

RV32 REFERENCE CARD

RV32IMFD Registers and Assembler Directives

Register	ABI Name	Description	Saver*
x0	zero	Zero constant	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	—
x3	gp	Global pointer	—
x4	tp	Thread pointer	Callee
x5-x7	t0-t2	Temporaries	Caller
x8	s0 / fp	Saved / frame pointer	Callee
x9	s1	Saved register	Callee
x10-x11	a0-a1	Funct. arguments/return values	Caller
x12-x17	a2-a7	Funct. arguments	Caller
x18-x27	s2-s11	Saved registers	Callee
x28-x31	t3-t6	Temporaries	Caller

* Within a given function f, all registers marked as caller are considered temporary and do not need to be saved. All registers marked as callee must be saved at function entrance, if used.

Register	ABI Name	Description	Saver*
f0-f7	ft0-ft7	FP temporaries	Caller
f8-f9	fs0-fs1	FP saved registers	Callee
f10-f11	fa0-fa1	FP Funct. arguments/return values	Caller
f12-f17	fa2-fa7	FP Funct. arguments	Caller
f18-f27	fs2-fs11	FP saved registers	Callee
f28-f31	ft8-ft11	FP temporaries	Caller

Assembler Directives:

- DECLARATION OF SEGMENTS:
.data (for RW data) and .text (for program)
- DECLARATION OF DATA (WITHIN .data SEGMENT):
.byte, .half, .word, .string – initialized list of values/characters
.zero – N bytes initialized with value 0

Instruction Formats

	31	27 26	25 24	20 19	15 14	12 11	7 6	0
R-Type	funct7			rb	ra	funct3	rd	opcode
I-Type	imm _[11:0]				ra	funct3	rd	opcode
S-Type	imm _[11:5]			rb	ra	funct3	imm _[4:0]	opcode
B-Type	imm _[12 10:5]			rb	ra	funct3	imm _[4:1 11]	opcode
U-Type	imm _[31:12]						rd	opcode
J-Type	imm _[20 10:1 11 19:12]						rd	opcode
R3-Type	funct5	fmt	rb		ra	funct3	rd	opcode
R4-Type	rc	fmt	rb		ra	funct3	rd	opcode

Note: on B-Type and J-Type instructions the encoding omits bit 0 of the immediate field.

RV32IM Instructions

	Instruction		Name	Encoding				RTL Description
				Type	Opcode	funct3	funct7	
-	nop		No Operation	<i>Pseudo-inst.:</i> addi x0,x0,0				-
Move	li	xd,imm	Load (move) Immediate	<i>Pseudo-inst.:</i> Myriad sequence				xd ← imm
	la	xd,symbol	Load Address	<i>Pseudo-inst.:</i> auipc + addi				xd ← imm (symbol)
	lui	xd,imm	Load Upper Imm	U	0110111			xd ← imm << 12
	auipc	xd,imm	Add Upper Imm to PC	U	0010111			xd ← PC + (imm << 12)
	mv	xd,rs	Move Register Value	<i>Pseudo-inst.:</i> addi xd, xs, 0				xd ← xs
Arithmetic, Logic, Shift	neg	xd,xs	Negate (2's complement)	<i>Pseudo-inst.:</i> sub xd, x0, xs				xd ← - xs
	add	xd,xa,xb	ADD	R	0110011	0x0	0x00	xd ← xa + xb
	addi	xd,xa,imm	ADD Immediate	I	0010011	0x0		xd ← xa + imm
	sub	xd,xa,xb	SUB	R	0110011	0x0	0x20	xd ← xa - xb
	not	xd,xs	NOT	<i>Pseudo-inst.:</i> xori xd, xs, -1				xd ← ~ xs
	and	xd,xa,xb	AND	R	0110011	0x7	0x00	xd ← xa & xb
	andi	xd,xa,imm	AND Immediate	I	0010011	0x7		xd ← xa & imm
	or	xd,xa,xb	OR	R	0110011	0x6	0x00	xd ← xa xb
	ori	xd,xa,imm	OR Immediate	I	0010011	0x6		xd ← xa imm
	xor	xd,xa,xb	XOR	R	0110011	0x4	0x00	xd ← xa ^ xb
	xori	xd,xa,imm	XOR Immediate	I	0010011	0x4		xd ← xa ^ imm
	sll	xd,xa,xb	Shift Left Logical	R	0110011	0x1	0x00	xd ← xa << xb _[4:0]
	slli	xd,xa,imm	Shift Left Logical Imm	I	0010011	0x1		xd ← xa << imm _[4:0]
	srl	xd,xa,xb	Shift Right Logical	R	0110011	0x5	0x00	xd ← xa >> xb _[4:0]
	srli	xd,xa,imm	Shift Right Logical Imm	I	0010011	0x5		xd ← xa >> imm _[4:0]
Multiply, Divide	sra	xd,xa,xb	Shift Right Arithmetic	R	0110011	0x5	0x20	xd ← xa >> xb _[4:0]
	srai	xd,xa,imm	Shift Right Arith Imm	I	0010011	0x5		xd ← xa >> imm _[4:0]
	mul	xd,xa,xb	Multiply	R	0110011	0x0	0x01	xd ← (xa × xb) _[31:0]
	mulh	xd,xa,xb	Multiply High (S×S)	R	0110011	0x1	0x01	xd ← (xa × xb) _[63:32]
	mulsu	xd,xa,xb	Multiply High (S×U)	R	0110011	0x2	0x01	xd ← (xa × xb) _[63:32]
	mulu	xd,xa,xb	Multiply High (U×U)	R	0110011	0x3	0x01	xd ← (xa × xb) _[63:32]
	div	xd,xa,xb	Divide	R	0110011	0x4	0x01	xd ← xa / xb
Load, Store	divu	xd,xa,xb	Divide (U)	R	0110011	0x5	0x01	xd ← xa / xb
	rem	xd,xa,xb	Remainder	R	0110011	0x6	0x01	xd ← xa % xb
	remu	xd,xa,xb	Remainder (U)	R	0110011	0x7	0x01	xd ← xa % xb
	l{b h w} xd,symbol		Load From Global Symbol	<i>Pseudo-inst.:</i> auipc + l{b h w}				xd ← M[symbol] (B/H/W)
	lb	xd,imm(xa)	Load Byte	I	0000011	0x0		xd ← M[xa+imm] (B)
	lh	xd,imm(xa)	Load Halfword	I	0000011	0x1		xd ← M[xa+imm] (H)
	lw	xd,imm(xa)	Load Word	I	0000011	0x2		xd ← M[xa+imm] (W)
Load, Store	lbu	xd,imm(xa)	Load Byte (U)	I	0000011	0x2		xd ← M[xa+imm] (BU)
	lhu	xd,imm(xa)	Load Halfword (U)	I	0000011	0x2		xd ← M[xa+imm] (HU)
	s{b h w} xs,symbol,xt		Store to Global Symbol	<i>Pseudo-inst.:</i> auipc + s{b h w}				M[symbol] ← xs (modifies xt)
	sb	xb,imm(xa)	Store Byte	S	0100011	0x0		M[xa+imm] ← xb _[7:0]
	sh	xb,imm(xa)	Store Half	S	0100011	0x1		M[xa+imm] ← xb _[15:0]
	sw	xb,imm(xa)	Store Word	S	0100011	0x2		M[xa+imm] ← xb _[31:0]

RV32IM Instructions (continued)

	Instruction	Name	Type	Opcode	funct3	funct7	RTL Description
Compare	slt <i>xd,xa,xb</i>	Set Less Than	R	0110011	0x2	0x00	$xd \leftarrow (xa < xb)?1:0$
	slti <i>xd,xa,imm</i>	Set Less Than Imm	I	0010011	0x2		$xd \leftarrow (xa < imm)?1:0$
	sltu <i>xd,xa,xb</i>	Set Less Than (U)	R	0110011	0x3	0x00	$xd \leftarrow (xa < xb)?1:0$
	sltiu <i>xd,xa,imm</i>	Set Less Than Imm (U)	I	0010011	0x3		$xd \leftarrow (xa < imm)?1:0$
	seqz <i>xd,xs</i>	Set Equal Zero	<i>Pseudo-inst.:</i> sltiu <i>xd, xs, 1</i>				$xd \leftarrow (xs == 0)?1:0$
	snez <i>xd,xs</i>	Set Not Equal Zero	<i>Pseudo-inst.:</i> sltu <i>xd, x0, xs</i>				$xd \leftarrow (xs != 0)?1:0$
	sltz <i>xd,xs</i>	Set Less Than Zero	<i>Pseudo-inst.:</i> slt <i>xd, xs, x0</i>				$xd \leftarrow (xs < 0)?1:0$
	sgtz <i>xd,xs</i>	Set Greater Than Zero	<i>Pseudo-inst.:</i> slt <i>xd, x0, xs</i>				$xd \leftarrow (xs > 0)?1:0$
Flow control (branch, jump, call, ret)	beq <i>xa,xb,imm</i>	Branch Equal	B	1100011	0x0		$\text{if}(xa==xb) \text{ PC} \leftarrow \text{PC} + \text{imm}$
	bne <i>xa,xb,imm</i>	Branch Not Equal	B	1100011	0x1		$\text{if}(xa!=xb) \text{ PC} \leftarrow \text{PC} + \text{imm}$
	bgt <i>xs, xt, offset</i>	Branch Greater Than	<i>Pseudo-inst.:</i> blt <i>xt, xs, offset</i>				$\text{if}(xs>xt) \text{ PC} \leftarrow \text{symbol}$
	bge <i>xa,xb,imm</i>	Branch Greater or Equal	B	1100011	0x5		$\text{if}(xa>=xb) \text{ PC} \leftarrow \text{PC} + \text{imm}$
	ble <i>xs, xt, offset</i>	Branch Less or Equal	<i>Pseudo-inst.:</i> bge <i>xt, xs, offset</i>				$\text{if}(xs<=xt) \text{ PC} \leftarrow \text{symbol}$
	blt <i>xa,xb,imm</i>	Branch Less Than	B	1100011	0x4		$\text{if}(xa<xb) \text{ PC} \leftarrow \text{PC} + \text{imm}$
	bgtu <i>xs, xt, offset</i>	Branch Less Than (U)	<i>Pseudo-inst.:</i> bltu <i>xt, xs, offset</i>				$\text{if}(xs>xt) \text{ PC} \leftarrow \text{symbol}$
	bgeu <i>xa,xb,imm</i>	Branch Greater or Equal (U)	B	1100011	0x7		$\text{if}(xa>=xb) \text{ PC} \leftarrow \text{PC} + \text{imm}$
	bleu <i>xs, xt, offset</i>	Branch Less or Equal (U)	<i>Pseudo-inst.:</i> bgeu <i>xt, xs, offset</i>				$\text{if}(xs<=xt) \text{ PC} \leftarrow \text{symbol}$
	bltu <i>xa,xb,imm</i>	Branch Less Than (U)	B	1100011	0x6		$\text{if}(xa<xb) \text{ PC} \leftarrow \text{PC} + \text{imm}$
	beqz <i>xs, symbol</i>	Branch Equal Zero	<i>Pseudo-inst.:</i> beq <i>xs, x0, offset</i>				$\text{if}(xs==0) \text{ PC} \leftarrow \text{symbol}$
	bnez <i>xs, symbol</i>	Branch Not Equal Zero	<i>Pseudo-inst.:</i> bne <i>xs, x0, offset</i>				$\text{if}(xs!=0) \text{ PC} \leftarrow \text{symbol}$
	blez <i>xs, symbol</i>	Branch Less or Equal Zero	<i>Pseudo-inst.:</i> bge <i>x0, xs, offset</i>				$\text{if}(xs<=0) \text{ PC} \leftarrow \text{symbol}$
	bgez <i>xs, symbol</i>	Branch Greater or Equal Zero	<i>Pseudo-inst.:</i> bge <i>xs, x0, offset</i>				$\text{if}(xs>=0) \text{ PC} \leftarrow \text{symbol}$
	bltz <i>xs, symbol</i>	Branch Less Than Zero	<i>Pseudo-inst.:</i> blt <i>xs, x0, offset</i>				$\text{if}(xs<0) \text{ PC} \leftarrow \text{symbol}$
	bgtz <i>xs, symbol</i>	Branch Greater Than Zero	<i>Pseudo-inst.:</i> blt <i>x0, xs, offset</i>				$\text{if}(xs>0) \text{ PC} \leftarrow \text{symbol}$
	j <i>symbol</i>	Jump	<i>Pseudo-inst.:</i> jal <i>x0, offset</i>				$\text{PC} \leftarrow \text{symbol}$
	jal <i>xd,imm</i>	Jump And Link	J	1101111			$xd \leftarrow \text{PC}+4; \text{ PC} \leftarrow \text{PC} + \text{imm}$
	jal <i>symbol</i>	Jump And Link To Symbol	<i>Pseudo-inst.:</i> jal <i>x1, offset</i>				$x1 \leftarrow \text{PC} + 4; \text{ PC} \leftarrow \text{symbol}$
	jr <i>xs</i>	Jump Register	<i>Pseudo-inst.:</i> jalr <i>x0, xs, 0</i>				$\text{PC} \leftarrow \text{xs}$
	jalr <i>xs</i>	Jump And Link Register	<i>Pseudo-inst.:</i> jalr <i>x1, xs, 0</i>				$x1 \leftarrow \text{PC} + 4; \text{ PC} \leftarrow \text{xs}$
	jalr <i>xd,xa,imm</i>	Jump And Link Register	I	1100111	0x0		$xd \leftarrow \text{PC}+4; \text{ PC} \leftarrow \text{xa} + \text{imm}$
	call <i>symbol</i>	Call subroutine	<i>Pseudo-inst.:</i> auipc + jalr				$x1 \leftarrow \text{PC} + 4; \text{ PC} \leftarrow \text{symbol}$
	ret	Return from subroutine	<i>Pseudo-inst.:</i> jalr <i>x0, x1, 0</i>				$\text{PC} \leftarrow x1$
OS	ecall	Environment Call	I	1110011	imm=0x0, others=0		$\text{SEPC} \leftarrow \text{PC} + 4; \text{ PC} \leftarrow \text{STVEC}$
	ebreak	Environment Break	I	1110011	imm=0x1, others=0		$\text{SEPC} \leftarrow \text{PC} + 4; \text{ PC} \leftarrow \text{STVEC}$
	sret	Exception return	I	1110011	imm=0x102, others=0		$\text{PC} \leftarrow \text{SEPC}$

RV32FD Floating Point Instructions (not available on Ripes)

	"F" "D" Instruction	Name	Type	Opcode	funct3	funct5	RTL Description
LD, ST	f1{w d} <i>fd,symbol</i>	FP Load from Symbol	<i>Pseudo-inst.:</i> auipc + f1{w d}				$fd \leftarrow M[\text{symbol}]$
	f1{w d} <i>fd,imm(fa)</i>	FP Load	I	0000111	{010 011}		$fd \leftarrow M[\text{fa}+\text{imm}]$
	f{sw d} <i>fb,symbol,xt</i>	FP Store to Symbol	<i>Pseudo-inst.:</i> auipc + f{sw d}				$M[\text{symbol}] \leftarrow fb \text{ (modifies xt)}$
Move	f{sw d} <i>fb,imm(fa)</i>	FP Store	S	0100111	{010 011}		$M[\text{fa}+\text{imm}] \leftarrow fb$
	fmv.{s d} <i>fd,fs</i>	FP Sign Injection	<i>Pseudo-inst.:</i> fsgnj.{s d} <i>fd,fs,fs</i>				$fd = fs$
	fsgnj.{s d} <i>fd,fa,fb</i>	FP Sign Injection	R3	1010011	000	{0x10 0x11}	$fd = \text{abs}(fa) * \text{sgn}(fb)$
	fsgnjn.{s d} <i>fd,fa,fb</i>	FP Sign Neg Injection	R3	1010011	001	{0x10 0x11}	$fd = \text{abs}(fa) * -\text{sgn}(fb)$
	fsgnjx.{s d} <i>fd,fa,fb</i>	FP Sign Xor Injection	R3	1010011	010	{0x10 0x11}	$fd = fa * \text{sgn}(fb)$
	fabs.{s d} <i>fd,fs</i>	FP Absolute Value	<i>Pseudo-inst.:</i> fsgnjx.{s d} <i>fd,fs,fs</i>				$fd = fs $
	fneg.{s d} <i>fd,fs</i>	FP Negative Value	<i>Pseudo-inst.:</i> fsgnjn.{s d} <i>fd,fs,fs</i>				$fd = -fs$
Arithmetic	fmin.{s d} <i>fd,fa,fb</i>	FP Minimum	R3	1010011	000	{0x10 0x11}	$fd = \min(fa, fb)$
	fmax.{s d} <i>fd,fa,fb</i>	FP Maximum	R3	1010011	001	{0x10 0x11}	$fd = \max(fa, fb)$
	fadd.{s d} <i>fd,fa,fb</i>	FP Add	R3	1010011	rm	{0x00 0x01}	$fd = fa + fb$
	fsub.{s d} <i>fd,fa,fb</i>	FP Sub	R3	1010011	rm	{0x04 0x05}	$fd = fa - fb$
	fmul.{s d} <i>fd,fa,fb</i>	FP Mul	R3	1010011	rm	{0x08 0x09}	$fd = fa * fb$
	fdiv.{s d} <i>fd,fa,fb</i>	FP Div	R3	1010011	rm	{0x0c 0x0d}	$fd = fa / fb$
Fused	fsqrt.{s d} <i>fd,fa</i>	FP Square Root	R3	1010011	rm	{0x2c 0x2d}	$fd = \text{sqrt}(fa)$
	fmadd.{s d} <i>fd,fa,fb,fc</i>	FP Fused Mul-Add	R4	1000011	rm		$fd = fa * fb + fc$
	fmsub.{s d} <i>fd,fa,fb,fc</i>	FP Fused Mul-Sub	R4	1000111	rm		$fd = fa * fb - fc$
	fnmadd.{s d} <i>fd,fa,fb,fc</i>	FP Neg Fused Mul-Add	R4	1001111	rm		$fd = -fa * fb + fc$
	fnmsub.{s d} <i>fd,fa,fb,fc</i>	FP Neg Fused Mul-Sub	R4	1001011	rm		$fd = -fa * fb - fc$
Convert	fcvt.{s d}.w <i>fd,xa</i>	* FP Conv from Sign Int	R3	1010011	rm	{0x68 0x69}	$fd = (\text{float}) \text{ xa}$
	fcvt.{s d}.wu <i>fd,xa</i>	* FP Conv from Uns Int	R3	1010011	rm	{0x68 0x69}	$fd = (\text{float}) \text{ xa}$
	fcvt.w.{s d} <i>xd,fa</i>	* FP Convert to Int	R3	1010011	rm	{0x60 0x61}	$xd = (\text{int32_t}) \text{ fa}$
	fcvt.wu.{s d} <i>xd,fa</i>	* FP Convert to Int	R3	1010011	rm	{0x60 0x61}	$xd = (\text{uint32_t}) \text{ fa}$
	fmv.x.w <i>xd,fa</i>	Move SP FP to Int	R3	1010011	000	0x70	$xd = *((\text{int}*) \&fa)$
	fmv.w.x <i>fd,xa</i>	Move Int to SP FP	R3	1010011	000	0x78	$fd = *((\text{float}*) \&xa)$
Compare	feq.{s d} <i>xd,fa,fb</i>	FP Equal	R3	1010011	010	{0x50 0x51}	$xd = (fa == fb) ? 1 : 0$
	flt.{s d} <i>xd,fa,fb</i>	FP Less Than	R3	1010011	001	{0x50 0x51}	$xd = (fa < fb) ? 1 : 0$
	fle.{s d} <i>xd,fa,fb</i>	FP Less or Equal	R3	1010011	000	{0x50 0x51}	$xd = (fa <= fb) ? 1 : 0$
	fclass.{s d} <i>xd,fa</i>	FP Classify	R3	1010011	001	{0x70 0x71}	$xd = (fa \text{ type}):0..9$

* – To encode fcvt.{s|d}.w and fcvt.w.{s|d} set xb=0; for fcvt.{s|d}.w and fcvt.w.{s|d} set xb=1.

rm – Floating point rounding mode. Set to "000" to select round to nearest.

Observation: Designations f1{w|d} and f{sw|d} represent the corresponding instructions for SP FP load/store (f1w/fsw) and for DP FP load/store (f1d/fsd). Similarly, designations f__.{s|d} are used to represent the corresponding SP (f__.{s}) and DP (f__.{d}) instructions.

Special purpose registers and instructions (not available on Ripes)

Relevant Control and Status Registers (CSRs):

Type	Register	Full Name	CSR ID
Exception handling	sepc	Supervisor Exception PC	0x141
	scause	Supervisor Exception Cause	0x142
	stvec	Supervisor Trap Vector Base Register	0x105
	sip	Supervisor Interrupt-Pending Register	0x144
	sie	Supervisor Interrupt-Enable Register	0x104
	sstatus	Supervisor Status Register	0x100
Virtual Memory	satp	Supervisor Address Translation and Protection	0x180

	31	...	5	4	3	2	1	0	
scause	<number>								states the number of the first pending interruption
stvec	<address>								states the address of the Interrupt Service Routine
sie	INT31	...	INT5	INT4	INT3	INT2	INT1	INT0	0 - Interruptions masked; 1 - Interruptions enabled
sip	INT31	...	INT5	INT4	INT3	INT2	INT1	INT0	0 - No pending interruption; 1 - Pending interruption
sstatus	-	...	SPIE	-	-	-	GIE	-	

GIE - Global Interrupt Enable (set to zero to disable ALL interruptions/exceptions). When entering an interrupt service routine (ISR), the value of GIE is first copied to SPIE and then set to zero. When leaving the ISR, its value is restored using SPIE.

SPIE - GIE backup copy. The value of GIE is automatically copied to SPIE when entering the interrupt service routine.

	Instruction	Name	Encoding				RTL Description
			Type	Opcode	funct3	funct5	
Handling of CSR registers	csrrc <i>xd,csr,xa</i> *	CSR read and clear	I	1110011	011		$xd \leftarrow CSR[csr]; CSR[csr] \leftarrow CSR[csr] \& \sim xa$
	csrrci <i>xd,csr,imm2</i> *	CSR read and clear	I	1110011	111		$xd \leftarrow CSR[csr]; CSR[csr] \leftarrow CSR[csr] \& \sim imm2$
	csrrs <i>xd,csr,xa</i> *	CSR read and set	I	1110011	010		$xd \leftarrow CSR[csr]; CSR[csr] \leftarrow CSR[csr] xa$
	csrrsi <i>xd,csr,imm2</i> *	CSR read and set	I	1110011	110		$xd \leftarrow CSR[csr]; CSR[csr] \leftarrow CSR[csr] imm2$
	csrrw <i>xd,csr,xa</i> *	CSR read and write	I	1110011	001		$xd \leftarrow CSR[csr]; CSR[csr] \leftarrow xa$
	csrrwi <i>xd,csr,imm2</i> *	CSR read and write	I	1110011	101		$xd \leftarrow CSR[csr]; CSR[csr] \leftarrow imm2$
	csrr <i>xd,csr</i>	CSR read	<i>Pseudo-inst.: csrrs rd,csr,x0</i>				$xd \leftarrow CSR[csr]$
	csrw <i>csr,xa</i>	CSR write	<i>Pseudo-inst.: csrrw x0,csr,xa</i>				$CSR[csr] \leftarrow xa$
	csrc <i>csr,xa</i>	CSR clear bits	<i>Pseudo-inst.: csrrw x0,csr,xa</i>				$CSR[csr] \leftarrow CSR[csr] \& \sim xa$
	csrs <i>csr,xa</i>	CSR set bits	<i>Pseudo-inst.: csrrs x0,csr,xa</i>				$CSR[csr] \leftarrow CSR[csr] xa$
	csrwi <i>csr,imm2</i>	CSR write	<i>Pseudo-inst.: csrrwi x0,csr,imm2</i>				$CSR[csr] \leftarrow imm2$
	csrci <i>csr,imm2</i>	CSR clear bits	<i>Pseudo-inst.: csrrwi x0,csr,imm2</i>				$CSR[csr] \leftarrow CSR[csr] \& \sim imm2$
	csrsi <i>csr,imm2</i>	CSR set bits	<i>Pseudo-inst.: csrrsi x0,csr,imm2</i>				$CSR[csr] \leftarrow CSR[csr] imm2$

* – To encode the instructions use field *imm* to specify *csr* and *ra* to specify *imm2*.