RISC-V Reference Card

Fundamentos de Sistemas Computacionais (FSC) - L.EIC004

Licenciatura em Engenharia Informática e Computação

February 2023

RISC-V Instruction Set

Core Instruction Formats

31 ——— 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	Opcode	funct3	funct7	Description
add	ADD	R	0110011	000	0000000	rd = rs1 + rs2
sub	SUB	R	0110011	000	0100000	rd = rs1 - rs2
xor	XOR	R	0110011	100	0000000	rd = rs1 ˆ rs2
or	OR	R	0110011	110	0000000	rd = rs1 — rs2
and	AND	R	0110011	111	0000000	rd = rs1 & rs2
sll	Shift Left Logical	R	0110011	001	0000000	rd = rs1 <<rs2
srl	Shift Right Logical	R	0110011	101	0000000	rd = rs1 >>rs2
sra	Shift Right Arith.	R	0110011	101	0100000	rd = rs1 >>rs2
slt	Set Less Than	R	0110011	010	0000000	rd = (rs1 <rs2)?1:0
sltu	Set Less Than (U)	R	0110011	011	0000000	rd = (rs1 <rs2)?1:0
addi	ADD Immediate	I	0010011	000		rd = rs1 + imm
xori	XOR Immediate	I	0010011	100		rd = rs1 ˆ imm
ori	OR Immediate	I	0010011	110		rd = rs1 — imm
andi	AND Immediate	I	0010011	111		rd = rs1 & imm
slli	Shift Left Logical Imm	I	0010011	001	0000000	rd = rs1 <<imm[0:4]
srli	Shift Right Logical Imm	I	0010011	101	0000000	rd = rs1 >>imm[0:4]
srai	Shift Right Arith. Imm	I	0010011	101	0100000	rd = rs1 >>imm[0:4]
slti	Set Less Than Imm	I	0010011	010		rd = (rs1 <imm)?1:0
sltiu	Set Less Than (U) Imm	I	0010011	011		rd = (rs1 <imm)?1:0
lb	Load Byte	I	0000011	000		rd = M[rs1+imm][0:7]
lh	Load Half	I	0000011	001		rd = M[rs1+imm][0:15]
lw	Load Word	I	0000011	010		rd = M[rs1+imm][0:31]
lbu	Load Byte (U)	I	0000011	100		rd = M[rs1+imm][0:7]
lhu	Load Half (U)	I	0000011	101		rd = M[rs1+imm][0:15]
sb	Store Byte	S	0100011	000		M[rs1+imm][0:7] = rs2[0:7]
sh	Store Half	S	0100011	001		M[rs1+imm][0:15] = rs2[0:15]
sw	Store Word	S	0100011	010		M[rs1+imm][0:31] = rs2[0:31]
beq	Branch ==	B	1100011	000		if(rs1 == rs2) PC += imm
bne	Branch ≠	B	1100011	001		if(rs1 != rs2) PC += imm
blt	Branch <	B	1100011	100		if(rs1 <rs2) PC += imm
bge	Branch ≥	B	1100011	101		if(rs1 >= rs2) PC += imm
bltu	Branch < (U)	B	1100011	110		if(rs1 <rs2) PC += imm
bgeu	Branch ≥ (U)	B	1100011	111		if(rs1 >= rs2) PC += imm
jal	Jump And Link	J	1101111	-		rd = PC+4; PC += imm
jalr	Jump And Link Reg	I	1100111	000		rd = PC+4; PC = rs1 + imm
lui	Load Upper Imm	U	0110111	-		rd = imm <<12
auipc	Add Upper Imm to PC	U	0010111	-		rd = PC + (imm <<12)

RV32M Multiply Extension

Inst	Name	FMT	Opcode	funct3	funct7	Description
mul	MUL	R	0110011	000	0000001	rd = (rs1 * rs2)[31:0]
mulh	MUL High	R	0110011	001	0000001	rd = (rs1 * rs2)[63:32]
mulsu	MUL High (S) (U)	R	0110011	010	0000001	rd = (rs1 * rs2)[63:32]
mulu	MUL High (U)	R	0110011	011	0000001	rd = (rs1 * rs2)[63:32]
div	DIV	R	0110011	100	0000001	rd = rs1 / rs2
divu	DIV (U)	R	0110011	101	0000001	rd = rs1 / rs2
rem	Remainder	R	0110011	110	0000001	rd = rs1 % rs2
remu	Remainder (U)	R	0110011	111	0000001	rd = rs1 % rs2

Pseudo Instructions

Pseudoinstruction	Base Instruction(s)	Meaning
la rd, symbol	addi rd, rd, symbol[11:0]	Load address
l{b—h—w—d} rd, symbol	l{b—h—w—d} rd, symbol[11:0](rd)	Load global
s{b—h—w—d} rd, symbol, rt	s{b—h—w—d} rd, symbol[11:0](rt)	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 ≠ 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 ≠ zero
blez rs, offset	bge x0, rs, offset	Branch if ≤ zero
bgez rs, offset	bge rs, x0, offset	Branch if ≥ 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 ≤
bgtu rs, rt, offset	bltu rt, rs, offset	Branch if >, unsigned
bleu rs, rt, offset	bgeu rt, rs, offset	Branch if ≤, 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] jalr x1, x1, offset[11:0]	Call far-away subroutine
tail offset	auipc x6, offset[31:12] jalr x0, x6, offset[11:0]	Tail call far-away subroutine

Register Calling Convention

Register	ABI Name	Description	Saver
x0	zero	Zero constant	—
x1	ra	Return address	Callee
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	Function arguments/return	Caller
x12-x17	a2-a7	Function args	Caller
x18-x27	s2-s11	Saved registers	Callee
x28-x31	t3-t6	Temporaries	Caller

ALU Control

ALU Control Lines	Function
0000	AND
0001	OR
0010	add
0110	subtract

opcode	ALUOp	Operation	funct7	funct3	ALU Action	ALU Control Input
lw	00	load word	xxxxxxx	xxx	add	0010
sw	00	store word	xxxxxxx	xxx	add	0010
beq	01	branch if equal	xxxxxxx	xxx	subtract	0110
R-Type	10	add	0000000	000	add	0010
R-Type	10	sub	0100000	000	subtract	0110
R-Type	10	and	0000000	-	AND	0000
R-Type	10	or	0000000	110	OR	0001