	funct	rs2	rs1	funct	rd	opcode	Assembly
	31 25	24 20	19 15	14-12	11 7	6 0	
R-type	0000 000	rs2	rs1	000	rd	011 0011	add rd, rs1, rs2 # rd = rs1 + rs2
	0100 000	rs2	rs1	000	rd	011 0011	sub rd, rs1, rs2 # rd = rs1 – rs2
	0000 001	rs2	rs1	000	rd	011 0011	mul rd, rs1, rs2 # rd = LOW(rs1 * rs2)
	0000 001	rs2	rs1	001	rd	011 0011	mulh rd, rs1, rs2 # rd = HIGH(rs1 * rs2) [signed x signed]
	0000 001	rs2	rs1	010	rd	011 0011	mulhsu rd, rs1, rs2 # rd = HIGH(rs1 * rs2) [signed x unsigned]
	0000 001	rs2	rs1	011	rd	011 0011	mulhu rd, rs1, rs2 # rd = HIGH(rs1 * rs2) [unsigned x unsigned]
	0000 001	rs2	rs1	100	rd	011 0011	div rd, rs1, rs2 # rd = LOW(rs1 / rs2) [signed, round to 0]
	0000 001	rs2	rs1	101	rd	011 0011	divu rd, rs1, rs2 # rd = LOW(rs1 / rs2) [unsigned, round to 0]
	0000 001	rs2	rs1	110	rd	011 0011	rem rd, rs1, rs2 # rd = LOW(rs1 % rs2) [signed]
	0000 001	rs2	rs1	111	rd	011 0011	remu rd, rs1, rs2 # rd = LOW(rs1 % rs2) [unsigned]
	0000 000	rs2	rs1	010	rd	011 0011	slt rd, rs1, rs2 # rd = rs1 < rs2 ? 1 : 0 *[signed]*
	0000 000	rs2	rs1	011	rd	011 0011	sltu rd, rs1, rs2 # rd = rs1 < rs2 ? 1 : 0 *[unsigned]*
	0000 000	rs2	rs1	111	rd	011 0011	and rd, rs1, rs2 # rd = rs1 & rs2
	0000 000	rs2	rs1	110	rd	011 0011	or rd, rs1, rs2 # rd = rs1 | rs2
	0000 000	rs2	rs1	100	rd	011 0011	xor rd, rs1, rs2 # rd = rs1 ^ rs2
	0000 000	rs2	rs1	001	rd	011 0011	sll rd, rs1, rs2 # rd = rs1 << (rs2 & 0x1f)
	0000 000	rs2	rs1	101	rd	011 0011	srl rd, rs1, rs2 # rd = rs1 >>> (rs2 & 0x1f) *[unsigned]*
	0100 000	rs2	rs1	101	rd	011 0011	sra rd, rs1, rs2 # rd = rs1 >> (rs2 & 0x1f) *[signed]*
I-type	imm[11:0]		rs1	000	rd	110 0111	jalr rd, imm(rs1) # rd = pc + 4; pc = (rs1 + signext(imm)) & (-2)
	imm[11:0]		rs1	010	rd	000 0011	lw rd, imm(rs1) # rd = mem32[rs1 + signext(imm)]
	imm[11:0]		rs1	001	rd	000 0011	lh rd, imm(rs1) # rd = signext(mem16[rs1 + signext(imm)])
	imm[11:0]		rs1	101	rd	000 0011	lhu rd, imm(rs1) # rd = zeroext(mem16[rs1 + signext(imm)])
	imm[11:0]		rs1	000	rd	000 0011	lb rd, imm(rs1) # rd = signext(mem8[rs1 + signext(imm)])
	imm[11:0]		rs1	100	rd	000 0011	lbu rd, imm(rs1) # rd = zeroext(mem8[rs1 + signext(imm)])
	imm[11:0]		rs1	000	rd	001 0011	addi rd, rs1, imm # rd = rs1 + signext(imm)
	imm[11:0]		rs1	010	rd	001 0011	slti rd, rs1, imm # rd = rs1 < signext(imm) ? 1 : 0 *[signed]*
	imm[11:0]		rs1	011	rd	001 0011	sltiu rd, rs1, imm # rd = rs1 < signext(imm) ? 1 : 0 *[unsigned]*
	imm[11:0]		rs1	111	rd	001 0011	andi rd, rs1, imm # rd = rs1 & signext(imm)
	imm[11:0]		rs1	110	rd	001 0011	ori rd, rs1, imm # rd = rs1 | signext(imm)
	imm[11:0]		rs1	100	rd	001 0011	xori rd, rs1, imm # rd = rs1 ^ signext(imm)
	0000 000	shamt	rs1	001	rd	001 0011	slli rd, rs1, shamt # rd = rs1 << shamt
	0000 000	shamt	rs1	101	rd	001 0011	srli rd, rs1, shamt # rd = rs1 >>> shamt *[unsigned]*
	0100 000	shamt	rs1	101	rd	001 0011	srai rd, rs1, shamt # rd = rs1 >> shamt *[signed]*

	0000 0000 0000			0000 0	000	0000 0		111 0011	ecall	# system call
	0000 0000 0001			0000 0	000	0000 0		111 0011	ebreak	# debug
U-type	imm[31:12]					rd		001 0111	auipc rd, imm	# rd = pc + (imm << 12)
	imm[31:12]					rd		011 0111	lui rd, imm	# rd = imm << 12
S-type	imm[11:5]		rs2	rs1	010	imm[4:0]		010 0011	sw rs2, imm(rs1)	# mem[rs1 + signext(imm)] = rs2
	imm[11:5]		rs2	rs1	001	imm[4:0]		010 0011	sh rs2, imm(rs1)	# mem[rs1 + signext(imm)] = LSH(rs2)
	imm[11:5]		rs2	rs1	000	imm[4:0]		010 0011	sb rs2, imm(rs1)	# mem[rs1 + signext(imm)] = LSB(rs2)
B-type	imm[12]	imm[10:5]	rs2	rs1	000	imm[4:1]	imm[11]	110 0011	beq rs1, rs2, imm	# pc = (rs1 == rs2) ? pc + signext(imm)
	imm[12]	imm[10:5]	rs2	rs1	001	imm[4:1]	imm[11]	110 0011	bne rs1, rs2, imm	# pc = (rs1 != rs2) ? pc + signext(imm)
	imm[12]	imm[10:5]	rs2	rs1	100	imm[4:1]	imm[11]	110 0011	blt rs1, rs2, imm	# pc = (rs1 < rs2) ? pc + signext(imm) *[signed]*
	imm[12]	imm[10:5]	rs2	rs1	101	imm[4:1]	imm[11]	110 0011	bge rs1, rs2, imm	# pc = (rs1 >= rs2) ? pc + signext(imm) *[signed]*
	imm[12]	imm[10:5]	rs2	rs1	110	imm[4:1]	imm[11]	110 0011	bltu rs1, rs2, imm	# pc = (rs1 < rs2) ? pc + signext(imm) *[unsigned]*
	imm[12]	imm[10:5]	rs2	rs1	111	imm[4:1]	imm[11]	110 0011	bgeu rs1, rs2, imm	# pc = (rs1 >= rs2)? pc + signext(imm) *[unsigned]*
J-type	imm[20]	imm[10:1]	imm[11]	imm[19:12]		rd		110 1111	jal rd, imm	# rd = pc + 4; pc = pc + signext(imm)

Notes:

- The Program Counter (PC) contains the address of the current instruction.
- Loads and stores operate in Little Endian byte ordering.
- All immediates are sign-extended, including the logical instructions andi, ori, xori.
- Arithmetic instructions, including multiplication and division, do not detect overflows and do not generate exceptions.
- Division by zero yields quotient 0xffffffff (all 1's) both for signed and unsigned divisions. The remainder is equal to the dividend. No exception is generated.
- Overflow can occur in signed division in only one case: (-2^31) / (-1). In this case, the quotient is -2^31 and the remainder is 0.
- All branches and the jal instruction use PC relative offsets. The jal instruction has an address range of +-1MiB relative to the instruction address and branches have range of +-4KiB. If the target address is not word-aligned, an instruction-address-misaligned exception is generated on an unconditional jump or on a taken branch (but not on an untaken branch).
- The jalr instruction uses an absolute address computed from a register and an immediate offset. The rightmost bit of the target address is always set to 0.
- The auipc and lui instructions set the lower 12 bits of the destination register to zero.

Register mnemonics and calling conventions

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`	`t0`	Temporary / Alternate link register	Caller
`x6-7`	`t1-2`	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 temporaries	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