Lösungsvorschlag
Thema: Hardwarearchitektur — Von-Neumann-Architektur (Kapitel 3)

Präsenzaufgaben (Musterlösungen / Erwartungshorizont)

Aufgabe 1: Hintergrund. John von Neumann (1903–1957), Mathematiker/Physiker. 1945 verfasste er den *First Draft of a Report on the EDVAC* (Entwurf für einen elektronischen Digitalrechner), der das heute klassische Rechnermodell beschreibt.

- **Kontext:** Zeit der ersten elektronischen Rechenanlagen (ENIAC, EDVAC). Übergang von fest verdrahteten Programmen zu *speicherprogrammierten* Maschinen.
- **Beitrag:** Klare Trennung von Rechenwerk (ALU), Steuerwerk, Speicher, I/O und Bus. Einführung eines Programmzählers (PC) und eines Befehlsholzyklus.
- **Beispiele/Projekte:** ENIAC (Ballistik), EDVAC, später IAS-Rechner (Princeton), Einfluss auf EDSAC/Manchester Mark I.

Aufgabe 2: Grundidee der Von-Neumann-Architektur (Speicherprogrammiertechnik).

- **Kernidee:** *Programme und Daten liegen im selben (adressierbaren) Speicher.* Der Prozessor liest nacheinander Befehle, die im Speicher als *Zahlen* (Bitmuster) abgelegt sind.
- **Bedeutung:** Programme können wie Daten behandelt werden (laden, speichern, verändern). Das macht Universalrechner flexibel (beliebige Algorithmen ohne Hardware-Umbau).
- **Schema (in Worten):** CPU (ALU + Steuerwerk + Register) ist über *Adress-*, *Daten-* und *Steuerbus* mit dem Speicher und den Ein-/Ausgabegeräten verbunden. Ein gemeinsamer Adressraum.

Aufgabe 3: Hauptkomponenten (präzise).

Aufgabe 3:a) **ALU (Rechenwerk):** Führt Operationen auf Registern/Daten aus (z. B. Addieren, Subtrahieren, logische Verknüpfungen AND/OR/XOR, Vergleiche). Liefert **Flags** (z. B. Zero, Carry, Overflow, Negative), die das Steuerwerk für Verzweigungen nutzt.

Aufgabe 3:b) **Steuerwerk (Control Unit):** Steuert den *Fetch–Decode–Execute*-Zyklus. Enthält typ. **PC** (Program Counter), **IR** (Instruction Register), ggf. **Decoder** und **Mikrosteuerwerk**. Erzeugt Steuersignale (z. B. *Read/Write*, Register laden).

Aufgabe 3:c) **Speicher:** Beinhaltet *Programm und Daten*; adressierbar in Worten/Bytes. Stack/Heap/Code-Bereich sind *logische* Aufteilungen im selben physischen/virtuellen Speicher.

Aufgabe 3:d) **Ein-/Ausgabe (I/O):** Schnittstellen zu Tastatur, Bildschirm, Netz, Sensoren … angebunden als *Memory-mapped I/O* (I/O-Register im Adressraum) oder über separaten I/O-Bus.

Aufgabe 3:e) **Bus-System:** *Adressbus* wählt Speicher-/I/O-Adresse, *Datenbus* transportiert Daten, *Steuerbus* koordiniert (z. B. *RD/ WR*, Takt, Interrupts).

Aufgabe 4: Arbeitsweise: Fetch–Decode–Execute.

Aufgabe 2: Beispielablauf — *Fetch–Decode–Execute* (ausführlich)

Annahme (einfaches Akkumulator-Modell): Register: PC (Program Counter), IR (Instruction Register), ACC (Akkumulator), MAR (Memory Address Register), MDR (Memory Data Register). Busse: Adressbus (ABUS), Datenbus (DBUS), Steuerbus (Signale RD/WR). Speicherzellen: Mem[A] = 5, Mem[B] = 7. Programm (ab PC = 100): `LOAD A`; `ADD B`; `STORE A`.

Ziel: Nach Ablauf soll im ACC und in Mem[A] der Wert $5 + 7 = 12$ stehen.

Instruktion 1: `LOAD A`

T1	*Fetch*	MAR $\leftarrow$ PC; ABUS $\leftarrow$ PC; RD = 1 (Befehl lesen)
T2	*Fetch*	MDR $\leftarrow$ Mem[MAR]; IR $\leftarrow$ MDR; PC $\leftarrow$ PC + 1
T3	*Decode*	Decoder erkennt `LOAD`, Operand = Adresse A
T4	*Exec*	MAR $\leftarrow A$; ABUS $\leftarrow A$; RD = 1 (Operandenwort lesen)
T5	*Exec*	MDR $\leftarrow$ Mem[A] (=5); ACC $\leftarrow$ MDR (ACC = 5)

Instruktion 2: `ADD B`

T6	*Fetch*	MAR $\leftarrow$ PC; ABUS $\leftarrow$ PC; RD = 1
T7	*Fetch*	MDR $\leftarrow$ Mem[MAR]; IR $\leftarrow$ MDR; PC $\leftarrow$ PC + 1
T8	*Decode*	Decoder erkennt `ADD`, Operand = Adresse B
T9	*Exec*	MAR $\leftarrow B$; ABUS $\leftarrow B$; RD = 1 (Operandenwort lesen)
T10	*Exec*	MDR $\leftarrow$ Mem[B] (=7); ACC $\leftarrow$ ACC + MDR (ACC = $5 + 7 = 12$); Flags (Z/C/V/N) setzen

Instruktion 3: `STORE A`

T11	*Fetch*	MAR $\leftarrow$ PC; ABUS $\leftarrow$ PC; RD = 1
T12	*Fetch*	MDR $\leftarrow$ Mem[MAR]; IR $\leftarrow$ MDR; PC $\leftarrow$ PC + 1
T13	*Decode*	Decoder erkennt `STORE`, Operand = Adresse A
T14	*Exec*	MAR $\leftarrow A$; MDR $\leftarrow$ ACC (=12); ABUS $\leftarrow A$
T15	*Exec*	WR = 1; Mem[A] $\leftarrow$ MDR (Speicherzelle A erhält 12)

Hinweis auf Niveau 11: Wichtig ist das Prinzip (Holen–Decodieren–Ausführen) und die Rolle von PC/IR/ALU, nicht die Mikroschritte einzelner Takte.

Aufgabe 5: Vor- und Nachteile. **Stärken:** Einfache, universelle Struktur; Programme sind leicht lad-/änderbar; kostengünstiger als fest verdrahtete Logik.
Grenzen (*Von-Neumann-Flaschenhals*): CPU und Speicher teilen sich *einen* Datenweg; nur *ein* Transfer pro Takt (Instruktion *oder* Daten) $\Rightarrow$ Bandbreite limitiert; Latenzen steigen. Gegenmittel: Caches, Prefetch, Pipeline, breitere Busse, Out-of-Order, …

Aufgabe 6: Vergleich Harvard vs. Von Neumann (optional). **Harvard:** Getrennter Speicher/Bus für *Instruktionen* und *Daten* $\Rightarrow$ paralleles Holen von Befehlen und Daten, oft in Mikrocontrollern/DSPs (z. B. AVR, PIC).
Von Neumann: Ein gemeinsamer Speicher/Bus (klassische PCs/Server). **Modern:** „Modified Harvard“ auf Cache-Ebene (getrennter L1 I-/D-Cache), aber gemeinsamer Hauptspeicher.

Hausaufgaben / Vertiefung (Musterlösungen)

Aufgabe 1: Skizze mit Legende (Beispielantwort). Minimalstruktur (in Worten, zeichnerisch ähnlich):

- **CPU** mit **ALU**, **Steuerwerk**, **Registern** (PC, IR, Akkumulator/Allzweckregister).
- **Speicher** (Programm + Daten), **I/O** (Tastatur, Display, Netz).
- **Busse:** Adressbus (CPU→Speicher/I/O), Datenbus (beide Richtungen), Steuerbus (RD/WR, Takt).

Legende (Beispiel): PC: Adresse des nächsten Befehls; IR: aktueller Befehl; MAR/MDR (optional): Speicheradresse/Datenwort; ALU: rechnet auf Registern; I/O: per Memory-mapped I/O eingebunden.

Aufgabe 2: Beispielablauf (Fetch–Decode–Execute). *Programm:* `LOAD A`; `ADD B`; `STORE A`.

Aufgabe 2:a) **LOAD A:** PC→MAR; Speicher liest Instruktion →IR; *Decode*; Adresse A →MAR; Speicher →MDR; MDR →Akkumulator.

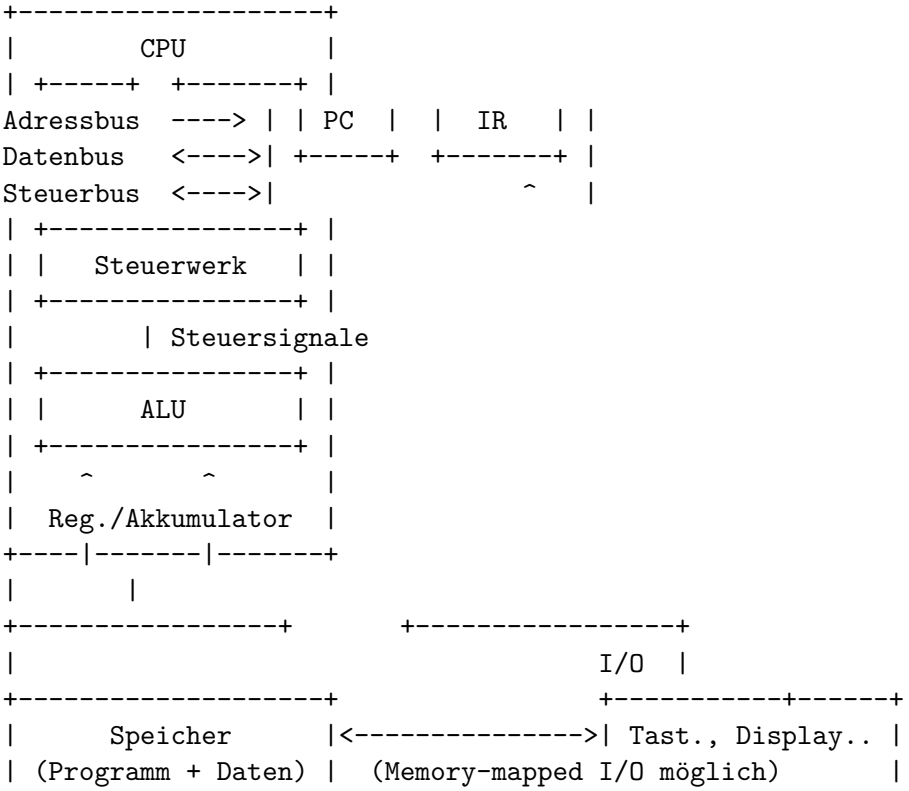
Aufgabe 2:b) **ADD B:** PC→MAR; Instruktion →IR; *Decode*; Speicher[B] →MDR; ALU: Acc := Acc + MDR; Flags aktualisieren (Zero/Carry/Overflow).

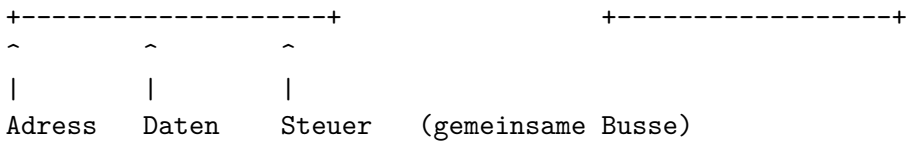
Aufgabe 2:c) **STORE A:** PC→MAR; Instruktion →IR; *Decode*; Acc →MDR; MAR:=A; Speicher schreibt MDR an A.

Wesentlich: PC zählt nach jedem *Fetch* weiter; *Decode* wählt Datenquellen/Ziele; *Execute* verändert Register/Speicher.

Aufgabe 3: Kurzvergleich I-/D-Caches. Was bringen getrennte L1-Caches? Instruktionen (I-Cache) und Daten (D-Cache) können gleichzeitig geholt/geschrieben werden ⇒ weniger Konflikte auf engstem Level, höhere effektive Bandbreite, bessere Ausnutzung von Lokalitäten (zeitlich/räumlich).
Aber: Dahinter bleibt der *gemeinsame* Hauptspeicher und Front-Side-/Memory-Bus; bei Cache-Misses zeigt sich der Von-Neumann-Flaschenhals weiterhin (Latenz/Bandbreite).

Anhang: ASCII-Skizze (Von-Neumann-Architektur)





Didaktischer Hinweis (Klasse 11): Es reicht, die *Rollen* der Bauteile (ALU, Steuerwerk, Speicher, I/O, Busse) und den *Ablauf* (Fetch–Decode–Execute) sicher zu erklären. Details wie Mikrocode, Pipeline-Stufen, Out-of-Order u. Ä. sind Nice-to-Know und werden später vertieft.