RISC-V Opcodes (RV32I)

by Klaus Kohl-Schöpe (v1.0 - 08.08.2021)

Mnemonics		Opcode																													Type	Operation	Description
		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	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>**	**funct7**							**rs2**					**rs1**					**funct3**			**rd**							**op**						**op = xxbbb11 with bbb<>111**	
ADD	<rd>,<rs1>,<rs2>	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>	0	**1**	0	0	0	0	0	rs2					rs1					0	0	0	rd					0	1	1	0	0	1	1	R	rd = rs1 - rs2	Substrace
SLL	<rd>,<rs1>,<rs2>	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>	0	0	0	0	0	0	0	rs2					rs1					0	1	0	rd					0	1	1	0	0	1	1	R	rd = rs1<rs2 ? Rs1 : r2	Set less than
SLTU	<rd>,<rs1>,<rs2>	0	0	0	0	0	0	0	rs2					rs1					0	1	1	rd					0	1	1	0	0	1	1	R	rd = rs1 < rs2 ? Rs1 : r2	Set less than unsigned
XOR	<rd>,<rs1>,<rs2>	0	0	0	0	0	0	0	rs2					rs1					1	0	0	rd					0	1	1	0	0	1	1	R	rd = rs1 ^ rs2	Xor
SRL	<rd>,<rs1>,<rs2>	0	0	0	0	0	0	0	rs2					rs1					1	0	1	rd					0	1	1	0	0	1	1	R	rd = rs1 >> rs2 (0)	Shift Right Logic (0)
SRA	<rd>,<rs1>,<rs2>	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>	0	0	0	0	0	0	0	rs2					rs1					1	1	0	rd					0	1	1	0	0	1	1	R	rd = rs1 | rs2	Or
AND	<rd>,<rs1>,<rs2>	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**	**imm[11:0]**												**rs1**					**funct3**			**rd**					**op**							**op = xxbbb11 with bbb<>111**	
LB	<rd>,imm(<rs1>)	imm[11:0]												rs1					0	0	0	rd					0	0	0	0	0	1	1	I	rd = Mem[imm + rs1] (signed)	Load Byte
LH	<rd>,imm(<rs1>)	imm[11:0]												rs1					0	0	1	rd					0	0	0	0	0	1	1	I	rd = Mem[imm + rs1] (signed)	Load Halfword
LW	<rd>,imm(<rs1>)	imm[11:0]												rs1					0	1	0	rd					0	0	0	0	0	1	1	I	rd = Mem[imm + rs1]	Load Word
LBU	<rd>,imm(<rs1>)	imm[11:0]												rs1					1	0	0	rd					0	0	0	0	0	1	1	I	rd = Mem[imm + rs1] (unsigned)	Load Byte Unsigned
LHU	<rd>,imm(<rs1>)	imm[11:0]												rs1					1	0	1	rd					0	0	0	0	0	1	1	I	rd = Mem[imm + rs1] (unsigned)	Load Halfword Unsigned
ADDI	<rd>,<rs1>,imm	imm[11:0]												rs1					0	0	0	rd					0	0	1	0	0	1	1	I	rd = rs1 + imm	Add Immediate
SLLI	<rd>,<rs1>,imm	0	0	0	0	0	0	0	imm[4:0]					rs1					0	0	1	rd					0	0	1	0	0	1	1	I	rd = rs1 << imm	Shift Left Logic Immediate
SLTI	<rd>,<rs1>,imm	imm[11:0]												rs1					0	1	0	rd					0	0	1	0	0	1	1	I	rd = rs1 < imm ? 1 : 0	Set Less Than Immediate
SLTIU	<rd>,<rs1>,imm	imm[11:0]												rs1					0	1	1	rd					0	0	1	0	0	1	1	I	rd = rs1 < imm ? 1 : 0	Set Less Than Immediate Unsigned
XORI	<rd>,<rs1>,imm	imm[11:0]												rs1					1	0	0	rd					0	0	1	0	0	1	1	I	rd = rs1 ^ imm	Xor Immediate
SRLI	<rd>,<rs1>,imm	0	0	0	0	0	0	0	imm[4:0]					rs1					1	0	1	rd					0	0	1	0	0	1	1	I	rd = rs1 >> imm	Shift Right Logic Immediate
SRAI	<rd>,<rs1>,imm	0	**1**	0	0	0	0	0	imm[4:0]					rs1					1	0	1	rd					0	0	1	0	0	1	1	I	rd = rs1 >> imm (signed)	Shift Right Arithmetic Immediate
ORI	<rd>,<rs1>,imm	imm[11:0]												rs1					1	1	0	rd					0	0	1	0	0	1	1	I	rd = rs1 | imm	Or Immediate
ANDI	<rd>,<rs1>,imm	imm[11:0]												rs1					1	1	1	rd					0	0	1	0	0	1	1	I	rd = rs1 & imm	And Immediate
JALR	<rd>,<rs1>,imm	imm[11:0]												rs1					0	0	0	rd					1	1	0	0	1	1	1	I	rd = PC+4; PC=rs1+imm&(-2)	Jump And Link Register
S-Type	**<rd>,<rs1>,<rs2>**	**imm[11:5]**							**rs2**					**rs1**					**func3**			**imm[4:0]**					**op**							**op = xxbbb11 with bbb<>111**	
SB	<rs2>,imm(<rs1>)	imm[11:5]							rs2					rs1					0	0	0	imm[4:0]					0	1	0	0	0	1	1	S	Mem[rs1 + imm] = rs2	Store Byte
SH	<rs2>,imm(<rs1>)	imm[11:5]							rs2					rs1					0	0	1	imm[4:0]					0	1	0	0	0	1	1	S	Mem[rs1 + imm] = rs2	Store Halfword
SW	<rs2>,imm(<rs1>)	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**	**imm[12,10:5]**							**rs2**					**rs1**					**funct3**			**imm[4:1,11]**					**op**							**op = xxbbb11 with bbb<>111**	
BEQ	<rs1>,<rs2>,imm	imm[12,10:5]							rs2					rs1					0	0	0	imm[4:1,11]					1	1	0	0	0	1	1	SB	PC = PC+imm<<1 if rs1=rs2	Branch if Equal
BNE	<rs1>,<rs2>,imm	imm[12,10:5]							rs2					rs1					0	0	1	imm[4:1,11]					1	1	0	0	0	1	1	SB	PC = PC+imm<<1 if rs1<>rs2	Branch if Not Equal
BLT	<rs1>,<rs2>,imm	imm[12,10:5]							rs2					rs1					1	0	0	imm[4:1,11]					1	1	0	0	0	1	1	SB	PC = PC+imm<<1 if rs1<rs2	Branch if Less
BGE	<rs1>,<rs2>,imm	imm[12,10:5]							rs2					rs1					1	0	1	imm[4:1,11]					1	1	0	0	0	1	1	SB	PC = PC+imm<<1 if rs1>=rs2	Branch if Greater or Equal
BLTU	<rs1>,<rs2>,imm	imm[12,10:5]							rs2					rs1					1	1	0	imm[4:1,11]					1	1	0	0	0	1	1	SB	PC = PC+imm<<1 if rs1<rs2	Brunch if Less Unsigned
BGEU	<rs1>,<rs2>,imm	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>**	**imm[31:12]**																				**rd**					**op**							**op = xxbbb11 with bbb<>111**	
AUIPC	<rd>,imm	imm[31:12]																				rd					0	0	1	0	1	1	1	U	rd = PC + imm << 12	Load Upper Immediate to PC
LUI	<rd>,imm	imm[31:12]																				rd					0	1	1	0	1	1	1	U	rd = imm << 12	Load Upper Immediate
UJ-Type	**<rd>,imm**	**imm[20]**	**imm[10-1]**										**imm[11]**	**imm[19-12]**								**rd**					**op**							**op = xxbbb11 with bbb<>111**	
JAL	<rd>,imm	20	10	9	8	7	6	5	4	3	2	1	11	19	18	17	16	15	14	13	12	rd					1	1	0	1	1	1	1	UJ	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	
FENCE		0				PI	PO	PR	PW	SI	SO	SR	SW	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	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	0	0	0	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	1	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	1	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	1	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	0	1	0	0	0	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.VMA	<rs1>,<rs2>	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>	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	addr[11:0]												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>	addr[11:0]												rs1					0	1	0	rd					1	1	1	0	0	1	1		rd = CSR; CSR = CSR | rs1	Atomic Read and Set CSR
CSRRSI	<rd>,addr,uimm	addr[11:0]												uimm					1	1	0	rd					1	1	1	0	0	1	1		rd = CSR; CSR = CSR | imm	Atomic Read and Set CSR immedate
CSRRC	<rd>,addr,<rs1>	addr[11:0]												rs1					0	1	1	rd					1	1	1	0	0	1	1		rd = CSR; CSR = CSR & ~rs1	Atomic Read and Clear CSR
CSRRCI	<rd>,addr,uimm	addr[11:0]												uimm					1	1	1	rd					1	1	1	0	0	1	1		rd = CSR; CSR = CSR & ~imm	Atomic Read and Clear CSR immedate

RISC-V Opcodes (RV32C)

by Klaus Kohl-Schöpe (v1.0 - 08.08.2021)

Mnemonics		Opcode																														Type	Operation	Description
		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	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>**	**---**																**func4**				**rd/rs**					**rs2**				**op**		**CR**	**op <> 11**	
C.MV	<rd>,<rs2>	---																1	0	0	0	rd<>0					rs2				1	0	CR	rd = rs2	Move (rd=0: HINT)
C.ADD	<rd>,<rs2>	---																1	0	0	1	rd					rs2<>0				1	0	CR	rd = rd + rs2	Add (rd=0: HINT)
C.JR	<rs>	---																1	0	0	0	rs1<>0					0				1	0	CR	PC = rs1	Jump Reg
C.SUB	<rd'>,<rs2'>	---																1	0	0	0	1	1	rd'			0	0	rs2'			0	1	CR	rd = rd - rs2	Sub
C.JALR	<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**	**---**																**func3**			**i**	**rd**					**imm**				**op**		**CI**	**op <> 11**	
C.LWSP	<rd>,uimm(r2)	---																0	1	0	i5	rd<>0					i4:2|7:6				1	0	CI	rd = mem(x2 + uimm)	Read value from (SP + uimm (*4))
C.LI	<rd>,imm	---																0	1	0	i5	rd<>0					i4:0				0	1	CI	rd = imm	Load immediate
C.LUI	<rd>,imm	---																0	1	1	i17	rd<>0					i16:12				0	1	CI	rd = imm	Load upper immediate (signed bit 17:12)
C.ADDI16SP	<rd>,imm	---																0	1	1	i9	2					i4|6|8:7|5				0	1	CI	SP = SP + imm	Add imm (*16) to SP
C.NOP		---																0	0	0	0	0					0				0	1	CI	No Operation	Pseudocode (r0 ,# 0 c.addi)
C.ADDI	<rd>,imm	---																0	0	0	i5	rd<>0					i4:0				0	1	CI	rd = rd + imm	Add immediate
C.SLLI	<rd>,uimm	---																0	0	0	i5	rd<>0					i4:0				1	0	CI	rd = rd << uimm (i<>0)	Logic shift left (i5=0 for RV32) (rd=0: HINT)
CS-Type	**<rd>,imm**	**---**																**func3**			**uimm**			**rd'**			**uimm**			**rs2'**	**op**		**CS**	**op <> 11**	
C.SW	<rs2'>,uimm(rs')	---																1	1	0	i5:3			rd'			i2,6			rs2'	0	0	CS	mem(rd' + uimm) = rs2'	Store rs2' to (rd' + uimm (*4))
C.XOR	<rd'>,<rs2'>	---																1	0	0	0	1	1	rd'			0	1	rs2'		0	1	CS	rd' = rd' ^ rs2'	Xor
C.OR	<rd'>,<rs2'>	---																1	0	0	0	1	1	rd'			1	0	rs2'		0	1	CS	rd' = rd' | rs2'	Or
C.AND	<rd'>,<rs2'>	---																1	0	0	0	1	1	rd'			1	1	rs2'		0	1	CS	rd' = rd' & rs2'	And
CSS-Type	**<rd>,<rs2>**	**---**																**func3**			**uimm**									rs2'	**op**		**CSS**	**op <> 11**	
C.SWSP	<rs2>,uimm(r2)	---																1	1	0	i5:2|7:6						rs2				1	0	CSS	mem(r2 + uimm) = rs2'	Store rs2' to (SP + uimm (*4))
CIW-Type	**<rs2'>,imm**	**---**																**func3**			**uimm**									**rs2´**	**op**		**CIW**	**op <> 11**	
C.ADDI4SPN	<rs2'>,uimm	---																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**	**---**																**func3**			**uimm**			**rs'**			**uimm**			**rs2'**	**op**		**CL**	**op <> 11**	
C.LW	<rs2'>,uimm(rs')	---																0	1	0	i5:3			rs'			i2,6			rs2'	0	0	CL	rd' = mem(rs' + uimm)	Load word from (rs' + uimm (*4))
CB-Type	**<rd>,imm**	**---**																**func3**			**offset**			**rd'/rs'**			**offset**				**op**		**CB**	**op <> 11**	
C.BEQZ	<rs'>, addr	---																1	1	0	o8|4:3			rs'			o7:6|2:1|5				0	1	CB	PC = PC + offset if rs1' = 0	Branch if rs' is 0 (offset*2)
C.BNEZ	<rs>, addr	---																1	1	1	o8|4:3			rs'			o7:6|2:1|5				0	1	CB	PC = PC + offset if rs1' <> 0	Branch if rs' not 0 (offset*2)
C.ANDI	<rd'>,imm	---																1	0	0	i5	1	0	rd'			i4:0				0	1	CB	rd' = rd' ^ imm	And Immediate
C.SRLI	<rd'>,uimm	---																1	0	0	i5	0	0	rd'			i4:0				0	1	CB	rd' = rd' >> imm	Logic shift right (i5=0 for RV32)
C.SRAI	<rd'>,uimm	---																1	0	0	i5	0	1	rd'			i4:0				0	1	CB	rd' = rd' >> imm (signed)	Arithmetic shift right (i5=0 for RV32)
CJ-Type	**offset**	**---**																**func3**			**jump offset**										**op**		**CJ**	**op <> 11**	
C.J	addr	---																1	0	1	o11|4|9:8|10|6|7|3:1|5										0	1	CJ	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)

Mnemonics		Opcode																															Type	Operation	Description
		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	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
C.ADDI4SPN	<rs2'>,uimm	---																0	0	0	i5:4	9:6	2	3						rs2'		0	0	CIW	rd' = SP + i	Add imm (*4) to SP (imm <> 0)
C.LW	<rs2'>,uimm(rs')	---																0	1	0	i5:3			rs'			i2,6			rs2'		0	0	CL	rd' = mem(rs' + uimm)	Load word from (rs' + imm (*4))
C.SW	<rs2'>,uimm(rd')	---																1	1	0	i5:3			rd'			i2,6			rs2'		0	0	CS	mem(rd' + uimm) = rs2'	Store rs2' to (rd' + uimm (*4))
C.NOP		---																0	0	0	0	0					0					0	1	CI	No Operation	Pseudocode (r0 ,# 0 c.addi)
C.ADDI	<rd>,imm	---																0	0	0	i5	rd<>0					i4:0					0	1	CI	rd = rd + imm	Add immediate
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)
C.LI	<rd>,imm	---																0	1	0	i5	rd<>0					i4:0					0	1	CI	rd = imm	Load immediate
C.LUI	<rd>,imm	---																0	1	1	i17	rd<>0 and <>2					i16:12					0	1	CI	rd = imm	Load upper immediate (signed bit 17:12)
C.ADDI16SP	sp,imm	---																0	1	1	i9	2					i4	6	8:7	5		0	1	CI	SP = SP + imm	Add imm (*16) to SP
C.SRLI	<rd'>,uimm	---																1	0	0	i5	0	0	rd'			i4:0					0	1	CB	rd' = rd' >> imm	Logic shift right (i5=0 for RV32)
C.SRAI	<rd'>,uimm	---																1	0	0	i5	0	1	rd'			i4:0					0	1	CB	rd' = rd' >> imm (signed)	Arithmetic shift right (i5=0 for RV32)
C.ANDI	<rd'>,imm	---																1	0	0	i5	1	0	rd'			i4:0					0	1	CB	rd' = rd' ^ imm	And Immediate
C.SUB	<rd'>,<rs2'>	---																1	0	0	0	1	1	rd'			0	0		rs2'		0	1	CR	rd = rd - rs2	Sub
C.XOR	<rd'>,<rs2'>	---																1	0	0	0	1	1	rd'			0	1		rs2'		0	1	CS	rd' = rd' ^ rs2'	Xor
C.OR	<rd'>,<rs2'>	---																1	0	0	0	1	1	rd'			1	0		rs2'		0	1	CS	rd' = rd' | rs2'	Or
C.AND	<rd'>,<rs2'>	---																1	0	0	0	1	1	rd'			1	1		rs2'		0	1	CS	rd' = rd' & rs2'	And
C.J	addr	---																1	0	1	o11	4	9	8	10	6	7	3	1	5		0	1	CJ	PC = PC + offset	Jump (offset*2)
C.BEQZ	<rs'>, addr	---																1	1	0	o8	4:3		rs'			o7:6	2:1	5			0	1	CB	PC = PC + offset if rs1' = 0	Branch if rs' is 0 (offset*2)
C.BNEZ	<rs'>, addr	---																1	1	1	o8	4:3		rs'			o7:6	2:1	5			0	1	CB	PC = PC + offset if rs1' <> 0	Branch if rs' not 0 (offset*2)
C.SLLI	<rd>,uimm	---																0	0	0	i5	rd<>0					i4:0<>0					1	0	CI	rd = rd << uimm (i<>0)	Logic shift left (i5=0 for RV32) (rd=0: HINT)
C.LWSP	<rd>,uimm(r2)	---																0	1	0	i5	rd<>0					i4:2	7:6				1	0	CI	rd = mem(x2 + uimm)	Read value from (SP + uimm (*4))
C.JR	<rs>	---																1	0	0	0	rs<>0					0					1	0	CR	PC = rs	Jump Reg
C.MV	<rd>,<rs2>	---																1	0	0	0	rd<>0					rs2					1	0	CR	rd = rs2	Move (rd=0: HINT)
C.EBREAK		---																1	0	0	1	0					0					1	0	CR	ebreak	Environment BREAK
C.JALR	<rd>	---																1	0	0	1	rs<>0					0					1	0	CR	rs = PC+2; PC = rs	Jump and Link
C.ADD	<rd>,<rs2>	---																1	0	0	1	rd<>0					rs2<>0					1	0	CR	rd = rd + rs2	Add (rd=0: HINT)
C.SWSP	<rs2>,uimm(r2)	---																1	1	0	i5:2	7:6					rs2					1	0	CSS	mem(r2 + uimm) = rs2'	Store word to (SP + uimm (*4))
LB	<rd>,imm(<rs1>)	imm[11:0]																rs1			0	0	0	rd			0	0	0	0	0	1	1	I	rd = Mem[imm + rs1] (signed)	Load Byte
LH	<rd>,imm(<rs1>)	imm[11:0]																rs1			0	0	1	rd			0	0	0	0	0	1	1	I	rd = Mem[imm + rs1] (signed)	Load Halfword
LW	<rd>,imm(<rs1>)	imm[11:0]																rs1			0	1	0	rd			0	0	0	0	0	1	1	I	rd = Mem[imm + rs1]	Load Word
LBU	<rd>,imm(<rs1>)	uimm[11:0]																rs1			1	0	0	rd			0	0	0	0	0	1	1	I	rd = Mem[imm + rs1] (unsigned)	Load Byte Unsigned
LHU	<rd>,imm(<rs1>)	uimm[11:0]																rs1			1	0	1	rd			0	0	0	0	0	1	1	I	rd = Mem[imm + rs1] (unsigned)	Load Halfword Unsigned
FENCE.I		0			0				0									0	0	0	0						0	0	0	1	1	1	1		Sync write before instruction read	
FENCE		0			PI	PO	PR	PW	SI	SO	SR	SW						0	0	0	0						0	0	0	1	1	1	1		Sync write before instruction read	Sync with additional parameter
ADDI	<rd>,<rs1>,imm	imm[11:0]																rs1			0	0	0	rd			0	0	1	0	0	1	1	I	rd = rs1 + imm	Add Immediate
SLLI	<rd>,<rs1>,imm	0	0	0	0	0	0	0	imm[4:0]									rs1			0	0	1	rd			0	0	1	0	0	1	1	I	rd = rs1 << imm	Shift Left Logic Immediate
SLTI	<rd>,<rs1>,imm	imm[11:0]																rs1			0	1	0	rd			0	0	1	0	0	1	1	I	rd = rs1 < imm ? 1 : 0	Set Less Than Immediate
SLTIU	<rd>,<rs1>,imm	imm[11:0]																rs1			0	1	1	rd			0	0	1	0	0	1	1	I	rd = rs1 < imm ? 1 : 0	Set Less Than Immediate Unsigned
XORI	<rd>,<rs1>,imm	imm[11:0]																rs1			1	0	0	rd			0	0	1	0	0	1	1	I	rd = rs1 ^ imm	Xor Immediate
SRLI	<rd>,<rs1>,imm	0	0	0	0	0	0	0	imm[4:0]									rs1			1	0	1	rd			0	0	1	0	0	1	1	I	rd = rs1 >> imm	Shift Right Logic Immediate
SRAI	<rd>,<rs1>,imm	0	1	0	0	0	0	0	imm[4:0]									rs1			1	0	1	rd			0	0	1	0	0	1	1	I	rd = rs1 >> imm (signed)	Shift Right Arithmetic Immediate
ORI	<rd>,<rs1>,imm	imm[11:0]																rs1			1	1	0	rd			0	0	1	0	0	1	1	I	rd = rs1 | imm	Or Immediate
ANDI	<rd>,<rs1>,imm	imm[11:0]																rs1			1	1	1	rd			0	0	1	0	0	1	1	I	rd = rs1 & imm	And Immediate
AUIPC	<rd>,imm	imm[31:12]																						rd			0	0	1	0	1	1	1	U	rd = PC + imm << 12	Load Upper Immediate to PC
SB	<rs2>,imm(<rs1>)	imm[11:5]							rs2									rs1			0	0	0	imm[4:0]			0	1	0	0	0	1	1	S	Mem[rs1 + imm] = rs2	Store Byte
SH	<rs2>,imm(<rs1>)	imm[11:5]							rs2									rs1			0	0	1	imm[4:0]			0	1	0	0	0	1	1	S	Mem[rs1 + imm] = rs2	Store Halfword
SW	<rs2>,imm(<rs1>)	imm[11:5]							rs2									rs1			0	1	0	imm[4:0]			0	1	0	0	0	1	1	S	rs1 + imm	Store Word
ADD	<rd>,<rs1>,<rs2>	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>	0	1	0	0	0	0	0	rs2									rs1			0	0	0	rd			0	1	1	0	0	1	1	R	rd = rs1 - rs2	Substrace
SLL	<rd>,<rs1>,<rs2>	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>	0	0	0	0	0	0	0	rs2									rs1			0	1	0	rd			0	1	1	0	0	1	1	R	rd = rs1<rs2 ? Rs1 : r2	Set less than
SLTU	<rd>,<rs1>,<rs2>	0	0	0	0	0	0	0	rs2									rs1			0	1	1	rd			0	1	1	0	0	1	1	R	rd = rs1 < rs2 ? Rs1 : r2	Set less than unsigned
XOR	<rd>,<rs1>,<rs2>	0	0	0	0	0	0	0	rs2									rs1			1	0	0	rd			0	1	1	0	0	1	1	R	rd = rs1 ^ rs2	Xor
SRL	<rd>,<rs1>,<rs2>	0	0	0	0	0	0	0	rs2									rs1			1	0	1	rd			0	1	1	0	0	1	1	R	rd = rs1 >> rs2 (0)	Shift Right Logic (0)
SRA	<rd>,<rs1>,<rs2>	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>	0	0	0	0	0	0	0	rs2									rs1			1	1	0	rd			0	1	1	0	0	1	1	R	rd = rs1 | rs2	Or
AND	<rd>,<rs1>,<rs2>	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
LUI	<rd>,imm	imm[31:12]																						rd			0	1	1	0	1	1	1	U	rd = imm << 12	Load Upper Immediate
BEQ	<rs1>,<rs2>,imm	imm[12,10:5]							rs2									rs1			0	0	0	imm[4:1,11]			1	1	0	0	0	1	1	SB	PC = PC+imm<<1 if rs1=rs2	Branch if Equal
BNE	<rs1>,<rs2>,imm	imm[12,10:5]							rs2									rs1			0	0	1	imm[4:1,11]			1	1	0	0	0	1	1	SB	PC = PC+imm<<1 if rs1<>rs2	Branch if Not Equal
BLT	<rs1>,<rs2>,imm	imm[12,10:5]							rs2									rs1			1	0	0	imm[4:1,11]			1	1	0	0	0	1	1	SB	PC = PC+imm<<1 if rs1<rs2	Branch if Less
BGE	<rs1>,<rs2>,imm	imm[12,10:5]							rs2									rs1			1	0	1	imm[4:1,11]			1	1	0	0	0	1	1	SB	PC = PC+imm<<1 if rs1>=rs2	Branch if Greater or Equal
BLTU	<rs1>,<rs2>,imm	imm[12,10:5]							rs2									rs1			1	1	0	imm[4:1,11]			1	1	0	0	0	1	1	SB	PC = PC+imm<<1 if rs1<rs2	Brunch if Less Unsigned
BGEU	<rs1>,<rs2>,imm	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
JALR	<rd>,<rs1>,imm	imm[11:0]																rs1			0	0	0	rd			1	1	0	0	1	1	1	I	rd = PC+4; PC=rs1+imm&(-2)	Jump And Link Register
JAL	<rd>,imm	20	10	9	8	7	6	5	4	3	2	1	11	19	18	17	16	15	14	13	12						rd					1	1	1	1	UJ	rd = PC+4; PC = PC + imm<<1	Jump and Link
ECALL		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0						0					1	1	1	1		MEPC = PC; Call vector 8 or 11	!!! MEPC + 4 required !!!
EBREAK		0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0						0					1	1	1	1		MEPC = PC; Call vector 3	!!! MEPC + 4 required !!!
URET		0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0						0					1	1	1	1		UIE = UPIE; UPIE = 1	Return from Traps (U-Mode)
SRET		0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0						0					1	1	1	1		SIE = SPIE; SPIE = 1	Return from Traps (S-Mode)
WFI		0	0	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0						0					1	1	1	1		Sleep till NMI, interrupt or event	Wait for Interrupt
SFENCE.VMA	<rs1>,<rs2>	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
MRET		0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0						0					1	1	1	1		MIE = MPIE; MPIE = 1	Return from Traps (M-Mode)
CSRRW	<rd>,addr,<rs1>	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	addr[11:0]																uimm			1	0	1	rd			1	1	1	0	0	1	1		rd = CSR; CSR = imm	Atomic Read and Write CSR immediate
CSRRS	<rd>,addr,<rs1>	addr[11:0]																rs1			0	1	0	rd			1	1	1	0	0	1	1		rd = CSR; CSR = CSR | rs1	Atomic Read and Set CSR
CSRRSI	<rd>,addr,uimm	addr[11:0]																uimm			1	1	0	rd			1	1	1	0	0	1	1		rd = CSR; CSR = CSR | imm	Atomic Read and Set CSR immediate
CSRRC	<rd>,addr,<rs1>	addr[11:0]																rs1			0	1	1	rd			1	1	1	0	0	1	1		rd = CSR; CSR = CSR & ~rs1	Atomic Read and Clear CSR
CSRRCI	<rd>,addr,uimm	addr[11:0]																uimm			1	1	1	rd			1	1	1	0	0	1	1		rd = CSR; CSR = CSR & ~imm	Atomic Read and Clear CSR immediate

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	**Stack Pointer**
R3	gp	---	Global Pointer
R4	tp	---	Thread Pointer
R5	t0	Caller	Temporaries
R6	t1	Caller	Temporaries
R7	t2	Caller	Temporaries
R8	s0/fp	Callee	Saved Register/Frame Pointer
R9	s1	Callee	Saved Register
R10	a0	Caller	Function Argument/Return Value
R11	a1	Caller	Function Argument/Return Value
R12	a2	Caller	Function Argument
R13	a3	Caller	Function Argument
R14	a4	Caller	Function Argument
R15	a5	Caller	Function Argument
R16	a6	Caller	Function Argument
R17	a7	Caller	Function Argument
R18	s2	Callee	Saved Registers
R19	s3	Callee	Saved Registers
R20	s4	Callee	Saved Registers
R21	s5	Callee	Saved Registers
R22	s6	Callee	Saved Registers
R23	s7	Callee	Saved Registers
R24	s8	Callee	Saved Registers
R25	s9	Callee	Saved Registers
R26	s10	Callee	Saved Registers
R27	s11	Callee	Saved Registers
R28	t3	Caller	Temporaries
R29	t4	Caller	Temporaries
R30	t5	Caller	Temporaries
R31	t6	Caller	Temporaries

FP Register			
Register	ABI	Saver	Descr.
F0	ft0	Caller	FP Temporaries
F1	ft1	Caller	Return Address
F2	ft2	Caller	Stack Pointer
F3	ft3	Caller	Global Pointer
F4	ft4	Caller	Thread Pointer
F5	ft5	Caller	Temporaries
F6	ft6	Caller	Temporaries
F7	ft7	Caller	Temporaries
F8	fs0	Callee	FP Saved Registers
F9	fs1	Callee	FP Saved Registers
F10	fa0	Caller	FP Argument/Return Value
F11	fa1	Caller	FP Argument/Return Value
F12	fa2	Caller	FP Argument
F13	fa3	Caller	FP Argument
F14	fa4	Caller	FP Argument
F15	fa5	Caller	FP Argument
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	Callee	FP Saved Registers
F22	fs6	Callee	FP Saved Registers
F23	fs7	Callee	FP Saved Registers
F24	fs8	Callee	FP Saved Registers
F25	fs9	Callee	FP Saved Registers
F26	fs10	Callee	FP Saved Registers
F27	fs11	Callee	FP Saved Registers
F28	ft8	Caller	FP Temporaries
F29	ft9	Caller	FP Temporaries
F30	ft10	Caller	FP Temporaries
F31	ft11	Caller	FP Temporaries

Register in compressed OpCode								
RVC	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
Register	x8	x9	x10	x11	x12	x13	x14	x15
Integer ABI	s0	s1	a0	a1	a2	a3	a4	a5
Float	f8	f9	f10	f11	f12	f13	f14	f15
Float ABI	fs0	fs1	fa0	fa1	fa2	fa3	fa4	fa5