RV32I Base Integer Instruction Set

Format	Name	Description	Pseudocode
`LUI rd,imm`	Load Upper Immediate	Load 20-bit immediate to upper bits	`rd ← imm`
`AUIPC rd,offset`	Add Upper Immediate to PC	Add upper immediate to PC	`rd ← pc + offset`

`JAL rd,offset`	Jump and Link	Jump and save return address	`rd ← pc + length(inst)`
			`pc ← pc + offset`

`JALR rd,rs1,offset`	Jump and Link Register	Jump to register + offset	`rd ← pc + length(inst)`
			`pc ← (rs1 + offset) ∧ -2`

`BEQ rs1,rs2,offset`	Branch Equal	Branch if equal	`if rs1 = rs2 then pc ← pc + offset`
`BNE rs1,rs2,offset`	Branch Not Equal	Branch if not equal	`if rs1 ≠ rs2 then pc ← pc + offset`
`BLT rs1,rs2,offset`	Branch Less Than	Branch if less than (signed)	`if rs1 < rs2 then pc ← pc + offset`
`BGE rs1,rs2,offset`	Branch Greater Equal	Branch if greater/equal (signed)	`if rs1 ≥ rs2 then pc ← pc + offset`
`BLTU rs1,rs2,offset`	Branch Less Than Unsigned	Branch if less than (unsigned)	`if rs1 < rs2 then pc ← pc + offset`
`BGEU rs1,rs2,offset`	Branch Greater Equal Unsigned	Branch if greater/equal (unsigned)	`if rs1 ≥ rs2 then pc ← pc + offset`
`LB rd,offset(rs1)`	Load Byte	Load 8-bit signed	`rd ← s8[rs1 + offset]`
`LH rd,offset(rs1)`	Load Half	Load 16-bit signed	`rd ← s16[rs1 + offset]`
`LW rd,offset(rs1)`	Load Word	Load 32-bit signed	`rd ← s32[rs1 + offset]`
`LBU rd,offset(rs1)`	Load Byte Unsigned	Load 8-bit unsigned	`rd ← u8[rs1 + offset]`
`LHU rd,offset(rs1)`	Load Half Unsigned	Load 16-bit unsigned	`rd ← u16[rs1 + offset]`
`SB rs2,offset(rs1)`	Store Byte	Store 8-bit	`u8[rs1 + offset] ← rs2`
`SH rs2,offset(rs1)`	Store Half	Store 16-bit	`u16[rs1 + offset] ← rs2`
`SW rs2,offset(rs1)`	Store Word	Store 32-bit	`u32[rs1 + offset] ← rs2`
`ADDI rd,rs1,imm`	Add Immediate	Add immediate	`rd ← rs1 + sx(imm)`
`SLTI rd,rs1,imm`	Set Less Than Immediate	Set if less than immediate (signed)	`rd ← sx(rs1) < sx(imm)`
`SLTIU rd,rs1,imm`	Set Less Than Immediate Unsigned	Set if less than immediate (unsigned)	`rd ← ux(rs1) < ux(imm)`
`XORI rd,rs1,imm`	XOR Immediate	XOR with immediate	`rd ← ux(rs1) ⊕ ux(imm)`
`ORI rd,rs1,imm`	OR Immediate	OR with immediate	`rd ← ux(rs1) ∨ ux(imm)`
`ANDI rd,rs1,imm`	AND Immediate	AND with immediate	`rd ← ux(rs1) ∧ ux(imm)`
`SLLI rd,rs1,imm`	Shift Left Logical Immediate	Shift left logical by immediate	`rd ← ux(rs1) « ux(imm)`
`SRLI rd,rs1,imm`	Shift Right Logical Immediate	Shift right logical by immediate	`rd ← ux(rs1) » ux(imm)`
`SRAI rd,rs1,imm`	Shift Right Arithmetic Immediate	Shift right arithmetic by immediate	`rd ← sx(rs1) » ux(imm)`
`ADD rd,rs1,rs2`	Add	Add registers	`rd ← sx(rs1) + sx(rs2)`
`SUB rd,rs1,rs2`	Subtract	Subtract registers	`rd ← sx(rs1) - sx(rs2)`
`SLL rd,rs1,rs2`	Shift Left Logical	Shift left logical by register	`rd ← ux(rs1) « rs2`
`SLT rd,rs1,rs2`	Set Less Than	Set if less than (signed)	`rd ← sx(rs1) < sx(rs2)`
`SLTU rd,rs1,rs2`	Set Less Than Unsigned	Set if less than (unsigned)	`rd ← ux(rs1) < ux(rs2)`
`XOR rd,rs1,rs2`	XOR	XOR registers	`rd ← ux(rs1) ⊕ ux(rs2)`
`SRL rd,rs1,rs2`	Shift Right Logical	Shift right logical by register	`rd ← ux(rs1) » rs2`
`SRA rd,rs1,rs2`	Shift Right Arithmetic	Shift right arithmetic by register	`rd ← sx(rs1) » rs2`
`OR rd,rs1,rs2`	OR	OR registers	`rd ← ux(rs1) ∨ ux(rs2)`
`AND rd,rs1,rs2`	AND	AND registers	`rd ← ux(rs1) ∧ ux(rs2)`
`FENCE pred,succ`	Fence	Memory fence	-
`FENCE.I`	Fence Instruction	Instruction fence	-

RV64I Base Integer Instruction Set (Additional)

Format	Name	Description	Pseudocode
`LWU rd,offset(rs1)`	Load Word Unsigned	Load 32-bit unsigned	`rd ← u32[rs1 + offset]`
`LD rd,offset(rs1)`	Load Double	Load 64-bit	`rd ← u64[rs1 + offset]`
`SD rs2,offset(rs1)`	Store Double	Store 64-bit	`u64[rs1 + offset] ← rs2`
`ADDIW rd,rs1,imm`	Add Immediate Word	Add immediate (32-bit result)	`rd ← s32(rs1) + imm`
`SLLIW rd,rs1,imm`	Shift Left Logical Immediate Word	Shift left logical immediate (32-bit)	`rd ← s32(u32(rs1) « imm)`
`SRLIW rd,rs1,imm`	Shift Right Logical Immediate Word	Shift right logical immediate (32-bit)	`rd ← s32(u32(rs1) » imm)`
`SRAIW rd,rs1,imm`	Shift Right Arithmetic Immediate Word	Shift right arithmetic immediate (32-bit)	`rd ← s32(rs1) » imm`
`ADDW rd,rs1,rs2`	Add Word	Add (32-bit result)	`rd ← s32(rs1) + s32(rs2)`
`SUBW rd,rs1,rs2`	Subtract Word	Subtract (32-bit result)	`rd ← s32(rs1) - s32(rs2)`
`SLLW rd,rs1,rs2`	Shift Left Logical Word	Shift left logical (32-bit)	`rd ← s32(u32(rs1) « rs2)`
`SRLW rd,rs1,rs2`	Shift Right Logical Word	Shift right logical (32-bit)	`rd ← s32(u32(rs1) » rs2)`

Format	Name	Description	Pseudocode
`SRAW rd,rs1,rs2`	Shift Right Arithmetic Word	Shift right arithmetic (32-bit)	`rd ← s32(rs1) » rs2`

RV32M/RV64M Multiply-Divide Extension

Format	Name	Description	Pseudocode
`MUL rd,rs1,rs2`	Multiply	Multiply (lower bits)	`rd ← ux(rs1) × ux(rs2)`
`MULH rd,rs1,rs2`	Multiply High Signed	Multiply high (signed × signed)	`rd ← (sx(rs1) × sx(rs2)) » xlen`
`MULHSU rd,rs1,rs2`	Multiply High Signed-Unsigned	Multiply high (signed × unsigned)	`rd ← (sx(rs1) × ux(rs2)) » xlen`
`MULHU rd,rs1,rs2`	Multiply High Unsigned	Multiply high (unsigned × unsigned)	`rd ← (ux(rs1) × ux(rs2)) » xlen`
`DIV rd,rs1,rs2`	Divide Signed	Divide (signed)	`rd ← sx(rs1) ÷ sx(rs2)`
`DIVU rd,rs1,rs2`	Divide Unsigned	Divide (unsigned)	`rd ← ux(rs1) ÷ ux(rs2)`
`REM rd,rs1,rs2`	Remainder Signed	Remainder (signed)	`rd ← sx(rs1) mod sx(rs2)`
`REMU rd,rs1,rs2`	Remainder Unsigned	Remainder (unsigned)	`rd ← ux(rs1) mod ux(rs2)`
`MULW rd,rs1,rs2`	Multiple Word	Multiply (32-bit result)	`rd ← u32(rs1) × u32(rs2)`
`DIVW rd,rs1,rs2`	Divide Signed Word	Divide signed (32-bit)	`rd ← s32(rs1) ÷ s32(rs2)`
`DIVUW rd,rs1,rs2`	Divide Unsigned Word	Divide unsigned (32-bit)	`rd ← u32(rs1) ÷ u32(rs2)`
`REMW rd,rs1,rs2`	Remainder Signed Word	Remainder signed (32-bit)	`rd ← s32(rs1) mod s32(rs2)`
`REMUW rd,rs1,rs2`	Remainder Unsigned Word	Remainder unsigned (32-bit)	`rd ← u32(rs1) mod u32(rs2)`

Assembler Pseudo-Instructions

Pseudo-instruction	Expansion	Description
`nop`	`addi zero,zero,0`	No operation
`li rd, expression`	(several expansions)	Load immediate
`la rd, symbol`	(several expansions)	Load address
`mv rd, rs1`	`addi rd, rs, 0`	Copy register
`not rd, rs1`	`xori rd, rs, -1`	One's complement
`neg rd, rs1`	`sub rd, x0, rs`	Two's complement
`negw rd, rs1`	`subw rd, x0, rs`	Two's complement Word
`sext.w rd, rs1`	`addiw rd, rs, 0`	Sign extend Word
`seqz rd, rs1`	`sltiu rd, rs, 1`	Set if = zero
`snez rd, rs1`	`sltu rd, x0, rs`	Set if ≠ zero
`sltz rd, rs1`	`slt rd, rs, x0`	Set if < zero
`sgtz rd, rs1`	`slt rd, x0, rs`	Set if > zero
`beqz rs1, offset`	`beq rs, x0, offset`	Branch if = zero
`bnez rs1, offset`	`bne rs, x0, offset`	Branch if ≠ zero
`blez rs1, offset`	`bge x0, rs, offset`	Branch if ≤ zero
`bgez rs1, offset`	`bge rs, x0, offset`	Branch if ≥ zero
`bltz rs1, offset`	`blt rs, x0, offset`	Branch if < zero
`bgtz rs1, 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`	`bltu rt, rs, offset`	Branch if ≤, unsigned
`j offset`	`jal x0, offset`	Jump
`jr offset`	`jal x1, offset`	Jump register
`ret`	`jalr x0, x1, 0`	Return from subroutine

CSR Instructions

Format	Name	Description
`CSRRW rd, csr, rs1`	CSR Read and Write	Atomic read and write CSR
`CSRRS rd, csr, rs1`	CSR Read and Set	Atomic read and set bits in CSR
`CSRRC rd, csr, rs1`	CSR Read and Clear	Atomic read and clear bits in CSR
`CSRRWI rd, csr, imm5`	CSR Read and Write Immediate	Atomic read and write CSR (immediate)
`CSRRSI rd, csr, imm5`	CSR Read and Set Immediate	Atomic read and set bits (immediate)
`CSRRCI rd, csr, imm5`	CSR Read and Clear Immediate	Atomic read and clear bits (immediate)

Registers

RISC-V Register Set and ABI Names

Register	ABI Name	Description	Saver
`x0`	`zero`	Hard-wired zero	-
`x1`	`ra`	Return address	Caller
`x2`	`sp`	Stack pointer	Callee
`x3`	`gp`	Global pointer	-
`x4`	`tp`	Thread pointer	-
`x5`	`t0`	Temporary/alternate link register	Caller
`x6-x7`	`t1-t2`	Temporaries	Caller
`x8`	`s0/fp`	Saved register/frame pointer	Callee
`x9`	`s1`	Saved register	Callee
`x10-x11`	`a0-a1`	Function arguments/return values	Caller
`x12-x17`	`a2-a7`	Function arguments	Caller
`x18-x27`	`s2-s11`	Saved registers	Callee
`x28-x31`	`t3-t6`	Temporaries	Caller