🧠 x86 PROCESSOR BASICS (HOW THE CPU ACTUALLY RUNS THE SHOW)

Imagine the CPU as the **brain** of your computer.

But not a chill brain — a **cracked-out microsecond freak** that runs everything on caffeine and electricity. Here's how it works:

🏛️ The CPU – Central Processing Unit

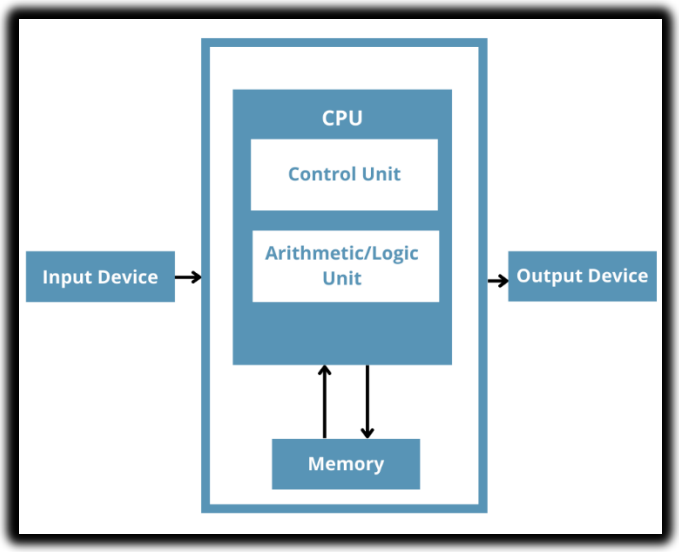
This is where all the **thinking**, **math**, and **decision-making** happens.

It has:

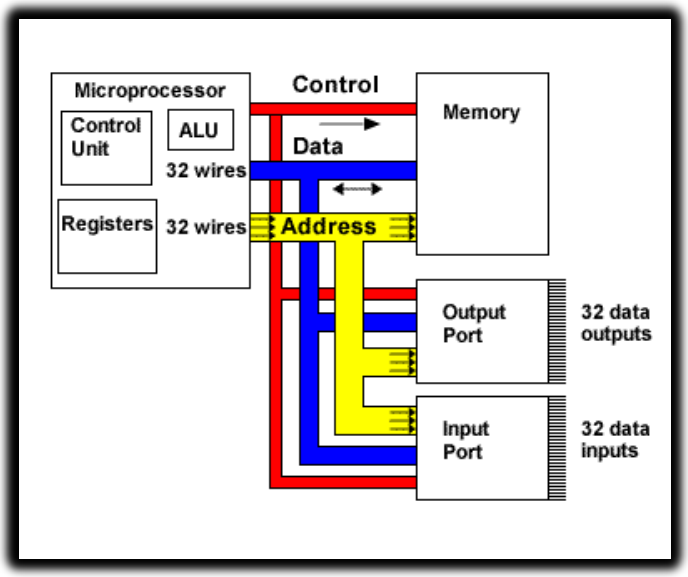
- **Registers** – tiny super-fast storage slots (think 32-bit pockets for numbers)
- **Clock** – keeps time like a heartbeat so stuff happens in sync
- **Control Unit (CU)** – the **boss** that decides what happens next
- **ALU (Arithmetic Logic Unit)** – the **muscle** that does all the math and logic ops (ADD, SUB, AND, OR, NOT, etc.)

How the CPU Connects to the World

The CPU talks to the rest of the PC through **pins** on its socket. These pins connect it to **buses** — long electric highways carrying signals.



The 3 main buses:



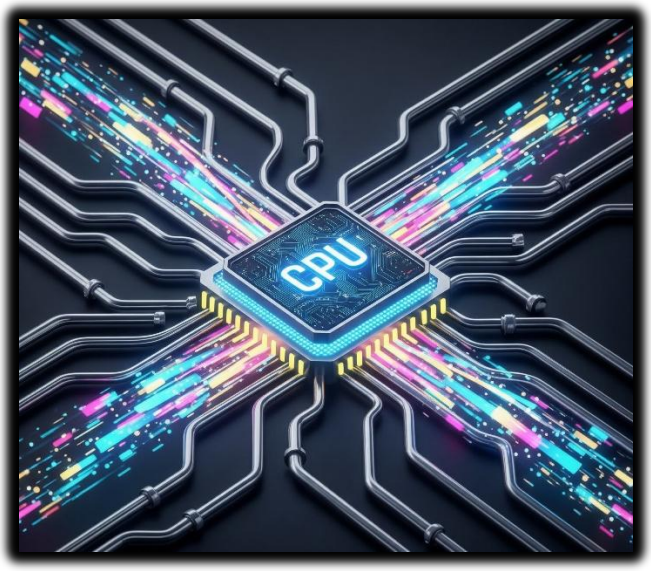
🟦 Data Bus

Moves the *actual data* and *instructions* between the CPU, memory and I/O devices.

The data bus is bidirectional, meaning information can flow in both directions.

The *"width" of the data bus* (how many parallel wires it has) determines how much data can be transferred at once.

A *64-bit data bus* can move 64 bits of data simultaneously.



Analogy: The data bus is like a fleet of delivery trucks that transport goods (data) and mail (instructions) between the city hall (CPU), the library (memory), and various businesses (I/O devices). These trucks can deliver or pick up cargo.

🟧 Address Bus

Says *where* in memory we're looking.

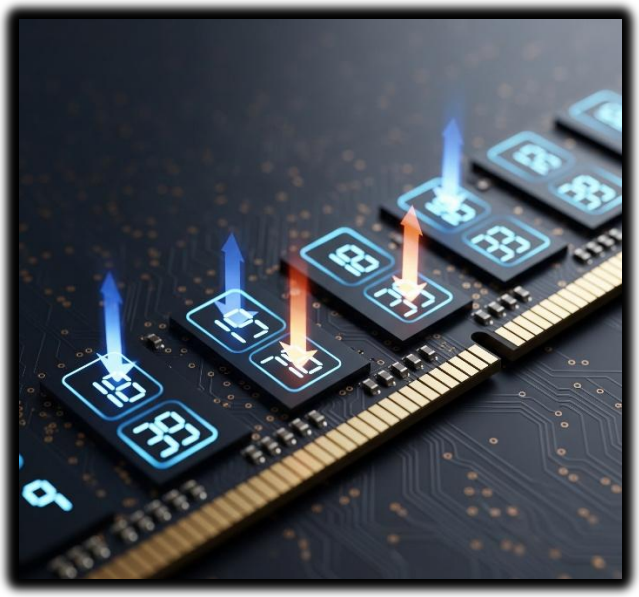
The address bus is *unidirectional*, meaning information flows only from the CPU to other components.

It carries the memory addresses **or** I/O port addresses where data is to be read from or written to.

When the CPU wants to access a specific piece of data or instruction, it places its memory address on the address bus, telling the memory unit *exactly where to find* or store that information.

The *width of the address bus* determines the maximum amount of memory the CPU can access.

A *32-bit address* bus can address 2^{32} unique memory locations (4 Gigabytes).



Imagine your *computer's RAM as a massive library*, and each book in that library has a unique shelf and position. When the CPU wants to read a specific piece of information (a "book"), it doesn't just shout out the book's title.

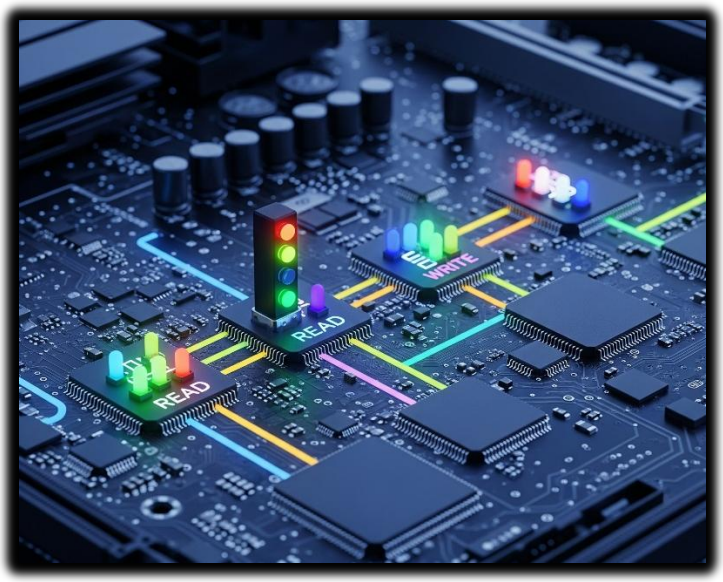
Instead, it sends out the exact *"shelf number"* and *"position"* through the address bus. This "shelf number and position" is what we call a **memory address**.

This *one-way communication* ensures that the CPU can accurately request data from, or send data to, a specific spot in memory.

🟥 Control Bus

Uses binary signals (on/off) to tell devices **when** to send or receive. It synchronizes the actions and manages the flow of information among all devices attached to the system bus.

Think: *"Hey RAM — CPU wants to read now!"*



It carries control signals that dictate operations like "memory read," "memory write," "I/O read," "I/O write," "interrupt request," and "bus grant."

These signals ensure that devices don't try to use the buses simultaneously or perform conflicting operations.

The control bus is like the city's traffic light system.

Other Buses

⚫ I/O Bus

Handles data moving between CPU and input/output devices (keyboard, mouse, etc.)

Also called the Peripheral bus, considered part of the system bus, but, yeah, it's a bit different coz its *dedicated* to transferring data between the CPU and the system I/O devices.

Modern systems often use high-speed serial buses like *PCI Express (PCIe)* for this purpose.

This bus is all about getting data to and from your **input/output devices**. Imagine:

- **Keyboard Input:** When you type "hello," that information needs to travel from your keyboard into the computer. The I/O bus is the route that data takes, like supplies being delivered to a restaurant. ⌨➡💻
- **Printer Output:** When you hit "print," the document data needs to go from your computer out to the printer. The I/O bus handles this, much like official documents being sent out to residents. 💻➡🖨

🧠 Memory – Where Programs & Data Live

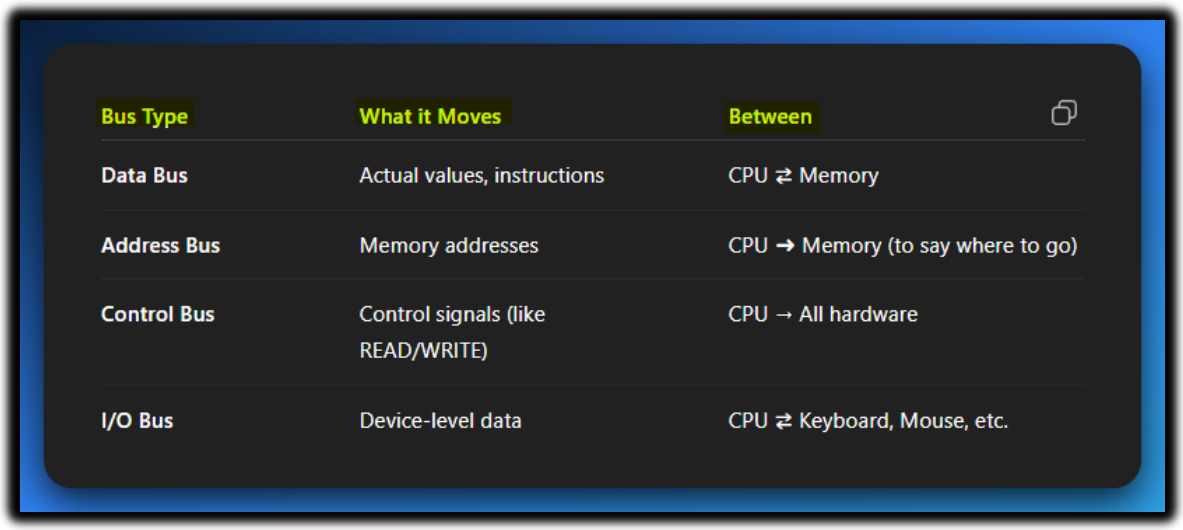
All your running programs and variables are stored in **RAM**. But here's the kicker:
The CPU **can't run them straight from RAM.**

It always does this:

1. Grabs the instruction from memory
2. Brings it into the CPU
3. Executes it
4. Maybe sends a result back to memory

So, your code doesn't *run in RAM*, it runs **inside the CPU** — one piece at a time, or in chunks.

📦 Buses Summary (Quick Table):



Bus Type	What it Moves	Between
Data Bus	Actual values, instructions	CPU ⇄ Memory
Address Bus	Memory addresses	CPU → Memory (to say where to go)
Control Bus	Control signals (like READ/WRITE)	CPU → All hardware
I/O Bus	Device-level data	CPU ⇄ Keyboard, Mouse, etc.

TLDR – Reverse Engineering Focus:

- Know the **ALU** is where bitwise ops live (AND, OR, SHL, etc.)
- Know that **registers** are the CPU's playground — what you see in disasm (like eax, edx, rsi, etc.)
- Remember: instructions **run inside** the CPU, not memory. Memory just holds them until they're needed.
- Buses = wires that move the ops around. If you're watching malware move code into memory and jump to it — that's this system in action.