RISC-V Register Conventions

REGISTER	NAME	USE	SAVER
x0	zero	The constant value 0	N.A.
x1	ra	return address	Caller
x2	sp	stack pointer	Callee
x3	gp	global data pointer	--
x4	tp	thread pointer	--
x5-x7	t0-t2	temporaries	Caller
x8	s0/fp	frame pointer	Callee
x9	s1	saved register	Callee
x10-x11	a0-a1	fn arguments / return values	Caller
x12-x17	a2-a7	function arguments	Caller
x18-x27	s2-s11	saved registers	Callee
x28-x31	t3-t6	temporaries	Caller

RISC-V Reference Sheet

31 25|24	20|19	15|14	12|11	7|6	0	
funct7	rs2	rs1	funct3	rd	opcode	R-Type
imm[31:20]		rs1	funct3	rd	opcode	I-Type
imm[31:25]	rs2	rs1	funct3	imm[11:7]	opcode	S-Type
imm[31:12]				rd	opcode	U-Type

R-Type Computational Instructions:

						Name	Mnemonic	Implementation
0000000	rs2	rs1	000	rd	0110011	ADD	ADD rd, rs1, rs2	`R[rd] = R[rs1] + R[rs2]`
0100000	rs2	rs1	000	rd	0110011	SUB	SUB rd, rs1, rs2	`R[rd] = R[rs1] - R[rs2]`
0000000	rs2	rs1	001	rd	0110011	Shift Left	SLL rd, rs1, rs2	`R[rd] = R[rs1] << R[rs2]`
0000000	rs2	rs1	010	rd	0110011	Set Less Than	SLT rd, rs1, rs2	`R[rd] = (R[rs1] < R[rs2]) ? 1:0`
0000000	rs2	rs1	100	rd	0110011	XOR	XOR rd, rs1, rs2	`R[rd] = R[rs1] ⊕ R[rs2]`
0000000	rs2	rs1	101	rd	0110011	Shift Right Logical	SRL rd, rs1, rs2	`R[rd] = R[rs1] >>> R[rs2]`
0100000	rs2	rs1	101	rd	0110011	Shift Right Arithmetic	SRA rd, rs1, rs2	`R[rd] = R[rs1] >>> R[rs2]`
0000000	rs2	rs1	110	rd	0110011	OR	OR rd, rs1, rs2	`R[rd] = R[rs1] | R[rs2]`
0000000	rs2	rs1	111	rd	0110011	AND	AND rd, rs1, rs2	`R[rd] = R[rs1] & R[rs2]`

I-Type Computational Instructions:

					Name	Mnemonic	Implementation
imm[31:20]	rs1	000	rd	0010011	Add Immediate	ADDI rd, rs1, imm	`R[rd] = R[rs1] + imm`
imm[31:20]	rs1	001	rd	0010011	Shift Left Immediate	SLLI rd, rs1, imm	`R[rd] = R[rs1] << imm`
imm[31:20]	rs1	010	rd	0010011	Set Less Than Immediate	SLTI rd, rs1, imm	`R[rd] = (R[rs1] < imm) ? 1:0`
imm[31:20]	rs1	100	rd	0010011	XOR Immediate	XORI rd, rs1, imm	`R[rd] = R[rs1] ⊕ imm`
imm[31:20]	rs1	101	rd	0010011	Shift Right Immediate	SRLI rd, rs1, imm	`R[rd] = R[rs1] >>> imm`
imm[31:20]	rs1	101	rd	0010011	Shift Right Arith Immediate	SRAI rd, rs1, imm	`R[rd] = R[rs1] >>> imm`
imm[31:20]	rs1	110	rd	0010011	OR Immediate	ORI rd, rs1, imm	`R[rd] = R[rs1] | sign_extend(imm)`
imm[31:20]	rs1	111	rd	0010011	AND Immediate	ANDI rd, rs1, imm	`R[rd] = R[rs1] & sign_extend(imm)`

Load and Store Instructions:

						Name	Mnemonic	Implementation
imm[31:20]		rs1	000	rd	0000011	Load Byte	LB rd, imm(rs1)	`R[rd] = Mem[imm+R[rs1]]`
imm[31:20]		rs1	010	rd	0000011	Load Word	LW rd, imm(rs1)	`R[rd] = Mem[imm+R[rs1]]`
imm[31:20]		rs1	011	rd	0000011	Load Double Word	LD rd, imm(rs1)	`R[rd] = Mem[imm+R[rs1]]`
imm[31:25]	rs2	rs1	000	imm[11:7]	0100011	Store Byte	SB rs2, imm(rs1)	`Mem[imm+R[rs1]] = R[rs2]`
imm[31:25]	rs2	rs1	010	imm[11:7]	0100011	Store Word	SW rs2, imm(rs1)	`Mem[imm+R[rs1]] = R[rs2]`
imm[31:25]	rs2	rs1	011	imm[11:7]	0100011	Store Double Word	SD rs2, imm(rs1)	`Mem[imm+R[rs1]] = R[rs2]`

31 25|24	20|19	15|14	12|11	7|6	0	
funct7	rs2	rs1	funct3	rd	opcode	R-Type
imm[31:20]		rs1	funct3	rd	opcode	I-Type
imm[31:25]	rs2	rs1	funct3	imm[11:7]	opcode	S-Type
imm[31:12]				rd	opcode	U-Type

U-Type Computational Instructions:

imm[31:12]	rd	0110111

Name	Mnemonic	Implementation
Load Upper Immediate	LUI rd, imm	`R[rd] = imm << 12`

Jump and Branch Instructions:

imm[31:12]			rd	1101111	
imm[31:20]		rs1	000	rd	1100111
imm[31:25]	rs2	rs1	000	imm[11:7]	1100011
imm[31:25]	rs2	rs1	101	imm[11:7]	1100011
imm[31:25]	rs2	rs1	100	imm[11:7]	1100011
imm[31:25]	rs2	rs1	101	imm[11:7]	1100011
imm[31:25]	rs2	rs1	100	imm[11:7]	1100011
imm[31:25]	rs2	rs1	001	imm[11:7]	1100011

Name	Mnemonic	Implementation
Jump and Link	JAL rd, imm	`R[rd] = PC+4; PC = PC + imm << 1`
Jump and Link Register	JALR rd, rs1, imm	`R[rd] = PC+4; PC = (R[rs1] + imm)`
Branch Equal	BEQ rs1, rs2, imm	`PC = if (R[rs1] == R[rs2]) PC + imm << 1`
Branch Greater than or Equal	BGE rs1, rs2, imm	`PC = if (R[rs1] >=s R[rs2]) PC + imm << 1`
Branch Less Than	BLT rs1, rs2, imm	`PC = if (R[rs1] <s R[rs2]) PC + imm << 1`
Branch GT or Equal Unsigned	BGEU rs1, rs2, imm	`PC = if (R[rs1] >=u R[rs2]) PC + imm << 1`
Branch Less Than Unsigned	BLTU rs1, rs2, imm	`PC = if (R[rs1] <u R[rs2]) PC + imm << 1`
Branch Not Equal	BNE rs1, rs2, imm	`PC = if (R[rs1] != R[rs2]) PC + imm << 1`

Pseudoinstructions:

These are simple assembly language instructions that do not have a direct machine language equivalent. During assembly, the assembler translates each pseudo instruction into one or more machine language instructions.

Name	Mnemonic	Implementation
Jump	J imm	`JAL x0, imm`
Jump Register	JR rs1	`JALR x0, rs, 0`
Branch Greater Than	BGT rs1, rs2, imm	`BLT rs2, rs1, imm`
Branch Less Than or Equal	BLE rs1, rs2, imm	`BGE rs2, rs1, imm`
Branch if Equal to Zero	BEQZ rs1, imm	`BEQ rs1, x0, imm`
Branch if Not Equal to Zero	BNEZ rs1, imm	`BNE rs1, x0 imm`
Negate	NEG rd, rs1	`SUB rd, x0, rs1`
NOP	NOP	`ADDI x0, x0, x0`
Return	RET	`jalr x0, x1, 0; auipc x6, offset[31:12]`
NOT	NOT rd, rs1	`ADDI rd, rs1, 0`
Load Immediate	LI rd, imm	Myriad Sequences
Move	MV rd, rs1	`ADDI rd, rs1, 0`
Set if Equal to Zero	SEQZ rd, rs1	`SLITU rd, rs1, 1`
Set if Equal not to Zero	SNEZ rd, rs1	`SLTU rd, x0, rs`