# RISC-V Reference

James Zhu <jameszhu@berkeley.edu>

## 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 | Usage | Description (C) | Note |
|---|---|---|---|---|---|
| add | ADD | R | add rd, rs1, rs2 | rd = rs1 + rs2 | |
| sub | SUB | R | sub rd, rs1, rs2 | rd = rs1 - rs2 | |
| xor | XOR | R | xor rd, rs1, rs2 | rd = rs1 ^ rs2 | |
| or | OR | R | or rd, rs1, rs2 | rd = rs1 \| rs2 | |
| and | AND | R | and rd, rs1, rs2 | rd = rs1 & rs2 | |
| sll | Shift Left Logical | R | sll rd, rs1, rs2 | rd = rs1 << rs2 | |
| srl | Shift Right Logical | R | srl rd, rs1, rs2 | rd = rs1 >> rs2 | |
| sra | Shift Right Arith* | R | sra rd, rs1, rs2 | rd = rs1 >> rs2 | msb-extends |
| slt | Set Less Than | R | slt rd, rs1, rs2 | rd = (rs1 < rs2)?1:0 | |
| sltu | Set Less Than (U) | R | sltu rd, rs1, rs2 | rd = (rs1 < rs2)?1:0 | zero-extends |
| addi | ADD Immediate | I | addi rd, rs1, imm | rd = rs1 + imm | |
| xori | XOR Immediate | I | xorii rd, rs1, imm | rd = rs1 ^ imm | |
| ori | OR Immediate | I | orii rd, rs1, imm | rd = rs1 \| imm | |
| andi | AND Immediate | I | andi rd, rs1, imm | rd = rs1 & imm | |
| slli | Shift Left Logical Imm | I | slli rd, rs1, imm | rd = rs1 << imm[0:4] | |
| srli | Shift Right Logical Imm | I | srli rd, rs1, imm | rd = rs1 >> imm[0:4] | |
| srai | Shift Right Arith Imm | I | srai rd, rs1, imm | rd = rs1 >> imm[0:4] | msb-extends |
| slti | Set Less Than Imm | I | slti rd, rs1, imm | rd = (rs1 < imm)?1:0 | |
| sltiu | Set Less Than Imm (U) | I | sltiu rd, rs1, imm | rd = (rs1 < imm)?1:0 | zero-extends |
| lb | Load Byte | I | lb rd, imm(rs1) | rd = M[rs1+imm][0:7] | |
| lh | Load Half | I | lh rd, imm(rs1) | rd = M[rs1+imm][0:15] | |
| lw | Load Word | I | lw rd, imm(rs1) | rd = M[rs1+imm][0:31] | |
| lbu | Load Byte (U) | I | lbu rd, imm(rs1) | rd = M[rs1+imm][0:7] | zero-extends |
| lhu | Load Half (U) | I | lhu rd, imm(rs1) | rd = M[rs1+imm][0:15] | zero-extends |
| sb | Store Byte | S | sb rd, imm(rs1) | M[rs1+imm][0:7] = rs2[0:7] | |
| sh | Store Half | S | sh rd, imm(rs1) | M[rs1+imm][0:15] = rs2[0:15] | |
| sw | Store Word | S | sw rd, imm(rs1) | M[rs1+imm][0:31] = rs2[0:31] | |
| beq | Branch == | B | beq rs1, rs2, imm | if(rs1 == rs2) PC += imm | |
| bne | Branch != | B | bne rs1, rs2, imm | if(rs1 != rs2) PC += imm | |
| blt | Branch < | B | blt rs1, rs2, imm | if(rs1 < rs2) PC += imm | |
| bge | Branch $\geq$ | B | bge rs1, rs2, imm | if(rs1 >= rs2) PC += imm | |
| bltu | Branch < (U) | B | bltu rs1, rs2, imm | if(rs1 < rs2) PC += imm | zero-extends |
| bgeu | Branch $\geq$ (U) | B | bgeu rs1, rs2, imm | if(rs1 >= rs2) PC += imm | zero-extends |
| jal | Jump And Link | J | jal rd, imm | rd = PC+4; PC += imm | |
| jalr | Jump And Link Reg | I | jalr rd, rs1, imm | rd = PC+4; PC = rs1 + imm | |
| lui | Load Upper Imm | U | lui rd, imm | rd = imm << 12 | |
| auipc | Add Upper Imm to PC | U | auioc rd, imm | rd = PC + (imm << 12) | |
| ecall | Environment Call | I | ecall | Transfer control to OS | |
| ebreak | Environment Break | I | ebreak | Transfer control to debugger | |

# 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` | *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, 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` | `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 |
| `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 | Callee |
| x3 | gp | Global pointer | — |
| x4 | tp | Thread pointer | — |
| x5-x7 | 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 |