

# RISC-V REFERENCE

## RISC-V Instruction Set

### Core Instruction Formats

|                       |    |    |    |     |    |     |    |        |    |             |   |        |   |        |
|-----------------------|----|----|----|-----|----|-----|----|--------|----|-------------|---|--------|---|--------|
| 31                    | 27 | 26 | 25 | 24  | 20 | 19  | 15 | 14     | 12 | 11          | 7 | 6      | 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 |

### RV32I Base Integer Instructions

| Inst   | Name                    | FMT | Opcode  | F3  | F7   | Description (C)              | Note         |
|--------|-------------------------|-----|---------|-----|------|------------------------------|--------------|
| add    | ADD                     | R   | 0000011 | 0x0 | 0x00 | rd = rs1 + rs2               |              |
| sub    | SUB                     | R   | 0000011 | 0x0 | 0x20 | rd = rs1 - rs2               |              |
| xor    | XOR                     | R   | 0000011 | 0x4 | 0x00 | rd = rs1 ^ rs2               |              |
| or     | OR                      | R   | 0000011 | 0x6 | 0x00 | rd = rs1   rs2               |              |
| and    | AND                     | R   | 0000011 | 0x7 | 0x00 | rd = rs1 & rs2               |              |
| sll    | Shift Left Logical      | R   | 0000011 | 0x1 | 0x00 | rd = rs1 << rs2              |              |
| srl    | Shift Right Logical     | R   | 0000011 | 0x2 | 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         |              |
| sltu   | Set Less Than (U)       | R   | 0110011 | 0x3 |      | rd = (rs1 < rs2)?1:0         | zero-extends |
| addi   | ADD Immediate           | I   | 0010011 | 0x0 | 0x00 | rd = rs1 + imm               |              |
| xori   | XOR Immediate           | I   | 0010011 | 0x0 | 0x00 | rd = rs1 ^ imm               |              |
| ori    | OR Immediate            | I   | 0010011 | 0x0 | 0x00 | rd = rs1   imm               |              |
| andi   | AND Immediate           | I   | 0010011 | 0x0 | 0x00 | rd = rs1 & imm               |              |
| slli   | Shift Left Logical Imm  | I   | 0010011 | 0x1 | 0x00 | rd = rs1 << imm              |              |
| srl    | 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 | 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 ==               | 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 ≤                | B   | 1100011 | 0x5 |      | if(rs1 ≥ rs2) PC += imm      |              |
| bltu   | 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         |              |
| jalr   | Jump And Link Reg       | I   | 1101111 | 0x0 |      | rd = PC+4; PC = rs1          |              |
| 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 | 0x00 | Transfer control to OS       | imm: 0x000   |
| 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

- **aq**: acquire access bit – this operation must occur before later memory ops
- **rl**: release access bit – this operation must occur after earlier memory ops

|           |                   |    |    |     |         |     |      |   |    |    |   |        |   |
|-----------|-------------------|----|----|-----|---------|-----|------|---|----|----|---|--------|---|
| 31        | 27                | 26 | 25 | 24  | 20      | 19  | 15   | 14  | 12 | 11 | 7 | 6      | 0 |
| funct5    |                   | aq | rl | rs2 |         | rs1 |      | funct3  |    | rd |   | opcode |   |
| 5         |                   | 1  | 1  | 5   |         | 5   |      | 3   |    | 5  |   | 7      |   |
| Inst      | Name              |    |    | FMT | Opcode  | F3  | F5   | Description (C)   |    |    |   |        |   |
| lr.w      | Load Reserved     |    |    | R   | 0101111 | 0x2 | 0x02 | rd = M[rs1], reserve M[rs1]                               |    |    |   |        |   |
| sc.w      | Store Conditional |    |    | R   | 0101111 | 0x2 | 0x03 | if (reserved) { M[rs1] = rs2; rd = 0 }<br>else { rd = 1 } |    |    |   |        |   |
| amoswap.w | Atomic Swap       |    |    | R   | 0101111 | 0x2 | 0x01 | rd = M[rs1]; swap(rd, rs2); M[rs1] = rd                   |    |    |   |        |   |
| amoadd.w  | Atomic ADD        |    |    | R   | 0101111 | 0x2 | 0x00 | rd = M[rs1] + rs2; M[rs1] = rd                            |    |    |   |        |   |
| amoand.w  | Atomic AND        |    |    | R   | 0101111 | 0x2 | 0x0C | rd = M[rs1] & rs2; M[rs1] = rd                            |    |    |   |        |   |
| amoor.w   | Atomic OR         |    |    | R   | 0101111 | 0x2 | 0x0A | rd = M[rs1]   rs2; M[rs1] = rd                            |    |    |   |        |   |
| amoxor.w  | Atomix XOR        |    |    | R   | 0101111 | 0x2 | 0x04 | rd = M[rs1] ^ rs2; M[rs1] = rd                            |    |    |   |        |   |
| amomax.w  | Atomic MAX        |    |    | R   | 0101111 | 0x2 | 0x14 | rd = max(M[rs1], rs2); M[rs1] = rd                        |    |    |   |        |   |
| amomin.w  | Atomic MIN        |    |    | R   | 0101111 | 0x2 | 0x10 | rd = min(M[rs1], rs2); 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     | *   |        |    |    | $rd = \text{abs}(rs1) * \text{sgn}(rs2)$  |
| fsgnjn.s  | Flt Sign Neg Injection | *   |        |    |    | $rd = \text{abs}(rs1) * -\text{sgn}(rs2)$ |
| fsgnjx.s  | Flt Sign Xor Injection | *   |        |    |    | $rd = rs1 * \text{sgn}(rs2)$              |
| fmin.s    | Flt Minimum            | *   |        |    |    | $rd = \min(rs1, rs2)$                     |
| fmax.s    | Flt Maximum            | *   |        |    |    | $rd = \max(rs1, rs2)$                     |
| fcvt.s.w  | Flt Conv from Sign Int | *   |        |    |    | $rd = (\text{float}) rs1$                 |
| fcvt.s.wu | Flt Conv from Uns Int  | *   |        |    |    | $rd = (\text{float}) rs1$                 |
| fcvt.w.s  | Flt Convert to Int     | *   |        |    |    | $rd = (\text{int32\_t}) rs1$              |
| fcvt.wu.s | Flt Convert to Int     | *   |        |    |    | $rd = (\text{uint32\_t}) rs1$             |
| fmv.x.w   | Move Float to Int      | *   |        |    |    | $rd = *((\text{int}*) \&rs1)$             |
| fmv.w.x   | Move Int to Float      | *   |        |    |    | $rd = *((\text{float}*) \&rs1)$           |
| feq.s     | Float Equality         | *   |        |    |    | $rd = (rs1 == rs2) ? 1 : 0$               |
| flt.s     | Float Less Than        | *   |        |    |    | $rd = (rs1 < rs2) ? 1 : 0$                |
| fle.s     | Float Less / Equal     | *   |        |    |    | $rd = (rs1 \leq rs2) ? 1 : 0$             |
| fclass.s  | Float Classify         | *   |        |    |    | $rd = 0..9$                               |

## Pseudo Instructions

| Pseudoinstruction         | Base Instruction(s)  | Meaning                         |
|---------------------------|--|---------------------------------|
| la rd, symbol             | auipc rd, symbol[31:12]<br>addi rd, rd, symbol[11:0]       | Load address                    |
| l{b h w d} rd, symbol     | auipc rd, symbol[31:12]<br>l{b h w d} rd, symbol[11:0](rd) | Load global                     |
| s{b h w d} rd, symbol, rt | auipc rt, symbol[31:12]<br>s{b h w d} rd, symbol[11:0](rt) | Store global                    |
| fl{w d} rd, symbol, rt    | auipc rt, symbol[31:12]<br>fl{w d} rd, symbol[11:0](rt)    | Floating-point load global      |
| fs{w d} rd, symbol, rt    | auipc rt, symbol[31:12]<br>fs{w d} rd, symbol[11:0](rt)    | Floating-point store global     |
| nop                       | addi x0, x0, 0   | No operation                    |
| li rd, immediate          | <i>Myriad sequences</i>                                    | 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, 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   | 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             | fsgnjd.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          |
| call offset               | auipc x1, offset[31:12]<br>jalr x1, x1, offset[11:0]       | Call far-away subroutine        |
| tail offset               | auipc x6, offset[31:12]<br>jalr x0, x6, offset[11:0]       | Tail call far-away subroutine   |
| fence                     | fence iorw, iorw   | 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         | —      |
| 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 |