RISC-V Opcodes (RV32I) by Klaus Kohl-Schöpe (v1.0 - 08.08.2021)

M	nemonics			Opcod	e			Type	Operation	Description
		31 30 29 28 27 26 25	24 23 22 21 20			11 10 9 8 7	6 5 4 3 2 1 0		imm is signed, uimm is unsigned	check bits - sometimes *2 or *4 or *16
R-Type	<rd>,<rs1>,<rs2></rs2></rs1></rd>	funct7	rs2	rs1	funct3	rd	ор		op = xxbbb11 with bbb<>111	
ADD	<rd>,<rs1>,<rs2></rs2></rs1></rd>	0 0 0 0 0 0 0	rs2	rs1	0 0 0	rd	0 1 1 0 0 1 1	R	rd = rs1 + rs2	Add
SUB	<rd>,<rs1>,<rs2></rs2></rs1></rd>	0 1 0 0 0 0 0	rs2	rs1	0 0 0	rd	0 1 1 0 0 1 1		rd = rs1 - rs2	Substrace
SLL	<rd>,<rs1>,<rs2></rs2></rs1></rd>	0 0 0 0 0 0 0	rs2	rs1	0 0 1	rd	0 1 1 0 0 1 1	R	rd = rs1 << rs2 (0)	Shift Left Logic (0)
SLT	<rd>,<rs1>,<rs2></rs2></rs1></rd>	0 0 0 0 0 0 0	rs2	rs1	0 1 0	rd	0 1 1 0 0 1 1	R	rd = rs1 <rs2 :="" ?="" r2<="" rs1="" td=""><td>Set less than</td></rs2>	Set less than
SLTU	<rd>,<rs1>,<rs2></rs2></rs1></rd>	0 0 0 0 0 0 0	rs2	rs1	0 1 1	rd	0 1 1 0 0 1 1		rd = rs1 < rs2 ? Rs1 : r2	Set less than unsigned
XOR	<rd>,<rs1>,<rs2></rs2></rs1></rd>	0 0 0 0 0 0 0	rs2	rs1	1 0 0	rd	0 1 1 0 0 1 1		rd = rs1 ^ rs2	Xor
SRL	<rd>,<rs1>,<rs2></rs2></rs1></rd>	0 0 0 0 0 0 0	rs2	rs1	1 0 1	rd	0 1 1 0 0 1 1		rd = rs1 >> rs2 (0)	Shift Right Logic (0)
SRA	<rd>,<rs1>,<rs2></rs2></rs1></rd>	0 1 0 0 0 0 0	rs2	rs1	1 0 1	rd	0 1 1 0 0 1 1	R	rd = rs1 >> rs2 (signed)	Shift Right Arithmetic (signed)
OR	<rd>,<rs1>,<rs2></rs2></rs1></rd>	0 0 0 0 0 0 0	rs2	rs1	1 1 0	rd	0 1 1 0 0 1 1		rd = rs1 rs2	Or
AND	<rd>,<rs1>,<rs2></rs2></rs1></rd>	0 0 0 0 0 0 0	rs2	rs1	1 1 1	rd	0 1 1 0 0 1 1	R	rd = rs1 & rs2	And
I-Type	<rd>,<rs1>,imm</rs1></rd>	imm[11:0	•	rs1	funct3	rd	ор		op = xxbbb11 with bbb<>111	
LB	<rd>,imm(<rs1>)</rs1></rd>	imm[11:0	- 1	rs1	0 0 0	rd	0 0 0 0 0 1 1		rd = Mem[imm + rs1] (signed)	Load Byte
LH	<rd>,imm(<rs1>)</rs1></rd>	imm[11:0		rs1	0 0 1	rd	0 0 0 0 0 1 1		rd = Mem[imm + rs1] (signed)	Load Halfword
LW	<rd>,imm(<rs1>)</rs1></rd>	imm[11:0		rs1	0 1 0	rd	0 0 0 0 0 1 1		rd = Mem[imm + rs1]	Load Word
LBU	<rd>,imm(<rs1>)</rs1></rd>	imm[11:0	- 1	rs1	1 0 0	rd	0 0 0 0 0 1 1		rd = Mem[imm + rs1] (unsigned)	Load Byte Unsigned
LHU	<rd>,imm(<rs1>)</rs1></rd>	imm[11:0	- 1	rs1	1 0 1	rd	0 0 0 0 0 1 1	-	rd = Mem[imm + rs1] (unsigned)	Load Halfword Unsigned
ADDI	<rd>,<rs1>,imm</rs1></rd>	imm[11:0		rs1	0 0 0	rd	0 0 1 0 0 1 1		rd = rs1 + imm	Add Immediate
SLLI	<rd>,<rs1>,imm</rs1></rd>	0 0 0 0 0 0 0	imm[4:0]	rs1	0 0 1	rd	0 0 1 0 0 1 1		rd = rs1 << imm	Shift Left Logic Immediate
SLTI	<rd>,<rs1>,imm</rs1></rd>	imm[11:0		rs1	0 1 0	rd	0 0 1 0 0 1 1	- !	rd = rs1 < imm ? 1 : 0	Set Less Than Immediate
SLTIU	<rd>,<rs1>,imm</rs1></rd>	imm[11:0		rs1	0 1 1	rd	0 0 1 0 0 1 1	+	rd = rs1 < imm ? 1 : 0	Set Less Than Immediate Unsigned
XORI	<rd>,<rs1>,imm</rs1></rd>	imm[11:0		rs1	1 0 0	rd	0 0 1 0 0 1 1		rd = rs1 ^ imm	Xor Immediate
SRLI	<rd>,<rs1>,imm</rs1></rd>	0 0 0 0 0 0 0	imm[4:0]	rs1	1 0 1	rd	0 0 1 0 0 1 1		rd = rs1 >> imm	Shift Right Logic Immediate
SRAI	<rd>,<rs1>,imm</rs1></rd>	0 1 0 0 0 0 0	imm[4:0]	rs1	1 0 1	rd	0 0 1 0 0 1 1		rd = rs1 >> imm (signed)	Shift Right Arithmetic Immediate
ORI	<rd>,<rs1>,imm</rs1></rd>	imm[11:0		rs1	1 1 0	rd	0 0 1 0 0 1 1	-	rd = rs1 imm	Or Immediate
ANDI	<rd>,<rs1>,imm</rs1></rd>	imm[11:0		rs1	1 1 1	rd	0 0 1 0 0 1 1		rd = rs1 & imm	And Immediate
JALR	<rd>,<rs1>,imm</rs1></rd>	imm[11:0		rs1	0 0 0	rd	1 1 0 0 1 1 1		rd = PC+4; PC=rs1+imm&(-2)	Jump And Link Register
S-Type	<rd>,<rs1>,<rs2></rs2></rs1></rd>	imm[11:5]	rs2	rs1	func3	imm[4:0]	ор		op = xxbbb11 with bbb<>111	
SB	<rs2>,imm(<rs1>)</rs1></rs2>	imm[11:5]	rs2	rs1	0 0 0	imm[4:0]	0 1 0 0 0 1 1		Mem[rs1 + imm] = rs2	Store Byte
SH	<rs2>,imm(<rs1>)</rs1></rs2>	imm[11:5]	rs2	rs1	0 0 1	imm[4:0]	0 1 0 0 0 1 1		Mem[rs1 + imm] = rs2	Store Halfword
SW	<rs2>,imm(<rs1>)</rs1></rs2>	imm[11:5]	rs2	rs1	0 1 0	imm[4:0]	0 1 0 0 0 1 1	S	rs1 + imm	Store Word
SB-Type	<rs1>,<rs2>,imm</rs2></rs1>	imm[12,10:5]	rs2	rs1	funct3	imm[4:1,11]	ор		op = xxbbb11 with bbb<>111	1
BEQ	<rs1>,<rs2>,imm</rs2></rs1>	imm[12,10:5]	rs2	rs1	0 0 0	imm[4:1,11]	1 1 0 0 0 1 1		PC = PC+imm<<1 if rs1=rs2	Branch if Equal
BNE	<rs1>,<rs2>,imm</rs2></rs1>	imm[12,10:5]	rs2	rs1	0 0 1	imm[4:1,11]	1 1 0 0 0 1 1		PC = PC+imm<<1 if rs1<>rs2	Branch if Not Equal
BLT	<rs1>,<rs2>,imm</rs2></rs1>	imm[12,10:5]	rs2	rs1	1 0 0	imm[4:1,11]	1 1 0 0 0 1 1		PC = PC+imm<<1 if rs1 <rs2< td=""><td>Branch if Less</td></rs2<>	Branch if Less
BGE	<rs1>,<rs2>,imm</rs2></rs1>	imm[12,10:5]	rs2	rs1	1 0 1	imm[4:1,11]	1 1 0 0 0 1 1		PC = PC+imm<<1 if rs1>=rs2	Branch if Greater or Equal
BLTU	<rs1>,<rs2>,imm</rs2></rs1>	imm[12,10:5]	rs2	rs1	1 1 0	imm[4:1,11]	1 1 0 0 0 1 1		PC = PC+imm<<1 if rs1 <rs2< td=""><td>Brunch if Less Unsigned</td></rs2<>	Brunch if Less Unsigned
BGEU	<rs1>,<rs2>,imm</rs2></rs1>	imm[12,10:5]	rs2	rs1	1 1 1	imm[4:1,11]	1 1 0 0 0 1 1	SB	PC = PC+imm<<1 if rs1>=rs2	Brunch if Greater or Equal Unsigned
U-Type	<rd>,<rs1>,<rs2></rs2></rs1></rd>		imm[31:12]			rd	op	.	op = xxbbb11 with bbb<>111	The addition on lower districts DO
AUIPC LUI	<rd>,imm</rd>		imm[31:12] imm[31:12]			rd rd	0 0 1 0 1 1 1		rd = PC + imm << 12 rd = imm << 12	Load Upper Immediate to PC
	<rd>,imm</rd>	imm[201.iz		11.imm[10.121		rd				Load Upper Immediate
UJ-Type	<rd>,imm</rd>		nm[10-1]:imm[1		444040		op		op = xxbbb11 with bbb<>111	livery and link
JAL	<rd>,imm</rd>	20 10 9 8 7 6 5			-	rd	1 1 0 1 1 1 1		rd = PC+4; PC = PC + imm<<1	Jump and Link
FENCE.I		0 0	0	0	0 0 0	0	0 0 0 1 1 1 1		Sync write before instruction read	<u> </u>
FENCE			PW SI SOSRSW	0	0 0 0	0	0 0 0 1 1 1 1		Sync write before instruction read	Sync with additional parameter
ECALL		0 0 0 0 0 0 0		0	0 0 0	0	1 1 1 0 0 1 1		MEPC = PC; Call vector 8 or 11	!!! MEPC + 4 required !!!
EBREAK			0 0 0 0 1	0	0 0 0	0	1 1 1 0 0 1 1		MEPC = PC; Call vector 3	!!! MEPC + 4 required !!!
URET		0 0 0 0 0 0 0		0	0 0 0	0	1 1 1 0 0 1 1	-	UIE = UPIE; UPIE = 1	Return from Traps (U-Mode)
SRET		0 0 0 1 0 0 0		0	0 0 0	0	1 1 1 0 0 1 1	-	SIE = SPIE; SPIE = 1	Return from Traps (S-Mode)
MRET		0 0 1 1 0 0 0		0	0 0 0	0	1 1 1 0 0 1 1		MIE = MPIE; MPIE = 1	Return from Traps (M-Mode)
WFI			0 0 1 0 1	0	0 0 0	0	1 1 1 0 0 1 1		Sleep till NMI, interrupt or event	Wait for Interrupt
SFENCE.VM	A <rs1>,<rs2></rs2></rs1>	0 0 0 1 0 0 1	rs2	rs1	0 0 0	rd	1 1 1 0 0 1 1		Flush cache	used before change TLB
CSRRW	<rd>,addr,<rs1></rs1></rd>	addr[11:0		rs1	0 0 1	rd	1 1 1 0 0 1 1		rd = CSR; CSR = rs1	Atomic Read and Write CSR
CSRRWI	<rd>,addr,uimm</rd>	addr[11:0	- 1	uimm	1 0 1	rd	1 1 1 0 0 1 1		rd = CSR; CSR = imm	Atomic Read and Write CSR immedate
CSRRS	<rd>,addr,<rs1></rs1></rd>	addr[11:0	- 1	rs1	0 1 0	rd	1 1 1 0 0 1 1		rd = CSR; CSR = CSR rs1	Atomic Read and Set CSR
	<rd>,addr,uimm</rd>	addr[11:0	01	uimm	1 1 0	rd	1 1 1 0 0 1 1		rd = CSR; CSR = CSR imm	Atomic Read and Set CSR immedate
CSRRSI										
CSRRSI CSRRC CSRRCI	<rd>,addr,dimm</rd> ,addr, <rs1>,addr,uimm</rs1>	addr[11:0 addr[11:0 addr[11:0	0]	rs1 uimm	0 1 1	rd rd	1 1 1 0 0 1 1 1 1 1 0 0 1 1		rd = CSR; CSR = CSR & ~rs1 rd = CSR; CSR = CSR & ~imm	Atomic Read and Clear CSR Atomic Read and Clear CSR immedate

RISC-V Opcodes (RV32C) by Klaus Kohl-Schöpe (v1.0 - 08.08.2021)

Mnemonics			pcode	Type		Description			
		31 30 29 28 27 26 25 24 23 22 21 20 19 18 17	16 15 14 13 13	2 11 10 9 8 7	6 5 4 3 2	1 0		imm = signed, uimm = unsignd	check bits - sometimes *2 or *4 or *16
CR-Type	<rd>,<rs2></rs2></rd>		func4	rd/rs	rs2	ор	CR	op <> 11	
C.MV	<rd>,<rs2></rs2></rd>		1 0 0 0) rd<>0	rs2	1 0	CR	rd = rs2	Move (rd=0: HINT)
C.ADD	<rd>,<rs2></rs2></rd>		1 0 0 1	l rd	rs2<>0	1 0	CR	rd = rd + rs2	Add (rd=0: HINT)
C.JR	<rs></rs>		1 0 0 0	rs1<>0	0	1 0	CR	PC = rs1	Jump Reg
C.SUB	<rd'>,<rs2'></rs2'></rd'>		1 0 0 0) 1 1 rd'	0 0 rs2'	0 1	CR	rd = rd - rs2	Sub
C.JALR	<rd></rd>		1 0 0 1	rs<>0	0	1 0	CR	rs = PC+2; PC = rs	Jump and Link
C.EBREAK			1 0 0 1	0	0	1 0	CR	ebreak	Environment BREAK
CI-Type	<rd>,imm</rd>		func3 i	i rd	imm	ор	CI	op <> 11	
C.LWSP	<rd>,uimm(r2)</rd>		0 1 0 i5		i4:2 7:6	1 0	CI	rd = mem(x2 + uimm)	Read value from (SP + uimm (*4))
C.LI	<rd>,imm</rd>		0 1 0 is		i4:0	0 1	CI	rd = imm	Load immediate
C.LUI	<rd>,imm</rd>		0 1 1 i1		i16:12	0 1		rd = imm	Load upper immediate (signed bit 17:12)
C.ADDI16SP	<rd>,imm</rd>		0 1 1 is		i4 6 8:7 5	0 1		SP = SP + imm	Add imm (*16) to SP
C.NOP			0 0 0 0		0	0 1		No Operation	Pseudocode (r0 ,# 0 c.addi)
C.ADDI	<rd>,imm</rd>	***	0 0 0 is		i4:0	0 1	_	rd = rd + imm	Add immediate
C.SLLI	<rd>,uimm</rd>		0 0 0 is		i4:0	1 0		rd = rd << uimm (i<>0)	Logic shift left (i5=0 for RV32) (rd=0: HINT)
CS-Type	<rd>,imm</rd>				uimm rs2'	ор		op <> 11	
C.SW	<rs2'>,uimm(rs')</rs2'>		1 1 0	i5:3 rd'	i2,6 rs2'	0 0		mem(rd' + uimm) = rs2'	Store rs2' to (rd' + uimm (*4))
C.XOR	<rd'>,<rs2'></rs2'></rd'>		1 0 0 0		0 1 rs2'	0 1		rd' = rd' ^ rs2'	Xor
C.OR	<rd'>,<rs2'></rs2'></rd'>		1 0 0 0		1 0 rs2'	0 1			Or
C.AND	<rd'>,<rs2'></rs2'></rd'>		1 0 0 0		1 1 rs2'	0 1		rd' = rd' & rs2'	And
CSS-Type	<rd>,<rs2></rs2></rd>		func3	uimm	rs2'	ор		op <> 11	
C.SWSP	<rs2>,uimm(r2)</rs2>		1 1 0	i5:2 7:6	rs2			mem(r2 + uimm) = rs2'	Store rs2' to (SP + uimm (*4))
CIW-Type	<rs2'>,imm</rs2'>		func3	uimm	rs2´			op <> 11	
C.ADDI4SPN	l <rs2'>,uimm</rs2'>		0 0 0	i5:4 9:6 2 3	rs2'	0 0	CIW	rd' = SP + i	Add uimm (*4) to SP (uimm <> 0)
CL-Type	<rs2'>,imm</rs2'>		func3	uimm rs'	uimm rs2'	ор		op <> 11	
C.LW	<rs2'>,uimm(rs')</rs2'>		0 1 0	i5:3 rs'	i2,6 rs2'	0 0			Load word from (rs' + uimm (*4))
CB-Type	<rd>,imm</rd>			offset rd'/rs'	offset	ор		op <> 11	
C.BEQZ	<rs'>, addr</rs'>			o8 4:3 rs'	07:6 2:1 5	0 1	CB	PC = PC + offset if rs1' = 0	Branch if rs' is 0 (offset*2)
C.BNEZ	<rs>, addr</rs>		1 1 1	o8 4:3 rs'	07:6 2:1 5	0 1			Branch if rs' not 0 (offset*2)
C.ANDI	<rd'>,imm</rd'>		1 0 0 is		i4:0	0 1		rd' = rd' ^ imm	And Immediate
C.SRLI	<rd'>,uimm</rd'>		1 0 0 is		i4:0	0 1	_	rd' = rd' >> imm	Logic shift right (i5=0 for RV32)
C.SRAI	<rd'>,uimm</rd'>		1 0 0 is		i4:0	0 1		rd' = rd' >> imm (signed)	Arithmetic shift right (i5=0 for RV32)
CJ-Type	offset		func3	jump of		ор		op <> 11	
C.J	addr		1 0 1	o11 4 9:8 10		0 1		PC = PC + offset	Jump (offset*2)
C.JAL	addr		0 0 1	o11 4 9:8 10	6 7 3:1 5	0 1	CJ	x1 = PC + 2; $PC = PC + offset$	Jump and Link (offset*2)

RISC-V Opcodes (RV32I/C sorted by Opcode) by Klaus Kohl-Schöpe (v1.0 - 08.08.2021)

Mne	emonics		Opcod	е					I	Type	Operation	Description	
		31 30 29 28 27 26 25 24 23 22 21 20	19 18 17 16 15	14 13	2 11 10 9 8	7	6 5 4 3	3 2 1			imm is signed, uimm is unsigned	check bits - sometimes *2 or *4 or *16	
C.ADDI4SPN	<rs2'>,uimm</rs2'>			0 0	i5:4 9:6		rs.		0 (CIW	rd' = SP + i	Add imm (*4) to SP (imm <> 0)	
C.LW	<rs2'>,uimm(rs')</rs2'>		0	1 0	i5:3 rs	3'	i2,6 rs:	2' (0 (CL	rd' = mem(rs' + uimm)	Load word from (rs' + imm (*4))	
C.SW	<rs2'>,uimm(rd')</rs2'>	-	1	1 0	i5:3 rc	ď'	i2,6 rs:	2' (0 (CS	mem(rd' + uimm) = rs2'	Store rs2' to (rd' + uimm (*4))	
C.NOP		0 0 0 0 0 0 0				No Operation	Pseudocode (r0 ,# 0 c.addi)						
C.ADDI	<rd>,imm</rd>			0 0	5 rd<>0		i4:0	(rd = rd + imm	Add immediate	
C.JAL	addr			0 1		:8 10	6 7 3:1 5) 1		x1 = PC + 2; PC = PC + offset	Jump and Link (offset*2)	
C.LI	<rd>,imm</rd>			1 0	5 rd<>0	_	i4:0		0 1 CI		rd = imm	Load immediate	
C.LUI	<rd>,imm</rd>		0			<>2	i16:12) 1		rd = imm	Load upper immediate (signed bit 17:12)	
C.ADDI16SP	sp,imm		0		9 2	_	i4 6 8:7 5			CI	SP = SP + imm	Add imm (*16) to SP	
C.SRLI	<rd'>,uimm</rd'>			0 0	5 0 0 rc		i4:0 i4:0) 1		rd' = rd' >> imm	Logic shift right (i5=0 for RV32)	
C.SRAI	<rd'>,uimm</rd'>		1		5 0 1 rd		i4:0	0		CB	rd' = rd' >> imm (signed)	Arithmetic shift right (i5=0 for RV32)	
C.ANDI	<rd'>,imm</rd'>			0 0	5 1 0 rd) 1		rd' = rd' ^ imm rd = rd - rs2	And Immediate	
C.SUB C.XOR	<rd'>,<rs2'> <rd'>,<rs2'></rs2'></rd'></rs2'></rd'>			0 0			0 0 rs: 0 1 rs:		0 1		rd' = rd' ^ rs2'	Sub Xor	
C.OR	<rd'>,<rs2'></rs2'></rd'>			0 0		_	1 0 rs:		0 1		rd' = rd' rs2'	Or	
C.AND	<rd'>,<rs2'></rs2'></rd'>				0 1 1 rc		1 1 rs:) 1		rd' = rd' & rs2'	And	
C.I	addr			0 1			6 7 3:1 5				PC = PC + offset	Jump (offset*2)	
C.BEQZ	<rs'>, addr</rs'>		1		o8 4:3 rs		07:6 2:1				PC = PC + offset if rs1' = 0	Branch if rs' is 0 (offset*2)	
C.BNEZ	<rs'>, addr</rs'>			1 1	08 4:3 rs		07:6 2:1				PC = PC + offset if rs1' <> 0	Branch if rs' not 0 (offset*2)	
C.SLLI	<rd>,uimm</rd>			0 0	5 rd<>0		i4:0<>0		1 0		rd = rd << uimm (i<>0)	Logic shift left (i5=0 for RV32) (rd=0: HINT	
C.LWSP	<rd>,uimm(r2)</rd>			1 0			i4:2 7:6		1 0		rd = mem(x2 + uimm)	Read value from (SP + uimm (*4))	
C.JR	<rs></rs>			0 0) rs<>0		Ö		1 0		PC = rs	Jump Reg	
C.MV	<rd>,<rs2></rs2></rd>		1	0 0		1	rs2	1	1 0		rd = rs2	Move (rd=0: HINT)	
C.EBREAK				0 0			0		1 0		ebreak	Environment BREAK	
C.JALR	<rd></rd>		1	0 0	1 rs<>0		0		1 0	CR	rs = PC+2; PC = rs	Jump and Link	
C.ADD	<rd>,<rs2></rs2></rd>			0 0			rs2<>0		1 0		rd = rd + rs2	Add (rd=0: HINT)	
C.SWSP	<rs2>,uimm(r2)</rs2>			1 0	i5:2 7:6		rs2		1 0		mem(r2 + uimm) = rs2'	Store word to (SP + uimm (*4))	
LB	<rd>,imm(<rs1>)</rs1></rd>	imm[11:0]	rs1) rd	耳	0 0 0 0		1 1		rd = Mem[imm + rs1] (signed)	Load Byte	
LH	<rd>,imm(<rs1>)</rs1></rd>	imm[11:0]	rs1	0 0			0 0 0 0		1 1		rd = Mem[imm + rs1] (signed)	Load Halfword	
LW	<rd>,imm(<rs1>)</rs1></rd>	imm[11:0]	rs1	0 1		_	0 0 0 0		1 1		rd = Mem[imm + rs1]	Load Word	
LBU	<rd>,imm(<rs1>)</rs1></rd>	uimm[11:0]	rs1	1 0			0 0 0 0		1 1		rd = Mem[imm + rs1] (unsigned)	Load Byte Unsigned	
LHU	<rd>,imm(<rs1>)</rs1></rd>	uimm[11:0]	rs1	1 0			0 0 0 0		1 1		rd = Mem[imm + rs1] (unsigned)	Load Halfword Unsigned	
FENCE.I		0 0 0	0	0 0			0 0 0 1		1 1		Sync write before instruction read		
FENCE		0 PI POPRPW SI SO SRSV	0	0 0			0 0 0 1		1 1		Sync write before instruction read	Sync with additional parameter	
ADDI	<rd>,<rs1>,imm</rs1></rd>	imm[11:0]	rs1	0 0			0 0 1 0		1 1		rd = rs1 + imm	Add Immediate	
SLLI	<rd>,<rs1>,imm</rs1></rd>	0 0 0 0 0 0 0 imm[4:0]	rs1	0 0			0 0 1 0		1 1		rd = rs1 << imm	Shift Left Logic Immediate	
SLTI	<rd>,<rs1>,imm</rs1></rd>	imm[11:0]	rs1	0 1			0 0 1 0		1 1		rd = rs1 < imm ? 1 : 0	Set Less Than Immediate	
SLTIU	<rd>,<rs1>,imm</rs1></rd>	imm[11:0]	rs1	0 1			0 0 1 0		1 1		rd = rs1 < imm ? 1 : 0	Set Less Than Immediate Unsigned	
XORI	<rd>,<rs1>,imm</rs1></rd>	imm[11:0]	rs1	1 0			0 0 1 0		1 1		rd = rs1 ^ imm	Xor Immediate	
SRLI	<rd>,<rs1>,imm</rs1></rd>	0 0 0 0 0 0 0 imm[4:0]	rs1	1 0			0 0 1 0		1 1		rd = rs1 >> imm	Shift Right Logic Immediate	
SRAI	<rd>,<rs1>,imm</rs1></rd>	0 1 0 0 0 0 0 imm[4:0]	rs1	1 0			0 0 1 0				rd = rs1 >> imm (signed)	Shift Right Arithmetic Immediate	
ORI	<rd>,<rs1>,imm</rs1></rd>	imm[11:0]	rs1	1 1			0 0 1 0		1 1		rd = rs1 imm	Or Immediate	
ANDI	<rd>,<rs1>,imm</rs1></rd>	imm[11:0]	rs1	1 1			0 0 1 0		1 1		rd = rs1 & imm	And Immediate	
AUIPC	<rd>,imm</rd>	imm[31:12] imm[11:5] rs2	ro1		rd imm(4:0		0 0 1 0		1 1 1 1		rd = PC + imm << 12	Load Upper Immediate to PC	
SB SH	<rs2>,imm(<rs1>) <rs2>,imm(<rs1>)</rs1></rs2></rs1></rs2>	imm[11:5] rs2 imm[11:5] rs2	rs1 rs1	0 0	0 imm[4:0 1 imm[4:0		0 1 0 0		1 1	S	Mem[rs1 + imm] = rs2 Mem[rs1 + imm] = rs2	Store Byte Store Halfword	
SW	<rs2>,imm(<rs1>)</rs1></rs2>	imm[11:5] rs2	rs1	0 1			0 1 0 0		1 1		rs1 + imm	Store Word	
ADD		0 0 0 0 0 0 0 rs2	rs1	0 0			0 1 1 0		1 1		rd = rs1 + rs2	Add	
SUB	<rd>,<rs1>,<rs2></rs2></rs1></rd>	0 1 0 0 0 0 0 rs2	rs1		o rd		0 1 1 0		1 1		rd = rs1 - rs2	Substrace	
SLL	<rd>,<rs1>,<rs2></rs2></rs1></rd>	0 0 0 0 0 0 0 rs2	rs1		1 rd		0 1 1 0		1 1		rd = rs1 << rs2 (0)	Shift Left Logic (0)	
SLT	<rd>,<rs1>,<rs2></rs2></rs1></rd>	0 0 0 0 0 0 0 rs2	rs1	0 1			0 1 1 0		1 1		rd = rs1 <rs2 :="" ?="" r2<="" rs1="" td=""><td>Set less than</td></rs2>	Set less than	
SLTU	<rd>,<rs1>,<rs2></rs2></rs1></rd>	0 0 0 0 0 0 0 rs2	rs1	0 1			0 1 1 0		1 1		rd = rs1 < rs2 ? Rs1 : r2	Set less than unsigned	
XOR	<rd>,<rs1>,<rs2></rs2></rs1></rd>	0 0 0 0 0 0 0 rs2	rs1	1 0			0 1 1 0		1 1		rd = rs1 ^ rs2	Xor	
SRL	<rd>,<rs1>,<rs2></rs2></rs1></rd>	0 0 0 0 0 0 0 ms2	rs1	1 0			0 1 1 0		1 1	R	rd = rs1 >> rs2 (0)	Shift Right Logic (0)	
SRA	<rd>,<rs1>,<rs2></rs2></rs1></rd>	0 1 0 0 0 0 0 rs2	rs1	1 0			0 1 1 0		1 1		rd = rs1 >> rs2 (signed)	Shift Right Arithmetic (signed)	
OR	<rd>,<rs1>,<rs2></rs2></rs1></rd>	0 0 0 0 0 0 0 ms2	rs1	1 1			0 1 1 0		1 1		rd = rs1 rs2	Or	
AND	<rd>,<rs1>,<rs2></rs2></rs1></rd>	0 0 0 0 0 0 0 ms2	rs1	1 1			0 1 1 0		1 1		rd = rs1 & rs2	And	
LUI	<rd>,imm</rd>	imm[31:12]			rd		0 1 1 0		1 1		rd = imm << 12	Load Upper Immediate	
BEQ	<rs1>,<rs2>,imm</rs2></rs1>	imm[12,10:5] rs2	rs1) imm[4:1,1		1 1 0 0		1 1		PC = PC+imm<<1 if rs1=rs2	Branch if Equal	
BNE	<rs1>,<rs2>,imm</rs2></rs1>	imm[12,10:5] rs2 imm[12.10:5] rs2	rs1	0 0	1 imm[4:1,1		1 1 0 0		1 1		PC = PC+imm<<1 if rs1<>rs2	Branch if Not Equal	
BLT	<rs1>,<rs2>,imm</rs2></rs1>		rs1 rs1) imm[4:1,1		1 1 0 0		1 1		PC = PC+imm<<1 if rs1 <rs2< td=""><td>Branch if Creater or Favel</td></rs2<>	Branch if Creater or Favel	
BGE BLTII	<rs1>,<rs2>,imm</rs2></rs1>	imm[12,10:5] rs2 imm[12,10:5] rs2	rs1	1 0			1 1 0 0		1 1		PC = PC+imm<<1 if rs1>=rs2	Branch if Greater or Equal	
BLTU BGEU	<rs1>,<rs2>,imm</rs2></rs1>	imm[12,10:5] IS2 imm[12,10:5] rs2	rs1	1 1			1 1 0 0		1 1 1 1		PC = PC+imm<<1 if rs1 <rs2 PC = PC+imm<<1 if rs1>=rs2</rs2 	Brunch if Creator or Equal Unsigned	
JALR	<rs1>,<rs2>,imm <rd>,<rs1>,imm</rs1></rd></rs2></rs1>	imm[11:0]	rs1	0 0							rd = PC+1; PC=rs1+imm&(-2)	Brunch if Greater or Equal Unsigned Jump And Link Register	
JALK JAL		20 10 9 8 7 6 5 4 3 2 1 11									rd = PC+4; PC=rs1+imm&(-2) rd = PC+4; PC = PC + imm<<1	Jump and Link Register Jump and Link	
JAL ECALL	<rd>,imm</rd>	0 0 0 0 0 0 0 0 0 0 0 0 0					1 1 1 0 1					!!! MEPC + 4 required !!!	
EBREAK	1	0 0 0 0 0 0 0 0 0 0 0 0 1	0	0 0			1 1 1 0			\vdash	MEPC = PC; Call vector 8 or 11 MEPC = PC; Call vector 3	!!! MEPC + 4 required !!! !!! MEPC + 4 required !!!	
URET		0 0 0 0 0 0 0 0 0 0 1 0	0	0 0			1 1 1 0			\vdash	UIE = UPIE; UPIE = 1	Return from Traps (U-Mode)	
SRET		0 0 0 1 0 0 0 0 0 0 1 0		0 0			1 1 1 0			\vdash	SIE = SPIE; SPIE = 1	Return from Traps (S-Mode)	
WFI	1	0 0 0 1 0 0 0 0 0 1 0 1	0	0 0			1 1 1 0			\vdash	Sleep till NMI, interrupt or event	Wait for Interrupt	
SFENCE.VMA	<rs1>,<rs2></rs2></rs1>	0 0 0 1 0 0 1 rs2	rs1	0 0			1 1 1 0			\vdash	Flush cache	used before change TLB	
MRET	1017,71027	0 0 1 1 0 0 0 0 0 0 1 0		0 0			1 1 1 0		1 1	\vdash	MIE = MPIE; MPIE = 1	Return from Traps (M-Mode)	
CSRRW	<rd>,addr,<rs1></rs1></rd>	addr[11:0]	rs1	0 0		-+	1 1 1 0	0 0 1	1 1	\vdash	rd = CSR; CSR = rs1	Atomic Read and Write CSR	
CSRRWI	<rd>,addr,vimm</rd>	addr[11:0]	uimm	1 0			1 1 1 0) () 1	1 1	\vdash	rd = CSR; CSR = imm	Atomic Read and Write CSR immedate	
CSRRS	<rd>,addr,diffilition <rd>,addr,<rs1> </rs1></rd></rd>	addr[11:0]	rs1	0 1		-1	1 1 1 0	0 1	1 1	\vdash	rd = CSR; CSR = CSR rs1	Atomic Read and Write CSR Immedate Atomic Read and Set CSR	
							. : : °	1 6		-	rd = CSR; CSR = CSR imm	Atomic Read and Set CSR immedate	
	<rd>.addr.uimm</rd>	addr[11:0]	uimm	1 1 1 1 1) Id		111110						
CSRRSI CSRRC	<rd>,addr,uimm <rd>,addr,<rs1></rs1></rd></rd>	addr[11:0] addr[11:0]	rs1	1 1 0 1			1 1 1 0				rd = CSR; CSR = CSR [IIIIIII	Atomic Read and Clear CSR	

RISC-V Registers

by Klaus Kohl-Schöpe (v1.0 - 28.06.2021)

	Register								
Register	ABI	Saver	Descr.						
R0	zero		Zero						
R1	ra	Caller	Return Address						
R2	sp	Callee							
R3	gp		Global Pointer						
R4	tp		Thread Pointer						
R5	t0	Caller	Temporaries						
R6	t1	Caller	Temporaries						
R7	t2	Caller							
R8	s0/fp		Saved Register/Frame Pointer						
R9	s1		Saved Register						
R10	a0		Function Argument/Return Value						
R11	a1		Function Argument/Return Value						
R12	a2		Function Argument						
R13	a3		Function Argument						
R14	a4	Caller	Function Argument						
R15	a5	Caller	Function Argument						
R16	a6	Caller	Function Argument						
R17	а7	Caller	Function Argument						
R18	s2	Callee	Saved Registers						
R19	s3		Saved Registers						
R20	s4	Callee	Saved Registers						
R21	s5		Saved Registers						
R22	s6		Saved Registers						
R23	s7		Saved Registers						
R24	s8		Saved Registers						
R25	s9		Saved Registers						
R26	s10		Saved Registers						
R27	s11	Callee	Saved Registers						
R28	t3	Caller	Temporaries						
R29	t4	Caller							
R30	t5	Caller	Temporaries						
R31	t6	Caller	Temporaries						

ED Pogistor								
Danistan	ABI		P Register					
Register F0	ft0	Saver Caller	Descr.					
F1	ft1	Caller						
F2	ft2	Caller						
F3	ft3	Caller						
F4	ft4	Caller						
F5	ft5		Temporaries					
F6	ft6	Caller	Temporaries					
F7	ft7		Temporaries					
F8	fs0		FP Saved Registers					
F9	fs1		FP Saved Registers					
F10	fa0		FP Argument/Return Value					
F11	fa1		FP Argument/Return Value					
F12	fa2		FP Argument					
F13	fa3		FP Argument					
F14	fa4	Caller						
F15	fa5	Caller						
F16	fa6	Caller	FP Argument					
F17	fa7	Caller	FP Argument					
F18	fs2	Callee	FP Saved Registers					
F19	fs3	Callee	FP Saved Registers					
F20	fs4	Callee	FP Saved Registers					
F21	fs5		FP Saved Registers					
F22	fs6	Callee	FP Saved Registers					
F23	fs7		FP Saved Registers					
F24	fs8		FP Saved Registers					
F25	fs9	Callee	FP Saved Registers					
F26	fs10		FP Saved Registers					
F27	fs11		FP Saved Registers					
F28	ft8		FP Temporaries					
F29	ft9		FP Temporaries					
F30	ft10		FP Temporaries					
F31	ft11	Caller	FP Temporaries					

Register in compressed OpCode										
RVC 000 001 010 011 100 101 110 1										
Register	х8	х9	x10	x11	x12	x13	x14	x15		
Integer ABI	s0	s1	a0	a1	a2	аЗ	a4	a5		
Float	f8	f9	f10	f11	f12	f13	f14	f15		
Float ABI	fs0	fs1	fa0	fa1	fa2	fa3	fa4	fa5		