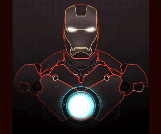
RISC-V Reference Sheet



MERL
Accelerating Engineering Innovation

General Registers

Register	ABI Name	Description	Saver
x0	zero	Constant zero	--
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	--
x4	tp	Thread pointer	--
x5	t0	Temporary	Caller
x6	t1	Temporary	Caller
x7	t2	Temporary	Caller
x8	s0/fp	Saved register / Frame pointer	Callee
x9	s1	Saved register	Callee
x10	a0	Function argument / Return value	Caller
x11	a1	Function argument / Return value	Caller
x12	a2	Function argument	Caller
x13	a3	Function argument	Caller
x14	a4	Function argument	Caller
x15	a5	Function argument	Caller
x16	a6	Function argument	Caller
x17	a7	Function argument	Caller
x18	s2	Saved register	Callee
x19	s3	Saved register	Callee
x20	s4	Saved register	Callee
x21	s5	Saved register	Callee
x22	s6	Saved register	Callee
x23	s7	Saved register	Callee
x24	s8	Saved register	Callee
x25	s9	Saved register	Callee
x26	s10	Saved register	Callee
x27	s11	Saved register	Callee
x28	t3	Temporary	Caller
x29	t4	Temporary	Caller
x30	t5	Temporary	Caller
x31	t6	Temporary	Caller

Floating Point Registers

Register	ABI Name	Description	Saver
f0	ft0	FP temporary	Caller
f1	ft1	FP temporary	Caller
f2	ft2	FP temporary	Caller
f3	ft3	FP temporary	Caller
f4	ft4	FP temporary	Caller
f5	ft5	FP temporary	Caller
f6	ft6	FP temporary	Caller
f7	ft7	FP temporary	Caller
f8	fs0	FP saved register	Callee
f9	fs1	FP saved register	Callee
f10	fa0	FP argument / Return value	Caller
f11	fa1	FP argument / Return value	Caller
f12	fa2	FP argument	Caller
f13	fa3	FP argument	Caller
f14	fa4	FP argument	Caller
f15	fa5	FP argument	Caller
f16	fa6	FP argument	
f17	fa7	FP argument	Caller
f18	fs2	FP saved register	Callee
f19	fs3	FP saved register	Callee
f20	fs4	FP saved register	Callee
f21	fs5	FP saved register	Callee
f22	fs6	FP saved register	Callee
f23	fs7	FP saved register	Callee
f24	fs8	FP saved register	Callee
f25	fs9	FP saved register	Callee
f26	fs10	FP saved register	Callee
f27	fs11	FP saved register	Callee
f28	ft8	FP temporary	Caller
f29	ft9	FP temporary	Caller
f30	ft10	FP temporary	Caller
f31	ft11	FP temporary	Caller

Assembler Directives

Directive	Arguments	Description
.align	integer	align to power of 2 (alias for .p2align)
.file	"filename"	emit filename FILE LOCAL symbol table
.globl	symbol_name	emit symbol_name to symbol table (scope GLOBAL)
.local	symbol_name	emit symbol_name to symbol table (scope LOCAL)
.common	symbol_name,size,align	emit common object to .bss section
.ident	"string"	accepted for source compatibility
.section	[{.text,.data,.rodata,.bss}]	emit section (if not present, default .text) and make current
.size	symbol, symbol	accepted for source compatibility
.text		emit .text section (if not present) and make current
.data		emit .data section (if not present) and make current
.rodata		emit .rodata section (if not present) and make current
.bss		emit .bss section (if not present) and make current
.string	"string"	emit string
.asciz	"string"	emit string (alias for .string)
.equ	name, value	constant definition
.macro	name arg1 [, argn]	begin macro definition \argname to substitute
.endm		end macro definition
.type	symbol, @function	accepted for source compatibility
.option	{arch, rvc, norvc, pic, nopic, relax, norelax, push, pop}	RISC-V options
.byte	expression [, expression]*	8-bit comma separated words
.2byte	expression [, expression]*	16-bit comma separated words
.4byte	expression [, expression]*	32-bit comma separated words
.word	expression [, expression]*	32-bit comma separated words
.8byte	expression [, expression]*	64-bit comma separated words
.dword	expression [, expression]*	64-bit comma separated words
.dtprelword	expression [, expression]*	32-bit thread local word
.dtpreldword	expression [, expression]*	64-bit thread local word
.sleb128	expression	signed little endian base 128, DWARF
.uleb128	expression	unsigned little endian base 128, DWARF
.p2align	p2,[pad_val=0],max	align to power of 2
.balign	b,[pad_val=0]	byte align
.zero	integer	zero bytes
.variant_cc	symbol_name	annotate the symbol with variant calling convention
.attribute	name, value	RISC-V object attributes

RV32I Base Integer Instruction

Inst.	Name	Type	Definition	Fields					
lui	Load Upper Immediate	U	R[rd] = {32'bimm[31], imm, 12'b0}	imm[31:12]				rd	0110111
auipc	Add Upper Immediate to PC	U	R[rd] = PC + {imm, 12'b0}	imm[31:12]				rd	0010111
jal	Jump And Link	J	R[rd] = PC + 4; PC = PC + {imm, 1'b0}	imm[20|10:1|11|19:12]				rd	1101111
jalr	Jump And Link Register	I	R[rd] = PC + 4; PC = PC + R[rs1] + imm	imm[11:0]		rs1	000	rd	1100111
beq	Branch EQual	B	if (R[rs1] == R[rs2]) PC = PC + {imm, 1'b0}	imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011
bne	Branch Not Equal	B	if (R[rs1] != R[rs2]) PC = PC + {imm, 1'b0}	imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011
blt	Branch Less Than	B	if (R[rs1] < R[rs2]) PC = PC + {imm, 1'b0}	imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011
bge	Branch Greater than or Equal	B	if (R[rs1] >= R[rs2]) PC = PC + {imm, 1'b0}	imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011
bltu	Branch Less Than Unsigned	B	if (R[rs1] < R[rs2]) PC = PC + {imm, 1'b0}	imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011
bgeu	Branch Greater than or Equal Unsigned	B	if (R[rs1] >= R[rs2]) PC = PC + {imm, 1'b0}	imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011
lb	Load Byte	I	R[rd] = {24'bM[][7], M[R[rs1]+imm][7:0]}	imm[11:0]		rs1	000	rd	0000011
lh	Load Halfword	I	R[rd] = {16'bM[][7], M[R[rs1]+imm][15:0]}	imm[11:0]		rs1	001	rd	0000011
lw	Load Word	I	R[rd] = M[R[rs1]+imm][31:0]	imm[11:0]		rs1	010	rd	0000011
lbu	Load Byte Unsigned	I	R[rd] = {24'b0, M[R[rs1]+imm][7:0]}	imm[11:0]		rs1	100	rd	0000011
lhu	Load Halfword Unsigned	I	R[rd] = {16'b0, M[R[rs1]+imm][15:0]}	imm[11:0]		rs1	101	rd	0000011
sb	Store Byte	S	M[R[rs1]+imm][7:0] = R[rs2][7:0]	imm[11:5]	rs2	rs1	000	imm[4:0]	0100011
sh	Store Halfword	S	M[R[rs1]+imm][15:0] = R[rs2][15:0]	imm[11:5]	rs2	rs1	001	imm[4:0]	0100011
sw	Store Word	S	M[R[rs1]+imm][31:0] = R[rs2][31:0]	imm[11:5]	rs2	rs1	010	imm[4:0]	0100011
addi	ADD Immediate	I	R[rd] = R[rs1] + imm	imm[11:0]		rs1	000	rd	0010011
slti	Set Less Than Immediate	I	R[rd] = (R[rs1] < imm) ? 1 : 0	imm[11:0]		rs1	010	rd	0010011
sltiu	Set Less Than Immediate Unsigned	I	R[rd] = (R[rs1] < imm) ? 1 : 0	imm[11:0]		rs1	011	rd	0010011
xori	XOR Immediate	I	R[rd] = R[rs1] ^ imm	imm[11:0]		rs1	100	rd	0010011
ori	OR Immediate	I	R[rd] = R[rs1] | imm	imm[11:0]		rs1	110	rd	0010011
andi	AND Immediate	I	R[rd] = R[rs1] & imm	imm[11:0]		rs1	111	rd	0010011
slli	Shift Left Logical Immediate	I	R[rd] = R[rs1] << imm	0000000	shamt	rs1	001	rd	0010011
srli	Shift Right Logical Immediate	I	R[rd] = R[rs1] >> imm	0000000	shamt	rs1	101	rd	0010011
srai	Shift Right Arithmetic Immediate	I	R[rd] = R[rs1] >>> imm	0100000	shamt	rs1	101	rd	0010011
add	ADD	R	R[rd] = R[rs1] + R[rs2]	0000000	rs2	rs1	000	rd	0110011
sub	SUBtract	R	R[rd] = R[rs1] - R[rs2]	0100000	rs2	rs1	000	rd	0110011
sll	Shift Left Logical	R	R[rd] = R[rs1] << R[rs2]	0000000	rs2	rs1	001	rd	0110011
slt	Set Less Than	R	R[rd] = (R[rs1] < R[rs2]) ? 1 : 0	0000000	rs2	rs1	010	rd	0110011
sltu	Set Less Than Unsigned	R	R[rd] = (R[rs1] < R[rs2]) ? 1 : 0	0000000	rs2	rs1	011	rd	0110011
xor	XOR	R	R[rd] = R[rs1] ^ R[rs2]	0000000	rs2	rs1	100	rd	0110011
srl	Shift Right Logical	R	R[rd] = R[rs1] >> R[rs2]	0000000	rs2	rs1	101	rd	0110011
sra	Shift Right Arithmetic	R	R[rd] = R[rs1] >>> R[rs2]	0100000	rs2	rs1	101	rd	0110011
or	OR	R	R[rd] = R[rs1] | R[rs2]	0000000	rs2	rs1	110	rd	0110011
and	AND	R	R[rd] = R[rs1] & R[rs2]	0000000	rs2	rs1	111	rd	0110011
fence	Memory Ordering	I	Ensure correct ordering of memory operations	fm, pred, succ		rs1	000	rd	0001111
ecall	Environment CALL	I	Transfer control to operating system	000000000000			000	00000	1110011
ebreak	Environment BREAK	I	Transfer control to debugger	000000000001			000	00000	1110011

RV64I Base Integer Instruction

Inst.	Name	Type	Definition	Fields					
lwu	Load Word Unsigned	I	R[rd] = M[R[rs1]+imm][31:0]	imm[11:0]		rs1	110	rd	0000011
ld	Load Doubleword	I	R[rd] = M[R[rs1]+imm][63:0]	imm[11:0]		rs1	011	rd	0000011
sd	Store Doubleword	S	M[R[rs1]+imm][63:0] = R[rs2][63:0]	imm[11:5]	rs2	rs1	011	imm[4:0]	0100011
slli	Shift Left Logical Immediate	I	R[rd] = R[rs1] << imm	000000	shamt	rs1	001	rd	0010011
srli	Shift Right Logical Immediate	I	R[rd] = R[rs1] >> imm	000000	shamt	rs1	101	rd	0010011
srai	Shift Right Arithmetic Immediate	I	R[rd] = R[rs1] >>> imm	010000	shamt	rs1	101	rd	0010011
addiw	ADD Immediate Word	I	R[rd] = R[rs1] + imm	imm[11:0]		rs1	000	rd	0010011
slliw	Shift Left Logical Immediate Word	I	R[rd] = R[rs1] + imm	0000000	shamt	rs1	001	rd	0010011
srli	Shift Right Logical Immediate Word	I	R[rd] = R[rs1] >> imm	0000000	shamt	rs1	101	rd	0010011
srai	Shift Right Arithmetic Immediate Word	I	R[rd] = R[rs1] >>> imm	0100000	shamt	rs1	101	rd	0010011
addw	ADD Word	R	R[rd] = R[rs1] + R[rs2]	0000000	rs2	rs1	000	rd	0111011
subw	SUBtract Word	R	R[rd] = R[rs1] - R[rs2]	0100000	rs2	rs1	000	rd	0111011
sllw	Shift Left Logical Word	R	R[rd] = R[rs1] << R[rs2]	0000000	rs2	rs1	001	rd	0111011
srlw	Shift Right Logical Word	R	R[rd] = R[rs1] >> R[rs2]	0000000	rs2	rs1	101	rd	0111011
sraw	Shift Right Arithmetic Word	R	R[rd] = R[rs1] >>> R[rs2]	0100000	rs2	rs1	101	rd	0111011

Integer Multiplication and Division Extension (M)

Inst.	Name	Type	Definition
mul	MULtiply	R	R[rd] = (R[rs1] * R[rs2])[31:0]
mulh	MULtiply High	R	R[rd] = (R[rs1] * R[rs2])[63:32]
mulhsu	MULtiply upper Half Signed / Unsigned	R	R[rd] = (R[rs1] * R[rs2])[63:32]
mulhu	MULtiply High Unsigned	R	R[rd] = (R[rs1] * R[rs2])[63:32]
div	DIVide	R	R[rd] = R[rs1] / R[rs2]
divu	DIVide Unsigned	R	R[rd] = R[rs1] / R[rs2]
rem	REMainder	R	R[rd] = R[rs1] % R[rs2]
remu	REMainder Unsigned	R	R[rd] = R[rs1] % R[rs2]

Single-Precision Floating Point Extension (F)

Inst.	Name	Type	Definition
flw	Load Word	I	F[rd] = M[R[rs1] + imm]
fsw	Store Word	S	M[R[rs1] + imm] = F[rd]
fmadd.s	Multiply-ADD	R	F[rd] = F[rs1] * F[rs2] + F[rs3]
fmsub.s	Multiply-SUBtract	R	F[rd] = F[rs1] * F[rs2] - F[rs3]
fnmsub.s	Negative Multiply-SUBtract	R	F[rd] = -(F[rs1] * F[rs2] - F[rs3])
fnmadd.s	Negative Multiply-ADD	R	F[rd] = -(F[rs1] * F[rs2] + F[rs3])
fadd.s	ADD	R	F[rd] = F[rs1] + F[rs2]
fsub.s	SUBtract	R	F[rd] = F[rs1] - F[rs2]
fmul.s	MULtiply	R	F[rd] = F[rs1] * F[rs2]
fdiv.s	DIVide	R	F[rd] = F[rs1] / F[rs2]
fsqrt.s	SQuare RooT	R	F[rd] = sqrt(F[rs1])
fsgnj.s	SiGN source	R	F[rd] = {F[rs2][31], F[rs1][30:0]}
fsgnjn.s	Negative SiGN source	R	F[rd] = {~F[rs2][31], F[rs1][30:0]}
fsgxjn.s	Xor SiGN source	R	F[rd] = {F[rs2][31] ^ F[rs1][31], F[rs1][30:0]}
fmin.s	MIN	R	F[rd] = (F[rs1] < F[rs2]) ? F[rs1] : F[rs2]
fmax.s	MAX	R	F[rd] = (F[rs1] > F[rs2]) ? F[rs1] : F[rs2]
fcvt.w.s	Convert to 32b Integer	R	R[rd] = integer(F[rs1])
fcvt.wu.s	Convert to 32b Integer Unsigned	R	R[rd] = integer(F[rs1])
fmv.x.w	Move to Integer	R	R[rd] = F[rs1]
feq.s	Compare Float EQual	R	R[rd] = (F[rs1] == F[rs2]) ? 1 : 0
flt.s	Compare Float Less Than	R	R[rd] = (F[rs1] < F[rs2]) ? 1 : 0
fle.s	Compare Float Less than or Equal	R	R[rd] = (F[rs1] <= F[rs2]) ? 1 : 0
fclass.s	Classify Type	R	R[rd] = class(F[rs1])
fcvt.s.w	Convert from 32b Integer	R	F[rd] = float(R[rs1])
fcvt.s.wu	Convert from 32b Integer Unsigned	R	F[rd] = float(R[rs1])
fmv.w.x	Move from Integer	R	F[rs1] = R[rd]

CSR Registers (Partial List)

Number	Privilege	Name	Description	Number	Privilege	Name	Description
0x001	URW	fflags	Floating-Point Accrued Exceptions	0x300	MRW	mstatus	Machine status register
0x002	URW	frm	Floating-Point Dynamic Rounding Mode	0x301	MRW	misa	ISA and extensions
0x003	URW	fcsr	Floating-Point Control and Status Register	0x302	MRW	medeleg	Machine exception delegation register
0xC00	URO	cycle	Cycle counter for RDCYCLE instruction	0x302	MRW	mideleg	Machine interrupt delegation register
0xC01	URO	time	Timer for RDTIME instruction	0x304	MRW	mie	Machine interrupt-enable register
0xC02	URO	instret	Instructions-retired counter for RDINSTRET instruction	0x305	MRW	mtvec	Machine trap-handler base address
0xC80	URO	cycleh	Upper 32 bits of cycle, RV32 only	0x306	MRW	mcounteren	Machine counter enable
0xC81	URO	timeh	Upper 32 bits of time, RV32 only	0x341	MRW	mepc	Machine exception program counter
0xC82	URO	instreth	Upper 32 bits of instret, RV32 only	0x342	MRW	mcause	Machine trap cause
0xF11	MRO	mvendorid	Vendor ID	0x343	MRW	mtval	Machine bad address or instruction
0xF12	MRO	marchid	Architecture ID	0x344	MRW	mip	Machine interrupt pending
0xF13	MRO	mimpid	Implementation ID	0xB00	MRW	mcycle	Machine cycle counter
0xF14	MRO	mhartid	Hardware thread ID	0xB02	MRW	minstret	Machine instructions-retired counter
0xF15	MRO	mconfigptr	Pointer to configuration data structure				

Control and Status Register Instruction Extension (Zicsr)

Inst.	Name	Type	Definition	Fields				
csrrw	CSR Read and Write	I	R[rd] = CSR; CSR = R[rs1]	csr	rs1	001	rd	1110011
csrrs	CSR Read and Set	I	R[rd] = CSR; CSR = CSR | R[rs1]	csr	rs1	010	rd	1110011
csrrc	CSR Read and Clear	I	R[rd] = CSR; CSR = CSR & ~R[rs1]	csr	rs1	011	rd	1110011
csrrwi	CSR Read and Write Immediate	I	R[rd] = CSR; CSR = {27'b0, imm}	csr	uimm	001	rd	1110011
csrrsi	CSR Read and Set Immediate	I	R[rd] = CSR; CSR = CSR | {27'b0, imm}	csr	uimm	010	rd	1110011
csrrci	CSR Read and Clear Immediate	I	R[rd] = CSR; CSR = CSR & ~{27'b0, imm}	csr	uimm	011	rd	1110011

Atomic Instruction Extension (A)

Inst.	Name	Type	Definition
lr.w	Load Reserved	R	R[rd] = M[R[rs1]], reservation on M[R[rs1]]
sc.w	Store Conditional	R	if reserved, M[R[rs1]] = R[rs2], R[rd] = 0; else R[rd] = 1
amoswap.w	Atomic Memory Operation SWAP	R	R[rd] = M[R[rs1]], M[R[rs1]] = R[rs2]
amoadd.w	Atomic Memory Operation ADD	R	R[rd] = M[R[rs1]], M[R[rs1]] = M[R[rs1]] + R[rs2]
amoxor.w	Atomic Memory Operation XOR	R	R[rd] = M[R[rs1]], M[R[rs1]] = M[R[rs1]] ^ R[rs2]
amoand.w	Atomic Memory Operation AND	R	R[rd] = M[R[rs1]], M[R[rs1]] = M[R[rs1]] & R[rs2]
amoxor.w	Atomic Memory Operation OR	R	R[rd] = M[R[rs1]], M[R[rs1]] = M[R[rs1]] | R[rs2]
amomin.w	Atomic Memory Operation MIN	R	R[rd] = M[R[rs1]], if(R[rs2] < M[R[rs1]]) M[R[rs1]] = R[rs2]
amomax.w	Atomic Memory Operation MAX	R	R[rd] = M[R[rs1]], if(R[rs2] > M[R[rs1]]) M[R[rs1]] = R[rs2]
amominu.w	Atomic Memory Operation MIN Unsigned	R	R[rd] = M[R[rs1]], if(R[rs2] < M[R[rs1]]) M[R[rs1]] = R[rs2]
amomaxu.w	Atomic Memory Operation MAX Unsigned	R	R[rd] = M[R[rs1]], if(R[rs2] > M[R[rs1]]) M[R[rs1]] = R[rs2]

Instruction-Fetch Fence Extension (Zifencei)

Inst.	Name	Type	Definition
fence.i	Fence Instruction-fetch	I	Ensure correct ordering of instruction fetch