Core Instruction Formats

31 25	24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	**R-type**
$imm_{[11:0]}$		rs1	funct3	rd	opcode	**I-type**
$imm_{[11:5]}$	rs2	rs1	funct3	$imm_{[4:0]}$	opcode	**S-type**
$imm_{[12,10:5]}$	rs2	rs1	funct3	$imm_{[4:1,11]}$	opcode	**B-type**
$imm_{[31:12]}$				rd	opcode	**U-type**
$imm_{[20,10:1,11,19:12]}$				rd	opcode	**J-type**
7bit	5bit	5bit	3bit	5bit	7bit	

RV32I Base Integer Instructions

Instruction		Name	Type	opcode	funct3	funct7	Description (C)
add	rd, rs1, rs2	add	R	0110011	000	0000000	rd = rs1 + rs2
sub	rd, rs1, rs2	sub	R	0110011	000	0100000	rd = rs1 - rs2
xor	rd, rs1, rs2	xor	R	0110011	100	0000000	rd = rs1 ^ rs2
or	rd, rs1, rs2	or	R	0110011	110	0000000	rd = rs1 | rs2
and	rd, rs1, rs2	and	R	0110011	111	0000000	rd = rs1 & rs2
sll	rd, rs1, rs2	shift left logical	R	0110011	001	0000000	rd = rs1 << rs2
srl	rd, rs1, rs2	shift right logical	R	0110011	101	0000000	rd = rs1 >> rs2
sra	rd, rs1, rs2	shift right arith.[a]	R	0110011	101	0100000	rd = rs1 >> rs2
slt	rd, rs1, rs2	set less than	R	0110011	010	0000000	rd = (rs1 < rs2) ? 1:0
sltu	rd, rs1, rs2	set less than unsigned[b]	R	0110011	011	0000000	rd = (rs1 < rs2) ? 1:0
addi	rd, rs1, imm	add immediate[a]	I	0010011	000	–	rd = rs1 + imm
xori	rd, rs1, imm	xor immediate[a]	I	0010011	100	–	rd = rs1 ^ imm
ori	rd, rs1, imm	or immediate[a]	I	0010011	110	–	rd = rs1 | imm
andi	rd, rs1, imm	and immediate[a]	I	0010011	111	–	rd = rs1 & imm
slli	rd, rs1, imm	shift left logical imm.	I	0010011	001	0000000[c]	rd = rs1 << $imm_{[4:0]}$
srli	rd, rs1, imm	shift right logical imm.	I	0010011	101	0000000[c]	rd = rs1 >> $imm_{[4:0]}$
srai	rd, rs1, imm	shift right arith. imm.[a]	I	0010011	101	0100000[c]	rd = rs1 >> $imm_{[4:0]}$
slti	rd, rs1, imm	set less than imm.	I	0010011	010	–	rd = (rs1 < imm) ? 1:0
sltiu	rd, rs1, uimm	set less than imm. unsign.[b]	I	0010011	011	–	rd = (rs1 < uimm) ? 1:0
lb	rd, imm(rs1)	load byte[a]	I	0000011	000	–	rd = M[rs1+imm]$_{[7:0]}$
lh	rd, imm(rs1)	load half[a]	I	0000011	001	–	rd = M[rs1+imm]$_{[15:0]}$
lw	rd, imm(rs1)	load word	I	0000011	010	–	rd = M[rs1+imm]$_{[31:0]}$
lbu	rd, uimm(rs1)	load byte unsigned[b]	I	0000011	100	–	rd = M[rs1+imm]$_{[7:0]}$[b]
lhu	rd, uimm(rs1)	load half unsigned[b]	I	0000011	101	–	rd = M[rs1+imm]$_{[15:0]}$[b]
sb	rs2, imm(rs1)	store byte	S	0100011	000	–	M[rs1+imm][7:0] = rs2$_{[7:0]}$
sh	rs2, imm(rs1)	store half	S	0100011	001	–	M[rs1+imm][15:0] = rs2$_{[15:0]}$
sw	rs2, imm(rs1)	store word	S	0100011	010	–	M[rs1+imm][31:0] = rs2$_{[31:0]}$
beq	rs1, rs2, label	branch ==	B	1100011	000	–	if(rs1 == rs2) PC += imm
bne	rs1, rs2, label	branch !=	B	1100011	001	–	if(rs1 != rs2) PC += imm
blt	rs1, rs2, label	branch <	B	1100011	100	–	if(rs1 < rs2) PC += imm
bge	rs1, rs2, label	branch ≥	B	1100011	101	–	if(rs1 >= rs2) PC += imm
bltu	rs1, rs2, label	branch < unsigned[b]	B	1100011	110	–	if(rs1 < rs2) PC += imm[b]
bgeu	rs1, rs2, label	branch ≥ unsigned[b]	B	1100011	111	–	if(rs1 >= rs2) PC += imm[b]
jal	rd, label	jump and link	J	1101111	–	–	rd = PC+4; PC += imm
jalr	rd, rs1, imm	jump and link reg	I	1100111	000	–	rd = PC+4; PC = rs1 + imm
lui	rd, upimm	load Upper imm.	U	0110111	–	–	rd = imm << 12
auipc	rd, upimm	add upper imm. to PC	U	0010111	–	–	rd = PC + (imm << 12)

[a] sign extended; [b] zero extended; [c] Encoded in instr$_{[31:25]}$, the upper seven bits of the immediate field

RV32M Multiply Extension

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

Privileged Instructions

Instruction	Name	Type	opcode	funct3	funct7	Description (C)
ecall	Env Call	I	1110011	000	0000000	Transfer control to OS
ebreak	Env Break	I	1110011	000	0000001	Transfer control to debugger

Registers

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

Environmental Calls

- write ID into register `a0`
- write parameter into register `a1`
- call **ecall**.

ID	Name	Register	Description
1	print_int	a1	prints integer
4	print_string	a1	prints null-terminated string with address
10	exit	-	ends the program
11	print_char	a1	prints ASCII character
17	exit2	a1	ends the program with return code

Pseudo Instructions

Pseudoinstruction	Base Instruction(s)	Meaning
la rd, symbol	**auipc** rd, symbol$_{[31:12]}$ **addi** rd, rd, symbol$_{[11:0]}$	Load address
l{b|h|w} rd, symbol	**auipc** rd, symbol$_{[31:12]}$ **l{b|h|w}** rd, symbol$_{[11:0]}$(rd)	Load global
s{b|h|w} rd, symbol, r1	**auipc** r1, symbol$_{[31:12]}$ **s{b|h|w}** rd, symbol$_{[11:0]}$(rs1)	Store global
nop	**addi** zero, zero, 0	No operation
li rd, imm	**addi** rd, zero, imm$_{11:0}$	Load 12-bit immediate
li rd, imm	**lui** rd, imm$_{31:12}$ **addi** rd, rd, imm$_{11:0}$	Load 32-bit immediate
mv rd, rs1	**addi** rd, rs1, 0	Copy register
not rd, rs1	**xori** rd, rs1, -1	One's complement
neg rd, rs1	**sub** rd, zero, rs1	Two's complement
negw rd, rs1	**subw** rd, zero, rs1	Two's complement word
seqz rd, rs1	**sltiu** rd, rs1, 1	Set if $=$ zero
snez rd, rs1	**sltu** rd, zero, rs1	Set if $\neq$ zero
sltz rd, rs1	**slt** rd, rs1, zero	Set if $<$ zero
sgtz rd, rs1	**slt** rd, zero, rs1	Set if $>$ zero
beqz rs1, label	**beq** rs1, zero, label	Branch if $=$ zero
bnez rs1, label	**bne** rs1, zero, label	Branch if $\neq$ zero
blez rs1, label	**bge** zero, rs1, label	Branch if $\leq$ zero
bgez rs1, label	**bge** rs1, zero, label	Branch if $\geq$ zero
bltz rs1, label	**blt** rs1, zero, label	Branch if $<$ zero
bgtz rs1, label	**blt** zero, rs1, label	Branch if $>$ zero
bgt rs1, rs2, label	**blt** rs2, rs1, label	Branch if $>$
ble rs1, rs2, label	**bge** rs2, rs1, label	Branch if $\leq$
bgtu rs1, rs2, label	**bltu** rs2, rs1, label	Branch if $>$, unsigned
bleu rs1, rs2, label	**bgeu** rs2, rs1, label	Branch if $\leq$, unsigned
j label	**jal** zero, label	Jump
jal label	**jal** ra, label	Jump and link
jr rs	**jalr** zero, rs1, 0	Jump register
jalr rs	**jalr** ra, rs1, 0	Jump and link register
ret	**jalr** zero, ra, 0	Return from subroutine
call label	**auipc** ra, label$_{[31:12]}$ **jalr** ra, ra, label$_{[11:0]}$	Call far-away subroutine

Floating-Point Instruction Formats

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
$imm_{[11:0]}$			rs1	funct3	rd	opcode	I-type
$imm_{[11:5]}$		rs2	rs1	funct3	$imm_{[4:0]}$	opcode	S-type
fs3	funct2	fs2	fs1	funct3	fd	opcode	R4-type*
5bit	2bit	5bit	5bit	3bit	5bit	7bit	

* floating only format

Floating-Point Registers

Name	Register	Description	Saver
ft0-7	f0-7	FP temporaries	Caller
fs0-1	f8-9	FP saved registers	Callee
fa0-1	f10-11	FP args/return values	Caller
fa2-7	f12-17	FP args	Caller
fs2-11	f18-27	FP saved registers	Callee
ft8-11	f28-31	FP temporaries	Caller

RV32F/D Floating-Point Extensions

Instruction		Name	Type	Opcode	funct3	funct7	Description (C)
fmadd.s/d	fd, fs1, fs2, fs3	Flt Fused Mul-Add	R4	1000011	rm[e]	fs3, fmt[d]	fd = fs1 * fs2 + fs3
fmsub.s/d	fd, fs1, fs2, fs3	Flt Fused Mul-Sub	R4	1000111	rm[e]	fs3, fmt[d]	fd = fs1 * fs2 - fs3
fnmadd.s/d	fd, fs1, fs2, fs3	Flt Neg Fused Mul-Add	R4	1001011	rm[e]	fs3, fmt[d]	fd = -(fs1 * fs2 + fs3)
fnmsub.s/d	fd, fs1, fs2, fs3	Flt Neg Fused Mul-Sub	R4	1001111	rm[e]	fs3, fmt[d]	fd = -(fs1 * fs2 - fs3)
fadd.s/d	fd, fs1, fs2	Flt Add	R	1010011	rm[e]	00000, fmt[d]	fd = fs1 + fs2
fsub.s/d	fd, fs1, fs2	Flt Sub	R	1010011	rm[e]	00001, fmt[d]	fd = fs1 - fs2
fmul.s/d	fd, fs1, fs2	Flt Mul	R	1010011	rm[e]	00010, fmt[d]	fd = fs1 * fs2
fdiv.s/d	fd, fs1, fs2	Flt Div	R	1010011	rm[e]	00011, fmt[d]	fd = fs1 / fs2
fsqrt.s/d	fd, fs1	Flt Square Root	R	1010011	rm[e]	01011, fmt[d]	fd = sqrt(fs1)
fsgnj.s/d	fd, fs1, fs2	Flt Sign Injection	R	1010011	000	00100, fmt[d]	fd = abs(fs1) * sign(fs2)[f]
fsgnjn.s/d	fd, fs1, fs2	Flt Sign Neg Injection	R	1010011	001	00100, fmt[d]	fd = abs(fs1) * -sign(fs2)[f]
fsgnjx.s/d	fd, fs1, fs2	Flt Sign Xor Injection	R	1010011	010	00100, fmt[d]	fd = fs1, sign(fd) = sign(fs1)^sign(fs2)[f]
fmin.s/d	fd, fs1, fs2	Flt Minimum	R	1010011	000	00101, fmt[d]	fd = min(fs1, fs2)
fmax.s/d	fd, fs1, fs2	Flt Maximum	R	1010011	001	00101, fmt[d]	fd = max(fs1, fs2)
feq.s/d	rd, fs1, fs2	Float Equality	R	1010011	010	10100, fmt[d]	rd = (fs1 == fs2) ? 1 : 0
flt.s/d	rd, fs1, fs2	Float Less Than	R	1010011	001	10100, fmt[d]	rd = (fs1 < fs2) ? 1 : 0
fle.s/d	rd, fs1, fs2	Float Less / Equal	R	1010011	000	10100, fmt[d]	rd = (fs1 <= fs2) ? 1 : 0
fclass.s/d	rd, fs1	Float Classify	R	1010011	001	11100, fmt[d]	rd = class (0..9)
		RVF only					
flw	fd, imm(rs1)	Flt Load Word	I	0000111	010	-	fd = M[rs1 + imm]
fsw	fs2, imm(rs1)	Flt Store Word	S	0100111	010	-	M[rs1 + imm] = fs2
fcvt.w.s	rd, fs1	Flt Convert to Int	R	1010011	rm[e]	1100000	rd = (int32_t) fs1
fcvt.wu.s	rd, fs1	Flt Convert to Int	R	1010011	rm[e]	1100000	rd = (uint32_t) fs1
fcvt.s.w	fd, rs1	Flt Conv from Sign Int	R	1010011	rm[e]	1101000	fd = (float) rs1
fcvt.s.wu	fd, rs1	Flt Conv from Uns Int	R	1010011	rm[e]	1101000	fd = (float) rs1
fmv.x.w	rd, fs1	Move Float to Int	R	1010011	000	1110000	rd = fs1[g]
fmv.w.x	fd, rs1	Move Int to Float	R	1010011	000	1111000	fd = rs1[g]
		RVD only					
fld	fd, imm(rs1)	Flt Load Double	I	0000111	011	-	fd = M[rs1 + imm]
fsd	fs2, imm(rs1)	Flt Store Double	S	0100111	011	-	M[rs1 + imm] = fs2
fcvt.w.d	rd, fs1	Flt Convert to Int	R	1010011	rm[e]	1100001	rd = (int32_t) fs1
fcvt.wu.d	rd, fs1	Flt Convert to Int	R	1010011	rm[e]	1100001	rd = (uint32_t) fs1
fcvt.d.w	fd, rs1	Flt Conv from Sign Int	R	1010011	rm[e]	1101001	fd = (double) rs1
fcvt.d.wu	fd, rs1	Flt Conv from Uns Int	R	1010011	rm[e]	1101001	fd = (double) rs1
fmv.s.d	fd, fs1	Move Double to Float	R	1010011	rm[e]	0100000	fd = fs1
fmv.d.s	fd, fs1	Move Float to Double	R	1010011	rm[e]	0100001	fd = fs1

fs1, fs2, fs3, fd: floating-point registers - fs1, fs2, and fd are encoded in fields rs1, rs2, and rd - only R4-type also encodes fs3;
[d] fmt: precision of computational instruction (single(.s)=00_2, double(.d)=01_2);
[e] rm: rounding mode (0=to nearest, 1=toward zero, 2=down, 3=up, 4=to nearest (max magnitude), 7=dynamic);
[f] sign(fs): the sign of fs
[g] Instructions FMV.S.X and FMV.X.S were renamed to FMV.W.X and FMV.X.W respectively to be more consistent with their semantics, which did not change. The old names will continue to be supported in the tools.