs1] = rd

RV32F / D Floating-Point Extensions

Inst	Name	FMT	Opcode	F3	F5	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	4	r	d/rs1		rs2	op	CR-type
funct3	imm	r	d/rs1		imm	op	CI-type
funct3		imn	ı		rs2	op	CSS-typ
funct3			imm		rd'	op	CIW-typ
funct3	in	ım	rs1'	imm	rd'	op	CL-type
funct3	in	ım	rd'/rs1'	imm	rs2'	op	CS-type
funct3	in	ım	rs1'		imm	op	CB-type
funct3			offset	t		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	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

Pseudoinstruction	Base Instruction(s)	Meaning
la rd, symbol	<pre>auipc rd, symbol[31:12] addi rd, rd, symbol[11:0]</pre>	Load address
l{b h w d} rd, symbol	<pre>auipc rd, symbol[31:12] l{b h w d} rd, symbol[11:0](rd)</pre>	Load global
s{b h w d} rd, symbol, rt	<pre>auipc rt, symbol[31:12] s{b h w d} rd, symbol[11:0](rt)</pre>	Store global
fl{w d} rd, symbol, rt	<pre>auipc rt, symbol[31:12] fl{w d} rd, symbol[11:0](rt)</pre>	Floating-point load global
fs{w d} rd, symbol, rt	auipc rt, symbol[31:12] fs{w d} rd, symbol[11:0](rt)	Floating-point store global
nop	addi x0, x0, 0	No operation
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
seqz rd, rs	sltiu rd, rs, 1	Set if = zero
snez rd, rs	sltu rd, rs, r	Set if $=$ zero
sltz rd, rs		Set if < zero
	slt rd, rs, x0	Set if > zero
sgtz rd, rs	slt rd, x0, rs	
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	fsgnjn.s rd, rs, rs	Single-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
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 $<$ 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	bltu rt, rs, offset	Branch if $>$, 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 x1, rs, 0	Jump and link register
ret	jalr x0, x1, 0	Return from subroutine
aall affaat	<pre>auipc x1, offset[31:12]</pre>	Call for avvey asherseins
call offset	jalr x1, x1, offset[11:0]	Call far-away subroutine
tail offset	auipc x6, offset[31:12] jalr x0, x6, offset[11:0]	Tail call far-away subroutine
fence	fence iorw, iorw	Fence on all memory and I/O
1 EIICE	TEHCE TOTW, TOTW	rence on an memory and 1/0

Registers

Register	ABI Name	Description	Saver
x0	zero	Zero constant	_
x1	ra	Return address	Caller
x2	sp	Stack pointer	_
x3	gp	Global pointer	_
x4	tp	Thread pointer	Callee
x5	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