

RISC-V Instruction-Set

Erik Engheim <erik.engheim@ma.com>

Arithmetic Operation

Mnemonic	Instruction	Type	Description
ADD	rd, rs1, rs2	R	rd ← rs1 + rs2
SUB	rd, rs1, rs2	R	rd ← rs1 - rs2
ADDI	rd, rs1, imm12	I	rd ← rs1 + imm12
SLT	rd, rs1, rs2	R	rd ← rs1 < rs2 ? 1 : 0
SLTI	rd, rs1, imm12	I	rd ← rs1 < imm12 ? 1 : 0
SLTU	rd, rs1, rs2	R	rd ← rs1 < rs2 ? 1 : 0
SLTIU	rd, rs1, imm12	I	rd ← rs1 < imm12 ? 1 : 0
LUI	rd, imm20	U	rd ← imm20 << 12
AUIP	rd, imm20	U	Add upper immediate to PC

Logical Operations

Mnemonic	Instruction	Type	Description
AND	rd, rs1, rs2	R	rd ← rs1 & rs2
OR	rd, rs1, rs2	R	rd ← rs1 rs2
XOR	rd, rs1, rs2	R	rd ← rs1 ^ rs2
ANDI	rd, rs1, imm12	I	rd ← rs1 & imm12
ORI	rd, rs1, imm12	I	rd ← rs1 imm12
XORI	rd, rs1, imm12	I	rd ← rs1 ^ imm12
SLL	rd, rs1, rs2	R	rd ← rs1 << rs2
SRL	rd, rs1, rs2	R	rd ← rs1 >> rs2
SRA	rd, rs1, rs2	R	rd ← rs1 >> rs2
SLLI	rd, rs1, shamt	I	rd ← rs1 << shamt
SRLI	rd, rs1, shamt	I	rd ← rs1 >> shamt
SRAI	rd, rs1, shamt	I	rd ← rs1 >> shamt

Load / Store Operations

Mnemonic	Instruction	Type	Description
LD	rd, imm12(rs1)	I	rd ← mem[rs1 + imm12]
LW	rd, imm12(rs1)	I	rd ← mem[rs1 + imm12]
LH	rd, imm12(rs1)	I	rd ← mem[rs1 + imm12]
LB	rd, imm12(rs1)	I	rd ← mem[rs1 + imm12]
LWU	rd, imm12(rs1)	I	rd ← mem[rs1 + imm12]
LHU	rd, imm12(rs1)	I	rd ← mem[rs1 + imm12]
LBU	rd, imm12(rs1)	I	rd ← mem[rs1 + imm12]
SD	rs2, imm12(rs1)	S	rs2 → mem[rs1 + imm12]
SW	rs2, imm12(rs1)	S	rs2(31:0) → mem[rs1 + imm12]
SH	rs2, imm12(rs1)	S	rs2(15:0) → mem[rs1 + imm12]
SB	rs2, imm12(rs1)	S	rs2(7:0) → mem[rs1 + imm12]

Branching

Mnemonic	Instruction	Type	Description
BEQ	rs1, rs2, imm12	SB	if rs1 = rs2 PC ← PC + imm12
BNE	rs1, rs2, imm12	SB	if rs1 ≠ rs2 PC ← PC + imm12
BGE	rs1, rs2, imm12	SB	if rs1 ≥ rs2 PC ← PC + imm12
BGEU	rs1, rs2, imm12	SB	if rs1 ≥ rs2 PC ← PC + imm12
BLT	rs1, rs2, imm12	SB	if rs1 < rs2 PC ← PC + imm12
BLTU	rs1, rs2, imm12	SB	if rs1 < rs2 PC ← PC + imm12 << 1
JAL	rd, imm20	UJ	rd ← PC + 4 PC ← PC + imm20
JALR	rd, imm12(rs1)	I	rd ← PC + 4 PC ← rs1 + imm12

Pseudo Instructions

Mnemonic	Instruction	Base instruction(s)
LI	rd, imm12	ADDI rd, zero, imm12
LI	rd, imm	LUI rd, imm[31:12] ADDI rd, rd, imm[11:0]
LA	rd, sym	AUIPC rd, sym[31:12] ADDI rd, rd, sym[11:0]
MV	rd, rs	ADDI rd, rs, 0
NOT	rd, rs	XORI rd, rs, -1
NEG	rd, rs	SUB rd, zero, rs
BGT	rs1, rs2, offset	BLT rs2, rs1, offset
BLE	rs1, rs2, offset	BGE rs2, rs1, offset
BGTU	rs1, rs2, offset	BLTU rs2, rs1, offset
BLEU	rs1, rs2, offset	BGEU rs2, rs1, offset
BEQZ	rs1, offset	BEQ rs1, zero, offset
BNEZ	rs1, offset	BNE rs1, zero, offset
BGEZ	rs1, offset	BGE rs1, zero, offset
BLEZ	rs1, offset	BGE zero, rs1, offset
BGTZ	rs1, offset	BLT zero, rs1, offset
J	offset	JAL zero, offset
CALL	offset12	JALR ra, ra, offset12
CALL	offset	AUIPC ra, offset[31:12] JALR ra, ra, offset[11:0]
RET		JALR zero, 0(ra)
NOP		ADDI zero, zero, 0

Register File

r0	r1	r2	r3
r4	r5	r6	r7
r8	r9	r10	r11
r12	r13	r14	r15
r16	r17	r18	r19
r20	r21	r22	r23
r24	r25	r26	r27
r28	r29	r30	r31

Register Aliases

zero	ra	sp	gp
tp	t0	t1	t2
s0/fp	s1	a0	a1
a2	a3	a4	a5
a6	a7	s2	s3
s4	s5	s6	s7
s8	s9	s10	s11
t3	t4	t5	t6

ra - return address
sp - stack pointer
gp - global pointer
tp - thread pointer

t0 - t6 - Temporary registers
s0 - s11 - Saved by callee
a0 - 17 - Function arguments
a0 - a1 - Return value(s)

32-bit instruction format

	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			
R	func							rs2				rs1				func		rd				opcode													
I	immediate												rs1				func		rd				opcode												
SB	immediate							rs2				rs1				func		immediate				opcode													
UJ	immediate																								rd				opcode						

AVR and RISC-V Instructions

Erik Engheim <erik.engheim@ma.com>

Arithmetic Operation

Mnemonic	Instruction	AVR	AVR Description
ADD	rd, rs1, rs2	Add	ADD rd, rs $rd \leftarrow rd + rs$
SUB	rd, rs1, rs2	Subtract	SUB rd, rs $rd \leftarrow rd - rs$
ADDI	rd, rs1, imm12	Add immediate	SUBI rd, -imm8 $rd \leftarrow rd - (-rs)$
ADDI	rd, rs1, -imm12	Subtract immediate	SUBI rd, imm8 $rd \leftarrow rd - imm8$
LUI	rd, imm20	Load upper immediate	LDI rd, imm8 $rd \leftarrow imm8$
AUIP	rd, imm20	Add upper immediate to PC	LDI rd, imm8 $rd \leftarrow imm8$

Logical Operations

Mnemonic	Instruction	AVR	AVR Description
AND	rd, rs1, rs2	AND	AND rd, rs $rd \leftarrow rd \& rs$
OR	rd, rs1, rs2	OR	OR rd, rs $rd \leftarrow rd rs$
XOR	rd, rs1, rs2	XOR	EOR rd, rs $rd \leftarrow rd \wedge rs$
ANDI	rd, rs1, imm12	AND immediate	ANDI rd, imm8 $rd \leftarrow rd \& imm8$
ORI	rd, rs1, imm12	OR immediate	ORI rd, imm8 $rd \leftarrow rd imm8$
XORI	rd, rs1, imm12	XOR immediate	LDI rs, imm8 EOR rd, rs
SLL	rd, rs1, rs2	Shift left logical	LSL rd $rd \leftarrow rd \ll 1$
SRL	rd, rs1, rs2	Shift right logical	LSR rd $rd \leftarrow rd \gg 1$
SRA	rd, rs1, rs2	Shift right arithmetic	ASR rd $rd \leftarrow rd \gg 1$
		Rotate Left through carry	ROL rd $rd \leftarrow rd \ll 1$ $rd[0] \leftarrow C, C \leftarrow rd[7]$
		Rotate Right through carry	ROR rd $rd \leftarrow rd \gg 1$ $rd[7] \leftarrow C, C \leftarrow rd[0]$
		Swap nibbles	SWAP rd $rd[3..0] \leftrightarrow rd[7..4]$

AVR Register File

r0	r1	r2	r3	r4	r5	r6	r7
r8	r9	r10	r11	r12	r13	r14	r15
r16	r17	r18	r19	r20	r21	r22	r23
r24	r25	r26	r27	r28	r29	r30	r31

AVR Register File

r0	r1	r2	r3	r4	r5	r6	r7
r8	r9	r10	r11	r12	r13	r14	r15
r16	r17	r18	r19	r20	r21	r22	r23
r24	r25	XL	XH	YL	YH	ZL	ZH

X

Y

Z

Callee saved registers
(subroutine must save)

Used for indirect addressing

Load / Store Operations

Mnemonic	Instruction	AVR	AVR Description
LB	rd, imm12(rs1)	Load byte	LDD rd, rs+imm6 $rd \leftarrow mem[rs + imm6]$
LB	rd, 0(rs1)	Load Indirect	LD rd, rs $rd \leftarrow mem[rs]$
LB	rd, 0(r1)	Load Indirect and post-increment	LD rd, rs+ $rd \leftarrow mem[rs], rs \leftarrow rs+1$
ADDI	r1, r1, -1	Load Indirect and pre-decrement	LD rd, -rs $rs \leftarrow rs-1, rd \leftarrow mem[rs]$
LI	rd, imm12	Load Immediate	LDI rd, imm8 $rd \leftarrow imm8$
LUI	r1, r1, imm20	Load Direct from data space (32-bit)	LDS rd, imm16 $rd \leftarrow mem[imm16]$
SB	rs2, imm12(rs1)	Store byte	STD rd+imm6, rs $mem[rd + imm6] \leftarrow rs$
SB	rs2, 0(rs1)	Store Indirect	ST rd, rs $mem[rd] \leftarrow rs$
SB	r2, 0(r1)	Store Indirect and post-increment	ST rd+, rs $mem[rd] \leftarrow rs, rd \leftarrow rd+1$
ADDI	r1, r1, -1	Store Indirect and pre-decrement	ST -rd, rs $rd \leftarrow rd-1, mem[rd] \leftarrow rs$
LUI	r1, r1, imm20	Store Direct to data space (32-bit)	STS imm16, rs $mem[imm16] \leftarrow rs$
SB	r2, imm12(r1)		

Branching

Mnemonic	Instruction	AVR	AVR Description
BEQ	rs1, rs2, imm12	Branch equal	BREQ imm7 if $Z == 1$ $pc \leftarrow pc + imm7$
BNE	rs1, rs2, imm12	Branch not equal	BRNE imm7 if $Z == 0$ $pc \leftarrow pc + imm7$
BGE	rs1, rs2, imm12	Branch greater than or equal	BRGE imm7 if $N \wedge V == 0$ $pc \leftarrow pc + imm7$
BGEU	rs1, rs2, imm12	Branch greater than or equal unsigned	BRSB imm7 if $C == 0$ $pc \leftarrow pc + imm12$
BLT	rs1, rs2, imm12	Branch less than	BRLT imm7 if $rs1 < rs2$ $pc \leftarrow pc + imm12$
BLTU	rs1, rs2, imm12	Branch less than unsigned	BRLO imm7 if $rs1 < rs2$ $pc \leftarrow pc + imm12 \ll 1$
JALR	zero, imm12(zero)	Jump	JMP imm16 $pc \leftarrow imm16$
JAL	zero, imm20	Relative jump	RJMP imm12 $pc \leftarrow pc + imm12$
JALR	zero, imm12(rs1)	Indirect jump	IJMP $pc \leftarrow r31:r30$
JAL	rd, imm20	Jump and link	RCALL imm12 $stack \leftarrow pc + 2$ $pc \leftarrow pc + imm12$
JALR	rd, imm12(zero)	Long call	CALL imm16 $stack \leftarrow pc + 2$ $pc \leftarrow imm16$
JALR	rd, imm12(rs1)	Jump and link register	RET $pc \leftarrow stack$
JALR	rd, imm12(rs1)	Jump and link register	ICALL $stack \leftarrow pc + 2$ $pc \leftarrow r31:r30$

16-bit RISC-V Compared with 16-bit AVR

Erik Engheim <erik.engheim@ma.com>

Arithmetic Operation

Instruction	RISC-V	Description	AVR	Description
ADD Immediate	ADDI Rd, K	$Rd \leftarrow Rd + K$	SUBI Rd, -K	$Rd \leftarrow Rd - (-K)$
ADD Immediate, scaled by 16 to SP	ADDI16SP K	$SP \leftarrow SP + 16 \times K$		
ADD Immediate, scaled by 4 to SP	ADDI4SPN K	$SP \leftarrow SP + 4 \times K$		
ADD	ADD Rd, Rs2	$Rd \leftarrow Rd + Rs2$	ADD Rd, Rr	
SUBstract	SUB Rd, Rs2	$Rd \leftarrow Rd - Rs2$	SUB Rd, Rr	$Rd \leftarrow Rd - Rr$

Logical Operations

Instruction	RISC-V	Description	AVR	Description
AND	AND Rd, Rs2	$Rd \leftarrow Rd \& Rs2$	AND Rd, Rs2	$Rd \leftarrow Rd \& Rr$
AND Immediate	ANDI Rd, K	$Rd \leftarrow Rd \& K$	ANDI Rd, K	$Rd \leftarrow Rd \& K$
OR	OR Rd, Rs2	$Rd \leftarrow Rd Rs2$	OR Rd, Rs2	$Rd \leftarrow Rd Re$
eXclusive OR	XOR Rd, Rs2	$Rd \leftarrow Rd \wedge Rs2$	EOR Rd, Rs2	$Rd \leftarrow Rd \wedge Re$
Shift Left Logical Immediate	SLLI Rd, K	$Rd \leftarrow Rd \ll K$	LSL Rd	$Rd \leftarrow Rd \ll 1$
Shift Right Logical Immediate	SRLI Rd, K	$Rd \leftarrow Rd \gg K$	LSR Rd	$Rd \leftarrow Rd \gg 1$
Shift Right Arithmetic Immediate	SRAI Rd, K	$Rd \leftarrow Rd \gg K$	ASR Rd	$Rd \leftarrow Rd \gg 1$

Branching

Instruction	RISC-V	Description	AVR	Description
Branch if EQual to Zero	BEQZ Rs1, K			
Branch if Not Equal to Zero	BNEZ Rs1, K			
Jump	J K	$PC \leftarrow PC + K$	RJMP K	$PC \leftarrow PC + K + 1$
Jump Register	JR Rs1	$PC \leftarrow Rs1$	IJMP	$PC \leftarrow Z$
Jump And Link	JAL K	$Ra \leftarrow PC + 2$ $PC \leftarrow PC + K$	RCALL K	$stack \leftarrow PC + 2$ $PC \leftarrow PC + K$
Jump And Link Register	JALR Rs1	$Ra \leftarrow PC + 2$ $PC \leftarrow Rs1$	ICALL	$stack \leftarrow PC + 2$ $PC \leftarrow Z$

Load / Store Operations

Instruction	RISC-V	Description	AVR	Description
Load Word indirect with displacement	LW Rd, K(Rs1)	$Rd \leftarrow M[Rs1 + K]$	LDD Rd, Y + K	$Rd \leftarrow M[Y + K]$
Load indirect			LD Rd, Y	$Rd \leftarrow M[Y]$
Load Indirect and post-increment			LD Rd, Y+	$Rd \leftarrow M[Y]$ $Y \leftarrow Y + 1$
Load Indirect and pre-decrement			LD Rd, -Y	$Y \leftarrow Y - 1$ $Rd \leftarrow M[Y]$
	LWSP Rd, K	$Rd \leftarrow M[SP + K]$		
Load Immediate	LI Rd, K	$Rd \leftarrow K$	LDI Rd, K	$Rd \leftarrow K$
Load Upper Immediate	LUI Rd, K	$Rd \leftarrow K \ll 12$		
Move (copy register)	MV Rd, Rs2	$Rd \leftarrow Rs2$	MOV Rd, Rr	$Rd \leftarrow Rr$
Copy register pair			MOVW Rd, Rr	$Rd+1:Rd \leftarrow Rr+1:Rr$
Store Word indirect with displacement	SW Rs2, K(Rs1)	$Rs2 \rightarrow M[Rs1 + K]$	STD Y + K, Rr	$M[Y + K] \leftarrow Rr$
Store indirect			ST Y, Rr	$M[Y] \leftarrow Rr$
Store Indirect and post-increment			ST Y+, Rr	$M[Y] \leftarrow Rr$ $X \leftarrow X + 1$
Store Indirect and pre-decrement			ST -Y, Rr	$Y \leftarrow Y - 1$ $M[Y] \leftarrow Rr$
Store Word, Stack Pointer relative	SWSP Rs2, K	$Rs2 \rightarrow M[K]$		

Legend

Rd - Destination register
Rr, Rs1, Rs2 - Source registers
K - Constant
M - Memory
PC - Program Counter
SP - Stack Pointer
Y - One of the X, Y, Z registers on AVR

AVR Register File

r0	r1	r2	r3	r4	r5	r6	r7	Callee saved registers (subroutine must save)
r8	r9	r10	r11	r12	r13	r14	r15	
r16	r17	r18	r19	r20	r21	r22	r23	
r24	r25	XL	XH	YL	YH	ZL	ZH	Used for indirect addressing
		X		Y		Z		

AVR Register File

Callee saved registers
(subroutine must save)

r0	r1	r2	r3	r4	r5	r6	r7
r8	r9	r10	r11	r12	r13	r14	r15
r16	r17	r18	r19	r20	r21	r22	r23
r24	r25	XL	XH	YL	YH	ZL	ZH

X Y Z
Used for indirect addressing

RISC-V Register File

r0	r1	r2	r3
r4	r5	r6	r7
r8	r9	r10	r11
r12	r13	r14	r15
r16	r17	r18	r19
r20	r21	r22	r23
r24	r25	r26	r27
r28	r29	r30	r31



RISC-V Register Aliases

zero	ra	sp	gp
tp	t0	t1	t2
s0/fp	s1	a0	a1
a2	a3	a4	a5
a6	a7	s2	s3
s4	s5	s6	s7
s8	s9	s10	s11
t3	t4	t5	t6

- ra - return address
- sp - stack pointer
- gp - global pointer
- tp - thread pointer

- t0 - t6 - Temporary registers
- s0 - s11 - Saved by callee
- a0 - 17 - Function arguments
- a0 - a1 - Return value(s)