

Lernzettel

Pipelines: Architektur, Hazard-Typen,
Forwarding, Stalling und Pipeline-Optimierung

Universität:	Technische Universität Berlin
Kurs/Modul:	Rechnerorganisation
Erstellungsdatum:	September 20, 2025



Zielorientierte Lerninhalte, kostenlos!
Entdecke zugeschnittene Materialien für deine Kurse:

<https://study.AllWeCanLearn.com>

Rechnerorganisation

Lernzettel: Pipelines: Architektur, Hazard-Typen, Forwarding, Stalling und Pipeline-Optimierung

(1) **Architektur einer Pipeline.** Die Grundidee ist, mehrere Instruktionen gleichzeitig in unterschiedlichen Stadien zu bearbeiten. Eine klassische 5-Stufen-Pipeline nutzt die Stufen *IF*, *ID*, *EX*, *MEM*, *WB*. Dabei laufen die Stufen in jedem Zyklus; eine Instruktionskette wird mit Abstand von je einem Zyklus durch die Stufen bearbeitet.

Stufen: IF, ID, EX, MEM, WB

Pipeline: IF → ID → EX → MEM → WB

(2) **Hazard-Typen.** Haupt-Hazards in einer Pipeline sind: - **Data Hazard** (RAW – Read After Write): Ein Operand wird gelesen, bevor der vorherige Befehl ihn geschrieben hat. - **Structural Hazard**: Nicht-genug Ressourcen (z.B. Speicherport, ALU) stehen gleichzeitig zur Verfügung. - **Control Hazard** (Branch-Hazard): Verzweigungen bestimmen die nächste Instruktionsfolge; die Vorhersage kann falsch sein.

(3) **Forwarding (Bypassing).** Forwarding sendet Zwischenergebnisse direkt aus einem späteren Pipeline-Stadium an ein früheres Stadium, um RAW-Hazards ohne Stalling zu eliminieren. Gängige Forwarding-Pfade sind:

Forwarding-Pfad: EX/MEM → EX

Forwarding-Pfad: MEM/WB → EX

Hinweis: Forwarding reduziert die Anzahl der benötigten Stall-Zyklen erheblich, indem es das Ergebnis aus EX oder MEM schon vor dem WB-Stage verfügbar macht.

(4) **Stalling.** Wenn kein Forwarding möglich ist oder eine RAW-Hazard-Lage so liegt, dass kein schneller Pfad existiert, werden Zyklen gestoppt und NOPs eingeschoben. Dies erhöht den effektiven Taktzyklus der Pipeline.

Beispiel: Instruktion 1: ADD R1, R2, R3 Instruktion 2: SUB R4, R1, R5

Ohne Forwarding-Pfad könnte Instruktion 2 erst im EX-Stadium von Instruktion 1 lesen, was zu einem Stopp führt. In vielen Fällen genügt jedoch Forwarding, sodass kein Stalling nötig ist. Falls kein Forwarding existiert, wird 1 Stalzuslag eingeführt, d. h. ein zusätzlicher Zyklus (NOP) wird eingefügt.

(5) **Pipeline-Optimierung.** Wichtige Ansätze zur Reduktion von Verzögerungen und zur Verbesserung der Ausführungsgeschwindigkeit:

- *Forwarding vollständig nutzen*: Entfernt viele RAW-Hazards ohne Stalls.
- *Hazard-Detektions-Einheit*: Erkennt Hazards früh und setzt gezielt NOPs nur dort, wo nötig.
- *Branch Prediction*: Vermeidet Control Hazards durch Vorhersage der Verzweigungsrichtung.
- *Static vs. Dynamic Prediction*: Stabilere Abläufe bei dynamischer Anpassung.
- *2-Bit-Sättigungszähler*: Beispiel-Modell für branch prediction; bei zwei Zuständen wird die Richtung je nach Verlauf angepasst.
- *Misprediction-Kosten*: $C_{\text{mis}} \approx \text{Mißprädien} \times \text{ZyklusKosten}$.
- *Delayschleifen und Schleifenunrollung*: Software-Optimierung kann die Häufigkeit von Verzweigungen senken.
- *Speculative Execution*: Vorwegnahme von Instruktionen in unsicheren Pfaden, mit späterer Verifikation.

Zusammenfassung in Formeln (jeweils own line):

$$\text{Ideal-CPI} \approx 1 \quad (\text{ohne Hazards})$$

$$\text{CPI}_{\text{mit Hazards}} \approx 1 + \text{Stalls/Zyklen}$$

Forwarding hilft gegen RAW-Hazards durch Pfade wie EX/MEM & MEM/WB

$$\text{Branch-Prediction-Kosten bei Fehlvorhersage: } C_{\text{mis}} = p \cdot t$$

$$\text{2-Bit-Sättigungszähler: } \text{State}_{n+1} = f(\text{State}_n, \text{Outcome})$$