Instruction Sets & CPU Architecture

How programs talk to hardware

What we will learn

- Instructions and programs
- Instruction Set Architecture (ISA)
- Assembly language basics
- CPU architecture ideas (RISC, CISC)
- Real CPUs: x86 and ARM
- Microcontrollers and why they exist

Section 1: Instructions and Programs

What we will see

- What instructions are
- How programs are built

What is an instruction?

An instruction is a **single command** given to the CPU.

It tells:

- What action to perform
- What data to use

Real-life analogy

Think of a recipe:

- Recipe = program
- Each step = instruction

CPU follows steps one by one.

Think About This

Why do CPUs only understand very small, simple instructions?

Section 1 Summary

- Programs are sequences of instructions
- Each instruction does one small job
- CPU follows instructions exactly

Section 2: Instruction Set Architecture (ISA)

What we will see

- What ISA means
- Why ISA matters

What is ISA?

Instruction Set Architecture defines:

- What instructions exist
- How they are written
- How data is accessed

ISA is a contract between software and hardware.

ISA as a contract

- Software follows ISA rules
- Hardware guarantees correct execution

This allows compatibility.

Real-life analogy

ISA is like traffic rules.

Same rules → many vehicles can drive safely.

Reflect

Why do companies try to keep ISAs stable for many years?

Section 2 Summary

- ISA defines the CPU language
- It separates software from hardware
- Same ISA allows software reuse

Section 3: Assembly Language

What we will see

- Why assembly exists
- Mnemonics and operands
- Arithmetic examples

Why assembly language?

Computers understand binary.
Humans prefer symbols.

Assembly uses short names for instructions.

Mnemonics

Examples:

- ADD – addition
- SUB – subtraction
- LD – load
- ST – store

Operands

Operands specify:

- Input data
- Output location

Usually registers or memory.

Simple addition example

```
ADD R3, R1, R2
```

Meaning:

R3 = R1 + R2

Think Deeper

Can multiplication be done using only addition?

Multiplication using addition (idea)

To compute:
$R = A \times B$

Add A repeatedly, B times.

8086-style assembly example

- AX, BX, CX, DX - Registers
- AX = the number you want to multiply (multiplicand)
- BX = how many times to add it (multiplier)
- Result will end up in DX

```
MOV CX, BX //
MOV DX, 0

LOOP:
ADD DX, AX
DEC CX
JNZ LOOP
```

Final result:

Reflect

Why include a MUL instruction if addition can do the job?

Section 3 Summary

- Assembly instructions are simple
- Complex work comes from repetition
- Old CPUs shaped modern ones

Section 4: CPU Architecture

What we will see

- Instruction design ideas
- RISC and CISC

Load/Store idea

- Load-Store architecture is a CPU design pattern that separates memory access from data processing.
- Memory accessed only by load/store
- Arithmetic works on registers

RISC idea

Reduced Instruction Set Computer:

- Few simple instructions
- Load-Store architecture, Operations happen in Registers
- Easier hardware
- Compiler does more work

Why RISC?

- Simple design
- Efficient execution
- Good performance per watt

CISC idea

Complex Instruction Set Computer:

- Many instructions
- Powerful individual instructions
- Shorter programs
- Pperations get performed in memory

RISC vs CISC

Aspect	RISC	CISC
Instruction design	Simple	Complex
Total instruction count	~50-150	~1000+
Code needed for a task	More	Fewer
Hardware complexity	Simple	Very complex

Think About This

Which is better: simple instructions or powerful instructions?

Section 4 Summary

- Architecture reflects design choices
- RISC and CISC differ in philosophy
- Both are still used today

Section 5: Real CPU Architectures

What we will see

- x86 and ARM
- Historical context

x86 architecture

Used by:

- Intel CPUs
- AMD CPUs

Common in desktops and laptops.

Where did x86 come from?

x86 began with Intel 8086 (1978).

Key idea:
Backward compatibility.

Is 8086 relevant today?

- Modern CPUs still understand it
- Used for teaching fundamentals
- Shapes modern CPU design

ARM architecture

Used in:

- Phones
- Tablets
- Embedded and servers

Based on RISC ideas.

x86 vs ARM

x86	ARM
Strong backward compatibility	Cleaner evolution
Complex ISA	Simpler ISA
Desktop legacy	Mobile first

Think About This

Why do phones prefer ARM while desktops used x86?

Section 5 Summary

- ISAs shape real CPUs
- x86 and ARM made different trade-offs
- Both dominate different domains

Section 6: Microcontrollers

What we will see

- What microcontrollers are
- Why they exist
- Examples

What is a microcontroller?

A microcontroller includes:

- CPU
- Memory
- Input/Output

All on one chip.

Real-life analogy

CPU system:
Many separate rooms.

Microcontroller:
Everything in one small box.

Why microcontrollers?

Used when:

- Task is simple
- Power must be low
- Cost must be low

Why not a full CPU?

Full CPUs are:

- Power-hungry
- Expensive
- Overkill

Microcontrollers are focused.

Example: Intel 8085

- Early microprocessor
- Popular in education
- Used in control systems

Is 8085 still used?

Rare in products.

Still taught for fundamentals.

Popular modern microcontrollers

- Arduino (ATmega)
- ESP32
- ARM Cortex-M

Microcontroller vs CPU

CPU	Microcontroller
General purpose	Dedicated
High performance	Low power
Many components	Single chip

Think About This

Would a traffic light need a CPU or a microcontroller?

Final Takeaways

- Instructions are the building blocks
- ISA defines how CPUs work
- Assembly shows logic step by step
- Architectures reflect design choices
- Microcontrollers fit simple tasks

Thank you!

Any questions?

Thejesh GN