Feather ISA

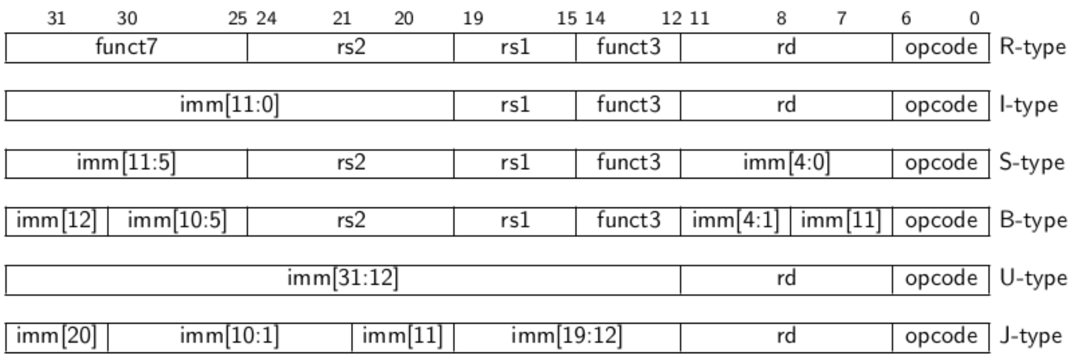
This custom ISA is designed based on `RV32I`.

32-bit Instructions, 32 General Registers of 32-bit width.

Some signed instructions beyond RV32I are implemented in this ISA.

- [ ] `ebreak` interruption implementation

Instruction Format


31 30 25 24 21 20 19 15 14 12 11 8 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] | imm[10:5] | rs2 | rs1 | funct3 | imm[4:1] | imm[11] | opcode | B-type
imm[31:12] | rd | opcode | U-type
imm[20] | imm[10:1] | imm[11] | imm[19:12] | rd | opcode | J-type

Instructions with * are custom instructions beyond `RV32I`.

R type

Inst	Name	Opcode	funct3	funct7	Description	Note
add	Add	0110011	000	0x00	rd = rs1 + rs2	
mul	Mul	0110011	000	0x01	rd = rs1 * rs2	low 32 bits
*addu	Unsigned Add	0110011	000	0x10	rd = rs1 + rs2	ignore overflow
sub	Sub	0110011	000	0x20	rd = rs1 - rs2	
*subu	Unsigned Sub	0110011	000	0x30	rd = rs1 - rs2	ignore overflow
sll	Shift Left Logical	0110011	001	0x00	rd = rs1 << rs2	
slt	Set Less Than	0110011	010	0x00	rd = rs1 < rs2	
sltu	Unsigned Set Less Than	0110011	011	0x00	rd = rs1 < rs2	zero-extends
xor	Xor	0110011	100	0x00	rd = rs1 ^ rs2	
div	Div	0110011	100	0x01	rs = rs1 / rs2	
srl	Shift Right Logical	0110011	101	0x00	rd = rs1 >> rs2	
sra	Shift Right Arithmetic	0110011	101	0x20	rd = rs1 >> rs2	msb-extends
or	Or	0110011	110	0x00	rd = rs1 | rs2	rs2
rem	Remainder	0110011	110	0x01	rd = rs1 % rs2	
and	And	0110011	111	0x00	rd = rs1 & rs2	

I type

Inst	Name	Opcode	funct3	imm[11:5]	Description	Note
addi	Add Imm	0010011	000		rd = rs1 + imm	
slli	Shift Left Logical Imm	0010011	001	0x00	rd = rs1 << imm	
slti	Set Less Than Imm	0010011	010		rd = rs1 < imm	
sltiu	Unsigned Set Less Than Imm	0010011	011		rd = rs1 < imm	zero-extends
xori	Xor Imm	0010011	100		rd = rs1 ^ imm	
srli	Shift Right Logical Imm	0010011	101	0x00	rd = rs1 >> imm	
srai	Shift Right Arithmetic Imm	0010011	101	0x20	rd = rs1 >> imm	msb-extends
ori	Or Imm	0010011	110		rd = rs1 \\| imm	
andi	And Imm	0010011	111		rd = rs1 & imm	
lb	Load Byte	0000011	000		rd = M[rs1+imm][0:7]	
lh	Load Half	0000011	001		rd = M[rs1+imm][0:15]	
lw	Load Word	0000011	010		rd = M[rs1+imm][0:31]	
lbu	Load Byte (U)	0000011	100		rd = M[rs1+imm][0:7]	zero-extends
lhu	Load Half (U)	0000011	101		rd = M[rs1+imm][0:15]	zero-extends
ecall	Environment Call	1110011	000		Transfer control to OS	

ebreak	Environment Break	1110011	000		Transfer control to debugger	

S type

Inst	Name	Opcode	funct3	Description
sb	Store Byte	0100011	000	M[rs1+imm][0:7]=rs2[0:7]
sh	Store Half	0100011	001	M[rs1+imm][0:15]=rs2[0:15]
sw	Store Word	0100011	010	M[rs1+imm][0:31]=rs2[0:31]

B type

Inst	Name	Opcode	funct3	Description	Note
beq	Branch ==	1100011	000	if(rs1 == rs2) PC+=imm	
bne	Branch !=	1100011	001	if(rs1 != rs2) PC+=imm	
blt	Branch <	1100011	100	if(rs1 < rs2) PC+=imm	
bge	Branch ≥	1100011	101	if(rs1 >= rs2) PC+=imm	
bltu	Unsigned Branch <	1100011	110	if(rs1 < rs2) PC+=imm	zero-extends
bgeu	Unsigned Branch ≥	1100011	111	if(rs1 >= rs2) PC+=imm	zero-extends

J type

Inst	Name	Opcode	funct3	Description
jal	Jump And Link	1101111		rd = PC + 4, PC += imm
jalr	Jump And Link Reg	1100111	000	rd = PC + 4, PC = rs1 + imm

U type

Inst	Name	Opcode	Description
lui	Load Upper Imm	0110111	rd = imm << 12
auipc	Add Upper Imm to PC	0010111	rd = PC + (imm << 12)