UnB Departamento de Ciência da Computação

RISC-V

Reference Data v1.7

MNEMONIC	FMT	NAME	DESCRIPTION (in Verilog)	Note	SYNTAX	OPCODE	FUNCT3	FUNCT7	RS2	HEX
lb	I	Load Byte	R[rd]={24'bM[](7),M[R[rs1]+ imm](7:0)}	4)	lb rd,imm(rs1)	0000011	000			03/0
lh	I	Load Halfword	R[rd]={16'bM[](15),M[R[rs1]+ imm](15:0)}	4)	lh rd,imm(rs1)	0000011	001			03/1
lw	I	Load Word	R[rd]=M[R[rs1]+imm]		lw rd,imm(rs1)	0000011	010			03/2
lbu	I	Load Byte Unsigned	R[rd]={24'b0,M[R[rs1]+imm](7:0)}		lbu rd,imm(rs1)	0000011	100			03/4
lhu	I	Load Halfword Unsigned	R[rd]={16'b0,M[R[rs1]+ imm](15:0)}		lhu rd,imm(rs1)	0000011	101			03/5
addi	I	Add Immediate	R[rd]=R[rs1]+imm		addi rd,rs1,imm	0010011	000			13/0
slli	I	Shift Left Logical Imm	R[rd]=R[rs1]<<imm		slli rd,rs1,imm	0010011	001	0000000		13/1/00
slti	I	Set Less Than Immediate	R[rd]=(R[rs1]<imm)?1:0		slti rd,rs1,imm	0010011	010			13/2
sltiu	I	Set Less Than Imm Unsig	R[rd]=(R[rs1]<imm)?1:0	2)	sltiu rd,rs1,imm	0010011	011			13/3
xori	I	XOR Immediate	R[rd]=R[rs1]^imm		xori rd,rs1,imm	0010011	100			13/4
srli	I	Shift Right Logical Imm	R[rd]=R[rs1]>>imm		srli rd,rs1,imm	0010011	101	0000000		13/5/00
srai	I	Shift Right Arith Imm	R[rd]=R[rs1]>>>imm	5)	srai rd,rs1,imm	0010011	101	0100000		13/5/20
ori	I	OR Immediate	R[rd]=R[rs1]|imm		ori rd,rs1,imm	0010011	110			13/6
andi	I	AND Immediate	R[rd]=R[rs1] & imm		andi rd,rs1,imm	0010011	111			13/7
auipc	U	Add Upper Immediate to PC	R[rd]=PC+{imm,12'b0}		auipc rd,imm	0010111				17
sb	S	Store Byte	M[R[rs1]+imm](7:0)=R[rs2](7:0)		sb rs2,imm(rs1)	0100011	000			23/0
sh	S	Store Halfword	M[R[rs1]+imm](15:0)=R[rs2](15:0)		sh rs2,imm(rs1)	0100011	001			23/1
sw	S	Store Word	M[R[rs1]+imm]=R[rs2]		sw rs2,imm(rs1)	0100011	010			23/2
add	R	Add	R[rd]=R[rs1]+R[rs2]		add rd,rs1,rs2	0110011	000	0000000		33/0/00
sub	R	Subtract	R[rd]=R[rs1]-R[rs2]		sub rd,rs1,rs2	0110011	000	0100000		33/0/20
sll	R	Shift Left Logical	R[rd]=R[rs1]<<R[rs2]		sll rd,rs1,rs2	0110011	001	0000000		33/1/00
slt	R	Set Less Than	R[rd]=(R[rs1]<R[rs2])?1:0		slt rd,rs1,rs2	0110011	010	0000000		33/2/00
sltu	R	Set Less Than Unsigned	R[rd]=(R[rs1]<R[rs2])?1:0	2)	sltu rd,rs1,rs2	0110011	011	0000000		33/3/00
xor	R	XOR	R[rd]=R[rs1]^R[rs2]		xor rd,rs1,rs2	0110011	100	0000000		33/4/00
srl	R	Shift Right Logical	R[rd]=R[rs1]>>R[rs2]		srl rd,rs1,rs2	0110011	101	0000000		33/5/00
sra	R	Shift Right Arithmetic	R[rd]=R[rs1]>>>R[rs2]	5)	sra rd,rs1,rs2	0110011	101	0100000		33/5/20
or	R	OR	R[rd]=R[rs1]|R[rs2]		or rd,rs1,rs2	0110011	110	0000000		33/6/00
and	R	AND	R[rd]=R[rs1] & R[rs2]		and rd,rs1,rs2	0110011	111	0000000		33/7/00
lui	U	Load Upper Immediate	R[rd]={imm,12'b0}		lui rd,imm	0110111				37
beq	SB	Branch if Equal	if(R[rs1]=R[rs2]) PC=PC+{imm,1'b0}		beq rs1,rs2,imm	1100011	000			63/0
bne	SB	Branch if Not Equal	if(R[rs1]!=R[rs2]) PC=PC+{imm,1'b0}		bne rs1,rs2,imm	1100011	001			63/1
blt	SB	Branch if Less Than	if(R[rs1]<R[rs2]) PC=PC+{imm,1'b0}		blt rs1,rs2,imm	1100011	100			63/4
bge	SB	Branch Greater or Equal	if(R[rs1]>=R[rs2]) PC=PC+{imm,1'b0}		bge rs1,rs2,imm	1100011	101			63/5
bltu	SB	Branch Less Than Unsign	if(R[rs1]<R[rs2]) PC=PC+{imm,1'b0}	2)	bltu rs1,rs2,imm	1100011	110			63/6
bgeu	SB	Branch Great or Eq Unsign	if(R[rs1]>=R[rs2]) PC=PC+{imm,1'b0}	2)	bgeu rs1,rs2,imm	1100011	111			63/7
jalr	I	Jump & Link Register	R[rd]=PC+4; PC=(R[rs1]+imm)&(!1)	3)	jalr rd,rs1,imm	1100111	000			67/0
jal	UJ	Jump & Link	R[rd]=PC+4; PC=PC+{imm,1'b0}		jal rd,imm	1101111				6F
ecall	I	Environment CALL	Transfer control to environment system		ecall	1110011	000	0000000	00000	73/0/000
csrrw	I	CSR Read & Write	R[rd]=CSR; CSR=R[rs1]		csrrw rd,rs1	1110011	001			73/1
csrrs	I	CSR Read & Set	R[rd]=CSR; CSR=CSR|R[rs1]		csrrs rd,rs1	1110011	010			73/2
csrrc	I	CSR Read & Clear	R[rd]=CSR; CSR=CSR&!R[rs1]		csrrc rd,rs1	1110011	011			73/3
csrrwi	I	CSR Read & Write Imm	R[rd]=CSR; CSR=imm		csrrwi rd,imm	1110011	101			73/5
csrrsi	I	CSR Read & Set Imm	R[rd]=CSR; CSR=CSR|imm		csrrsi rd,imm	1110011	110			73/6
csrrci	I	CSR Read & Clear Imm	R[rd]=CSR; CSR=CSR&!imm		csrrci rd,imm	1110011	111			73/7
mul	R	Multiply	R[rd]=R[rs1]*R[rs2](31:0)		mul rd,rs1,rs2	0110011	000	0000001		33/0/01
mulh	R	Multiply upper Half	R[rd]=R[rs1]*R[rs2](63:32)		mulh rd,rs1,rs2	0110011	001	0000001		33/1/01
mulhsu	R	Mult upper Half Sign/Uns	R[rd]=R[rs1]*R[rs2](63:32)	6)	mulhsu rd,rs1,rs2	0110011	010	0000001		33/2/01
mulhu	R	Mult upper Half Unsig	R[rd]=R[rs1]*R[rs2](63:32)	2)	mulhu rd,rs1,rs2	0110011	011	0000001		33/3/01
div	R	Divide	R[rd]=(R[rs1]/R[rs2])		div rd,rs1,rs2	0110011	100	0000001		33/4/01
divu	R	Divide Unsigned	R[rd]=(R[rs1]/R[rs2])	2)	divu rd,rs1,rs2	0110011	101	0000001		33/5/01
rem	R	Remainder	R[rd]=(R[rs1]%R[rs2])		rem rd,rs1,rs2	0110011	110	0000001		33/6/01
remu	R	Remainder Unsigned	R[rd]=(R[rs1]%R[rs2])	2)	remu rd,rs1,rs2	0110011	111	0000001		33/7/01
fadd.s	R	Float Point Add	F[rd]=F[rs1]+F[rs2]		fadd.s rd,rs1,rs2	1010011	rm	0000000		53/rm/00
fclass.s	R	Classify type	R[rd]=class(F[rs1])	8)	fclass.s rd,rs1	1010011	001	1110000		53/1/E0
fcvt.s.w	R	Convert from Integer	F[rd]=float(R[rs1])		fcvt.s.w rd,rs1	1010011	rm	1101000	00000	53/rm/D00
fcvt.s.wu	R	Conv from Unsign Integer	F[rd]=float(R[rs1])	2)	fcvt.s.wu rd,rs1	1010011	rm	1101000	00001	53/rm/D01
fcvt.w.s	R	Convert to Integer	R[rd]=integer(F[rs1])		fcvt.w.s rd,rs1	1010011	rm	1100000	00000	53/rm/C00
fcvt.wu.s	R	Convert to Unsign Integer	R[rd]=integer(F[rs1])	2)	fcvt.wu.s rd,rs1	1010011	rm	1100000	00001	53/rm/C01
fdiv.s	R	Float Point Divide	F[rd]=F[rs1]/F[rs2]		fdiv.s rd,rs1,rs2	1010011	rm	0001100		53/rm/0C
feq.s	R	Compare Float Equal	R[rd]=(F[rs1]==F[rs2])?1:0		feq.s rd,rs1,rs2	1010011	010	1010000		53/2/50
fle.s	R	Comp Float Less or Equal	R[rd]=(F[rs1]<=F[rs2])?1:0		fle.s rd,rs1,rs2	1010011	000	1010000		53/0/50
flt.s	R	Compare Float Less Than	R[rd]=(F[rs1]<F[rs2])?1:0		flt.s rd,rs1,rs2	1010011	001	1010000		53/1/50
flw	I	Float Point Load	F[rd]=M[R[rs1]+imm]		flw rd,imm(rs1)	0000111	010			07/2
fmax.s	R	FP Maximum	F[rd]=(F[rs1]>F[rs2])?F[rs1]: F[rs2]		fmax.s rd,rs1,rs2	1010011	001	0010100		53/1/14
fmin.s	R	FP Minimum	F[rd]=(F[rs1]<F[rs2])?F[rs1]: F[rs2]		fmin.s rd,rs1,rs2	1010011	000	0010100		53/0/14
fmul.s	R	FP Multiply	F[rd]=F[rs1]*F[rs2]		fmul.s rd,rs1,rs2	1010011	rm	0001000		53/rm/08
fmv.s.x	R	Move from Integer	F[rd]=R[rs1]		fmv.s.x rd,rs1	1010011	000	1111000	00000	53/0/F00
fmv.x.s	R	Move to Integer	R[rd]=F[rs1]		fmv.x.s rd,rs1	1010011	000	1110000	00000	53/0/E00
fsgnj.s	R	Sign source	F[rd]={F[rs2](31),F[rs1](30:0)}		fsgnj.s rd,rs1,rs2	1010011	000	0010000		53/0/10
fsgnjn.s	R	Negative Sign source	F[rd]={!F[rs2](31),F[rs1](30:0)}		fsgnjn.s rd,rs1,rs2	1010011	001	0010000		53/1/10
fsgnjx.s	R	XOR Sign source	F[rd]={F[rs2](31)^F[rs1](31), F[rs1](30:0)}		fsgnjx.s rd,rs1,rs2	1010011	010	0010000		53/2/10
fsqrt.s	R	FP Square Root	F[rd]=sqrt(F[rs1])		fsqrt.s rd,rs1	1010011	rm	0101100	00000	53/rm/580
fsub.s	R	Float Point Subtract	F[rd]=F[rs1]-F[rs2]		fsub.s rd,rs1,rs2	1010011	rm	0000100		53/rm/04
fsw	S	Float Point Store	M[R[rs1]+imm]=F[rs2]		fsw rs2,imm(rs1)	0100111	010			27/2
uret	R	User Return	PC=CSR[UEPC] and other settings		uret	1110011	000	0010000	00000	73/0/200

	NOTES
2)	*Operation assumes unsigned integers (instead 2's complement)*
3)	*The least significant bit of the branch address in jalr is set to 0*
4)	*(signed) Load instructions extend the sign bit of data*
5)	*Replicates the sign bit to fill in the leftmost bits of the result during right shift*
6)	*Multiply with one operand signed and one unsigned*
8)	*Classify writes a 10-bit mask to show which properties are true (e.g. –inf, -0, +0, +inf, denorm…)*
	The immediate field is sign-extended in RISC-V

CORE INSTRUCTION FORMATS

	31 25	24 20	19 15	14 12	11 7	6 0
R	funct7	rs2	rs1	funct3	Rd	opcode
I	imm[11:0]		rs1	funct3	Rd	opcode
S	imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]				Rd	opcode
UJ	imm[20|10:1|11|19:12]				Rd	opcode

PSEUDO INSTRUCTIONS

MNEMONIC	NAME	DESCRIPTION	USES
`beqz`	Branch == Zero	If(R[rs1]==0) PC=PC+{imm,1'b0}	`beq`
`bnez`	Branch != Zero	If(R[rs1]!=0) PC=PC+{imm,1'b0}	`bne`
`fabs.s`	Absolut Value	F[rd]=(F[rs1]<0)?-F[rs1]:F[rs1]	`fsgnx`
`fmv.s`	FP move	F[rd]=F[rs1]	`fsgnj`
`fneg.s`	FP negate	F[rd]=-F[rs1]	`fsgnjn`
`j`	Jump	PC={imm,1'b0}	`jal`
`jr`	Jump Register	PC=R[rs1]	`jalr`
`la`	Load Address	R[rd]=address	`auipc`
`li`	Load Immediate	R[rd]=immediate	`addi`
`mv`	Move	R[rd]=R[rs1]	`addi`
`neg`	Negate	R[rd]=-R[rs1]	`sub`
`nop`	No Operation	R[zero]=R[zero]+zero	`addi`
`not`	Not	R[rd]=!R[rs1]	`xori`
`ret`	Return	PC=R[ra]	`jalr`
`seqz`	Set if == Zero	R[rd]=(R[rs1]==0)?1:0	`sltiu`
`snez`	Set if != Zero	R[rd]=(R[rs1]!=0)?1:0	`sltu`

Decimal Prefix				Binary Prefix	
mili(m)	10^{-3}	kilo(k)	10^{3}	kibi(ki)	2^{10}
micro(μ)	10^{-6}	Mega(M)	10^{6}	Mebi(Mi)	2^{20}
nano(n)	10^{-9}	Giga(G)	10^{9}	Gibi(Gi)	2^{30}
pico(p)	10^{-12}	Tera(T)	10^{12}	Tebi(Ti)	2^{40}
femto(f)	10^{-15}	Peta(P)	10^{15}	Pebi(Pi)	2^{50}
atto(a)	10^{-18}	Exa(E)	10^{18}	Exbi(Ei)	2^{60}
zepto(z)	10^{-21}	Zetta(Z)	10^{21}	Zebi(Zi)	2^{70}
yocto(y)	10^{-24}	Yotta(Y)	10^{24}	Yobi(Yi)	2^{80}

IEEE 754 FLOATING-POINT STANDARD

$(-1)^{S} \times (1 + \text{Fraction}) \times 2^{(\text{Expoent}-\text{Bias})}$

Half-precision Bias	= 15
Single-Precision Bias	= 127
Double-Precision Bias	= 1023
Quad-Precision Bias	= 16383

IEEE Half, Single, Double, and Quad-Precision Formats:

S	Exponent	Fraction
15	14:10	9:0

S	Exponent	Fraction
31	30:23	22:0

S	Exponent	Fraction
63	62:52	51:0

S	Exponent	Fraction
127	126:112	111:0

FCSR (Float-point Control and Status Register)

31	...	8	7	6	5	4	3	2	1	0
Reserved			Round Mode			NV	DZ	OF	UF	NX

Round Mode(rm)

000	to even
001	to zero
010	to -∞
011	to +∞
100	to max mag
111	N.A. (Rars)

Flags

NV	Invalid Operation
DZ	Divide by Zero
OF	Overflow
UF	Underflow
NX	Inexact

REGISTER NAME, USE, CALLING CONVENTION

REGISTER	NAME	USE	SAVED?
`x0`	`zero`	The constant value 0	N.A.
`x1`	`ra`	Return Address	No
`x2`	`sp`	Stack Pointer	Yes
`x3`	`gp`	Global Pointer	--
`x4`	`tp`	Thread Pointer	--
`x5-x7`	`t0-t2`	Temporaries	No
`x8`	`s0/fp`	Saved Register/Frame Pointer	Yes
`x9`	`s1`	Saved Register	Yes
`x10-x11`	`a0-a1`	Function Arguments/Return Values	No
`x12-x17`	`a2-a7`	Function Arguments	No
`x18-x27`	`s2-s11`	Saved Registers	Yes
`x28-x31`	`t3-t6`	Temporaries	No
`f0-f7`	`ft0-ft7`	FP Temporaries	No
`f8-f9`	`fs0-fs1`	FP Saved Registers	Yes
`f10-f11`	`fa0-fa1`	FP Function Arguments/Return Values	No
`f12-f17`	`fa2-fa7`	FP Function Arguments	No
`f18-f27`	`fs2-fs11`	Saved Registers	Yes
`f28-f31`	`ft8-ft11`	Temporaries	No

Service	**a7**	**Input**	**Output**
Print Integer	1	a0=integer	Print an Integer on console
Print Float	2	fa0=float	Print a Float on console
Print String	4	a0=address of the string	Print a null-terminated string
Read Integer	5		Return in a0 the integer read from console
Read Float	6		Return in fa0 the float read from console
Read String	8	a0=buffer address, a1=max num characters	Return in a0 address the string read from console
Print Char	11	a0=char (ASCII)	Print a char a0 (ASCII)
Exit	10		Return to operational system
Read Char	12		Return in a0 the ASCII code of a pressed key
Time	30		Return in {a1,a0} the system time
Sleep	32	a0=time(ms)	Sleep for a0 milliseconds
Print Int Hex	34	a0=integer	Print an integer a0 in hexadecimal
Rand	41		Return a random number in a0