MyISA Instruction Set Architecture

Reference Manual

June 2026

Revision History

Version	Date	Description
2.0.0	June 2026	Initial MyISA release

Introduction

This document defines the **MyISA Instruction Set Architecture (ISA)** version 2.0.0.

Scope

This specification covers:

- Instruction formats and encoding
- Register file and calling convention
- All base and extension instructions
- Control and status registers (CSRs)
- Exception and trap handling

ISA Overview

The MyISA ISA is a load-store architecture with:

- 32 general-purpose registers (x0–x31)
- Fixed-width 32-bit instructions
- Three operand formats: register (R-type), immediate (I-type), and store (S-type)
- Branch (B-type) and upper-immediate (U-type) variants

- A separate jump-and-link (J-type) format

Registers

General-Purpose Registers

The architecture provides 32 general-purpose registers (x0–x31), each 32 bits wide. Register x0 is hardwired to the constant zero.

Register	ABI Name	Description	Callee-Saved
x0	zero	Always-zero register — reads return 0, writes are discarded	—
x1	ra	Return address link register	No
x2	sp	Stack pointer	Yes
x3	gp	Global pointer	Yes
x4	tp	Thread pointer	Yes
x5	t0	Temporary register 0	No
x6	t1	Temporary register 1	No
x7	t2	Temporary register 2	No
x8	s0	Saved register 0 / frame pointer	Yes
x9	s1	Saved register 1	Yes
x10	a0	Function argument 0 / return value 0	No
x11	a1	Function argument 1 / return value 1	No
x12	a2	Function argument 2	No
x13	a3	Function argument 3	No
x14	a4		No

Register	ABI Name	Description	Callee-Saved
		Function argument 4	
x15	a5	Function argument 5	No
x16	a6	Function argument 6	No
x17	a7	Function argument 7	No
x18	s2	Saved register 2	Yes
x19	s3	Saved register 3	Yes
x20	s4	Saved register 4	Yes
x21	s5	Saved register 5	Yes
x22	s6	Saved register 6	Yes
x23	s7	Saved register 7	Yes
x24	s8	Saved register 8	Yes
x25	s9	Saved register 9	Yes
x26	s10	Saved register 10	Yes
x27	s11	Saved register 11	Yes
x28	t3	Temporary register 3	No
x29	t4	Temporary register 4	No
x30	t5	Temporary register 5	No
x31	t6	Temporary register 6	No

ABI Calling Convention

Register	ABI Name	Caller-Saved	Argument
x0	zero	No	No
x1	ra	No	No
x2	sp	No	No
x3	gp	No	No
x4	tp	No	No

Register	ABI Name	Caller-Saved	Argument
x5	t0	Yes	No
x6	t1	Yes	No
x7	t2	Yes	No
x8	s0	No	No
x9	s1	No	No
x10	a0	Yes	Yes
x11	a1	Yes	Yes
x12	a2	Yes	Yes
x13	a3	Yes	Yes
x14	a4	Yes	Yes
x15	a5	Yes	Yes
x16	a6	Yes	Yes
x17	a7	Yes	Yes
x18	s2	No	No
x19	s3	No	No
x20	s4	No	No
x21	s5	No	No
x22	s6	No	No
x23	s7	No	No
x24	s8	No	No
x25	s9	No	No
x26	s10	No	No
x27	s11	No	No
x28	t3	Yes	No
x29	t4	Yes	No
x30	t5	Yes	No
x31	t6	Yes	No

Instruction Formats

Instructions are encoded in one of several fixed formats. All formats share the same opcode placement (bits 6:0) to allow efficient decoding.

R-Type Format

Bit width: **32 bits**

```
 31:25 |24:20|19:15|14:12|11:7| 6:0
funct7 | rs2 | rs1 |funct3| rd  |opcode
   7   |  5  |  5  | 3 |  5  |   7
```

I-Type Format

Bit width: **32 bits**

```
   31:20    |19:15|14:12|11:7| 6:0
    imm     | rs1 |funct3| rd  |opcode
     12     |  5  | 3 |  5  |   7
```

S-Type Format

Bit width: **32 bits**

```
 31:25 |24:20|19:15|14:12|11:7| 6:0
imm_hi | rs2 | rs1 |funct3|imm_lo|opcode
   7   |  5  |  5  | 3 |  5  |   7
```

B-Type Format

Bit width: **32 bits**

```
31:31|30:25 |24:20|19:15|14:12|11:7| 6:0
imm_12|imm_10_5| rs2 | rs1 |funct3|imm_4_1|opcode
1|  6   |  5  |  5  | 3 |  5  |   7
```

U-Type Format

Bit width: **32 bits**

```
       31:12        |11:7| 6:0
        imm         | rd  |opcode
         20         |  5  |   7
```

J-Type Format

Bit width: **32 bits**

```
31:31|  30:21   |20:20| 19:12  |11:7| 6:0
imm_20| imm_10_1 |imm_11|imm_19_12| rd  |opcode
1|    10    |1|   8    |  5  |   7
```

Instruction Set

Branch Operations

Mnemonic	Format	Opcode	Funct3	Funct7	Operands	Description
`BNE`	B	`0x63`	0x1	—	rs1, rs2, label	Branch not equal
`BEQ`	B	`0x63`	0x0	—	rs1, rs2, label	Branch equal
`BLT`	B	`0x63`	0x4	—	rs1, rs2, label	Branch less than (signed)
`BGE`	B	`0x63`	0x5	—	rs1, rs2, label	Branch greater than or equal (signed)
`BLTU`	B	`0x63`	0x6	—	rs1, rs2, label	Branch less than (unsigned)
`BGEU`	B	`0x63`	0x7	—	rs1, rs2, label	Branch greater than or equal (unsigned)

Integer Operations

Mnemonic	Format	Opcode	Funct3	Funct7	Operands	Description
`AUIPC`	U	`0x17`	—	—	rd, imm20	Add upper immediate to PC — forms PC-relative address
`SLT`	R	`0x33`	0x2	0x00	rd, rs1, rs2	Set if rs1 is less than rs2 (signed)
`SLTU`	R	`0x33`	0x3	0x00	rd, rs1, rs2	Set if rs1 is less than rs2 (unsigned)

Mnemonic	Format	Opcode	Funct3	Funct7	Operands	Description
SLTI	I	0x13	0x2	—	rd, rs1, imm12	Set if rs1 is less than immediate (signed)
SLTIU	I	0x13	0x3	—	rd, rs1, imm12	Set if rs1 is less than immediate (unsigned)
LUI	U	0x37	—	—	rd, imm20	Load upper immediate — places 20-bit immediate in upper 20 bits of rd
ADD	R	0x33	0x0	0x00	rd, rs1, rs2	Add registers
SUB	R	0x33	0x0	0x20	rd, rs1, rs2	Subtract registers
ADDI	I	0x13	0x0	—	rd, rs1, imm12	Add sign-extended 12-bit immediate to register rs1

Jump Operations

Mnemonic	Format	Opcode	Funct3	Funct7	Operands	Description
JAL	J	0x6F	—	—	rd, label	Jump and link — jump to PC+offset, save return address to rd
JALR	I	0x67	0x0	—	rd, rs1, offset	Jump and link register — jump to rs1+offset,

Mnemonic	Format	Opcode	Funct3	Funct7	Operands	Description
						save return address

Logical Operations

Mnemonic	Format	Opcode	Funct3	Funct7	Operands	Description
`XOR`	R	`0x33`	0x4	0x00	rd, rs1, rs2	Bitwise XOR
`OR`	R	`0x33`	0x6	0x00	rd, rs1, rs2	Bitwise OR
`AND`	R	`0x33`	0x7	0x00	rd, rs1, rs2	Bitwise AND
`ORI`	I	`0x13`	0x6	—	rd, rs1, imm12	Bitwise OR with immediate
`XORI`	I	`0x13`	0x4	—	rd, rs1, imm12	Bitwise XOR with immediate
`ANDI`	I	`0x13`	0x7	—	rd, rs1, imm12	Bitwise AND with immediate

Memory Operations

Mnemonic	Format	Opcode	Funct3	Funct7	Operands	Description
`SW`	S	`0x23`	0x2	—	rs2, offset(rs1)	Store word
`LB`	I	`0x03`	0x0	—	rd, offset(rs1)	Load byte (sign-extended)
`LH`	I	`0x03`	0x1	—	rd, offset(rs1)	Load halfword (sign-extended)
`LW`	I	`0x03`	0x2	—	rd, offset(rs1)	Load word
`LBU`	I	`0x03`	0x4	—	rd, offset(rs1)	

Mnemonic	Format	Opcode	Funct3	Funct7	Operands	Description
						Load byte (zero-extended)
`LHU`	I	`0x03`	0x5	—	rd, offset(rs1)	Load halfword (zero-extended)
`SB`	S	`0x23`	0x0	—	rs2, offset(rs1)	Store byte
`SH`	S	`0x23`	0x1	—	rs2, offset(rs1)	Store halfword

Shift Operations

Mnemonic	Format	Opcode	Funct3	Funct7	Operands	Description
`SRAI`	I	`0x13`	0x5	—	rd, rs1, shamt5	Arithmetic right shift by immediate shift amount
`SLLI`	I	`0x13`	0x1	—	rd, rs1, shamt5	Logical left shift by immediate shift amount
`SRA`	R	`0x33`	0x5	0x20	rd, rs1, rs2	Arithmetic right shift by lower 5 bits of rs2
`SRL`	R	`0x33`	0x5	0x00	rd, rs1, rs2	Logical right shift by lower 5 bits of rs2
`SLL`	R	`0x33`	0x1	0x00	rd, rs1, rs2	Logical left shift by lower 5 bits of rs2
`SRLI`	I	`0x13`	0x5	—	rd, rs1, shamt5	Logical right shift by

Mnemonic	Format	Opcode	Funct3	Funct7	Operands	Description
						immediate shift amount

Synchronization Operations

Mnemonic	Format	Opcode	Funct3	Funct7	Operands	Description
`FENCEI`	I	`0x0F`	0x1	—	—	Instruction fence — synchronizes instruction and data streams
`FENCE`	I	`0x0F`	0x0	—	pred, succ	Memory ordering fence

System Operations

Mnemonic	Format	Opcode	Funct3	Funct7	Operands	Description
`CSRRCI`	I	`0x73`	0x7	—	rd, csr, uimm5	Atomic read and clear bits in CSR with immediate
`ECALL`	I	`0x73`	0x0	—	—	Environment call — raises a system call exception
`EBREAK`	I	`0x73`	0x0	—	—	Breakpoint — raises a breakpoint exception
`CSRRW`	I	`0x73`	0x1	—	rd, csr, rs1	Atomic read/write CSR — writes rs1 to CSR, returns old value in rd
`CSRRS`	I	`0x73`	0x2	—	rd, csr, rs1	

Mnemonic	Format	Opcode	Funct3	Funct7	Operands	Description
						Atomic read and set bits in CSR
`CSRRC`	I	`0x73`	0x3	—	rd, csr, rs1	Atomic read and clear bits in CSR
`CSRRWI`	I	`0x73`	0x5	—	rd, csr, uimm5	Atomic read/write CSR with immediate
`CSRRSI`	I	`0x73`	0x6	—	rd, csr, uimm5	Atomic read and set bits in CSR with immediate

Instruction Details

BNE

- **Format:** B
- **Opcode:** `0x63`
- **Funct3:** `0x1`
- **Operands:** `rs1, rs2, label`
- **Description:** Branch not equal
- **Operation:** `if (rs1 != rs2) PC += sext(offset)`

BEQ

- **Format:** B
- **Opcode:** `0x63`
- **Funct3:** `0x0`
- **Operands:** `rs1, rs2, label`
- **Description:** Branch equal
- **Operation:** `if (rs1 == rs2) PC += sext(offset)`

BLT

- **Format:** B
- **Opcode:** `0x63`
- **Funct3:** `0x4`
- **Operands:** `rs1, rs2, label`
- **Description:** Branch less than (signed)

- **Operation:** `if (rs1 < rs2) PC += sext(offset)`

BGE

- **Format:** B
- **Opcode:** `0x63`
- **Funct3:** `0x5`
- **Operands:** `rs1, rs2, label`
- **Description:** Branch greater than or equal (signed)
- **Operation:** `if (rs1 >= rs2) PC += sext(offset)`

BLTU

- **Format:** B
- **Opcode:** `0x63`
- **Funct3:** `0x6`
- **Operands:** `rs1, rs2, label`
- **Description:** Branch less than (unsigned)
- **Operation:** `if (rs1 < rs2) PC += sext(offset)`

BGEU

- **Format:** B
- **Opcode:** `0x63`
- **Funct3:** `0x7`
- **Operands:** `rs1, rs2, label`
- **Description:** Branch greater than or equal (unsigned)
- **Operation:** `if (rs1 >= rs2) PC += sext(offset)`

AUIPC

- **Format:** U
- **Opcode:** `0x17`
- **Operands:** `rd, imm20`
- **Description:** Add upper immediate to PC — forms PC-relative address
- **Operation:** `rd = PC + (imm20 << 12)`

SLT

- **Format:** R
- **Opcode:** `0x33`
- **Funct3:** `0x2`
- **Funct7:** `0x00`
- **Operands:** `rd, rs1, rs2`

- **Description:** Set if rs1 is less than rs2 (signed)
- **Operation:** `rd = (rs1 < rs2) ? 1 : 0`

SLTU

- **Format:** R
- **Opcode:** `0x33`
- **Funct3:** `0x3`
- **Funct7:** `0x00`
- **Operands:** `rd, rs1, rs2`
- **Description:** Set if rs1 is less than rs2 (unsigned)
- **Operation:** `rd = (rs1 < rs2) ? 1 : 0`

SLTI

- **Format:** I
- **Opcode:** `0x13`
- **Funct3:** `0x2`
- **Operands:** `rd, rs1, imm12`
- **Description:** Set if rs1 is less than immediate (signed)
- **Operation:** `rd = (rs1 < sext(imm12)) ? 1 : 0`

SLTIU

- **Format:** I
- **Opcode:** `0x13`
- **Funct3:** `0x3`
- **Operands:** `rd, rs1, imm12`
- **Description:** Set if rs1 is less than immediate (unsigned)
- **Operation:** `rd = (rs1 < sext(imm12)) ? 1 : 0`

LUI

- **Format:** U
- **Opcode:** `0x37`
- **Operands:** `rd, imm20`
- **Description:** Load upper immediate — places 20-bit immediate in upper 20 bits of rd
- **Operation:** `rd = imm20 << 12`

ADD

- **Format:** R
- **Opcode:** `0x33`
- **Funct3:** `0x0`

- **Funct7:** `0x00`
- **Operands:** `rd, rs1, rs2`
- **Description:** Add registers
- **Operation:** `rd = rs1 + rs2`

SUB

- **Format:** R
- **Opcode:** `0x33`
- **Funct3:** `0x0`
- **Funct7:** `0x20`
- **Operands:** `rd, rs1, rs2`
- **Description:** Subtract registers
- **Operation:** `rd = rs1 - rs2`

ADDI

- **Format:** I
- **Opcode:** `0x13`
- **Funct3:** `0x0`
- **Operands:** `rd, rs1, imm12`
- **Description:** Add sign-extended 12-bit immediate to register rs1
- **Operation:** `rd = rs1 + sext(imm12)`

JAL

- **Format:** J
- **Opcode:** `0x6F`
- **Operands:** `rd, label`
- **Description:** Jump and link — jump to PC+offset, save return address to rd
- **Operation:** `rd = PC + 4; PC += sext(offset)`

JALR

- **Format:** I
- **Opcode:** `0x67`
- **Funct3:** `0x0`
- **Operands:** `rd, rs1, offset`
- **Description:** Jump and link register — jump to rs1+offset, save return address
- **Operation:** `rd = PC + 4; PC = (rs1 + sext(offset)) & ~1`

XOR

- **Format:** R
- **Opcode:** `0x33`
- **Funct3:** `0x4`
- **Funct7:** `0x00`
- **Operands:** `rd, rs1, rs2`
- **Description:** Bitwise XOR
- **Operation:** `rd = rs1 ^ rs2`

OR

- **Format:** R
- **Opcode:** `0x33`
- **Funct3:** `0x6`
- **Funct7:** `0x00`
- **Operands:** `rd, rs1, rs2`
- **Description:** Bitwise OR
- **Operation:** `rd = rs1 | rs2`

AND

- **Format:** R
- **Opcode:** `0x33`
- **Funct3:** `0x7`
- **Funct7:** `0x00`
- **Operands:** `rd, rs1, rs2`
- **Description:** Bitwise AND
- **Operation:** `rd = rs1 & rs2`

ORI

- **Format:** I
- **Opcode:** `0x13`
- **Funct3:** `0x6`
- **Operands:** `rd, rs1, imm12`
- **Description:** Bitwise OR with immediate
- **Operation:** `rd = rs1 | sext(imm12)`

XORI

- **Format:** I
- **Opcode:** `0x13`
- **Funct3:** `0x4`
- **Operands:** `rd, rs1, imm12`

- **Description:** Bitwise XOR with immediate
- **Operation:** `rd = rs1 ^ sext(imm12)`

ANDI

- **Format:** I
- **Opcode:** `0x13`
- **Funct3:** `0x7`
- **Operands:** `rd, rs1, imm12`
- **Description:** Bitwise AND with immediate
- **Operation:** `rd = rs1 & sext(imm12)`

SW

- **Format:** S
- **Opcode:** `0x23`
- **Funct3:** `0x2`
- **Operands:** `rs2, offset(rs1)`
- **Description:** Store word
- **Operation:** `MEM[rs1 + offset][31:0] = rs2[31:0]`

LB

- **Format:** I
- **Opcode:** `0x03`
- **Funct3:** `0x0`
- **Operands:** `rd, offset(rs1)`
- **Description:** Load byte (sign-extended)
- **Operation:** `rd = sext(MEM[rs1 + offset][7:0])`

LH

- **Format:** I
- **Opcode:** `0x03`
- **Funct3:** `0x1`
- **Operands:** `rd, offset(rs1)`
- **Description:** Load halfword (sign-extended)
- **Operation:** `rd = sext(MEM[rs1 + offset][15:0])`

LW

- **Format:** I
- **Opcode:** `0x03`
- **Funct3:** `0x2`
- **Operands:** `rd, offset(rs1)`

- **Description:** Load word
- **Operation:** `rd = MEM[rs1 + offset][31:0]`

LBU

- **Format:** I
- **Opcode:** `0x03`
- **Funct3:** `0x4`
- **Operands:** `rd, offset(rs1)`
- **Description:** Load byte (zero-extended)
- **Operation:** `rd = MEM[rs1 + offset][7:0]`

LHU

- **Format:** I
- **Opcode:** `0x03`
- **Funct3:** `0x5`
- **Operands:** `rd, offset(rs1)`
- **Description:** Load halfword (zero-extended)
- **Operation:** `rd = MEM[rs1 + offset][15:0]`

SB

- **Format:** S
- **Opcode:** `0x23`
- **Funct3:** `0x0`
- **Operands:** `rs2, offset(rs1)`
- **Description:** Store byte
- **Operation:** `MEM[rs1 + offset][7:0] = rs2[7:0]`

SH

- **Format:** S
- **Opcode:** `0x23`
- **Funct3:** `0x1`
- **Operands:** `rs2, offset(rs1)`
- **Description:** Store halfword
- **Operation:** `MEM[rs1 + offset][15:0] = rs2[15:0]`

SRAI

- **Format:** I
- **Opcode:** `0x13`
- **Funct3:** `0x5`
- **Operands:** `rd, rs1, shamt5`

- **Description:** Arithmetic right shift by immediate shift amount
- **Operation:** `rd = rs1 >>> shamt`

SLLI

- **Format:** I
- **Opcode:** `0x13`
- **Funct3:** `0x1`
- **Operands:** `rd, rs1, shamt5`
- **Description:** Logical left shift by immediate shift amount
- **Operation:** `rd = rs1 << shamt`

SRA

- **Format:** R
- **Opcode:** `0x33`
- **Funct3:** `0x5`
- **Funct7:** `0x20`
- **Operands:** `rd, rs1, rs2`
- **Description:** Arithmetic right shift by lower 5 bits of rs2
- **Operation:** `rd = rs1 >>> rs2[4:0]`

SRL

- **Format:** R
- **Opcode:** `0x33`
- **Funct3:** `0x5`
- **Funct7:** `0x00`
- **Operands:** `rd, rs1, rs2`
- **Description:** Logical right shift by lower 5 bits of rs2
- **Operation:** `rd = rs1 >> rs2[4:0]`

SLL

- **Format:** R
- **Opcode:** `0x33`
- **Funct3:** `0x1`
- **Funct7:** `0x00`
- **Operands:** `rd, rs1, rs2`
- **Description:** Logical left shift by lower 5 bits of rs2
- **Operation:** `rd = rs1 << rs2[4:0]`

SRLI

- **Format:** I

- **Opcode:** `0x13`
- **Funct3:** `0x5`
- **Operands:** `rd, rs1, shamt5`
- **Description:** Logical right shift by immediate shift amount
- **Operation:** `rd = rs1 >> shamt`

FENCEI

- **Format:** I
- **Opcode:** `0x0F`
- **Funct3:** `0x1`
- **Operands:** –
- **Description:** Instruction fence — synchronizes instruction and data streams
- **Operation:** `Flushes instruction cache after data writes`

FENCE

- **Format:** I
- **Opcode:** `0x0F`
- **Funct3:** `0x0`
- **Operands:** `pred, succ`
- **Description:** Memory ordering fence
- **Operation:** `Orders memory accesses as specified by pred and succ fields`

CSRRCI

- **Format:** I
- **Opcode:** `0x73`
- **Funct3:** `0x7`
- **Operands:** `rd, csr, uimm5`
- **Description:** Atomic read and clear bits in CSR with immediate
- **Operation:** `rd = CSR[csr]; CSR[csr] &= ~zimm`

ECALL

- **Format:** I
- **Opcode:** `0x73`
- **Funct3:** `0x0`
- **Operands:** –
- **Description:** Environment call — raises a system call exception
- **Operation:** `Traps to the configured exception handler`

EBREAK

- **Format:** I
- **Opcode:** `0x73`
- **Funct3:** `0x0`
- **Operands:** –
- **Description:** Breakpoint — raises a breakpoint exception
- **Operation:** `Used by debuggers to halt program execution`

CSRRW

- **Format:** I
- **Opcode:** `0x73`
- **Funct3:** `0x1`
- **Operands:** `rd, csr, rs1`
- **Description:** Atomic read/write CSR — writes rs1 to CSR, returns old value in rd
- **Operation:** `rd = CSR[csr]; CSR[csr] = rs1`

CSRRS

- **Format:** I
- **Opcode:** `0x73`
- **Funct3:** `0x2`
- **Operands:** `rd, csr, rs1`
- **Description:** Atomic read and set bits in CSR
- **Operation:** `rd = CSR[csr]; CSR[csr] |= rs1`

CSRRC

- **Format:** I
- **Opcode:** `0x73`
- **Funct3:** `0x3`
- **Operands:** `rd, csr, rs1`
- **Description:** Atomic read and clear bits in CSR
- **Operation:** `rd = CSR[csr]; CSR[csr] &= ~rs1`

CSRRWI

- **Format:** I
- **Opcode:** `0x73`
- **Funct3:** `0x5`
- **Operands:** `rd, csr, uimm5`
- **Description:** Atomic read/write CSR with immediate

- **Operation:** `rd = CSR[csr]; CSR[csr] = zimm`

CSRRSI

- **Format:** I
- **Opcode:** `0x73`
- **Funct3:** `0x6`
- **Operands:** `rd, csr, uimm5`
- **Description:** Atomic read and set bits in CSR with immediate
- **Operation:** `rd = CSR[csr]; CSR[csr] |= zimm`

Opcode Map

The following table shows the top-level opcode mapping. The opcode occupies bits 6:0 of every instruction.

opcode[6:0]	Type	Instructions
`0x33` (51)	R	ADD, SUB, SLT, SLTU, AND, OR, XOR, SLL, SRL, SRA (10 inst)s
`0x13` (19)	I	ADDI, SLTI, SLTIU, ANDI, ORI, XORI, SLLI, SRLI, SRAI (9 inst)s
`0x37` (55)	U	LUI (1 inst)
`0x17` (23)	U	AUIPC (1 inst)
`0x03` (3)	I	LB, LH, LW, LBU, LHU (5 inst)s
`0x23` (35)	S	SB, SH, SW (3 inst)s
`0x63` (99)	B	BEQ, BNE, BLT, BGE, BLTU, BGEU (6 inst)s
`0x6F` (111)	J	JAL (1 inst)
`0x67` (103)	I	JALR (1 inst)
`0x0F` (15)	I	FENCE, FENCEI (2 inst)s
`0x73` (115)	I	ECALL, EBREAK, CSRRW, CSRRS, CSRRC, CSRRWI, CSRRSI, CSRRCI (8 inst)s

Encoding Matrix

opcode funct3	0	1	2	3	4	5	6	7
0x33	ADD	SLL	SLT	SLTU	XOR	SRL	OR	AND
0x13	ADDI	SLLI	SLTI	SLTIU	XORI	SRLI	ORI	ANDI
0x37	—	—	—	—	—	—	—	—
0x17	—	—	—	—	—	—	—	—
0x03	LB	LH	LW	—	LBU	LHU	—	—
0x23	SB	SH	SW	—	—	—	—	—
0x63	BEQ	BNE	—	—	BLT	BGE	BLTU	BGEU
0x6F	—	—	—	—	—	—	—	—
0x67	JALR	—	—	—	—	—	—	—
0x0F	FENCE	FENCEI	—	—	—	—	—	—
0x73	ECALL	CSRRW	CSRRS	CSRRC	—	CSRRWI	CSRRSI	CSRRCI

Exception and Trap Handling

Exception Types

Exception Code	Name	Description
0	Instruction address misaligned	PC not aligned to 4 bytes
1	Instruction access fault	Failed to fetch instruction
2	Illegal instruction	Unrecognized opcode
3	Breakpoint	EBREAK instruction executed
4	Load address misaligned	Load address not properly aligned
5	Load access fault	Failed to load from memory
6	Store address misaligned	Store address not properly aligned
7	Store access fault	Failed to store to memory

Exception Code	Name	Description
8	Environment call from U-mode	ECALL in user mode
9	Environment call from S-mode	ECALL in supervisor mode
11	Environment call from M-mode	ECALL in machine mode

Trap Vector

On taking a trap, the implementation:

1. Sets `mepc` to the PC of the trapping instruction (or next PC for ECALL/EBREAK)
2. Sets `mcause` to the exception code
3. Sets `mtval` to exception-specific information
4. Sets `mstatus.MPP` to the current privilege mode
5. Sets `mstatus.MPIE` to `mstatus.MIE`
6. Clears `mstatus.MIE`
7. Sets PC to `mtvec`

Pseudo-Instructions

Pseudo-instruction	Expansion	Description
`NOP`	`ADDI x0, x0, 0`	No operation
`MV rd, rs`	`ADDI rd, rs, 0`	Copy register
`NOT rd, rs`	`XORI rd, rs, -1`	Bitwise NOT
`NEG rd, rs`	`SUB rd, x0, rs`	Negate register
`LI rd, imm`	(multiple)	Load immediate
`LA rd, label`	(multiple)	Load address
`RET`	`JALR x0, x1, 0`	Return from subroutine
`CALL label`	(multiple)	Call subroutine
`J label`	`JAL x0, label`	Unconditional jump
`JR rs`	`JALR x0, rs, 0`	Jump register

Control and Status Registers

The following CSRs are accessible via the `CSRRW`, `CSRRS`, `CSRRC`, and their immediate variants.

Address	Name	Description
`0x300`	mstatus	Machine status register — holds global interrupt enable and privilege state
`0x301`	misa	Machine ISA register — encodes supported ISA extensions
`0x302`	medeleg	Machine exception delegation register
`0x303`	mideleg	Machine interrupt delegation register
`0x304`	mie	Machine interrupt-enable register
`0x305`	mtvec	Machine trap-handler base address
`0x306`	mcounteren	Machine counter enable register
`0x340`	mscratch	Scratch register for machine-mode trap handlers
`0x341`	mepc	Machine exception program counter — holds PC of trapping instruction
`0x342`	mcause	Machine trap cause — encodes exception or interrupt cause
`0x343`	mtval	Machine trap value — exception-specific information
`0x344`	mip	Machine interrupt pending register
`0x3A0`	pmpcfg0	Physical memory protection configuration 0
`0x3B0`	pmpaddr0	Physical memory protection address 0
`0x3B1`	pmpaddr1	

Address	Name	Description
		Physical memory protection address 1
0x3B2	pmpaddr2	Physical memory protection address 2
0x3B3	pmpaddr3	Physical memory protection address 3
0xB00	mcycle	Machine cycle counter — counts number of clock cycles
0xB02	minstret	Machine instructions-retired counter
0xB03	mhpmcounter3	Machine hardware performance counter 3
0xB04	mhpmcounter4	Machine hardware performance counter 4
0xB05	mhpmcounter5	Machine hardware performance counter 5
0x323	mhpmevent3	Machine hardware performance event selector 3
0x324	mhpmevent4	Machine hardware performance event selector 4
0x325	mhpmevent5	Machine hardware performance event selector 5
0xC00	ucycle	User-mode cycle counter read
0xC02	uinstret	User-mode instructions-retired read
0xC80	ucycleh	Upper 32 bits of user-mode cycle counter
0xC82	uinstreth	Upper 32 bits of user-mode instructions-retired counter
0x100	sstatus	Supervisor status register
0x104	sie	Supervisor interrupt-enable register
0x105	stvec	Supervisor trap-handler base address

Address	Name	Description
0x140	sscratch	Scratch register for supervisor-mode trap handlers
0x141	sepc	Supervisor exception program counter
0x142	scause	Supervisor trap cause register
0x143	stval	Supervisor trap value register
0x144	sip	Supervisor interrupt pending register
0x180	satp	Supervisor address translation and protection — controls page tables

Quick Reference

Instruction Encoding Quick Reference

R-Type

```
 31:27   | 26:25  | 24:20  | 19:15  | 14:12  | 11:7   | 6:0
 funct7 |  —     | rs2    | rs1    | funct3 | rd     |
opcode
   7    |  2     |   5    |   5    |   3    |   5    |   7
```

I-Type

```
 31:20   | 19:15  | 14:12  | 11:7   | 6:0
 imm[11:0]| rs1    | funct3 | rd     | opcode
   12    |   5    |   3    |   5    |   7
```

S-Type

```
 31:27  | 26:25  | 24:20  | 19:15  | 14:12  | 11:7   | 6:0
 imm[11:5]| —     | rs2    | rs1    | funct3 | imm[4:0]|
opcode
   7    |  2     |   5    |   5    |   3    |   5    |   7
```

B-Type

```
 31    | 30:27 | 26:25 | 24:20  | 19:15  | 14:12  | 11:8  |
7    | 6:0
 imm[12]| imm[10:5]| —  | rs2    | rs1    | funct3 |
imm[4:1]| imm[11]| opcode
   1   |   6   |  2    |   5    |   5    |   3    |   4
|  1   |   7
```

U-Type

```
 31:12      | 11:7   | 6:0
 imm[31:12] | rd     | opcode
   20       |   5    |   7
```

J-Type

```
 31    | 30:21  | 20   | 19:12     | 11:7   | 6:0
 imm[20]| imm[10:1]| imm[11]| imm[19:12]| rd     | opcode
   1   |   10   |  1   |   8       |   5    |   7
```