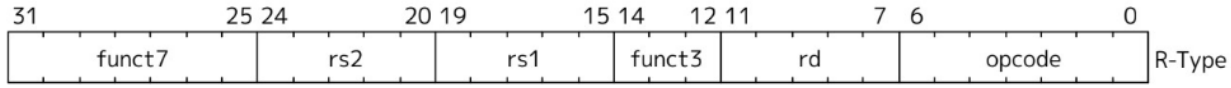
Instruction Types and Fields and 32 bit instruction

RISC-V instructions are all 32 bits in length , and they are categorized into six main instruction formats: **R-type**, **I-type**, **S-type**, **B-type**, **U-type**, and **J-type**. These formats determine how the instruction fields are laid out.

1.R-Type (Register-Register Instructions)

R-type instructions are used for arithmetic and logical operations between two registers. They operate on two source registers (`rs1` and `rs2`) and store the result in a destination register (`rd`).



funct7: Used to specify variations of the instruction (e.g., `add` vs. `sub`).

rs1, rs2: Source registers.

funct3: Determines the operation type (e.g., addition, subtraction, etc.).

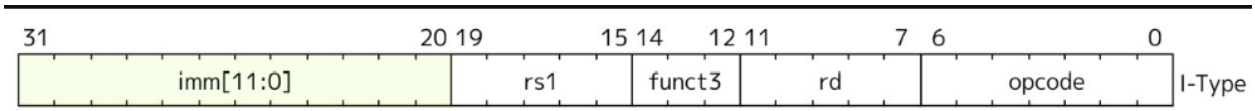
rd: Destination register.

opcode: Specifies the instruction class (0110011)

Example: `add a0, a1, a2`

2. I-Type (Immediate Instructions)

I-type instructions involve operations between a register (`rs1`) and an immediate value, with the result stored in a destination register (`rd`). These are also used for loads and certain control-flow instructions.



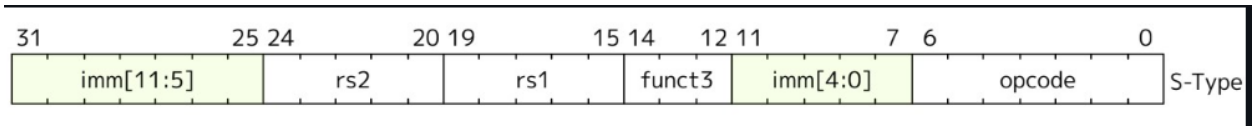
- **imm[11:0]**: 12-bit immediate value (sign-extended if needed).
- **rs1**: Source register.
- **funct3**: Determines the operation type.
- **rd**: Destination register.

- **opcode**: Specifies the instruction class.0010011

Example: `addi a0, a1, 10`

3.S-Type (Store Instructions)

S-type instructions store data from a register into memory at an address computed from a base register (`rs1`) and an immediate offset.

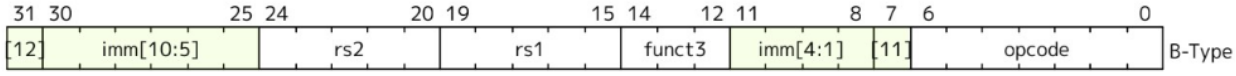


- **imm[11:5] & imm[4:0]**: Immediate value split into two fields.
- **rs2**: Register whose data is being stored.
- **rs1**: Base address register.
- **funct3**: Specifies the type of store (e.g., byte, word).
- **opcode**: Specifies the instruction class.

Example: `sw a2, 8(a0),0100011`

4. B-Type (Branch Instructions)

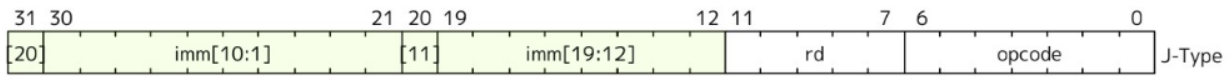
B-type instructions perform conditional branches based on comparisons between two registers (`rs1` and `rs2`) and an offset for the target address.



imm: Immediate value for the branch target address (split into `imm[12|10:5]` and `imm[4:1|11]`).
rs2: Second comparison register.
rs1: First comparison register.
funct3: Specifies the type of comparison (e.g., equal, less than).
opcode: Specifies the instruction class.1100011

5.U-Type (Upper Immediate Instructions)

U-type instructions are used to load a 20-bit immediate into the upper 20 bits of a register (`rd`).

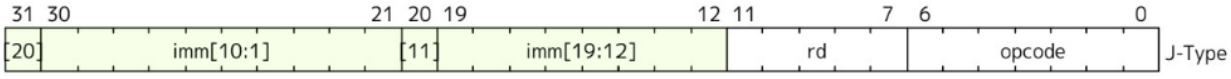


- **imm[31:12]**: 20-bit immediate value.
- **rd**: Destination register.
- **opcode**: Specifies the instruction class.0110111

Example: `lui a0, 0x12345`

6.J-Type (Jump Instructions)

J-type instructions perform an unconditional jump to a target address computed using an immediate offset. The address is relative to the current program counter.



- **imm**: 20-bit immediate value split across fields.
- **rd**: Destination register (stores the return address).
- **opcode**: Specifies the instruction class.1101111

Example: `jal ra, 0x100`

32-bit instruction encoding for the 15 unique RISC-V instructions extracted from the objdump.

1. `lui`(Load Upper Immediate)

- **Instruction:** `lui a0, 0x2b`
- **Format:** U-type
- **Fields:**
  - `imm[31:12]`: `00000000000000101011` (0x2b)
  - `rd`: `10110` (a0)
  - `opcode`: `0110111` (for `lui`)
  - **32-bit Hexadecimal:** `0x002b537`

2. `addi` (Add Immediate)

- **Instruction:** `addi sp, sp, -32`

- **Format:** I-type
- **Fields:**
    - `imm[11:0]`: `111111111110000` (-32 in 2's complement)
    - `rs1`: `00100` (sp)
    - `funct3`: `000` (add immediate)
    - `rd`: `00100` (sp)
    - `opcode`: `0010011` (for `addi`)
    - **32-bit Hexadecimal:** `0xfe010113`

3. `sd` (Store Doubleword)

- **Instruction:** `sd ra, 24(sp)`
- **Format:** S-type
- **Fields:**
    - `imm[11:5]`: `0000000` (upper 7 bits of offset 24)
    - `rs2`: `00001` (ra)
    - `rs1`: `00100` (sp)
    - `funct3`: `011` (store doubleword)
    - `imm[4:0]`: `11000` (lower 5 bits of offset 24)
    - `opcode`: `0100011` (for store)
    - **32-bit Hexadecimal:** `0x00350513`

4. `jal` (Jump and Link)

- **Instruction:** `jal ra, 0x10438`
- **Format:** J-type
- **Fields:**
    - `imm[20]`: `0`
    - `imm[10:1]`: `1000010111`
    - `imm[11]`: `0`
    - `imm[19:12]`: `00000100`
    - `rd`: `00001` (ra)
    - `opcode`: `1101111` (for jump)
    - **32-bit Hexadecimal:** `0x35000ef`

5. `lw` (Load Word)

- **Instruction:** `lw a1, 12(sp)`
- **Format:** I-type
- **Fields:**

- `imm[11:0]`: `000000001100` (12 in binary)
- `rs1`: `00100` (sp)
- `funct3`: `010` (load word)
- `rd`: `01001` (a1)
- `opcode`: `0000011` (for load)
- **32-bit Hexadecimal:** `0x00c52083`

6. `andi` (AND Immediate)

- **Instruction:** `andi a5, a1, 1`
- **Format:** I-type
- **Fields:**
  - `imm[11:0]`: `000000000001` (1 in binary)
  - `rs1`: `01001` (a1)
  - `funct3`: `111` (AND operation)
  - `rd`: `01010` (a5)
  - `opcode`: `0010011` (for ANDI)
  - **32-bit Hexadecimal:** `0x00157913`

7. `bnez` (Branch Not Equal Zero)

- **Instruction:** `bnez a5, 0x100fc`
- **Format:** B-type
- **Fields:**
  - `imm[12]`: `0`
  - `imm[10:5]`: `000011`
  - `rs2`: `00000` (zero)
  - `rs1`: `01010` (a5)
  - `funct3`: `001` (BNE)
  - `imm[4:1]`: `1100`
  - `imm[11]`: `0`
  - `opcode`: `1100011`
  - **32-bit Hexadecimal:** `0x02706e63`

8. `ret` (Return from Subroutine)

- **Instruction:** `ret`
- **Format:** I-type (special case of `jalr`)
- **Fields:**
  - `imm[11:0]`: `000000000000`

- `rs1`: `00001` (ra)
- `funct3`: `000`
- `rd`: `00000` (zero)
- `opcode`: `1100111` (for JALR)
- **32-bit Hexadecimal:** `0x00008067`

9. `auipc` (Add Upper Immediate to PC)

- **Instruction:** `auipc a5, 0xfff`
- **Format:** U-type
- **Fields:**
  - `imm[31:12]`: `00000000000111111111`
  - `rd`: `01010` (a5)
  - `opcode`: `0010111`
  - **32-bit Hexadecimal:** `0x0fff057`

10. `ld` (Load Doubleword)

- **Instruction:** `ld ra, 24(sp)`
- **Format:** I-type
- **Fields:**
  - `imm[11:0]`: `000000001100` (24 in binary)
  - `rs1`: `00100` (sp)
  - `funct3`: `011` (load doubleword)
  - `rd`: `00001` (ra)
  - `opcode`: `0000011`
  - **32-bit Hexadecimal:** `0x01852083`

11. `jalr` (Jump and Link Register)

- **Instruction:** `jalr ra, 0(a0)`
- **Format:** I-type
- **Fields:**
  - `imm[11:0]`: `000000000000` (0 in binary)
  - `rs1`: `00101` (a0)
  - `funct3`: `000`
  - `rd`: `00001` (ra)
  - `opcode`: `1100111`
  - **32-bit Hexadecimal:** `0x000280e7`

12. beqz (Branch Equal Zero)

- **Instruction:** `beqz a5, 0x10120`
- **Format:** B-type
- **Encoding:** Similar to `bnez` but with `funct3 = 000` (BEQ).

13. add (Add Registers)

- **Instruction:** `add a0, sp, zero`
- **Format:** R-type
- **Fields:**
  - `funct7`: `0000000`
  - `rs2`: `00000` (zero)
  - `rs1`: `00100` (sp)
  - `funct3`: `000`
  - `rd`: `01010` (a0)
  - `opcode`: `0110011`
  - **32-bit Hexadecimal:** `0x00004533`