1. RISC-V (RV32I) Cheatsheet

1.1. Registers

1.1.1. General-Purpose Registers (x0–x31)

ABI Name	Register	Description
x0	zero	Hard-wired zero
x1	ra	Return address
x2	sp	Stack pointer
x3	gp	Global pointer
x4	tp	Thread pointer
x5–x7	t0–t2	Temporary registers (caller-saved)
x8–x9	s0/fp	Saved register/Frame pointer (callee-saved)
x10–x11	a0–a1	Function arguments/Return values
x12–x17	a2–a7	Function arguments
x18–x27	s2–s11	Saved registers (callee-saved)
x28–x31	t3–t6	Temporary registers (caller-saved)

1.1.2. Register Conventions

- **Caller-saved (t0–t6, a0–a7, ra):** Must be saved by the caller if needed after a function call.
- **Callee-saved (s0–s11, sp, gp, tp):** Must be preserved by the callee.

1.2. Instruction Types

Type	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R-Type	funct7							rs2					rs1					funct3			rd					opcode						
I-Type	imm[11:0]												rs1					funct3			rd					opcode						
S-Type	imm[11:5]							rs2					rs1					funct3			imm[4:0]					opcode						
B-Type	i[12]	imm[10:5]						rs2					rs1					funct3			imm[4:1]				i[11]	opcode						
U-Type	imm[31:12]																				rd					opcode						
J-Type	i[20]	imm[10:1]										i[11]	imm[19:12]								rd					opcode						

Note: i[...] is the same as imm[...]

1.3. RV32I Base Instructions

1.3.1. Arithmetic/Logic

Type	Instruction	Syntax	Semantics	Description
R-Type	add	`add rd, rs1, rs2`	`rd = rs1 + rs2`	Add
R-Type	sub	`sub rd, rs1, rs2`	`rd = rs1 - rs2`	Subtract
R-Type	sll	`sll rd, rs1, rs2`	`rd = rs1 << rs2`	Shift left logical
R-Type	slt	`slt rd, rs1, rs2`	`rd = (rs1 < rs2) ? 1 : 0`	Set less than (signed)
R-Type	sltu	`sltu rd, rs1, rs2`	`rd = (rs1 < rs2) ? 1 : 0`	Set less than (unsigned)
R-Type	xor	`xor rd, rs1, rs2`	`rd = rs1 ^ rs2`	Bitwise XOR
R-Type	srl	`srl rd, rs1, rs2`	`rd = rs1 >> rs2`	Shift right logical
R-Type	sra	`sra rd, rs1, rs2`	`rd = rs1 >>> rs2`	Shift right arithmetic
R-Type	or	`or rd, rs1, rs2`	`rd = rs1 | rs2`	Bitwise OR
R-Type	and	`and rd, rs1, rs2`	`rd = rs1 & rs2`	Bitwise AND

1.3.2. Immediate Instructions

Type	Instruction	Syntax	Semantics	Description
I-Type	addi	`addi rd, rs1, imm`	`rd = rs1 + sign_ext(imm)`	Add immediate
I-Type	slti	`slti rd, rs1, imm`	`rd = (rs1 < sign_ext(imm)) ? 1 : 0`	Set less than immediate (signed)
I-Type	sltiu	`sltiu rd, rs1, imm`	`rd = (rs1 < imm) ? 1 : 0`	Set less than immediate (unsigned)
I-Type	xori	`xori rd, rs1, imm`	`rd = rs1 ^ sign_ext(imm)`	Bitwise XOR immediate
I-Type	ori	`ori rd, rs1, imm`	`rd = rs1 | sign_ext(imm)`	Bitwise OR immediate
I-Type	andi	`andi rd, rs1, imm`	`rd = rs1 & sign_ext(imm)`	Bitwise AND immediate
I-Type	slli	`slli rd, rs1, shamt`	`rd = rs1 << shamt`	Shift left logical immediate
I-Type	srli	`srli rd, rs1, shamt`	`rd = rs1 >> shamt`	Shift right logical immediate
I-Type	srai	`srai rd, rs1, shamt`	`rd = rs1 >>> shamt`	Shift right arithmetic immediate

1.3.3. Load/Store

Type	Instruction	Syntax	Semantics	Description
I-Type	lb	`lb rd, offset(rs1)`	`rd = sign_ext(M[rs1 + offset][7:0])`	Load byte (signed)
I-Type	lh	`lh rd, offset(rs1)`	`rd = sign_ext(M[rs1 + offset][15:0])`	Load halfword (signed)
I-Type	lw	`lw rd, offset(rs1)`	`rd = sign_ext(M[rs1 + offset][31:0])`	Load word

Type	Instruction	Syntax	Semantics	Description
I-Type	lbu	`lbu rd, offset(rs1)`	`rd = zero_ext(M[rs1 + offset][7:0])`	Load byte (unsigned)
I-Type	lhu	`lhu rd, offset(rs1)`	`rd = zero_ext(M[rs1 + offset][15:0])`	Load halfword (unsigned)
S-Type	sb	`sb rs2, offset(rs1)`	`M[rs1 + offset][7:0] = rs2[7:0]`	Store byte
S-Type	sh	`sh rs2, offset(rs1)`	`M[rs1 + offset][15:0] = rs2[15:0]`	Store halfword
S-Type	sw	`sw rs2, offset(rs1)`	`M[rs1 + offset][31:0] = rs2[31:0]`	Store word

1.3.4. Control Flow

Type	Instruction	Syntax	Semantics	Description
B-Type	beq	`beq rs1, rs2, offset`	`if (rs1 == rs2) PC += offset`	Branch if equal
B-Type	bne	`bne rs1, rs2, offset`	`if (rs1 != rs2) PC += offset`	Branch if not equal
B-Type	blt	`blt rs1, rs2, offset`	`if (rs1 < rs2) PC += offset`	Branch if less than (signed)
B-Type	bge	`bge rs1, rs2, offset`	`if (rs1 >= rs2) PC += offset`	Branch if greater or equal (signed)
B-Type	bltu	`bltu rs1, rs2, offset`	`if (rs1 < rs2) PC += offset`	Branch if less than (unsigned)
B-Type	bgeu	`bgeu rs1, rs2, offset`	`if (rs1 >= rs2) PC += offset`	Branch if greater or equal (unsigned)
J-Type	jal	`jal rd, offset`	`rd = PC + 4; PC += offset`	Jump and link
I-Type	jalr	`jalr rd, rs1, offset`	`rd = PC + 4; PC = rs1 + offset`	Jump and link register

1.3.5. System

Type	Instruction	Syntax	Semantics	Description
I-Type	ecall	`ecall`	Transfer control to OS	Environment call
I-Type	ebreak	`ebreak`	Trigger breakpoint	Environment break
I-Type	fence	`fence`	Memory ordering fence	Memory fence
I-Type	fence.i	`fence.i`	Instruction ordering fence	Instruction fence

1.4. Pseudoinstructions

Type	Instruction	Syntax	Equivalent Base Instructions	Description
I-Type	li	`li rd, imm`	`addi rd, x0, imm`	Load immediate (pseudo)
R-Type	mv	`mv rd, rs`	`addi rd, rs, 0`	Move (pseudo)
R-Type	not	`not rd, rs`	`xori rd, rs, -1`	Bitwise NOT (pseudo)
R-Type	neg	`neg rd, rs`	`sub rd, x0, rs`	Negate (pseudo)
I-Type	seqz	`seqz rd, rs`	`sltiu rd, rs, 1`	Set if equal to zero (pseudo)
I-Type	snez	`snez rd, rs`	`sltu rd, x0, rs`	Set if not equal to zero (pseudo)
R-Type	sltz	`sltz rd, rs`	`slt rd, rs, x0`	Set if less than zero (pseudo)
R-Type	sgtz	`sgtz rd, rs`	`slt rd, x0, rs`	Set if greater than zero (pseudo)
B-Type	beqz	`beqz rs, offset`	`beq rs, x0, offset`	Branch if equal to zero (pseudo)
B-Type	bnez	`bnez rs, offset`	`bne rs, x0, offset`	Branch if not equal to zero (pseudo)
B-Type	blez	`blez rs, offset`	`bge x0, rs, offset`	Branch if less or equal to zero (pseudo)
B-Type	bgez	`bgez rs, offset`	`bge rs, x0, offset`	Branch if greater or equal to zero (pseudo)
B-Type	bltz	`bltz rs, offset`	`blt rs, x0, offset`	Branch if less than zero (pseudo)
B-Type	bgtz	`bgtz rs, offset`	`blt x0, rs, offset`	Branch if greater than zero (pseudo)
J-Type	j	`j offset`	`jal x0, offset`	Jump (pseudo)
I-Type	ret	`ret`	`jalr x0, ra, 0`	Return (pseudo)
J-Type	call	`call offset`	`jal ra, offset`	Call subroutine (pseudo)

1.5. Notes

- Pseudoinstructions are translated into one or more base instructions by the assembler.
- RV32I is the base integer instruction set for 32-bit RISC-V.