Appendix B. RISC-V Instruction Set Summary

31:25		24:20	19:15	14:12	11:7	6:0	
funct7		rs2	rs1	funct3	rd	op	**R-Type**
imm$_{11:0}$			rs1	funct3	rd	op	**I-Type**
imm$_{11:5}$ / imm$_{12,10:5}$		rs2	rs1	funct3	imm$_{4:0}$ / imm$_{4:1,11}$	op	**S/B-Type**
imm$_{31:12}$ / imm$_{20,10:1,11,19:12}$					rd	op	**U/J-Type**
rs3	funct2	rs2	rs1	funct3	rd	op	**R4-Type**
31:27	26:25	24:20	19:15	14:12	11:7	6:0	

Figure B.1. RISC-V 32-bit instruction formats

• `imm`:	signed immediate
• `zimm`:	5-bit unsigned immediate in **rs2** field
• `Address`:	memory address (**rs1** + SignExt(**imm$_{11:0}$**))
• `[Address]`:	contents of memory location `Address`
• `BTA`:	branch target address (PC + SignExt(**imm$_{12:0}$**))
• `JTA`:	jump target address (PC + SignExt(**imm$_{20:0}$**))
• `label`:	text indicating instruction address
• `SignExt`:	value sign-extended to 32 bits
• `ZeroExt`:	value zero-extended to 32 bits
• `CSR`:	control & status register: number in immediate field

Table B.1. RV32I: RISC-V integer and privileged/system instructions

op	funct3	funct7	Type	Instruction	Description	Operation
0000011 (3)	000	-	I	`lb rd, imm(rs1)`	load byte	rd = SignExt([Address]$_{7:0}$)
0000011 (3)	001	-	I	`lh rd, imm(rs1)`	load half	rd = SignExt([Address]$_{15:0}$)
0000011 (3)	010	-	I	`lw rd, imm(rs1)`	load word	rd = [Address]
0000011 (3)	100	-	I	`lbu rd, imm(rs1)`	load byte unsigned	rd = ZeroExt([Address]$_{7:0}$)
0000011 (3)	101	-	I	`lhu rd, imm(rs1)`	load half unsigned	rd = ZeroExt([Address]$_{15:0}$)
0010011 (19)	000	-	I	`addi rd, rs1, imm`	add immediate	rd = rs1 + SignExt(imm)
0010011 (19)	001	0000000	R	`slli rd, rs1, zimm`	shift left logical immediate	rd = rs1 << zimm
0010011 (19)	010	-	I	`slti rd, rs1, imm`	set less than immediate	rd = (rs1 < SignExt(imm))
0010011 (19)	011	-	I	`sltiu rd, rs1, imm`	set less than immediate unsigned	rd = (rs1 < SignExt(imm))
0010011 (19)	100	-	I	`xori rd, rs1, imm`	xor immediate	rd = rs1 ^ SignExt(imm)
0010011 (19)	101	0000000	R	`srli rd, rs1, zimm`	shift right logical immediate	rd = rs1 >> zimm
0010011 (19)	101	0100000	R	`srai rd, rs1, zimm`	shift right arithmetic immediate	rd = rs1 >>> zimm
0010011 (19)	110	-	I	`ori rd, rs1, imm`	or immediate	rd = rs1 | SignExt(imm)
0010011 (19)	111	-	I	`andi rd, rs1, imm`	and immediate	rd = rs1 & SignExt(imm)
0010111 (23)	-	-	U	`auipc rd, imm`	add upper immediate to PC	rd = {imm$_{31:12}$, 12'b0} + PC
0100011 (35)	000	-	S	`sb rs2, imm(rs1)`	store byte	[Address]$_{7:0}$ = rs2$_{7:0}$
0100011 (35)	001	-	S	`sh rs2, imm(rs1)`	store half	[Address]$_{15:0}$ = rs2$_{15:0}$
0100011 (35)	010	-	S	`sw rs2, imm(rs1)`	store word	[Address] = rs2
0110011 (51)	000	0000000	R	`add rd, rs1, rs2`	add	rd = rs1 + rs2
0110011 (51)	000	0100000	R	`sub rd, rs1, rs2`	sub	rd = rs1 - rs2
0110011 (51)	001	0000000	R	`sll rd, rs1, rs2`	shift left logical	rd = rs1 << rs2$_{4:0}$
0110011 (51)	010	0000000	R	`slt rd, rs1, rs2`	set less than	rd = (rs1 < rs2)
0110011 (51)	011	0000000	R	`sltu rd, rs1, rs2`	set less than unsigned	rd = (rs1 < rs2)
0110011 (51)	100	0000000	R	`xor rd, rs1, rs2`	xor	rd = rs1 ^ rs2
0110011 (51)	101	0000000	R	`srl rd, rs1, rs2`	shift right logical	rd = rs1 >> rs2$_{4:0}$
0110011 (51)	101	0100000	R	`sra rd, rs1, rs2`	shift right arithmetic	rd = rs1 >>> rs2$_{4:0}$
0110011 (51)	110	0000000	R	`or rd, rs1, rs2`	or	rd = rs1 | rs2
0110011 (51)	111	0000000	R	`and rd, rs1, rs2`	and	rd = rs1 & rs2
0110111 (55)	-	-	U	`lui rd, imm`	load upper immediate	rd = {imm$_{31:12}$, 12'b0}
1100011 (99)	000	-	B	`beq rs1, rs2, label`	branch if =	if (rs1 == rs2) PC = BTA
1100011 (99)	001	-	B	`bne rs1, rs2, label`	branch if !=	if (rs1 != rs2) PC = BTA
1100011 (99)	100	-	B	`blt rs1, rs2, label`	branch if <	if (rs1 < rs2) PC = BTA
1100011 (99)	101	-	B	`bge rs1, rs2, label`	branch if ≥	if (rs1 ≥ rs2) PC = BTA
1100011 (99)	110	-	B	`bltu rs1, rs2, label`	branch if < unsigned	if (rs1 < rs2) PC = BTA
1100011 (99)	111	-	B	`bgeu rs1, rs2, label`	branch if ≥ unsigned	if (rs1 ≥ rs2) PC = BTA
1100111 (103)	000	-	I	`jalr rd, rs1, imm`	jump and link register	rd = PC + 4, PC = rs1 + SignExt(imm)
1101111 (111)	-	-	J	`jal rd, label`	jump and link	rd = PC + 4, PC = JTA
1110011 (115)	000	-	I	`ecall`	transfer control to OS (**imm**=0)	
1110011 (115)	000	-	I	`ebreak`	transfer control to debugger (**imm**=1)	
1110011 (115)	000	-	I	`uret`	return from user exception (**rs1**=0, **rd**=0, **imm**=2)	PC = uepc
1110011 (115)	000	-	I	`sret`	return from supervisor exception (**rs1**=0, **rd**=0, **imm**=258)	PC = sepc
1110011 (115)	000	-	I	`hret`	return from hypervisor exception (**rs1**=0, **rd**=0, **imm**=514)	PC = hepc
1110011 (115)	000	-	I	`mret`	return from machine exception (**rs1**=0, **rd**=0, **imm**=770)	PC = mepc
1110011 (115)	001	-	I	`csrrw rd, csr, rs1`	CSR read/write (**imm**=CSR)	rd = CSR, CSR = rs1
1110011 (115)	010	-	I	`csrrs rd, csr, rs1`	CSR read/set (**imm**=CSR)	rd = CSR, CSR = CSR | rs1
1110011 (115)	011	-	I	`csrrc rd, csr, rs1`	CSR read/clear (**imm**=CSR)	rd = CSR, CSR = CSR & ~rs1
1110011 (115)	101	-	I	`csrrwi rd, csr, zimm`	CSR read/write immediate (**imm**=CSR)	rd = CSR, CSR = ZeroExt(zimm)
1110011 (115)	110	-	I	`csrrsi rd, csr, zimm`	CSR read/set immediate (**imm**=CSR)	rd = CSR, CSR = CSR | ZeroExt(zimm)
1110011 (115)	111	-	I	`csrrci rd, csr, zimm`	CSR read/clear immediate (**imm**=CSR)	rd = CSR, CSR = CSR & ~ZeroExt(zimm)

Table B.2. Register Names and Numbers

Name	Register Number	Use
`zero`	`x0`	Constant value 0
`ra`	`x1`	Return address
`sp`	`x2`	Stack pointer
`gp`	`x3`	Global pointer
`tp`	`x4`	Thread pointer
`t0-2`	`x5-7`	Temporary variables
`s0/fp`	`x8`	Saved variable / Frame pointer
`s1`	`x9`	Saved variable
`a0-1`	`x10-11`	Function arguments / Return values
`a2-7`	`x12-17`	Function arguments
`s2-11`	`x18-27`	Saved variables
`t3-6`	`x28-31`	Temporary variables

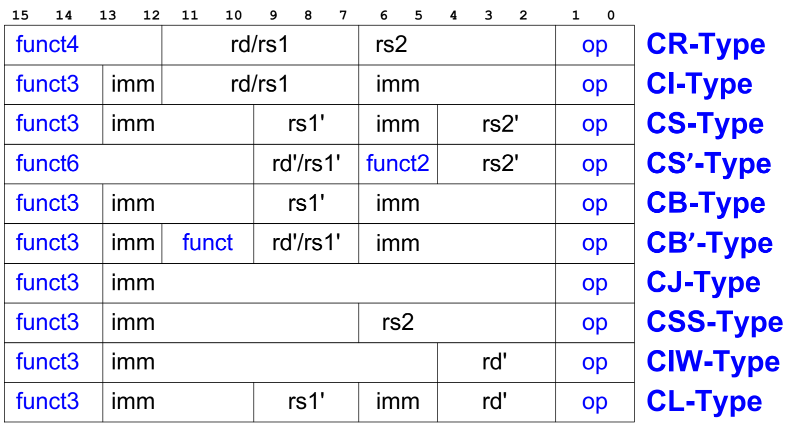
Table B.3. RVM: RISC-V multiply and divide instructions

op	funct3	funct7	Type	Instruction	Description	Operation
0110011 (51)	000	0000001	R	`mul rd, rs1, rs2`	multiply	`rd = (rs1 x rs2)`$_{31:0}$
0110011 (51)	001	0000001	R	`mulh rd, rs1, rs2`	multiply high (signed signed)	`rd = (rs1 x rs2)`$_{63:32}$
0110011 (51)	010	0000001	R	`mulhsu rd, rs1, rs2`	multiply high signed unsigned	`rd = (rs1 x rs2)`$_{63:32}$
0110011 (51)	011	0000001	R	`mulhu rd, rs1, rs2`	multiply high unsigned	`rd = (rs1 x rs2)`$_{63:32}$
0110011 (51)	100	0000001	R	`div rd, rs1, rs2`	divide (signed)	`rd = rs1 / rs2`
0110011 (51)	101	0000001	R	`divu rd, rs1, rs2`	divide unsigned	`rd = rs1 / rs2`
0110011 (51)	110	0000001	R	`rem rd, rs1, rs2`	remainder (signed)	`rd = rs1 % rs2`
0110011 (51)	111	0000001	R	`remu rd, rs1, rs2`	remainder unsigned	`rd = rs1 % rs2`

Table B.4. RVF/D: RISC-V single- and double-precision floating-point instructions

op	funct3	funct7	rs2	Type	Instruction	Description	Operation
1000011 (67)	rm	`fs3`, fmt	-	R4	`fmadd fd,fs1,fs2,fs3`	multiply-add	`fd = fs1 x fs2 + fs3`
1000111 (71)	rm	`fs3`, fmt	-	R4	`fmsub fd,fs1,fs2,fs3`	multiply-subtract	`fd = fs1 x fs2 - fs3`
1001011 (75)	rm	`fs3`, fmt	-	R4	`fnmsub fd,fs1,fs2,fs3`	negative multiply-add	`fd = -(fs1 x fs2 + fs3)`
1001111 (79)	rm	`fs3`, fmt	-	R4	`fnmadd fd,fs1,fs2,fs3`	negative multiply-subtract	`fd = -(fs1 x fs2 - fs3)`
1010011 (83)	rm	00000, fmt	-	R	`fadd fd,fs1,fs2`	Add	`fd = fs1 + fs2`
1010011 (83)	rm	00001, fmt	-	R	`fsub fd,fs1,fs2`	Subtract	`fd = fs1 - fs2`
1010011 (83)	rm	00010, fmt	-	R	`fmul fd,fs1,fs2`	multiply	`fd = fs1 x fs2`
1010011 (83)	rm	00011, fmt	-	R	`fdiv fd,fs1,fs2`	Divide	`fd = fs1 / fs2`
1010011 (83)	rm	01011, fmt	00000	R	`fsqrt fd,fs1`	square root	`fd = sqrt(fs1)`
1010011 (83)	000	00100, fmt	-	R	`fsgnj fd,fs1,fs2`	sign injection	`fd = fs1, sign = sign(fs2)`
1010011 (83)	001	00100, fmt	-	R	`fsgnjn fd,fs1,fs2`	negative sign injection	`fd = fs1, sign = -sign(fs2)`
1010011 (83)	010	00100, fmt	-	R	`fsgnjx fd,fs1,fs2`	xor sign injection	`fd = fs1, sign = sign(fs2)`⊕`sign(fs1)`
1010011 (83)	000	00101, fmt	-	R	`fmin fd,fs1,fs2`	Min	`fd = min(fs1, fs2)`
1010011 (83)	001	00101, fmt	-	R	`fmax fd,fs1,fs2`	Max	`fd = max(fs1, fs2)`
1010011 (83)	010	10100, fmt	-	R	`feq rd,fs1,fs2`	compare =	`rd = (fs1 == fs2)`
1010011 (83)	001	10100, fmt	-	R	`flt rd,fs1,fs2`	compare <	`rd = (fs1 < fs2)`
1010011 (83)	000	10100, fmt	-	R	`fle rd,fs1,fs2`	compare ≤	`rd = (fs1 ≤ fs2)`
1010011 (83)	001	11100, fmt	00000	R	`fclass rd,fs1`	Classify	`rd = classification of fs1`
RVF only							
0000111 (7)	010	-	-	I	`flw fd, imm(rs1)`	load float	`fd = [Address]`
0100111 (39)	010	-	-	S	`fsw fs2,imm(rs1)`	store float	`[Address] = fd`
1010011 (83)	rm	1100000	00000	R	`fcvt.w.s rd, fs1`	convert to integer	`rd = integer(fs1)`
1010011 (83)	rm	1100000	00001	R	`fcvt.wu.s rd, fs1`	convert to unsigned integer	`rd = unsigned(fs1)`
1010011 (83)	rm	1101000	00000	R	`fcvt.s.w fd, rs1`	convert int to float	`fd = float(rs1)`
1010011 (83)	rm	1101001	00001	R	`fcvt.s.wu fd, rs1`	convert unsigned to float	`fd = float(rs1)`
1010011 (83)	000	1110000	00000	R	`fmv.x.w rd, fs1`	move to integer register	`rd = fs1`
1010011 (83)	000	1111000	00000	R	`fmv.w.x fd, rs1`	move to f.p. register	`fd = rs1`
RVD only							
0000111 (7)	011	-	-	I	`fld fd, imm(rs1)`	load double	`fd = [Address]`
0100111 (39)	011	-	-	S	`fsd fs2,imm(rs1)`	store double	`[Address] = fd`
1010011 (83)	rm	1100001	00000	R	`fcvt.w.d rd, fs1`	convert to integer	`rd = integer(fs1)`
1010011 (83)	rm	1100001	00001	R	`fcvt.wu.d rd, fs1`	convert to unsigned integer	`rd = unsigned int(fs1)`
1010011 (83)	rm	1101001	00000	R	`fcvt.d.w fd, rs1`	convert int to double	`fd = double(rs1)`
1010011 (83)	rm	1101001	00001	R	`fcvt.d.wu fd, rs1`	convert unsigned to double	`fd = double(rs1)`
1010011 (83)	rm	0100000	00001	R	`fcvt.s.d fd, fs1`	convert double to float	`fd = integer(fs1)`
1010011 (83)	rm	0100001	00000	R	`fcvt.d.s fd, fs1`	convert float to double	`fd = unsigned int(fs1)`

`fs1, fs2, fs3, fd`: floating-point registers (encoded in fields **rs1**, **rs2**, **rs3**, **rd**); **fmt**: precision of computational instruction (S=00$_2$, D=01$_2$, Q=11$_2$);
rm: rounding mode; (0=to nearest, 1=toward zero, 2=down, 3=up, 4=to nearest (max magnitude), 7=dynamic); `sign(fs2)`: sign of `fs2`;



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
funct4				rd/rs1					rs2					op		CR-Type
funct3			imm	rd/rs1					imm					op		CI-Type
funct3			imm			rs1'			imm		rs2'			op		CS-Type
funct6						rd'/rs1'			funct2		rs2'			op		CS'-Type
funct3			imm			rs1'			imm					op		CB-Type
funct3			imm	funct		rd'/rs1'			imm					op		CB'-Type
funct3			imm											op		CJ-Type
funct3			imm						rs2					op		CSS-Type
funct3			imm								rd'			op		CIW-Type
funct3			imm			rs1'			imm		rd'			op		CL-Type

Figure B.2. RISC-V compressed (16-bit) instruction formats

Table B.5. RVC: RISC-V compressed (16-bit) instructions

op	$\text{instr}_{15:10}$	funct2	Type	RVC Instruction	32-Bit Equivalent
00 (0)	000---	--	CIW	`c.addi4spn rd',imm`	`addi rd', sp, ZeroExt(imm) x 4`
00 (0)	001---	--	CL	`c.fld fd', imm(rs1')`	`fld fd', (ZeroExt(imm)x8)(rs1')`
00 (0)	010---	--	CL	`c.lw rd', imm(rs1')`	`lw rd', (ZeroExt(imm)x4)(rs1')`
00 (0)	011---	--	CL	`c.flw fd', imm(rs1')`	`flw fd', (ZeroExt(imm)x4)(rs1')`
00 (0)	101---	--	CS	`c.fsd fs2', imm(rs1')`	`fsd fs2',(ZeroExt(imm)x8)(rs1')`
00 (0)	110---	--	CS	`c.sw rs2', imm(rs1')`	`sw rs2',(ZeroExt(imm)x4)(rs1')`
00 (0)	111---	--	CS	`c.fsw fs2', imm(rs1')`	`fsw fs2',(ZeroExt(imm)x4)(rs1')`
01 (1)	000---	--	CI	`c.addi rd, imm`	`addi rd, rd, SignExt(imm)`
01 (1)	001---	--	CJ	`c.jal label`	`jal ra, label`
01 (1)	010---	--	CI	`c.li rd, imm`	`addi rd, x0, SignExt(imm)`
01 (1)	011---	--	CI	`c.lui rd, imm`	`lui rd, {14{imm`$_5$`}, imm}`
01 (1)	011---	--	CI	`c.addi16sp x0, imm`	`addi sp, sp, SignExt(imm)x16`
01 (1)	100-00	--	CB'	`c.srli rd', imm`	`srli rd', rd', imm`
01 (1)	100-01	--	CB'	`c.srai rd', imm`	`srai rd', rd', imm`
01 (1)	100-10	--	CB'	`c.andi rd', imm`	`andi rd', rd', SignExt(imm)`
01 (1)	100011	00	CS'	`c.sub rd', rs2'`	`sub rd', rd', rs2'`
01 (1)	100011	01	CS'	`c.xor rd', rs2'`	`xor rd', rd', rs2'`
01 (1)	100011	10	CS'	`c.or rd', rs2'`	`or rd', rd', rs2'`
01 (1)	100011	11	CS'	`c.and rd', rs2'`	`and rd', rd', rs2'`
01 (1)	101---	--	CJ	`c.j label`	`jal x0, label`
01 (1)	110---	--	CB	`c.beqz rs1', label`	`beq rs1',x0, label`
01 (1)	111---	--	CB	`c.bnez rs1', label`	`bne rs1',x0, label`
10 (2)	000---	--	CI	`c.slli rd, imm`	`slli rd, rd, imm`
10 (2)	001---	--	CI	`c.fldsp fd, imm`	`fld fd, (ZeroExt(imm)x8)(sp)`
10 (2)	010---	--	CI	`c.lwsp rd, imm`	`lw rd, (ZeroExt(imm)x4)(sp)`
10 (2)	011---	--	CI	`c.flwsp fd, imm`	`flw fd, (ZeroExt(imm)x4)(sp)`
10 (2)	1000--	--	CR	`c.jr rs1` (rs2=0)	`jalr x0, rs1, 0`
10 (2)	1000--	--	CR	`c.mv rd, rs2` (rs2≠0)	`add rd, x0, rs2`
10 (2)	1001--	--	CR	`c.ebreak` (rs1=0, rs2=0)	`ebreak`
10 (2)	1001--	--	CR	`c.jalr rs1` (rs1≠0, rs2=0)	`jalr ra, rs1, 0`
10 (2)	1001--	--	CR	`c.add rd, rs2`	`add rd, rd, rs2`
10 (2)	101---	--	CSS	`c.fsdsp fs2, imm`	`fsd fs2, (ZeroExt(imm)x8)(sp)`
10 (2)	110---	--	CSS	`c.swsp rs2, imm`	`sw rs2, (ZeroExt(imm)x4)(sp)`
10 (2)	111---	--	CSS	`c.fswsp fs2, imm`	`fsw fs2, (ZeroExt(imm)x4)(sp)`

- `rs1'`, `rs2'`, `rd'`: 3-bit register designator for registers 8-15; (000_2 = `x8` or `f8`, 001_2 = `x9` or `f9`, etc.)

Table B.6. RISC-V pseudoinstructions

Pseudoinstruction	RISC-V Instruction	Description
nop	addi x0, x0, 0	no operation
li rd, $\text{imm}_{11:0}$	addi rd, zero, $\text{imm}_{31:0}$	load 12-bit immediate
li rd, $\text{imm}_{31:0}$	lui rd, $\text{imm}_{31:12}{}^*$ addi rd, rd, $\text{imm}_{11:0}$	load 32-bit immediate
mv rd, rs1	addi rd, rs1, 0	register copy
not rd, rs1	xori rd, rs1, -1	one's complement: **rd** = ~**rs1**
neg rd, rs2	sub rd, x0, rs2	two's complement: **rd** = - **rs1**
seqz rd, rs1	sltiu rd, rs1, 1	set if == 0
snez rd, rs2	sltu rd, x0, rs2	set if != 0
sltz rd, rs1	slt rd, rs1, x0	set if < 0
sgtz rd, rs2	slt rd, x0, rs2	set if > 0
beqz rs1, label	beq rs1, x0, label	branch if == 0
bnez rs1, label	bne rs1, x0, label	branch if != 0
blez rs2, label	bge x0, rs2, label	branch if ≤ 0
bgez rs1, label	bge rs1, x0, label	branch if ≥ 0
bltz rs1, label	blt rs1, x0, label	branch if < 0
bgtz rs2, label	blt x0, rs2, label	branch if > 0
ble rs2, rs1, label	bge rs1, rs2, label	branch if ≤
bgt rs2, rs1, label	blt rs1, rs2, label	branch if >
bleu rs2, rs1, label	bgeu rs1, rs2, label	branch if ≤ (unsigned)
bgtu rs2, rs1, label	bltu rs1, rs2, offset	branch if > (unsigned)
j label	jal x0, label	jump: PC = label
jal label	jal ra, label	jump and link: PC = label, ra = PC+4
jr rs1	jalr x0, rs1, 0	jump register: PC = [**rs1**]
jalr rs1	jalr ra, rs1, 0	jump and link register: PC = [**rs1**], ra = PC+4
ret	jalr x0, ra, 0	return from function: PC = [ra]
call label	auipc ra, $\text{offset}_{31:12}{}^*$ jalr ra, ra, $\text{offset}_{11:0}$	call (potentially far-away) function
la rd, symbol	lui rd, $\text{symbol}_{31:12}{}^*$ addi rd, rd, $\text{symbol}_{11:0}$	load address of global variable
l{b|h|w} rd, symbol	lui rd, $\text{symbol}_{31:12}{}^*$ l{b|h|w} rd, $\text{symbol}_{11:0}$(rd)	load global variable (symbol is a 32-bit address)
s{b|h|w} rs2, symbol, rs1	lui rs1, $\text{symbol}_{31:12}{}^*$ s{b|h|w} rs2, $\text{symbol}_{11:0}$(rs1)	store global variable (symbol is a 32-bit address)
csrr rd, csr	csrrw rd, csr, x0	read CSR: **rd** = **csr**
csrw csr, rs1	csrrw x0, csr, rs1	write CSR: **csr** = **rs1**

* If **imm_{11}** is 1, the upper immediate is incremented by 1.