Task-3 RISC V Instruction types

➢ Introduction

RISC-V (Reduced Instruction Set Computer - V) is an open-source instruction set architecture (ISA) grounded in reduced instruction set computing principles. Its simplicity, modularity, and extensibility make it popular in academia, industry, and research.

To fully leverage RISC-V, understanding its instruction formats is essential. These formats define how instructions are structured, encoded, and executed, affecting everything from CPU design to compiler optimization.

➢ Understanding Instruction Formats

RISC-V uses six primary instruction types: R-type, I-type, S-type, B-type, U-type, and J-type. Each type has a unique structure tailored for specific operations.

➢ Key Fields in Instructions

RISC-V instructions are encoded in 32 bits, divided into distinct fields. Each field serves a specific purpose, such as identifying the type of instruction, defining the operation, or specifying registers and immediate values. Here's a detailed breakdown of these key fields:

1. Opcode Field (7 bits)

- **Purpose:** The opcode determines the basic category or class of an instruction (e.g., arithmetic, memory access, or control flow).
- **Size:** 7 bits (bit positions 0–6).
- **Example:**
  - 0110011 → Indicates an **R-type instruction** for register-based operations.
  - 0010011 → Indicates an **I-type instruction** for immediate operations.

2. Register Fields

RISC-V instructions specify registers involved in operations using these fields:

a. Destination Register (rd)

- **Purpose:** Specifies the register where the result of the operation will be stored.
- **Size:** 5 bits (bit positions 7–11).
- **Range:** x0 to x31 (32 general-purpose registers).
- **Example:**
  - rd = 00001 → Refers to register x1.

b. Source Registers (rs1 and rs2)

- **Purpose:** Specify the registers containing operands for the operation.

- **Size:**
  - rs1: 5 bits (bit positions 15–19).
  - rs2: 5 bits (bit positions 20–24).
- **Range:** x0 to x31.
- **Example:**
  - rs1 = 00010 → Refers to register x2.
  - rs2 = 00011 → Refers to register x3.

3. Immediate Field

The **immediate field** encodes constant values directly within an instruction.

- **Purpose:** Used for:
  - Immediate arithmetic (e.g., addi).
  - Memory addressing (e.g., lw, sw).
  - Branch and jump offsets (e.g., beq, jal).
- **Size and Placement:** Varies depending on the instruction type:
  - **I-Type:** 12 bits (bit positions 20–31).
  - **S-Type:** Split between bit positions 7–11 and 25–31.
  - **B-Type:** Split across multiple fields (bits 7, 8–11, 25–30, and 31).
  - **U-Type:** 20 bits (bit positions 12–31).
  - **J-Type:** 20 bits, split across several positions.
- **Sign Extension:** Immediate values are sign-extended to match the operand size.
- **Example:**
  - addi x5, x6, 10: The immediate value 10 is encoded in the instruction and added to the value in register x6.

4. Function Fields (func3 and func7)

The **function fields** refine the operation defined by the opcode.

a. func3 (3 bits)

- **Purpose:** Specifies the operation within the instruction type.
- **Size:** 3 bits (bit positions 12–14).
- **Example:**
  - func3 = 000 → Addition (add in R-type, addi in I-type).
  - func3 = 010 → Load/Store word (lw or sw).

b. func7 (7 bits)

- **Purpose:** Provides additional specificity for operations in R-type instructions.
- **Size:** 7 bits (bit positions 25–31).
- **Example:**
  - func7 = 0000000 → Regular addition (add).
  - func7 = 0100000 → Subtraction (sub).

5. Shamt Field (Shift Amount)

- **Purpose:** Specifies the number of positions to shift in shift instructions (e.g., slli, srli, srai).
- **Size:** 5 bits (subset of immediate field in I-type).
- **Example:**
  - slli x5, x6, 3 → Shift the value in x6 left by 3 bits and store the result in x5.

6. Offset/Target Field

For branch (B-Type) and jump (J-Type) instructions, the offset/target field determines the address of the next instruction to execute if the branch or jump is taken.

- **Purpose:** Specifies the relative or absolute address to jump to.
- **Size:** Varies by instruction type:
  - **B-Type:** 12 bits (split across bits 7, 8–11, 25–30, and 31).
  - **J-Type:** 20 bits (split across bits 12–19, 20, 21–30, and 31).

- **Sign Extension:** Offsets are sign-extended for compatibility with negative values.
- **Example:**
    - beq x1, x2, label: The offset to label is encoded, and the jump occurs if x1 == x2.

Example: Decoding an R-Type Instruction

Assembly Instruction:

add x5, x6, x7

Encoded Machine Code:

0000000 00111 00110 000 00101 0110011

Field Name	Bits	Value	Description
func7	25–31	0000000	Indicates addition.
rs2	20–24	00111	Source register 2 (x7).
rs1	15–19	00110	Source register 1 (x6).
func3	12–14	000	Addition operation.
rd	7–11	00101	Destination register (x5).
opcode	0–6	0110011	R-type instruction.

➢ Immediate field in detail

The Immediate Field in RISC-V instructions is a critical component that allows the encoding of constant values directly into instructions. These constants (or immediate values) can represent numbers, memory offsets, or branch/jump targets, depending on the instruction type.

Purpose of the Immediate Field

The immediate field is used in the following scenarios:

1. Immediate Arithmetic and Logical Operations:
    - Immediate values are combined with register values to perform operations without needing another source register.
    - Example:
        - addi x5, x6, 10 → Adds 10 (an immediate value) to the value in x6 and stores the result in x5.
2. Memory Addressing:
    - Used as an offset for memory operations like load (lw) and store (sw).
    - Example:
        - lw x7, 16(x8) → Loads the word from the memory address x8 + 16 into x7.
3. Branch and Jump Offsets:
    - Specifies the offset from the current program counter (PC) for control flow instructions like branch (beq, bne) and jump (jal).
    - Example:
        - beq x1, x2, label → If x1 == x2, jumps to the instruction at the offset defined by label.

Size and Placement of the Immediate Field

The size and exact placement of the immediate field vary depending on the instruction type:

1. I-Type Instructions
   - Size: 12 bits.
   - Placement: Bit positions 20–31.
   - Usage: For arithmetic, logical operations, and memory addressing.
   - Example: addi x5, x6, 10
     - Encodes the immediate value 10 (12 bits) in bits 20–31.
     - Adds 10 to the value in x6 and stores it in x5.
2. S-Type Instructions
   - Size: 12 bits (split into two parts).
   - Placement:
     - Lower 5 bits (imm[4:0]): Bit positions 7–11.
     - Upper 7 bits (imm[11:5]): Bit positions 25–31.
   - Usage: For memory store instructions like sw.
   - Example: sw x7, 16(x8)
     - Encodes 16 as the immediate value, split across bits 7–11 and 25–31.
3. B-Type Instructions
   - Size: 12 bits (split across multiple positions).
   - Placement:
     - Bit 11 (imm[11]): Bit position 7.
     - Bits 4–1 (imm[4:1]): Bit positions 8–11.
     - Bits 10–5 (imm[10:5]): Bit positions 25–30.
     - Bit 12 (imm[12]): Bit position 31.
   - Usage: For branch instructions like beq and bne.
   - Example: beq x1, x2, label
     - Encodes the offset to label in the immediate field, split into the specified positions.
4. U-Type Instructions
   - Size: 20 bits.
   - Placement: Bit positions 12–31.
   - Usage: For instructions like lui (Load Upper Immediate).
   - Example: lui x5, 0x12345
     - Encodes 0x12345 in the upper 20 bits, with the lower 12 bits set to zero.
5. J-Type Instructions
   - Size: 20 bits (split across multiple positions).
   - Placement:
     - Bit 20 (imm[20]): Bit position 31.
     - Bits 10–1 (imm[10:1]): Bit positions 21–30.
     - Bit 11 (imm[11]): Bit position 20.
     - Bits 19–12 (imm[19:12]): Bit positions 12–19.
   - Usage: For jump instructions like jal.
   - Example: jal x1, label
     - Encodes the offset to label in the immediate field, split into the specified positions.

Sign Extension

- Purpose: The immediate field often uses fewer bits than the register width (e.g., 12-bit immediate in a 32-bit system). To ensure proper computation with signed numbers, the immediate value is sign-extended.
- How It Works:
  - The most significant bit (MSB) of the immediate field is repeated to fill the higher-order bits of the 32-bit register.
  - For positive values, the MSB is 0 (e.g., 10 → 00000000...01010).
  - For negative values, the MSB is 1 (e.g., -10 → 11111111...10110).

➢ Func3 and fun7

func3 (3 bits) helps differentiate between operations that share the same opcode. **func7** (7 bits) provides additional distinction when necessary, especially for similar operations like add and sub.

1. R-Type Instructions

Used for register-to-register operations.

Instruction	Opcode	func3	func7	Description
add	0110011	000	0000000	Addition
sub	0110011	000	0100000	Subtraction
sll	0110011	001	0000000	Shift left logical
slt	0110011	010	0000000	Set less than (signed)
sltu	0110011	011	0000000	Set less than (unsigned)
xor	0110011	100	0000000	Bitwise XOR
srl	0110011	101	0000000	Shift right logical
sra	0110011	101	0100000	Shift right arithmetic
or	0110011	110	0000000	Bitwise OR
and	0110011	111	0000000	Bitwise AND

2. I-Type Instructions

Used for immediate arithmetic, logical operations, and loads.

- **Arithmetic and Logical Operations**

Instruction	Opcode	func3	Description
addi	0010011	000	Add immediate
slti	0010011	010	Set less than immediate (signed)
sltiu	0010011	011	Set less than immediate (unsigned)
xori	0010011	100	Bitwise XOR immediate
ori	0010011	110	Bitwise OR immediate
andi	0010011	111	Bitwise AND immediate
slli	0010011	001	Shift left logical immediate
srli	0010011	101	Shift right logical immediate

Instruction	Opcode	func3	Description
Srai	0010011	101	Shift right arithmetic immediate

For slli, srli, and srai, **func7** is also used:

Instruction	func7
slli	0000000
srli	0000000
srai	0100000

- **Load Instructions**

Instruction	Opcode	func3	Description
lb	0000011	000	Load byte
lh	0000011	001	Load halfword
lw	0000011	010	Load word
ld	0000011	011	Load double word
lbu	0000011	100	Load byte unsigned
lhu	0000011	101	Load halfword unsigned

3. S-Type Instructions

Used for store operations.

Instruction	Opcode	func3	Description
sb	0100011	000	Store byte
sh	0100011	001	Store halfword
sw	0100011	010	Store word

4. B-Type Instructions

Used for conditional branches.

Instruction	Opcode	func3	Description
beq	1100011	000	Branch if equal
bne	1100011	001	Branch if not equal
blt	1100011	100	Branch if less than (signed)
bge	1100011	101	Branch if greater or equal (signed)

Instruction	Opcode	func3	Description
bltu	1100011	110	Branch if less than (unsigned)
bgeu	1100011	111	Branch if greater or equal (unsigned)

5. U-Type Instructions

Used for upper immediate operations.

Instruction	Opcode	func3	Description
lui	0110111	N/A	Load upper immediate
auipc	0010111	N/A	Add upper immediate to PC

6. J-Type Instructions

Used for unconditional jumps.

Instruction	Opcode	func3	Description
jal	1101111	N/A	Jump and link

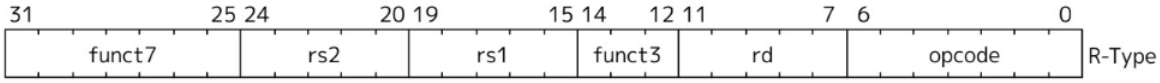
7. System Instructions

Used for system calls and environment handling.

Instruction	Opcode	func3	func7	Description
ecall	1110011	000	0000000	Environment call
ebreak	1110011	000	0000000	Environment break

➢ Instruction types

1. R-Type (Register Type) Instructions



- Purpose: Perform operations between two registers and store the result in a third register.
- Fields:
  - opcode: Identifies the instruction category.
  - rd: Destination register.
  - func3: Differentiates operations within the same category.
  - rs1, rs2: Source registers.
  - func7: Further differentiates operations.
- Example Instructions:

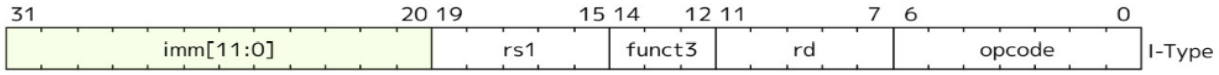
- add, sub: Arithmetic operations.
- and, or, xor: Bitwise operations.
- sll, srl, sra: Shift operations.

- Encoding Example:

add x1, x2, x3 // x1 = x2 + x3

- Encoded in 32 bits with the appropriate opcode, func3, and func7 values.

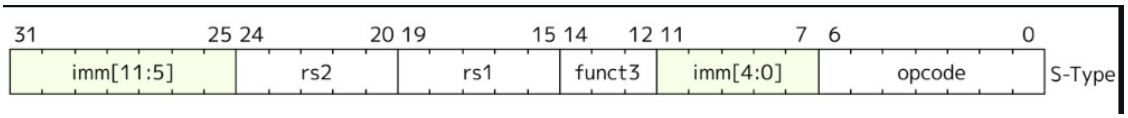
2. I-Type (Immediate Type) Instructions



- Purpose: Perform operations using an immediate (constant) value and a register.
- Fields:
  - opcode: Identifies the instruction category.
  - rd: Destination register.
  - func3: Differentiates operations.
  - rs1: Source register.
  - imm[11:0]: 12-bit immediate value.
- Example Instructions:
  - addi: Add immediate value to a register.
  - andi, ori: Perform bitwise operations with an immediate value.
  - lw: Load word from memory.
- Encoding Example:

addi x1, x2, 10 // x1 = x2 + 10

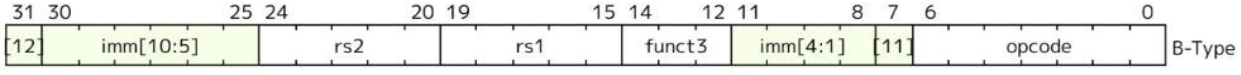
3. S-Type (Store Type) Instructions



- Purpose: Store data from a register into memory.
- Fields:
  - opcode: Identifies the instruction category.
  - func3: Differentiates store operations (e.g., byte, word).
  - rs1: Base address register.
  - rs2: Source register (contains the data to store).
  - imm[11:0]: 12-bit immediate value (split into two parts in encoding).
- Example Instructions:
  - sb: Store byte.
  - sh: Store halfword.
  - sw: Store word.
- Encoding Example:

sw x1, 8(x2) // Store the value of x1 at the address x2 + 8

4. B-Type (Branch Type) Instructions



- Purpose: Perform conditional branching based on comparisons.
- Fields:
  - opcode: Identifies the instruction category.
  - func3: Differentiates branch conditions (e.g., equal, less than).

- rs1, rs2: Registers to compare.
- imm[12:1]: 12-bit immediate value specifying the branch offset.

- Example Instructions:
  - beq: Branch if equal.
  - bne: Branch if not equal.
  - blt, bge: Branch on less-than/greater-than conditions.
- Encoding Example:

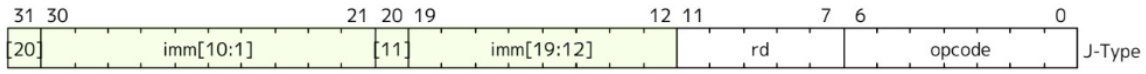
```
beq x1, x2, 16  // Branch to PC + 16 if x1 == x2
```

5. U-Type (Upper Immediate Type) Instructions

- Purpose: Load or manipulate the upper 20 bits of a register.
- Fields:
  - opcode: Identifies the instruction category.
  - rd: Destination register.
  - imm[31:12]: 20-bit immediate value.
- Example Instructions:
  - lui: Load upper immediate.
  - auipc: Add upper immediate to the program counter.
- Encoding Example:

```
lui x1, 0x12345  // x1 = 0x12345000
```

6. J-Type (Jump Type) Instructions



- Purpose: Perform unconditional jumps with a link (saving the return address in a register).
- Fields:
  - opcode: Identifies the instruction category.
  - rd: Destination register (stores return address).
  - imm[20:1]: 20-bit immediate value specifying the jump offset.
- Example Instructions:
  - jal: Jump and link.
- Encoding Example:

```
jal x1, 100  // Jump to PC + 100 and save return address in x1
```

➢ RISC-V Register Names and Numeric Indices

RISC-V registers are divided into categories based on their use, with both a name and an index:

Register Name	Description	Index
x0	**Zero**	**0**
ra	**Return Address**	**1**
sp	**Stack Pointer**	**2**
gp	**Global Pointer**	**3**
tp	**Thread Pointer**	**4**
t0 to t6	**Temporary Registers**	**5–7, 28–31**

Register Name	Description	Index
s0/fp	Saved Register / Frame Pointer	8
s1 to s11	Saved Registers	9–18
a0 to a7	Argument Registers	10–17

How to Map Register Names

1. **Identify the Register Name: The instruction specifies a human-readable register name like a0, sp, or s0.**
2. **Convert the Register Name to Index: Use the table above to find the corresponding index:**
    - **a0 → x10**
    - **sp → x2**
    - **s0 → x8**
3. **Convert the Index to Binary: Registers are encoded as 5-bit binary values in machine code:**
    - **a0 → x10 → 01010 (5 bits)**
    - **sp → x2 → 00010 (5 bits)**
    - **s0 → x8 → 01000 (5 bits)**

Examples of Register Encoding

Example 1: addi a0, sp, 384

- **rd = a0 → x10 → 01010**
- **rs1 = sp → x2 → 00010**

Example 2: ld a0, -32(s0)

- **rd = a0 → x10 → 01010**
- **rs1 = s0 → x8 → 01000**

Example 3: sd ra, 408(sp)

- **rs1 = sp → x2 → 00010**
- **rs2 = ra → x1 → 00001**

Document By : Sarvamangala B

Department of Electronics and Communication

RV institute of technology and management