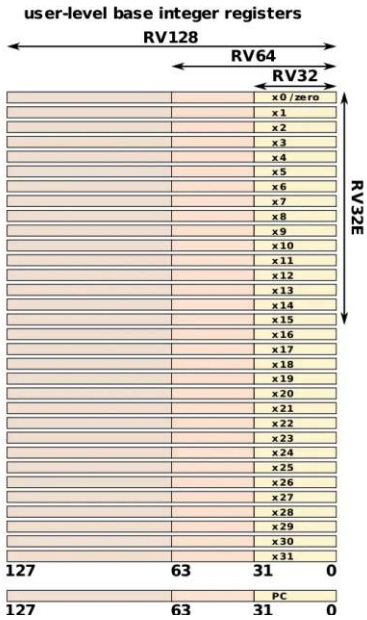
REGISTROS



Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5–7	t0–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller
f0–7	ft0–7	FP temporaries	Caller
f8–9	fs0–1	FP saved registers	Callee
f10–11	fa0–1	FP arguments/return values	Caller
f12–17	fa2–7	FP arguments	Caller
f18–27	fs2–11	FP saved registers	Callee
f28–31	ft8–11	FP temporaries	Caller

MIPS	RISC-V
Registros “R0 a R31”	Registros “x0 a x31”
Float de “f0 a f31”	Se mantiene
r0 con valor a 0	x0 con valor a 0
X31 como return address	X1 como return address

INSTRUCCIONES

Tipo	pseudo-instrucción	Equivalencia
salto	j offset	jalr x0, rs1, offset
retorno de subrutina	ret	jr ra
no operación	nop	addi x0, x0, 0
carga de inmediato	li rd, imm	muchas secuencias
copiar registro	mv rd, rs	addi rd, rs, 0
llamada subrutina	call offset	auipc x1, offset[31:12] jalr x1, x1, offset[11:0]

Tipo	Instrucción	Ejemplo	Descripción
mult.	mul	mul rd, rs1, rs2	$rd = (rs1 \times rs2)_{63..0}$
	mulh	mulh rd, rs1, rs2	$rd = (rs1 \times rs2)_{127..64}$
	mulhsu	mulhsu rd, rs1, rs2	$rd = (rs1 \times arg\_sin\_signo(rs2))_{127..64}$
	mulhu	mulhu rd, rs1, rs2	$rd = (arg\_sin\_signo(rs1) \times arg\_sin\_signo(rs2))_{127..64}$
	mulw	mulw rd, rs1, rs2	$rd = ext\_signo((rs1_{31..0} \times rs2_{31..0})_{31..0})$
división	div	div rd, rs1, rs2	$rd = rs1 \div rs2$
	divu	divu rd, rs1, rs2	$rd = arg\_sin\_signo(rs1) \div arg\_sin\_signo(rs2)$
	divw	divw rd, rs1, rs2	$rd = ext\_signo((rs1_{31..0} \div rs2_{31..0})_{31..0})$
	divuw	divuw rd, rs1, rs2	$rd = ext\_signo((arg\_sin\_signo(rs1_{31..0}) \div arg\_sin\_signo(rs2_{31..0}))_{31..0})$
resto	rem	rem rd, rs1, rs2	$rd = rs1 \,\%\, rs2$
	remu	remu rd, rs1, rs2	$rd = arg\_sin\_signo(rs1) \,\%\, arg\_sin\_signo(rs2)$
	remw	remw rd, rs1, rs2	$rd = ext\_signo((rs1_{31..0} \,\%\, rs2_{31..0})_{31..0})$
	remuw	remuw rd, rs1, rs2	$rd = ext\_signo((arg\_sin\_signo(rs1_{31..0}) \,\%\, arg\_sin\_signo(rs2_{31..0}))_{31..0})$

Tipo	Instrucción	Ejemplo	Descripción
suma	add	add rd, rs1, rs2	$rd = rs1 + rs2$
	addw	addw rd, rs1, rs2	$rd = ext\_signo(rs1_{31..0} + rs2_{31..0})$
	addi	addi rd, rd1, imm	$rd = rs1 + ext\_signo(imm_{11..0})$
	addiw	addiw rd, rs1, imm	$rd = ext\_signo(rs1_{31..0} + ext\_signo(imm_{11..0}))$
resta	sub	sub rd, rs1, rs2	$rd = rs1 - rs2$
	subw	subw rd, rs1, rs2	$rd = ext\_signo(rs1_{31..0} - rs2_{31..0})$
lógicas (bit-wise)	xor	xor rd, rs1, rs2	$rd = rs1$ *xor* $rs2$
	xori	xori rd, rs1, imm	$rd = rs1$ *xor* $ext\_signo(imm_{11..0})$
	or	or rd, rs1, rs2	$rd = rs1$ *or* $rs2$
	ori	ori rd, rs1, imm	$rd = rs1$ *or* $ext\_signo(imm_{11..0})$
	and	and rd, rs1, rs2	$rd = rs1$ *and* $rs2$
	andi	andi rd, rs1, imm	$rd = rs1$ *and* $ext\_signo(imm_{11..0})$
salto	jal	jal rd, imm	$rd = PC + 4$; $PC = PC + ext\_signo(imm_{20..1} << 1)$
	jalr	jalr rd, rs1, imm	$rd = PC + 4$; $PC = rs1 + ext\_signo(imm_{11..0})$
salto cond.	beq	beq rs1, rs2, imm	*si* $(rs1 == rs2)$ $PC = PC + ext\_signo(imm_{12..1} << 1)$
	bne	bne rs1, rs2, imm	*si* $(rs1 <> rs2)$ $PC = PC + ext\_signo(imm_{12..1} << 1)$
	blt	blt rs1, rs2, imm	*si* $(rs1 < rs2)$ $PC = PC + ext\_signo(imm_{12..1} << 1)$
	bge	bge rs1, rs2, imm	*si* $(rs1 >= rs2)$ $PC = PC + ext\_signo(imm_{12..1} << 1)$
	bltu	bltu rs1, rs2, imm	*si* $(rs1 < rs2)$ $PC = PC + ext\_signo(imm_{12..1} << 1)$ (*)
	bgeu	bgeu rs1, rs2, imm	*si* $(rs1 >= rs2)$ $PC = PC + ext\_signo(imm_{12..1} << 1)$ (*)
comp.	slt	slt rd, rs1, rs2	*si* $(rs1 < rs2)$ $rd = 1$ *caso contrario* $rd = 0$
	slti	slti rd, rs1, imm	*si* $(rs1 < ext\_signo(imm_{11..0}))$ $rd = 1$ *caso contrario* $rd = 0$
	sltu	sltu rd, rs1, rs2	*si* $(rs1 < rs2)$ $rd = 1$ *caso contrario* $rd = 0$ (*)
	sltiu	stliu rd, rs1, imm	*si* $(rs1 < ext\_signo(imm_{11..0}))$ $rd = 1$ *caso contrario* $rd = 0$ (*)

Tipo	Instrucción	Ejemplo	Descripción
carga/alm. byte	lb	lb rd, imm(rs1)	$rd = ext\_signo(Memoria[rs1 + ext\_signo(imm_{11..0})][7:0])$
	lbu	lbu rd, imm(rs1)	$rd = ext\_ceros(Mem[rs1 + ext\_signo(imm_{11..0})][7:0])$
	sb	sb rs2, imm(rs1)	$Mem[rs1 + ext\_signo(imm_{11..0})][7:0] = rs2_{7..0}$
carga/alm. media palabra	lh	lh rd, imm(rs1)	$rd = ext\_signo(Mem[rs1 + ext\_signo(imm_{11..0})][15:0])$
	lhu	lhu rd, imm(rs1)	$rd = ext\_ceros(Mem[rs1 + ext\_signo(imm_{11..0})][15:0])$
	sh	sh rs2, imm(rs1)	$Mem[rs1 + ext\_signo(imm_{11..0})][15:0] = rs2_{15..0}$
carga/alm. palabra	lw	lw rd, imm(rs1)	$rd = ext\_signo(Mem[rs1 + ext\_signo(imm_{11..0})][31:0])$
	lwu	lwu rd, imm(rs1)	$rd = ext\_ceros(Mem[rs1 + ext\_signo(imm_{11..0})][31:0])$
	sw	sw rs2. imm(rs1)	$Mem[rs1 + ext\_signo(imm_{11..0})][31:0] = rs2_{31..0}$
carga/alm. doble palabra	ld	ld rd, imm(rs1)	$rd = Mem[rs1 + ext\_signo(imm_{11..0})][63:0]$
	sd	sd rs2, imm(rs1)	$Mem[rs1 + ext\_signo(imm_{11..0})][63:0] = rs2$
carga registros	lui	lui rd, imm	$rd = imm_{19..0} << 12$
	auipc	auipc rd, imm	$rd = PC + (imm_{19..0} << 12)$
desplazamiento lógico izquierda	sll	sll rd, rs1, rs2	$rd = rs1 << rs2_{5..0}$
	slli	slli rd, rs1, imm	$rd = rs1 << imm_{5..0}$
	sllw	sllw rd, rs1, rs2	$rd_{31..0} = rs1_{31..0} << rs2_{4..0}$
	slliw	slliw rd, rs1, imm	$rd_{31..0} = rs1_{31..0} << imm_{4..0}$
desplazamiento lógico derecha	srl	srl rd, rs1, rs2	$rd = rs1 >> rs2_{5..0}$ (se insertan 0s)
	srli	srli rd, rs1, imm	$rd = rs1 >> imm_{5..0}$ (se insertan 0s)
	srlw	srlw rd, rs1, rs2	$rd_{31..0} = rs1_{31..0} >> rs2_{4..0}$ (se insertan 0s)
	srliw	srliw rd, rs1, imm	$rd_{31..0} = rs1_{31..0} >> imm_{4..0}$ (se insertan 0s)
desplazamiento aritmético derecha	sra	sra rd, rs1, rs2	$rd = rs1 >> rs2_{5..0}$ (se extiende el bit de signo)
	srai	srai rd, rs1, imm	$rd = rs1 >> imm_{5..0}$ (se extiende el bit de signo)
	sraw	sraw rd, rs1, rs2	$rd_{31..0} = rs1_{31..0} >> rs2_{4..0}$ (se extiende el bit de signo)
	sraiw	sraiw rd, rs1, imm	$rd_{31..0} = rs1_{31..0} >> imm_{4..0}$ (se extiende el bit de signo)
otras	fence	fence	sincroniza accesos a memoria de varios núcleos
	ecall	ecall	petición de servicio al entorno de ejecución
	ebreak	ebreak	devuelve el control a un entorno de depuración

Tipo	rv64f	rv64d	Ejemplo (rv64f)	Descripción
carga/alm.	flw	fld	flw fd, imm(rs1)	$fd = Mem[rs1 + ext\_signo(imm_{11..0})]$
	fsw	fsd	fsw fd, imm(rs1)	$Mem[rs1 + ext\_signo(imm_{11..0})] = fd$
copia registros	fmv.x.w	fmv.x.d	fmv.x.w rd, fs1	rv64f: $rd_{31..0} = fs1$; rv64d: $rd = fs1$
	fmv.w.x	fmv.d.x	fmv.w.x fd, rs1	rv64f: $fd = rs1_{31..0}$; rv64d: $fd = rs1$
aritméticas	fadd.s	fadd.d	fadd.s fd, fs1, fs2	$fd = fs1 + fs2$
	fsub.s	fsub.d	fsub.s fd, fs1, fs2	$fd = fs1 - fs2$
	fmul.s	fmul.d	fmul.s fd, fs1, fs2	$fd = fs1 \times fs2$
	fdiv.s	fdiv.d	fdiv.s fd, fs1, fs2	$fd = fs1 \div fs2$
	fsqrt.s	fsqrt.d	fsqrt.s fd, fs1	$fd = \sqrt{fs1}$
	fmadd.s	fmadd.d	fmadd.s fd, fs1, fs2, fs3	$fd = (fs1 \times fs2) + fs3$
	fmsub.s	fmsub.d	fmsub.s fd, fs1, fs2, fs3	$fd = (fs1 \times fs2) - fs3$
	fnmadd.s	fnmadd.d	fnmadd.s fd, fs1, fs2, fs3	$fd = -(fs1 \times fs2) - fs3$
	fnmsub.s	fnmsub.d	fnmsub.s fd, fs1, fs2, fs3	$fd = -(fs1 \times fs2) + fs3$
signo	fsgnj.s	fsgnj.d	fsnj.s fd, fs1, fs2	$fd = fs1$, $fd_{signo} = fs2_{signo}$
	fsgnjn.s	fsgnjn.d	fsnjn.s fd, fs1, fs2	$fd = fs1$, $fd_{signo} = fs2_{signo}$ *xor* 1
	fsgnjx.s	fsgnjx.d	fsgnjx.s fd, fs1, fs2	$fd = fs1$, $fd_{signo} = fs2_{signo}$ *xor* $fs1_{signo}$
comparación	feq.s	feq.d	feq.s fd, fs1, fs2	$si(fs1 == fs2)$ $fd = 1$*caso contrario* $fd = 0$
	flt.s	flt.d	flt.s fd, fs1, fs2	$si(fs1 < fs2)$ $fd = 1$*caso contrario* $fd = 0$
	fle.s	fle.d	fle.s fd, fs1, fs2	$si(fs1 <= fs2)$ $fd = 1$*caso contrario* $fd = 0$
	fmin.s	fmin.d	fmin.s fd, fs1, fs2	$fd = min(fs1, fs2)$
	fmax.s	fmax.d	fmax.s fd, fs1, fs2	$fd = max(fs1, fs2)$
conversión float a entero	fcvt.w.s	fcvt.w.d	fcvt.w.s rd, fs1	$rd_{31..0} = flotante\_a\_entero(fs1)$
	fcvt.l.s	fcvt.l.d	fcvt.l.s rd, fs1	$rd = flotante\_a\_entero(rs1)$
	fcvt.wu.s	fcvt.wu.d	fcvtwu.s rd, fs1	$rd_{31..0} = arg\_sin\_signo(flotante\_a\_entero(fs1)$
	fcvt.lu.s	fcvt.lu.d	fcvtlu.s rd, fs1	$rd = arg\_sin\_signo(flotante\_a\_entero(fs1))$
conversión entero a float	fcvt.s.w	fcvt.d.w	fcvt.s.w fd, rs1	$fd = entero\_a\_flotante(rs1_{31..0})$
	fcvt.s.wu	fcvt.d.wu	fcvt.s.wu fd, rs1	$fd = arg\_sin\_signo(entero\_a\_flotante(rs1_{31..0}))$
	fcvt.s.l	fcvt.d.l	fcvt.s.l fd, rs1	$fd = entero\_a\_flotante(rs1_{63..0})$
	fcvt.s.lu	fcvt.d.lu	fcvt.s.lu fd, rs1	$fd = arg\_sin\_signo(entero\_a\_flotante(rs1_{63..0}))$
conversión float - double		fcvt.s.d	fcvt.s.d fd, fs1	$fd = double\_a\_simple\_precision(fs1)$
		fcvt.d.s	fcvt.d.s fd, fs1	$fd = simple\_a\_doble\_precision(fs1)$
clase	fclass.s	fclass.d	fclass.s rd, fs1	$rd = clase(fs1)$: $-\infty, -0, +0, -\infty, \ldots$