RISC-V Instruction Set Summary



This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](http://creativecommons.org/licenses/by-nc-sa/4.0/).

Copyright © Codecubix.eu, fbourge

[Table 1 : RISC-V Registers names and numbers 2](#_Toc115710768)

[Table 2 : RISC-V (32-bits) instruction formats 3](#_Toc115710769)

[Table 3 : RV32I RISC-V Integer instructions 4](#_Toc115710770)

[Table 4 : RISC-V compressed (16-bits) instruction formats 4](#_Toc115710771)

Table 1 : RISC-V Registers names and numbers

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | RISC-V Base | Ext | ABI Name | Register | Description |
| 0 | Integer  Base  -  **RV32I**  **RV64I**  **RV128I** | **RV32-E** | zero | x0 | Hard-wired zero |
| 1 | ra | x1 | Return address |
| 2 | sp | x2 | Stack pointer |
| 3 | gp | x3 | Global pointer |
| 4 | tp | x4 | Thread pointer |
| 5 | t0 | x5 | Temporary/alternate link register |
| 6-7 | t1-2 | x6-7 | Temporaries |
| 8 | s0 / fp | x8 | Saved register/frame pointer |
| 9 | s1 | x9 | Saved register |
| 10-11 | a0-a1 | x10-11 | Function arguments/return values |
| 12-15 | a2-a5 | x12-15 | Function arguments |
| 16-17 |  | | a6-a7 | x16-17 | Function arguments |
| 18-27 | s2-s11 | x18-27 | Saved registers |
| 28-31 | t3-t6 | x28-31 | Temporaries |
| 32-39 | Floating  Point  -  **F** | | ft0-7 | f0-7 | FP temporaries |
| 40-41 | fs0-1 | f8-9 | FP saved registers |
| 42-43 | fa0-1 | f10-11 | FP arguments/return values |
| 44-49 | fa2-7 | f12-17 | FP arguments |
| 50-59 | fs2-11 | f18-27 | FP saved registers |
| 60-63 | ft8-11 | f28-31 | FP temporaries |

Table 2 : RISC-V (32-bits) instruction formats

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 31:25 | | 24:20 | 19:15 | 14:12 | 11:7 | 6:0 |  |
| **func7** | | rs2 | rs1 | **func3** | rd | **op** | **R-TYPE** |
| imm11:0 | | | rs1 | **func3** | rd | **op** | **I-TYPE** |
| imm11:5 | | rs2 | rs1 | **func3** | imm4:0 | **op** | **S-TYPE** |
| imm12,10:5 | | rs2 | rs1 | **func3** | imm4:1,11 | **op** | **SB-TYPE** |
| imm31:12 | | | | | rd | **op** | **U-TYPE** |
| imm20,10:1,11,19:12 | | | | | rd | **op** | **UJ-TYPE** |
| fs3 | **func2** | fs2 | fs1 | **func3** | fd | **op** | **R4-TYPE** |
| 5bits | 2bits | 5bits | 5bits | 3bits | 5bits | 7bits | 32bits |

**Instruction formats details**

In the base ISA, there are four core instruction formats (R/I/S/U) :

|  |  |
| --- | --- |
| * R-TYPE | Register-register ALU instructions : add, xor, mul |
| * I-TYPE | Immediate ALU instructions, load instructions : addi, lw, jalr, slli |
| * S-TYPE | Store instructions : sw, sb |
| * SB-TYPE | Comparison and branch instructions: beq, bge |
| * U-TYPE | Instructions with upper immediates |
| * UJ-TYPES | Jump instructions: jal |
|  |  |
| * func2 | type of operation on 2bits |
| * func3 | type of operation on 3bits |
| * func4 | type of operation on 4bits |
| * func6 | type of operation on 6bits |
| * func7 | type of operation on 7bits |

**Glossary of instruction descriptions**

|  |  |
| --- | --- |
| * rs1, rs2 | Register descriptors\* : Source operands 1 and 2 |
| * rd | Register descriptor\* : Destination operand |
| * **op** | Operation code |

\*Register descriptors (rs1,rs2, rd) always need 5 bits to address all possible 32 (2^5) working registers (from x0 to x31). That is partly why it is not possible to use CSR directly into regular instructions. In “Priviledge / CSR instructions” that are all I-TYPE, the csr operand is coded by imm11:0 (on 12 bits), and that is what theorically allows to address up to 4096 CSR (2^12). In addition, in the “Priviledge / CSR instructions”, the 5-bit unsigned immediate (uimm) is coded in the rs1 field and not in the imm11:0 field as it should be because of its previous use.

|  |  |
| --- | --- |
| * imm | signed immediate in imm11:0 |
| * uimm | 5-bit unsigned immediate in imm4:0 |
| * upimm | 20 upper bits of a 32-bit immediate, in imm31:12 |
| * Address | memory address : rs1 + SignExt(imm11:0) |
| * [Address] | data at memory location Address |
| * BTA | branch target address : PC + SignExt({imm12:1, 1’b0}) |
| * JTA | jump target address : PC + SignExt({imm20:1, 1’b0}) |
| * Label | text indicating instruction address |
| * SignExt | value sign-extended to 32 bits |
| * ZeroExt | value zero-extended to 32 bits |
| * csr | constrol and status register |

Table 3 : RV32I RISC-V Integer instructions

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| op | Func3 | Func7 | Type | Mnemonic | Description | Operation |
| 0000011(**3**) | 000 |  | **I** | **lb** rd, imm(rs1) | **Load** byte | rd = SignExt([Address]7:0) |
| 0000011(**3**) | 001 |  | **I** | **lh** rd, imm(rs1) | **Load** half | rd = SignExt([Address]15:0) |
| 0000011(**3**) | 010 |  | **I** | **lw** rd, imm(rs1) | **Load** word | rd = ([Address]31:0) |
| 0000011(**3**) | 100 |  | **I** | **lbu** rd, imm(rs1) | **Load** byte unsigned | rd = ZeroExt([Address]7:0) |
| 0000011(**3**) | 101 |  | **I** | **lhu** rd, imm(rs1) | **Load** half unsigned | rd = ZeroExt([Address]15:0) |
| 0010011(**19**) | 000 |  | **I** | addi rd, rs1, imm | ADD immediate | rd = rs1 + SignExt(imm) |
| 0010011(**19**) | 001 |  | **I** | slli rd, rs1, uimm | Shift left logical immediate | rd = rs1 << uimm |
| 0010011(**19**) | 010 |  | **I** | slti rd, rs1, imm | Set less than immediate | rd = rs1 < SignExt(imm) |
| 0010011(**19**) | 011 | 0000000 | **I** | sltiu rd, rs1, imm | Set less than imm. unsigned | rd = rs1 < SignExt(imm) |
| 0010011(**19**) | 100 |  | **I** | xori rd, rs1, imm | XOR immediate | rd = rs1 ^ SignExt(imm) |
| 0010011(**19**) | 101 | 0000000 | **I** | srli rd, rs1, uimm | Shift right logical immediate | rd = rs1 >> uimm |
| 0010011(**19**) | 101 | 0100000 | **I** | srai rd, rs1, uimm | Shift right arithmetic immediate | rd = rs1 >> uimm |
| 0010011(**19**) | 110 |  | **I** | ori rd, rs1, uimm | OR immediate | rd = rs1 | SignExt(imm) |
| 0010011(**19**) | 111 |  | **I** | andi rd, rs1, uimm | AND immediate | rd = rs1 & SignExt(imm) |
| 0010111(**23**) |  |  | **U** | auipc rd, rs1, uimm | ADD upper immediate to PC | rd = (upimm, 12’b0) + PC |
| 0100011(**35**) |  |  | **S** | **sb** rs2,imm(rs1) | **Store** byte | [Address]7:0 = rs27:0 |
| 0100011(**35**) |  |  | **S** | **sh** rs2,imm(rs1) | **Store** half | [Address]15:0 = rs215:0 |
| 0100011(**35**) |  |  | **S** | **sw** rs2,imm(rs1) | **Store** word | [Address]31:0 = rs2 |
| 0110011(**51**) | 000 | 0000000 | **R** | add rd, rs1, rs2 | ADD | rd = rs1 + rs2 |
| 0110011(**51**) | 000 | 0100000 | **R** | sub rd, rs1, rs2 | SUB | rd = rs1 - rs2 |
| 0110011(**51**) | 001 | 0000000 | **R** | sll rd, rs1, rs2 | Shift left logical | rd = rs1 << rs24:0 |
| 0110011(**51**) | 010 | 0000000 | **R** | slt rd, rs1, rs2 | Set less than | rd = rs1 < rs2 |
| 0110011(**51**) | 011 | 0000000 | **R** | sltu rd, rs1, rs2 | Set less than unsigned | rd = rs1 < rs2 |
| 0110011(**51**) | 100 | 0000000 | **R** | xor rd, rs1, rs2 | XOR | rd = rs1 ^ rs2 |
| 0110011(**51**) | 101 | 0000000 | **R** | srl rd, rs1, rs2 | Shift right logical | rd = rs1 >> rs24:0 |
| 0110011(**51**) | 101 | 0100000 | **R** | sra rd, rs1, rs2 | Shift right arithmetic | rd = rs1 >>>rs24:0 |
| 0110011(**51**) | 110 | 0000000 | **R** | or rd, rs1, rs2 | OR | rd = rs1 | rs2 |
| 0110011(**51**) | 111 | 0000000 | **R** | and rd, rs1, rs2 | AND | rd = rs1 & rs2 |
| 0110111(**55**) | - |  | **U** | lui rd, upimm | Load upper immediate | rd = {upimm, 12’b0} |
| 1100011(**99**) | 000 |  | **B** | **beq** rs1,rs2, label | **Branch** if equal = | if (rs1 == rs2) PC = BTA |
| 1100011(**99**) | 001 |  | **B** | **bne** rs1,rs2, label | **Branch** if not equal ≠ | if (rs1 != rs2) PC = BTA |
| 1100011(**99**) | 010 |  | **B** | **blt** rs1,rs2, label | **Branch** if lower than < | if (rs1 < rs2) PC = BTA |
| 1100011(**99**) | 011 |  | **B** | **bge** rs1,rs2, label | **Branch** if greater / equal ≥ | if (rs1 ≥ rs2) PC = BTA |
| 1100011(**99**) | 100 |  | **B** | **bltu** rs1,rs2, label | **Branch** if lower than unsigned < | if (rs1 < rs2) PC = BTA |
| 1100011(**99**) | 101 |  | **B** | **bgeu** rs1,rs2, label | **Branch** if greater / equal unsign. ≥ | if (rs1 ≥ rs2) PC = BTA |
| 1100111(**103**) | 000 |  | **I** | jalr rd, rs1, label | Jump and link register | PC = rs1 + SignExt(imm)  rd = PC + 4 |
| 1101111(**111**) | - |  | **J** | jal rd, label | Jump and link | PC = JTA  rd = PC + 4 |

Table 4 : RISC-V compressed (16-bits) instruction formats

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15:13 | 12 | 11 | 10 | 9 | 8 | 7 | 6:5 | 4:2 | 1:0 |  |
| **func4** | | rd/rs1 | | | | | rs2 | | **op** | **CR-TYPE** |
| **func3** | imm | rd/rs1 | | | | | imm | | **op** | **CI-TYPE** |
| **func3** | imm | | | rs1’ | | | imm | rs2’ | **op** | **CS-TYPE** |
| **func6** | | | | rd/rs1 | | | **func2** | rs2’ | **op** | **CS’-TYPE** |
| **func3** | imm | | | rs1’ | | | imm | | **op** | **CB-TYPE** |
| **func3** | imm | **func** | | rd’/rs1’ | | | imm | | **op** | **CB’-TYPE** |
| **func3** | imm | | | | | | | | **op** | **CJ-TYPE** |
| **func3** | imm | | | | | | rs2 | | **op** | **CSS-TYPE** |
| **func3** | imm | | | | | | | rd’ | **op** | **CIW-TYPE** |
| **func3** | imm | | | rs1’ | | | imm | rd’ | **op** | **CL-TYPE** |
| 3bits | 1bit | - | - | - | - | - | 2bits | 3bits | 2bits | 16bits |