VSDSquadron Mini Internship 2025

by

VLSI System Design

Documented By: Chandrakiran G
St. Joseph's Institute of Technology
Chennai-600119
Department of Electronics and Communication Engineering

WHAT IS RISC-V?

- RISC-V is an open-source instruction set architecture (ISA) that allows developers to processors for specific applications.
- RISC-V is based on reduced instruction set computer principles and is the fifth generation of processors built on this concept.
- RISC-V can also be understood as an alternative processor technology which is free and open, meaning that it does not require you to purchase the license of RISC-V to use it.

INSTRUCTIONS FORMAT IN RISC-V

The instructions format of a processor is the way in which machine language instructions are structured and organized for a processor to execute. It is made up of series of 0s and 1s, each containing information about the location and operation of data.

There are 6 instruction formats in RISC-V:

1. R-format
2. I-format
3. S-format
4. B-format
5. U-format
6. J-format

Let's discuss each of the instruction formats in detail with examples.

R-type Instruction

In RV32, each instruction is of size 32 bits. In R-type instruction, R stands for register which means that operations are carried on the Registers and not on memory location. This instruction

type is used to execute various arithmetic and logical operations. The entire 32 bits instruction is divided into 6 fields as shown below.

- The first field in the instruction format is known as **opcode,** also referred as operation code. The opcode is of length 7 bits and is used to determine the type of instruction format.
- The next subfield is known as **rd** field which is referred as Destination Register. The rd field is of length 5 bits and is used to store the final result of operation.
- The next subfield is **func3** also referred as function 3. Here the '3' represents the size of this field. This field tells the detail about the operation, i.e., the type of arithmetic and logical that is performed.
- The next two subfields are the source registers, **rs1 and rs2** each of length 5 bits. These are mainly used to store and manipulate the data during the execution of instructions.
- The last subfield is **func7** also referred as function 7. Here '7' represents the size of the field. The function of func7 field is same as that of func3 field.

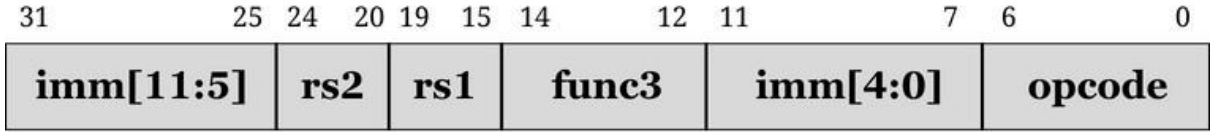
I-type Instruction

In RV32, each instruction is of size 32 bits. In I-type instruction, I stand for immediate which means that operations use Registers and Immediate value for their execution and are not related with memory location. This instruction type is used in immediate and load operations. The entire 32 bits instruction is divided into 5 fields as shown below.

- The first field in the instruction format is known as **opcode,** also referred as operation code. The opcode is of length 7 bits and is used to determine the type of instruction format.
- The next subfield is known as **rd** field which is referred as Destination Register. The rd field is of length 5 bits and is used to store the final result of operation.
- The next subfield is **func3** also referred as function 3. Here the '3' represents the size of this field. This field tells the detail about the operation, i.e., the type of arithmetic and logical that is performed.
- The next subfield is the source registers, **rs1** of length 5 bits. It is mainly used to store and manipulate the data during the execution of instructions.
- The only difference between R-type and I-type is rs2 and func7 field of R-type has been replaced by 12-bits signed immediate, **imm[11:0]**

S-type Instruction

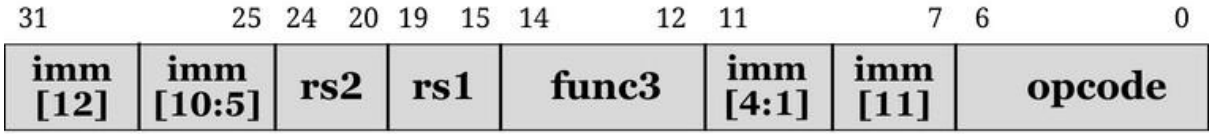
In RV32, each instruction is of size 32 bits. In S-type instruction, S stand for store which means it is store type instruction that helps to store the value of register into the memory. Mainly, this instruction type is used for store operations. The entire 32 bits instruction is divided into 6 fields as shown below.



- The first field in the instruction format is known as **opcode,** also referred as operation code. The opcode is of length 7 bits and is used to determine the type of instruction format.
- S-type instructions encode a 12-bit signed immediate, with the top seven bits imm[11:5] in bits [31:25] of the instruction and the lower five bits imm[4:0] in bits [11:7] of the instruction.
- S-type instruction doesn't have rd fields which states that these instructions are not used to write value to a register, but to write/store a value to a memory.
- The value to be stored is defined in **rs1** field and address to which we have to store this value is calculated using **rs1 and immediate** field. The width of the operation and types of instruction is defined by **func3,** it can be a word, half-word or byte.

B-type Instruction

In RV32, each instruction is of size 32 bits. In B-type instruction, B stand for branching which means it is mainly used for branching based on certain conditions. The entire 32 bits instruction is divided into 8 fields as shown below.



- The first field in the instruction format is known as **opcode,** also referred as operation code. The opcode is of length 7 bits and is used to determine the type of instruction format.
- B-type instructions encode a 12-bit signed immediate, with the most significant bit imm[12] in bit [31] of the instruction, six bits imm[10:5] in bits [25:30] of the instruction, four bits imm[4:1] in bits [11:8] and one bit imm[11] on bit[7].
- There are two source registers **rs1 and rs2** on which various operations are performed based on certain conditions, and those conditions are defined by **func3** field.
- After performing operations on the source register based on the conditions, it is evaluated that if the condition is true, Program Counter value gets updated by PC = Present PC Value + Immediate Value, and if the condition is false then PC will be given as PC = Present PC value + 4 bytes, which states that PC will move to next instruction set.

- RV32 instructions are word-aligned, which means that address is always defined in the multiple of 4 bytes.

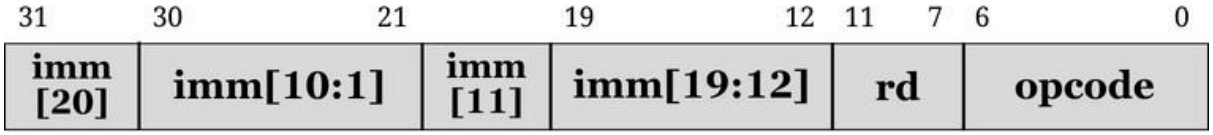
U-type Instruction

In RV32, each instruction is of size 32 bits. In U-type instruction, U stand for Upper Immediate instructions which means it is simply used to transfer the immediate data into the destination register. The entire 32 bits instruction is divided into 3 fields as shown below.

- The first field in the instruction format is known as **opcode,** also referred as operation code. The opcode is of length 7 bits and is used to determine the type of instruction format.
- The U-type instruction only consists of two instructions, i.e., LUI and AUIPC.
- For Example, lets take the instruction **lui rd, imm** and understand this instruction. **lui x15, 0x13579:** This instruction will be executed and the immediate value 0x13579 will be written in the MSB of the rd x15, and it will look like x15 = 0x13579000

J-type Instruction

In RV32, each instruction is of size 32 bits. In U-type instruction, J stand for jump, which means that this instruction format is used to implement jump type instruction. The entire 32 bits instruction is divided into 6 fields as shown below.



- The first field in the instruction format is known as **opcode,** also referred as operation code. The opcode is of length 7 bits and is used to determine the type of instruction format.
- The J-type instruction only consists of single instruction, JAL.
- J-type instruction encode 20 bits signed immediate which is divided into four fields.
- The J-type instructions are often used to perform jump to the desired memory location. The address of the desired memory location is defined in the instruction. These instructions are also used to implement loops.

Base RISC-V Instruction Sets

RV32I (Base Integer ISA - 32-bit)

- **I = Integer**
- 32 general-purpose registers: x0 to x31 (each 32-bit wide)
- x0 is hardwired to 0.

Instruction Types:

1. **R-type**: Register-Register arithmetic
    - e.g., add rd, rs1, rs2
2. **I-type**: Immediate instructions & Load
    - e.g., addi rd, rs1, imm, lw rd, offset(rs1)
3. **S-type**: Store
    - e.g., sw rs2, offset(rs1)
4. **B-type**: Branch
    - e.g., beq rs1, rs2, offset
5. **U-type**: Upper Immediate
    - e.g., lui rd, imm
6. **J-type**: Jump
    - e.g., jal rd, offset

Optional Extensions

RISC-V allows extending the base set with standardized modules:

Extension	Meaning	Example Instrs
M	Integer Multiplication/Division	mul, div, rem
A	Atomic Instructions	lr, sc, amoswap
F	Single-Precision Floating Point	fadd.s, fmul.s
D	Double-Precision Float	fadd.d, fmul.d
C	Compressed Instructions (16-bit)	c.add, c.li
V	Vector Extension (AI/ML apps)	vadd.vv, vmul.vv
B	Bit Manipulation	andn, xnor, rol

Register Summary

Register	Alias	Description
x0	zero	Constant 0
x1	ra	Return Address
x2	sp	Stack Pointer
x5-x7	t0-t2	Temporaries
x8	s0/fp	Saved reg / frame pointer
x10-x17	a0-a7	Argument registers
x28-x31	t3-t6	Temporaries

Example Code Snippet (Assembly)

```
addi x5, x0, 10   ; x5 = 10
addi x6, x0, 20   ; x6 = 20
add x7, x5, x6    ; x7 = x5 + x6 = 30
sw x7, 0(x2)      ; store x7 into memory[sp]
```

Why RISC-V for VLSI?

- **Open-source**: No license cost for silicon tape-out.
- **Modular**: Add only what you need (saves area/power).
- **Simple**: Great for learning processor design.
- **Scalable**: From microcontrollers (RV32I) to supercomputers (RV128).

Tools for RISC-V Learning

Tool	Use
RARS	RISC-V simulator for assembly
Spike	Official ISA simulator
QEMU	Full system emulation
SiFive Core Designer	Build your own RISC-V core
Xilinx/Vivado	RISC-V on FPGA

RISC-V instruction set:

The **RISC-V (Reduced Instruction Set Computer - Five)** instruction set architecture (ISA) is a **free and open-source** architecture that has gained significant traction in both academia and industry due to its simplicity, modularity, and extensibility. At its core, RISC-V is a **load/store architecture** based on the principles of reduced instruction complexity, which makes it highly efficient for hardware implementation. The **base instruction set**, known as **RV32I**, provides 32 general-purpose registers (x0 to x31) and includes fundamental arithmetic, logic, control, and memory operations. These instructions are categorized into several formats: **R-type** for register-to-register operations like add and sub, **I-type** for immediate-based instructions and loads such as addi and lw, **S-type** for store operations, **B-type** for conditional branches, **U-type** for operations like lui that load upper immediate values, and **J-type** for jumps. Importantly, x0 is a hardwired zero register, simplifying many operations at the hardware level.

What makes RISC-V particularly powerful is its **modular design**. Beyond the RV32I base, designers can optionally include standard **ISA extensions** like **M** for multiplication and division, **A** for atomic operations (useful in multi-threaded environments), **F** and **D** for floating-point support, **C** for compressed 16-bit instructions that reduce code size and improve cache performance, and **V** for vector processing, which is vital for AI and high-performance computing. The modular nature allows SoC and processor designers to pick only the features they need, resulting in **area- and power-efficient custom silicon**—a crucial advantage in embedded and IoT applications.

RISC-V also promotes **educational clarity and industrial scalability**. Its open licensing model enables universities to use and modify the ISA for teaching processor design and architecture courses without legal or financial constraints. Similarly, startups and companies can use RISC-V to build custom processors without paying royalties, unlike ARM or x86. Additionally, several tools support RISC-V development, such as **RARS** for assembly simulation, **Spike** (the official ISA simulator), **QEMU** for system-level emulation, and even FPGA-based development using platforms like **Xilinx Vivado** or **Intel Quartus**. This makes RISC-V an ideal ISA not just for CPUs, but for implementing **custom accelerators, co-processors, and even AI-specific hardware blocks**.

From a VLSI perspective, RISC-V's clean instruction set and open ecosystem allow RTL engineers to **design, synthesize, and verify custom processors**, and integrate them into SoCs with confidence. The well-defined nature of the ISA also means **easier formal verification**, reducing risk during silicon design and tape-out. As a result, RISC-V is now being adopted in everything from **tiny embedded controllers to high-performance vector cores**, making it one of the most promising instruction sets of the future.