

Computer Architecture Week 3

Khaula Molapo

Year 2 semester 2

1.

Imagine an IoT system powered by the RISC-V Instruction Set Architecture (ISA). Devices such as smart sensors and wearables benefit from RISC-V's reduced instruction set, which makes execution faster and energy efficient—key requirements for battery-powered systems. RISC-V instructions are divided into R-type (register), I-type (immediate), S-type (store), B-type (branch), U-type (upper immediate), and J-type (jump). For example, an R-type instruction performs arithmetic operations between registers, while an I-type uses constants directly. Addressing modes define how operands are accessed. In RISC-V, these include immediate addressing (operand embedded in the instruction), register addressing (operands in registers), and base + offset addressing (commonly used for memory operations). For instance, a sensor storing temperature data in memory can use base + offset addressing to fetch values efficiently. This combination of well-structured instruction formats and simple addressing modes makes RISC-V highly adaptable for IT tasks like optimizing IoT applications, where performance and power savings are critical.

2.

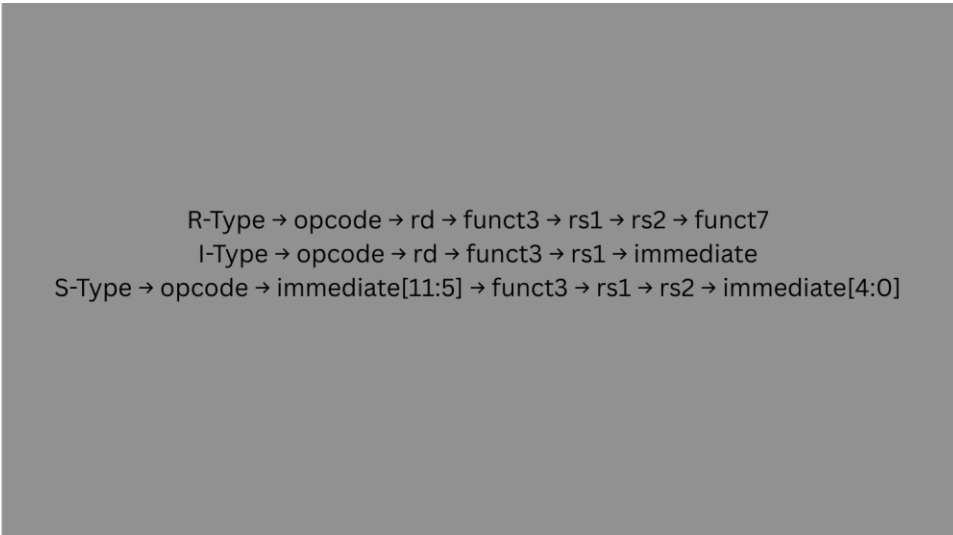
RISC (Reduced Instruction Set Computer) and CISC (Complex Instruction Set Computer) are two main approaches to ISA design. RISC focuses on simple, uniform instructions that execute in a single clock cycle, making it efficient and ideal for devices like smartphones and IoT systems. CISC, on the other hand, uses complex, multi-step instructions, reducing the number of lines of code needed but often requiring more clock cycles. RISC emphasizes hardware simplicity and speed, while CISC emphasizes flexibility and compact code size. A common example of CISC is Intel's x86 architecture, while RISC is embodied by ARM and RISC-V. Both play essential roles in computing: RISC dominates low-power embedded systems, while CISC remains common in desktops and servers.

3.

Writing this RISC-V program in RARS was a practical way to understand ISA concepts. The process showed me how instructions interact directly with registers and memory. For example, loading values into t0 and t1, performing an addition with the add instruction, and storing the result highlighted the structured simplicity of RISC-V. Unlike high-level languages, I had to manually manage registers and memory, which made me appreciate the efficiency of the instruction set. RARS also provided a clear interface for running and debugging the program, helping me trace execution step by step. Overall, the hands-on exercise strengthened my understanding of how low-level machine operations support IT tasks.

4.

This diagram represents three common instruction formats in RISC-V. The R-type format is used for register-to-register operations, such as addition or subtraction. It includes two source registers (rs1, rs2) and one destination register (rd). The I-type format is used for instructions with immediate values, such as loading a constant into a register or performing arithmetic with an embedded operand. The S-type format is used for storing data from a register into memory, splitting the immediate field into two parts. Each format uses the opcode field to identify the type of operation. Visualizing these formats helps understand how instructions are structured, showing the balance between simplicity and flexibility in RISC-V's ISA design.



R-Type → opcode → rd → funct3 → rs1 → rs2 → funct7
I-Type → opcode → rd → funct3 → rs1 → immediate
S-Type → opcode → immediate[11:5] → funct3 → rs1 → rs2 → immediate[4:0]