

RISC-V: Base Integer Instructions

Opcode	Instruction	Fmt	Example	Description	Notes
add	add	R	add a0, a1, a2	a0 = a1 + a2	sign-extends unsigned
sub	subtract	R	sub a0, a1, a2	a0 = a1 - a2	
xor	bitwise exclusive or	R	xor a0, a1, a2	a0 = a1 ^ a2	
or	bitwise or	R	or a0, a1, a2	a0 = a1 a2	
and	bitwise and	R	and a0, a1, a2	a0 = a1 & a2	
sll	shift left logical	R	sll a0, a1, a2	a0 = a1 << a2	
srl	shift right logical	R	srl a0, a1, a2	a0 = a1 >> a2	
sra	shift right arith	R	sra a0, a1, a2	a0 = a1 >> a2	
slt	set less than	R	slt a0, a1, a2	a0 = (a1 < a2) ? 1 : 0	
sltu	set less than (u)	R	sltu a0, a1, a2	a0 = (a1 < a2) ? 1 : 0	
addi	add immediate	I	addi a0, a1, -2	a0 = a1 + -2	sign-extends unsigned
xori	xor immediate	I	xori a0, a1, 2	a0 = a1 ^ 2	
ori	or immediate	I	ori a0, a1, 2	a0 = a1 2	
andi	and immediate	I	andi a0, a1, 2	a0 = a1 & 2	
slli	shift left logical imm	I	slli a0, a1, 2	a0 = a1 << 2	
srli	shift right logical imm	I	srli a0, a1, 2	a0 = a1 >> 2	
srai	shift right arith imm	I	srai a0, a1, 2	a0 = a1 >> 2	
slti	set less than imm	I	slti a0, a1, 2	a0 = (a1 < 2) ? 1 : 0	
sltiu	set less than imm (u)	I	sltiu a0, a1, 2	a0 = (a1 < 2) ? 1 : 0	
li	load immediate		li a0, 2	addi a0, zero, 2	<i>pseudo</i> , multiple encodings
la	load address		la a0, symbol	a0 = symbol	<i>pseudo</i> , multiple encodings
mv	move (copy)		mv a0, a1	addi a0, a1, 0	<i>pseudo</i>
neg	2s-complement negation		neg a0, a1	sub a0, zero, a1	<i>pseudo</i>
not	bitwise not		not a0, a1	xori a0, a1, -1	<i>pseudo</i>
lb	load byte	I	lb a0, 1(a1)	a0 = M[a1+1]	8 bits
lh	load half	I	lh a0, 2(a1)	a0 = M[a1+2]	16 bits
lw	load word	I	lw a0, 4(a1)	a0 = M[a1+4]	32 bits
lbu	load byte (u)	I	lbu a0, 1(a1)	a0 = M[a1+1]	zero-extends
lhu	load half (u)	I	lhu a0, 2(a1)	a0 = M[a1+2]	zero-extends
sb	store byte	S	sb a0, 1(a1)	M[a1+1] = a0	8 bits
sh	store half	S	sh a0, 2(a1)	M[a1+2] = a0	16 bits
sw	store word	S	sw a0, 4(a1)	M[a1+4] = a0	32 bits
beq	branch if =	B	beq a0, a1, label	if (a0 == a1) goto label	<i>pseudo</i> (bge a1, a0, label) <i>pseudo</i> (blt a1, a0, label)
bne	branch if ≠	B	bne a0, a1, label	if (a0 != a1) goto label	
blt	branch if <	B	blt a0, a1, label	if (a0 < a1) goto label	
ble	branch if ≤		ble a0, a1, label	if (a0 <= a1) goto label	
bgt	branch if >		bgt a0, a1, label	if (a0 > a1) goto label	
bge	branch if ≥	B	bge a0, a1, label	if (a0 >= a1) goto label	
bltu	branch if < (u)	B	bltu a0, a1, label	if (a0 < a1) goto label	
bleu	branch if ≤ (u)		bleu a0, a1, label	if (a0 <= a1) goto label	
bgtu	branch if > (u)		bgtu a0, a1, label	if (a0 > a1) goto label	
bgeu	branch if ≥ (u)	B	bgeu a0, a1, label	if (a0 >= a1) goto label	unsigned
beqz	branch if = 0		beqz a0, label	if (a0 == 0) goto label	<i>pseudo</i> using zero register
bnez	branch if ≠ 0		bnez a0, label	if (a0 != 0) goto label	<i>pseudo</i>
bltz	branch if < 0		bltz a0, label	if (a0 < 0) goto label	<i>pseudo</i>
blez	branch if ≤ 0		blez a0, label	if (a0 <= 0) goto label	<i>pseudo</i>
bgtz	branch if > 0		bgtz a0, label	if (a0 > 0) goto label	<i>pseudo</i>
bgez	branch if ≥ 0		bgez a0, label	if (a0 >= 0) goto label	<i>pseudo</i>
jal	jump and link	J	jal ra, label	ra = pc + 4; jump to label	<i>pseudo</i> , based on jal <i>pseudo</i> , multiple encodings <i>pseudo</i> , based on jalr
jalr	jump and link reg	I	jalr ra, 0(a1)	ra = pc + 4; jump to a1	
j	jump		j label	jump to label	
call	call subroutine		call label	ra = pc + 4; jump to label	
ret	return		ret	jump to address in ra	
lui	load upper imm	U	lui a0, 1234	a0 = 1234 << 12	20-bit immediate
auipc	add upper imm to pc	U	auipc a0, 1234	a0 = pc + (1234 << 12)	20-bit immediate
ecall	environment call	I	ecall	system call	memory synchronization
ebreak	environment break	I	ebreak	break to debugger	
fence	fence	I	fence	prevent memory reordering	

Multiply Extension

Opcode	Instruction	Fmt	Example	Description	Notes
mul	multiply	R	mul a0, a1, a2	$a0 = (a1 \times a2) [31:0]$	lower 32 bits
mulh	multiply high signed	R	mulh a0, a1, a2	$a0 = (a1 \times a2) [63:32]$	upper bits, signed
mulhsu	multiply high signed $\times$ unsigned	R	mulhsu a0, a1, a2	$a0 = (a1 \times a2) [63:32]$	a1 signed, a2 unsigned
mulhu	multiply high unsigned	R	mulhu a0, a1, a2	$a0 = (a1 \times a2) [63:32]$	upper bits, unsigned
div	divide signed	R	div a0, a1, a2	$a0 = a1 / a2$	signed division
divu	divide unsigned	R	divu a0, a1, a2	$a0 = a1 / a2$	unsigned division
rem	remainder signed	R	rem a0, a1, a2	$a0 = a1 \% a2$	signed modulo
remu	remainder unsigned	R	remu a0, a1, a2	$a0 = a1 \% a2$	unsigned modulo

Registers

Register	ABI Name	Description	Saver
x0	zero	constant zero	
x1	ra	return address	Caller
x2	sp	stack pointer	Callee
x3	gp	global pointer	
x4	tp	thread pointer	
x5-x7	t0-t2	temporary	Caller
x8	s0 / fp	saved register / frame pointer	Callee
x9	s1	saved register	Callee
x10-x11	a0-a1	argument / return value	Caller
x12-x17	a2-a7	argument	Caller
x18-x27	s2-s11	saved register	Callee
x28-x31	t3-t6	temporary	Caller

Instruction Formats

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0							
funct7							rs2					rs1					funct3					rd					opcode						R-type					
imm[11:0]												rs1					funct3					rd					opcode						I-type					
imm[11:5]							rs2					rs1					funct3					imm[4:0]					opcode						S-type					
imm[12:10:5]							rs2					rs1					funct3					imm[4:1 11]					opcode						B-type					
imm[31:12]																											rd					opcode						U-type
imm[20 10:1 11 19:12]																											rd					opcode						J-type