Kapitel 1

Hardwarearchitektur

1.1 Worum geht es?

Unter **Hardwarearchitektur** versteht man den grundsätzlichen Aufbau eines Rechners: Welche Bausteine gibt es (z. B. Prozessor, Speicher, Ein-/Ausgabe), wie sind sie *organisiert* und *verbunden*, und wie arbeiten sie zusammen, um Programme auszuführen? Die heute dominierende Grundidee ist die **von-Neumann-Architektur**.

1.2 John von Neumann und die Grundidee

In den 1940er Jahren formulierte John von Neumann (gemeinsam mit weiteren Pionieren um ENIAC/EDVAC) eine einfache, aber revolutionäre Idee: **Programm und Daten liegen im gleichen Speicher.** Das heißt, ein Programm ist selbst nur eine Folge von Zahlen (Maschinenbefehlen), die genau wie Daten im Hauptspeicher abgelegt und von der CPU geholt werden. Diese *Stored-Program*-Idee macht Rechner *flexibel* (beliebige Programme ladbar) und *universell*.

Bausteine im von-Neumann-Modell

- **CPU (Prozessor)** mit
  - **Steuerwerk** (kontrolliert den Ablauf, interpretiert Befehle),
  - **Rechenwerk/ALU** (führt Operationen wie Addieren, Vergleichen aus),
  - **Registern** (kleinste, sehr schnelle Speicherplätze, z. B. Akkumulator, Program Counter).
- **Hauptspeicher (RAM)**: enthält *sowohl* Daten *als auch* Befehle.
- **Ein-/Ausgabe (I/O)**: Tastatur, Bildschirm, Netz, Sensoren, Aktoren …
- **Bus-System**: Verbindet die Bausteine (Adress-, Daten- und Steuerbus).

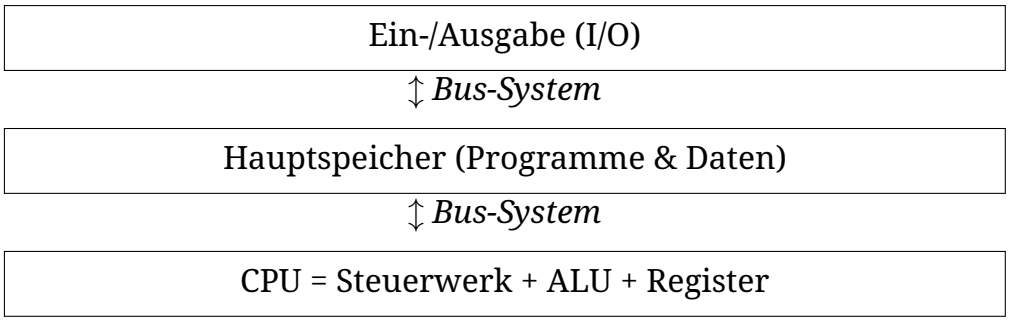

Abbildung 1.1: Vereinfachtes von-Neumann-Modell.

1.3 Der Befehlszyklus (Fetch–Decode–Execute)

Jeder Maschinenbefehl läuft in drei Schritten durch die CPU:

1. **Fetch** (Holen): Der *Program Counter (PC)* zeigt auf die nächste Befehlsadresse. Der Befehl wird aus dem Speicher gelesen und in das *Befehlsregister (IR)* gelegt.

2. **Decode** (Dekodieren): Das Steuerwerk „versteht“, welcher Operationstyp gemeint ist (z. B. ADD, LOAD, STORE) und welche Operanden/Adressen beteiligt sind.

3. **Execute** (Ausführen): Die ALU rechnet bzw. I/O/Speicherzugriffe passieren. Der PC wird auf den nächsten Befehl gesetzt (oder bei Sprüngen angepasst).

Mini-Beispiel (gedankliches Maschinenprogramm).

```
1         LOAD R0, [x]    ; lade x in Register R0
2         LOAD R1, [y]    ; lade y in Register R1
3         ADD  R0, R1     ; R0 := R0 + R1
4         STORE R0, [z]   ; speichere Ergebnis in z
```

Listing 1.1: Addition zweier Speicherstellen und Ablage des Ergebnisses

Hier holt die CPU nacheinander die Befehle (Fetch), dekodiert sie (Decode) und führt sie aus (Execute).

1.4 Speicher, Wortbreite und Adressierung

- **Wortbreite** (z. B. 32 Bit, 64 Bit) gibt an, wie viele Bits die CPU in einem Schritt besonders effizient verarbeitet (Register- und ALU-Breite). Sie beeinflusst u. a. den darstellbaren Adressraum und Zahlenbereich.

- **Adressbus/Datenbus**: Mit n Adressleitungen kann man 2^n Speicheradressen ansprechen.

- **Speicherhierarchie**: Register $\rightarrow$ Caches (L1/L2/L3) $\rightarrow$ RAM $\rightarrow$ SSD/HDD. Je näher an der CPU, desto schneller (aber kleiner/teurer).

1.5 Warum ist das so erfolgreich?

- **Einfachheit**: Ein einheitlicher Speicher für Programme und Daten macht die Hardware und das Laden von Programmen einfach.
- **Flexibilität**: Beliebige Programme können nachgeladen werden; Selbstmodifizierender Code ist (theoretisch) möglich.
- **Universalität**: Mit genug Speicher und Zeit kann ein solcher Rechner jede berechenbare Aufgabe lösen (Church–Turing-Idee).

1.6 Grenzen: der von-Neumann-Flaschenhals

Weil *Programm* und *Daten* über *denselben* Speicher-/Busweg kommen, konkurrieren sie um Bandbreite. Das bremst: Die CPU könnte schneller rechnen, als Daten/Befehle nachgeliefert werden. Gegenmaßnahmen:

- **Caches** und **Vorabruf** (Prefetch),
- **Pipelining** und **Superskalarität** (mehrere Befehle gleichzeitig in verschiedenen Stufen),
- **Mehrkerner** (Multi-Core) und **Vektor-/SIMD**-Einheiten,
- **Breitere Busse** und schnellere Speicher (DDR, HBM).

1.7 Harvard vs. von Neumann (und die Praxis heute)

Die **Harvard-Architektur** trennt *Befehls-* und *Datenspeicher* (je eigener Bus). Vorteil: Gleichzeitige Zugriffe, kein Flaschenhals an dieser Stelle. Viele *Mikrocontroller/DSPs* und auch *moderne CPUs intern* nutzen eine **modifizierte Harvard-Architektur**: z. B. getrennte *Instruktions-* und *Datencaches*, obwohl der *Hauptspeicher* gemeinsam ist. Damit kombiniert man die Programmierfreundlichkeit des von-Neumann-Modells mit Leistungsgewinnen.

1.8 Merke

- **von Neumann**: ein Speicher für *Programme und Daten*, CPU holt Befehle und führt sie im *Fetch–Decode–Execute*-Zyklus aus.
- **Bausteine**: Steuerwerk, ALU, Register, Speicher, I/O, Busse.
- **Vorteile**: Einfach, flexibel, universell. **Nachteil**: *von-Neumann-Flaschenhals*.
- **Heute**: Praktisch überall Grundlage; oft intern mit *Harvard-Elementen* (getrennte Caches) beschleunigt.