

The "**I-type**" instruction format in RV32I is a crucial category used for operations that involve a register and an **immediate (constant)** value, as well as for load instructions and the JALR (Jump and Link Register) instruction.

Here's a detailed explanation of I-type instructions in RV32I:

### 1. Purpose:

I-type instructions serve several key purposes in RV32I:

- **Immediate Arithmetic/Logical Operations:** Performing arithmetic or bitwise logical operations between a source register and a small, signed immediate value.
- **Load Instructions:** Loading data from memory into a register. The immediate value is used as an offset from a base address in a register.
- **Jump and Link Register (JALR):** Used for indirect jumps, often for function returns or switch statements.

### Key Characteristics of I-type Format:

- **12-bit Signed Immediate:** The immediate value is 12 bits wide and is always sign-extended to 32 bits before being used in an operation. This allows for immediate values in the range of -2048 to +2047.
- **rs1 and rd fields:** The source register (rs1) and destination register (rd) fields are in the same fixed positions as in R-type instructions, simplifying decoding.
- **funct3:** Similar to R-type, funct3 helps to differentiate between various I-type operations that share the same opcode.

### 3. Common I-type Instructions in RV32I:

Here are some prominent examples of I-type instructions:

- **Immediate Arithmetic/Logical Operations (Opcode: OP-IMM):**

- **ADDI rd, rs1, imm:** Adds the signed immediate value imm to the contents of rs1, stores the result in rd.
  - *Example:* addi x10, x11, 100 ( $x10 = x11 + 100$ )
- **SLTI rd, rs1, imm:** Sets rd to 1 if rs1 (signed) is less than the signed imm, otherwise sets rd to 0.
  - *Example:* slti x10, x11, -50
- **SLTIU rd, rs1, imm:** Sets rd to 1 if rs1 (unsigned) is less than the unsigned imm (after sign-extension, which can be tricky for unsigned comparisons), otherwise sets rd to 0. *Note: For unsigned comparison with immediate, the immediate is treated as a signed 12-bit number, then zero-extended for the comparison if rs1 is unsigned. This can sometimes lead to non-intuitive results if the immediate is negative.*
  - *Example:* sltiu x10, x11, 200
- **ANDI rd, rs1, imm:** Performs a bitwise AND operation between rs1 and the sign-extended imm, stores the result in rd.
  - *Example:* andi x10, x11, 0xFF
- **ORI rd, rs1, imm:** Performs a bitwise OR operation between rs1 and the sign-extended imm, stores the result in rd.
  - *Example:* ori x10, x11, 0x1000
- **XORI rd, rs1, imm:** Performs a bitwise XOR operation between rs1 and the sign-extended imm, stores the result in rd.
  - *Example:* xori x10, x11, 0xFFFF
- **SLLI rd, rs1, shamt:** Logically shifts rs1 left by shamt (shift amount, derived from the immediate), stores the result in rd. For 32-bit RV32I, shamt is restricted to the lower 5 bits of the

immediate.

- *Example:* slli x10, x11, 4
- **SRLI rd, rs1, shamt:** Logically shifts rs1 right by shamt, stores the result in rd.
  - *Example:* srlx x10, x11, 2
- **SRAI rd, rs1, shamt:** Arithmetically shifts rs1 right by shamt, stores the result in rd.
  - *Example:* srai x10, x11, 1
- **Load Instructions (Opcode: LOAD):**
  - **LB rd, imm(rs1):** Loads a signed byte from memory at address rs1 + imm, sign-extends it, and stores it in rd.
    - *Example:* lb x10, 4(x11) (Load byte from address x11 + 4 into x10)
  - **LH rd, imm(rs1):** Loads a signed half-word (16-bit) from memory at address rs1 + imm, sign-extends it, and stores it in rd.
    - *Example:* lh x10, 8(x11)
  - **LW rd, imm(rs1):** Loads a word (32-bit) from memory at address rs1 + imm, stores it in rd.
    - *Example:* lw x10, 0(x11)
  - **LBU rd, imm(rs1):** Loads an unsigned byte from memory at address rs1 + imm, zero-extends it, and stores it in rd.
    - *Example:* lbu x10, 1(x11)
  - **LHU rd, imm(rs1):** Loads an unsigned half-word from memory at address rs1 + imm, zero-extends it, and stores it in rd.
    - *Example:* lhu x10, 6(x11)
- **Jump and Link Register (Opcode: JALR):**
  - **JALR rd, rs1, imm:** Jumps to the address (rs1 + imm) and sets the return address (PC + 4) into rd. The effective jump target address has its least significant bit (LSB) set to 0. This is typically used for function returns (jalr ra, x0, 0) or indirect jumps/calls.
    - *Example:* jalr x1, x10, 0 (Jump to x10, store PC+4 in x1, typical return for ret instruction)

## Common U-type Instructions in RV32I:

There are two primary U-type instructions in RV32I:

- **LUI rd, imm (Load Upper Immediate)**
  - **Operation:** Loads the 20-bit imm into bits 31-12 of rd, and sets bits 11-0 of rd to zero.
  - **Pseudocode:**  $rd = imm \ll 12$
  - **Example:** lui x10, 0x12345
    - This would place 0x12345 into the upper 20 bits of x10, resulting in x10 = 0x12345000.

## AUIPC rd, imm (Add Upper Immediate to PC)

- **Operation:** Adds the 20-bit imm (shifted left by 12) to the current Program Counter (PC), and stores the result in rd.
- **Pseudocode:**  $rd = PC + (imm \ll 12)$
- **Example:** auipc x10, 0x10000
  - If PC is 0x80000000, then x10 would become  $0x80000000 + (0x10000 \ll 12)$  which is  $0x80000000 + 0x10000000 = 0x90000000$ .

## B-type instructions

### Key Characteristics of B-type Format:

- **12-bit Signed Immediate:** The immediate value represents a **signed offset** from the current PC. This offset is **multiplied by 2** (meaning the actual jump target is  $\text{PC} + (\text{immediate} \times 2)$ ) because all instructions are 4-byte aligned, so the least significant bit (LSB) of a branch target address is always 0. By omitting the LSB (which is always 0), the ISA gains an extra bit of range for the branch offset, effectively allowing a 13-bit offset using a 12-bit immediate field. The range is approximately  $\pm 4\text{KB}$  (since the immediate is multiplied by 2).
- **PC-Relative Addressing:** Branch targets are always calculated relative to the PC (Program Counter). Specifically,  $\text{PC} + \text{sign\_extended(immediate)}$ .
- **rs1 and rs2 fields:** These fields specify the two registers whose values will be compared.
- **funct3:** This field determines the specific comparison type (e.g., equal, not equal, less than, greater than or equal).

### 3. Common B-type Instructions in RV32I:

Here are the six conditional branch instructions in RV32I:

- **BEQ rs1, rs2, label (Branch if Equal)**
  - **Operation:** If the value in rs1 is equal to the value in rs2, branch to label.
  - **Pseudocode:**  $\text{if } (\text{rs1} == \text{rs2}) \text{ PC} = \text{PC} + (\text{signed\_imm} \ll 1)$
  - **Example:** `beq x5, x6, loop_start`
    - If x5 equals x6, jump to the instruction at `loop_start`.
- **BNE rs1, rs2, label (Branch if Not Equal)**
  - **Operation:** If the value in rs1 is not equal to the value in rs2, branch to label.
  - **Pseudocode:**  $\text{if } (\text{rs1} != \text{rs2}) \text{ PC} = \text{PC} + (\text{signed\_imm} \ll 1)$
  - **Example:** `bne x7, x8, end_program`
    - If x7 is not equal to x8, jump to the instruction at `end_program`.
- **BLT rs1, rs2, label (Branch if Less Than - Signed)**
  - **Operation:** If rs1 (signed) is less than rs2 (signed), branch to label.
  - **Pseudocode:**  $\text{if } (\text{signed(rs1)} < \text{signed(rs2)}) \text{ PC} = \text{PC} + (\text{signed\_imm} \ll 1)$
  - **Example:** `blt x10, x11, negative_val`
    - If x10 is signed less than x11, jump to `negative_val`.
- **BGE rs1, rs2, label (Branch if Greater Than or Equal - Signed)**
  - **Operation:** If rs1 (signed) is greater than or equal to rs2 (signed), branch to label.
  - **Pseudocode:**  $\text{if } (\text{signed(rs1)} \geq \text{signed(rs2)}) \text{ PC} = \text{PC} + (\text{signed\_imm} \ll 1)$
  - **Example:** `bge x10, x0, positive_val` (This is a common way to check if a number in x10 is positive or zero)
    - If x10 is signed greater than or equal to x0 (zero), jump to `positive_val`.
- **BLTU rs1, rs2, label (Branch if Less Than - Unsigned)**
  - **Operation:** If rs1 (unsigned) is less than rs2 (unsigned), branch to label.
  - **Pseudocode:**  $\text{if } (\text{unsigned(rs1)} < \text{unsigned(rs2)}) \text{ PC} = \text{PC} + (\text{signed\_imm} \ll 1)$
  - **Example:** `bltu x12, x13, smaller_unsigned`
    - If x12 is unsigned less than x13, jump to `smaller_unsigned`.
- **BGEU rs1, rs2, label (Branch if Greater Than or Equal - Unsigned)**
  - **Operation:** If rs1 (unsigned) is greater than or equal to rs2 (unsigned), branch to label.
  - **Pseudocode:**  $\text{if } (\text{unsigned(rs1)} \geq \text{unsigned(rs2)}) \text{ PC} = \text{PC} + (\text{signed\_imm} \ll 1)$
  - **Example:** `bgeu x14, x15, larger_unsigned`
    - If x14 is unsigned greater than or equal to x15, jump to `larger_unsigned`.

## Key Characteristics of S-type Format:

- **12-bit Signed Immediate:** The imm value (reconstructed from imm[11:5] and imm[4:0]) is a 12-bit signed offset, which is added to the contents of rs1 to form the effective memory address. The range is -2048 to +2047 bytes.
- **Base + Offset Addressing:** Memory addresses are calculated by adding a signed immediate offset to a base address held in rs1.
- **rs2 as Source:** Unlike load instructions where rd is the destination, in store instructions, rs2 specifies the register whose *value* is to be written to memory.

### 3. Common S-type Instructions in RV32I:

There are three primary store instructions in RV32I, differing by the size of the data being stored:

- **SB rs2, imm(rs1) (Store Byte)**
  - **Operation:** Stores the least significant byte (8 bits) of the value in rs2 to the memory address calculated as  $rs1 + \text{sign\_extended}(\text{imm})$ .
  - **Pseudocode:**  $\text{Memory}[rs1 + \text{sign\_extended}(\text{imm})] = rs2[7:0]$
  - **Example:** `sb x10, 0(x11)`
    - Stores the lower byte of register x10 into the memory location pointed to by x11.
- **SH rs2, imm(rs1) (Store Half-word)**
  - **Operation:** Stores the least significant half-word (16 bits) of the value in rs2 to the memory address calculated as  $rs1 + \text{sign\_extended}(\text{imm})$ . The address must be half-word aligned (i.e., address  $\% 2 == 0$ ).
  - **Pseudocode:**  $\text{Memory}[rs1 + \text{sign\_extended}(\text{imm})] = rs2[15:0]$
  - **Example:** `sh x10, 2(x11)`
    - Stores the lower half-word of register x10 into the memory location pointed to by x11 + 2.
- **SW rs2, imm(rs1) (Store Word)**
  - **Operation:** Stores the entire word (32 bits) from rs2 to the memory address calculated as  $rs1 + \text{sign\_extended}(\text{imm})$ . The address must be word-aligned (i.e., address  $\% 4 == 0$ ).
  - **Pseudocode:**  $\text{Memory}[rs1 + \text{sign\_extended}(\text{imm})] = rs2[31:0]$
  - **Example:** `sw x10, 4(x11)`
    - Stores the full 32-bit value of register x10 into the memory location pointed to by x11 + 4.