Semiconductor Technology from A to Z

Digital technology

Processor Architecture

Overview

A processor (CPU, Central Processing Unit) is the "brain" of a computer – but contrary to what that image suggests, it does not think for itself. Instead, it doggedly works through a list of instructions held in memory. This list is called a program, and each individual entry is a machine instruction (e.g. "add two numbers" or "load a value from memory"). A memory device such as SRAM or DRAM can only hold data and hand it back on request – the processor, by contrast, actively processes that data: it reads instructions, interprets them, and executes them, one after another, billions of times per second. Everything a computer "does" – from a simple addition to rendering a video – ultimately boils down to a very long chain of such simple individual steps.

Basic Architecture

For a processor to access instructions and data at all, it needs a path to memory. Most processors today follow the *von Neumann architecture*: the program (instructions) and data (the values being computed on) reside in the same memory and are loaded over the same pathway. This simplifies the design considerably but creates a bottleneck: instructions and data cannot be transported over this single path at the same time, so the processor must wait its turn (the so-called von Neumann bottleneck). The *Harvard architecture*, by contrast, physically separates program memory from data memory and allows genuine parallel access to both at once. Many modern processors use a compromise: they appear von-Neumann-like from the outside, but internally have separate caches for instructions and data, combining the advantages of both approaches.

Core Components

To understand how an instruction is actually executed, it helps to look at three central building blocks:

The *control unit* (CU) is something like the processor's conductor. It reads the next instruction from memory, identifies what kind of operation it represents, and generates the appropriate electrical control signals that activate all the other blocks at the right moment – without doing any computation itself.

The *arithmetic logic unit* (ALU) is the block that actually computes: addition, subtraction, comparisons, logical operations (AND, OR, etc.). It is told by the

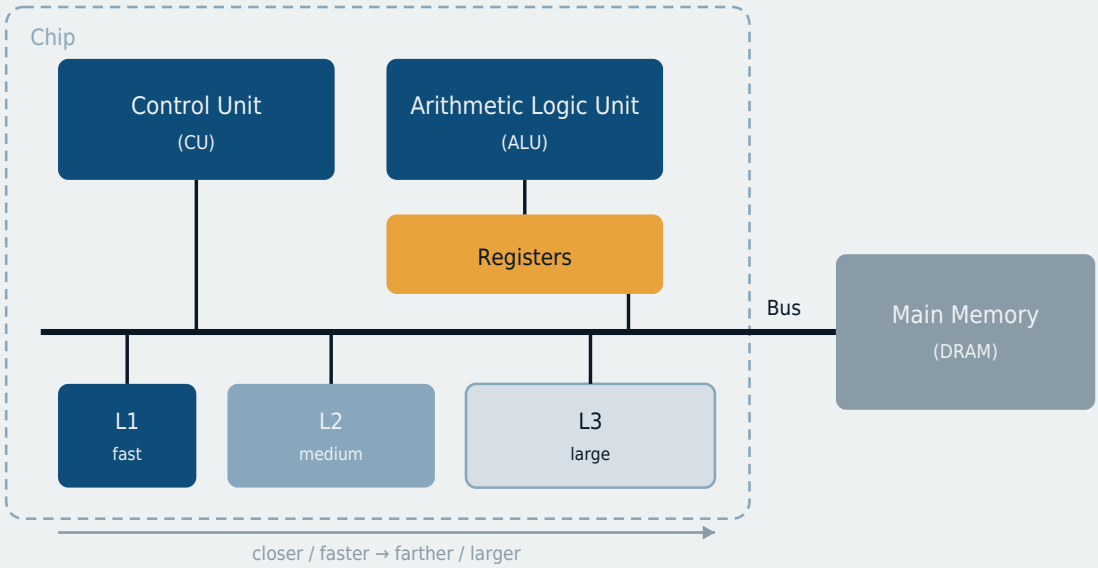
control unit which operation to perform and where the input values come from.

Registers are tiny, extremely fast memory cells located directly inside the processor – comparable to a scratch pad where the ALU stores its intermediate results while computing. There are only a handful of them (typically 16–32), because they require a lot of chip area per bit; in return, they are orders of magnitude faster than any other memory in the system.

For these blocks to exchange data at all, a *bus* is needed – a collection of electrical lines that transport addresses (where is a value?), data (which value?), and control signals (what should happen with it?) back and forth between the control unit, ALU, registers, cache, and memory. You can picture the bus as a shared road on which only a certain amount of "traffic" (bits) can flow at any given moment – its width (e.g. 32 or 64 bits) determines how much data can be transferred per clock cycle.

A single instruction always passes through three phases: *Fetch* (the control unit retrieves the next instruction from memory via the bus), *Decode* (the instruction is interpreted, and the control unit identifies the operation to be performed), and *Execute* (the ALU carries out the operation, with the result ending up in a register or being written back to memory).

Processor Architecture – Block Diagram



The bus links all units; cache bridges the gap between registers and main memory

Memory Hierarchy

Registers are extremely fast, but there are only a few of them – for everything not currently being worked on, the processor needs larger memory. This is

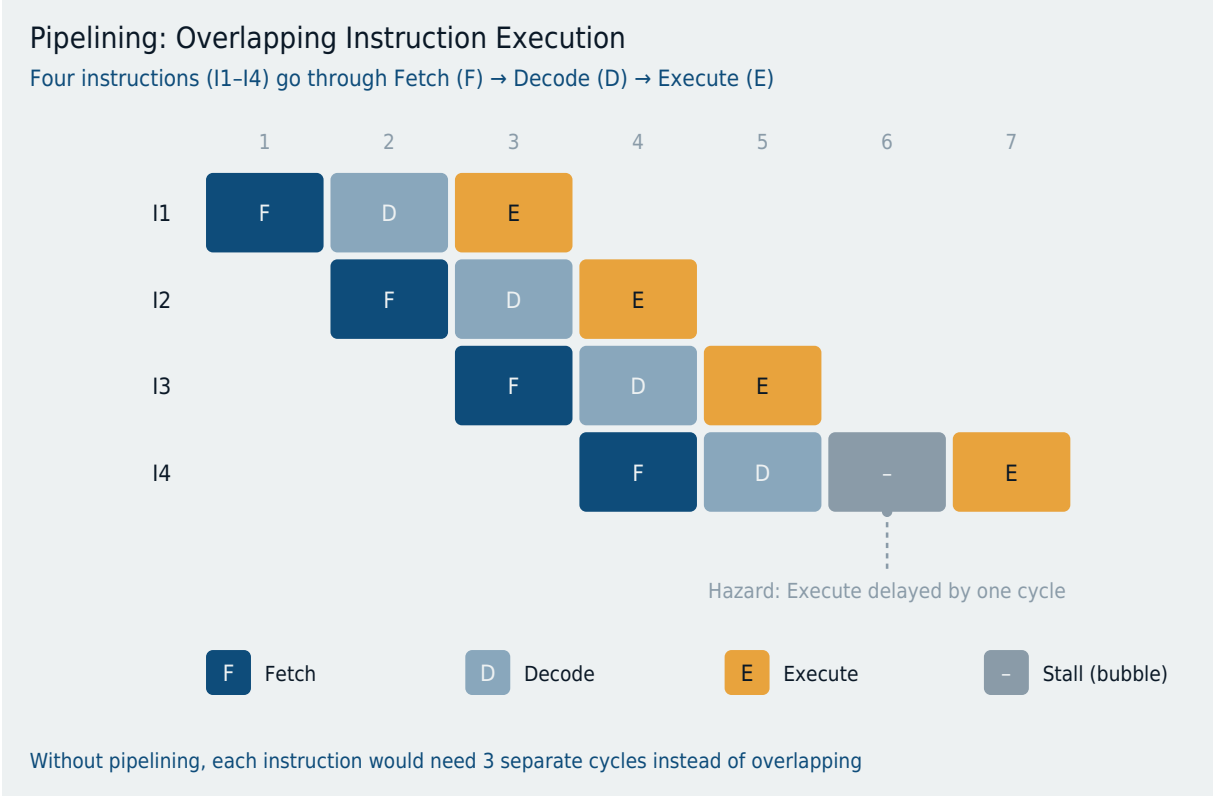
where *cache* comes in: a small but very fast intermediate memory that keeps frequently needed data and instructions close to the ALU, so the processor does not have to consult the slow main memory on every access. Cache is usually staged across several levels: L1 (level 1) sits directly next to the ALU, is the fastest but also the smallest (often just 32–64 KB); L2 is larger but somewhat slower; L3 is larger and slower still, and is often shared between several cores. Main memory (DRAM, on separate memory chips outside the processor) is orders of magnitude larger than any cache, but also considerably slower, because signals have to travel a much longer physical distance.

The reason this hierarchy works at all is the principle of *locality*: programs typically access the same memory regions repeatedly (temporal locality) and addresses that lie close together (spatial locality) – for example, in a loop that keeps using the same handful of variables. Cache exploits this by keeping recently used data close at hand, intercepting most accesses before they need to reach the slow main memory.

Clock, Pipelining and Parallelism

A clock generator sets the processor's rhythm: the clock frequency (e.g. 3 GHz = 3 billion cycles per second) determines how often a new processing stage is triggered each second. Higher clock frequencies mean more processing steps per second, but they also generate more power dissipation (heat) and run into physical limits, because signals cannot propagate through the circuit arbitrarily fast within the available time.

To extract more performance regardless, modern processors use *pipelining*: instead of completing an instruction fully, from fetch to execute, before starting the next one, execution is broken down into several stages that work simultaneously – like an assembly line – but offset across different instructions. While one instruction is being executed (Execute), the next is already being decoded (Decode), and a third is being fetched from memory (Fetch) – all within the same cycle, but at different stages. This noticeably increases throughput without making any single instruction faster. Problems arise when an instruction depends on the result of a previous one that is not yet ready (so-called *hazards*) – the pipeline then has to pause briefly or reorder work, which requires additional control logic.



Because individual cores can barely be clocked any faster without generating impractical amounts of heat, modern processors instead rely on *multicore design*: rather than a single, ever-faster core, they integrate several complete but somewhat simpler cores on one die, which can work on independent tasks in parallel.

From Die to Chip

In the finished layout, the control unit, ALU, registers, and cache are arranged as clearly delineated, repeatedly used blocks on the die – much like rooms in a building plan. Cache often takes up a surprisingly large share of the area, because each individual SRAM cell consists of six transistors and therefore requires comparatively large area. I/O areas (I/O pads), through which the chip connects to the outside world – memory, power supply, other components – sit at the edge of the die, so that the bond wires or solder bumps to the package connections stay short. How these blocks are planned and arranged in detail is covered in the chapter on Circuit Layouts.