RISC-V Instruction Set Summary

31:25		24:20	19:15	14:12	11:7	6:0	
funct7		rs2	rs1	funct3	rd	op	**R-Type**
$imm_{11:0}$			rs1	funct3	rd	op	**I-Type**
$imm_{11:5}$		rs2	rs1	funct3	$imm_{4:0}$	op	**S-Type**
$imm_{12,10:5}$		rs2	rs1	funct3	$imm_{4:1,11}$	op	**B-Type**
$imm_{31:12}$					rd	op	**U-Type**
$imm_{20,10:1,11,19:12}$					rd	op	**J-Type**
fs3	funct2	fs2	fs1	funct3	fd	op	**R4-Type**
5 bits	2 bits	5 bits	5 bits	3 bits	5 bits	7 bits	

Figure B.1 RISC-V 32-bit instruction formats

- `imm`: signed immediate in $\mathbf{imm}_{11:0}$
- `uimm`: 5-bit unsigned immediate in $\mathbf{imm}_{4:0}$
- `upimm`: 20 upper bits of a 32-bit immediate, in $\mathbf{imm}_{31:12}$
- `Address`: memory address: **rs1** + SignExt($\mathbf{imm}_{11:0}$)
- `[Address]`: data at memory location Address
- `BTA`: branch target address: PC + SignExt({$\mathbf{imm}_{12:1}$, 1'b0})
- `JTA`: jump target address: PC + SignExt({$\mathbf{imm}_{20:1}$, 1'b0})
- `label`: text indicating instruction address
- `SignExt`: value sign-extended to 32 bits
- `ZeroExt`: value zero-extended to 32 bits
- `csr`: control and status register

Table B.1 RV32I: RISC-V integer instructions

op	funct3	funct7	Type	Instruction	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	0000000*	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	–	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 imm.	rd = rs1 >>> uimm
0010011 (19)	110	–	I	`ori rd, rs1, imm`	or immediate	rd = rs1 | SignExt(imm)
0010011 (19)	111	–	I	`andi rd, rs1, imm`	and immediate	rd = rs1 & SignExt(imm)
0010111 (23)	–	–	U	`auipc rd, upimm`	add upper immediate to PC	rd = {upimm, 12'b0} + PC
0100011 (35)	000	–	S	`sb rs2, imm(rs1)`	store byte	[Address]$_{7:0}$ = rs2$_{7:0}$
0100011 (35)	001	–	S	`sh rs2, imm(rs1)`	store half	[Address]$_{15:0}$ = rs2$_{15:0}$
0100011 (35)	010	–	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 << rs2$_{4:0}$
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 >> rs2$_{4:0}$
0110011 (51)	101	0100000	R	`sra rd, rs1, rs2`	shift right arithmetic	rd = rs1 >>> rs2$_{4:0}$
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 =	if (rs1 == rs2) PC = BTA
1100011 (99)	001	–	B	`bne rs1, rs2, label`	branch if ≠	if (rs1 ≠ rs2) PC = BTA
1100011 (99)	100	–	B	`blt rs1, rs2, label`	branch if <	if (rs1 < rs2) PC = BTA
1100011 (99)	101	–	B	`bge rs1, rs2, label`	branch if ≥	if (rs1 ≥ rs2) PC = BTA
1100011 (99)	110	–	B	`bltu rs1, rs2, label`	branch if < unsigned	if (rs1 < rs2) PC = BTA
1100011 (99)	111	–	B	`bgeu rs1, rs2, label`	branch if ≥ unsigned	if (rs1 ≥ rs2) PC = BTA
1100111 (103)	000	–	I	`jalr rd, rs1, imm`	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

*Encoded in instr$_{31:25}$, the upper seven bits of the immediate field