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 ← rd + rs`
`SUB rd, rs1, rs2`	Subtract	`SUB rd, rs`	`rd ← rd - rs`
`ADDI rd, rs1, imm12`	Add immediate		
	Subtract immediate	`SUBI rd, imm`	`rd ← rd - imm`
`LUI rd, imm20`	Load upper immediate	`LDI rd, imm8`	`rd ← imm8`
`AUIP rd, imm20`	Add upper immediate to PC	`LDI rd, imm8`	`rd ← imm8`

Logical Operations

Mnemonic	Instruction	AVR	AVR Description
`AND rd, rs1, rs2`	AND	`AND rd, rs`	`rd ← rd & rs`
`OR rd, rs1, rs2`	OR	`OR rd, rs`	`rd ← rd | rs`
`XOR rd, rs1, rs2`	XOR	`EOR rd, rs`	`rd ← rd ^ rs`
`ANDI rd, rs1, imm12`	AND immediate	`ANDI rd, imm8`	`rd ← rd & imm8`
`ORI rd, rs1, imm12`	OR immediate	`ORI rd, imm8`	`rd ← rd | imm8`
`XORI rd, rs1, imm12`	XOR immediate		
`SLL rd, rs1, rs2`	Shift left logical	`LSL rd`	`rd ← rd << 1`
`SRL rd, rs1, rs2`	Shift right logical	`LSR rd`	`rd ← rd >> 1`
`SRA rd, rs1, rs2`	Shift right arithmetic	`ASR rd`	`rd ← rd >> 1`
	Rotate Left through carry	`ROL rd`	`rd ← rd << 1` `rd[0] ← C, C ← rd[7]`
	Rotate Right through carry	`ROR rd`	`rd ← rd >> 1` `rd[7] ← C, C ← rd[0]`
	Swap nibbles	SWAP rd	`rd[3..0] ⟷ 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

Register Aliases

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 ← mem[rs + imm6]`
	Load Indirect	`LD rd, rs`	`rd ← mem[rs]`
	Load Indirect and post-increment	`LD rd, rs+`	`rd ← mem[rs], rs ← rs+1`
	Load Indirect and pre-decrement	`LD rd, -rs`	`rs ← rs-1, rd ← mem[rs]`
`LI rd, imm12`	Load Immediate	`LDI rd, imm8`	`rd ← imm8`
	Load Direct from data space (32-bit)	`LDS rd, imm16`	`rd ← mem[imm16]`
`SB rs2, imm12(rs1)`	Store byte	`STD rd+imm6, rs`	`mem[rd + imm6] ← rs`
	Store Indirect	`ST rd, rs`	`mem[rd] ← rs`
	Store Indirect and post-increment	`ST rd+, rs`	`mem[rd] ← rs, rd ← rd+1`
	Store Indirect and pre-decrement	`ST -rd, rs`	`rd ← rd-1, mem[rd] ← rs`
	Store Direct to data space (32-bit)	`STS imm16, rs`	`mem[imm16] ← rs`

Branching

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