ÜBUNG 6

29.05.2026

Grundlagen der Rechnerarchitektur und Betriebssysteme

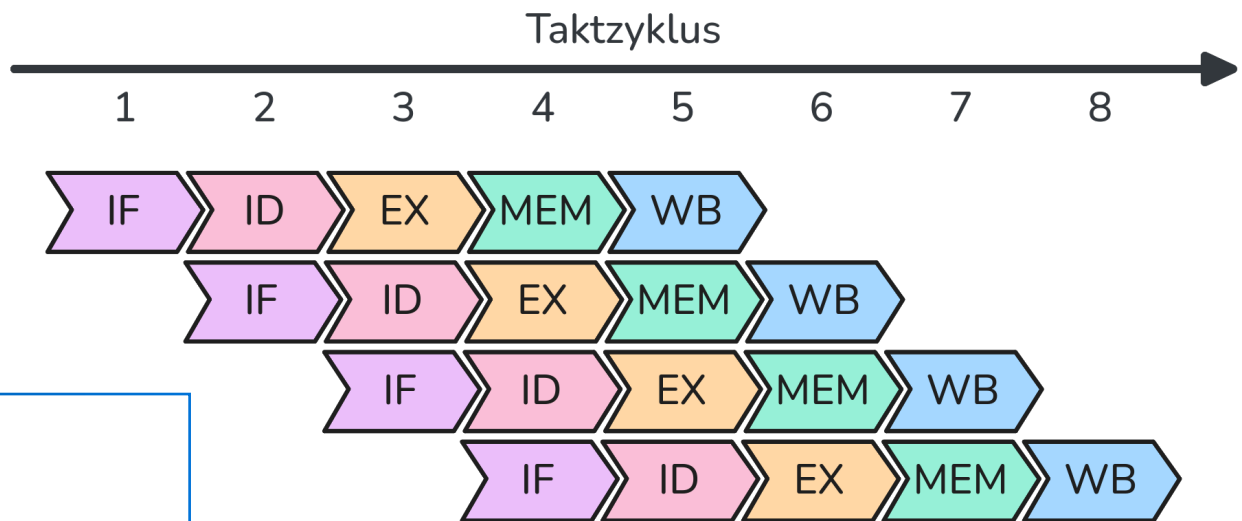
Pipelines und I/O

Florian Knoch · Lehrstuhl für Praktische Informatik,
insbes. Systemnahe Programmierung, Universität Bamberg

Pipelines

Fünfstufige Pipeline

Stufen

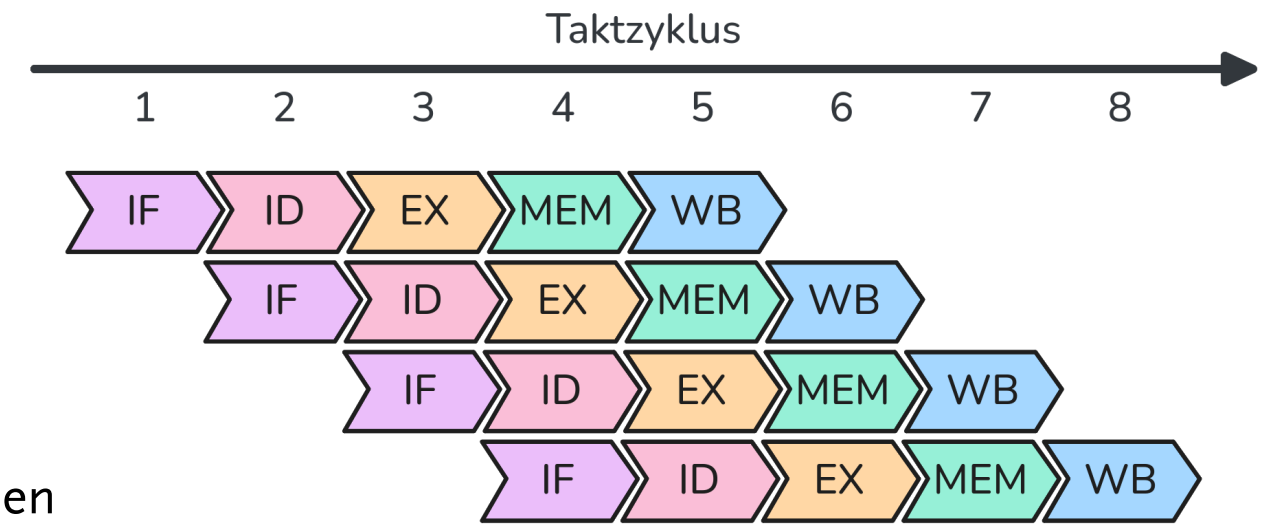


AUFGABE 1A

Nennen und erläutern Sie kurz die fünf aus der Vorlesung bekannten Pipeline-Stufen in der Reihenfolge, in der die Instruktionen durch die Pipeline wandern.

Fünfstufige Pipeline

Stufen



IF – Instruction Fetch: Instruktion holen
ID – Instruction Decode: Instruktion dekodieren
EX – Execute: Instruktion ausführen
MEM – Memory: auf den Speicher zugreifen
WB – Write Back: in Register zurückschreiben

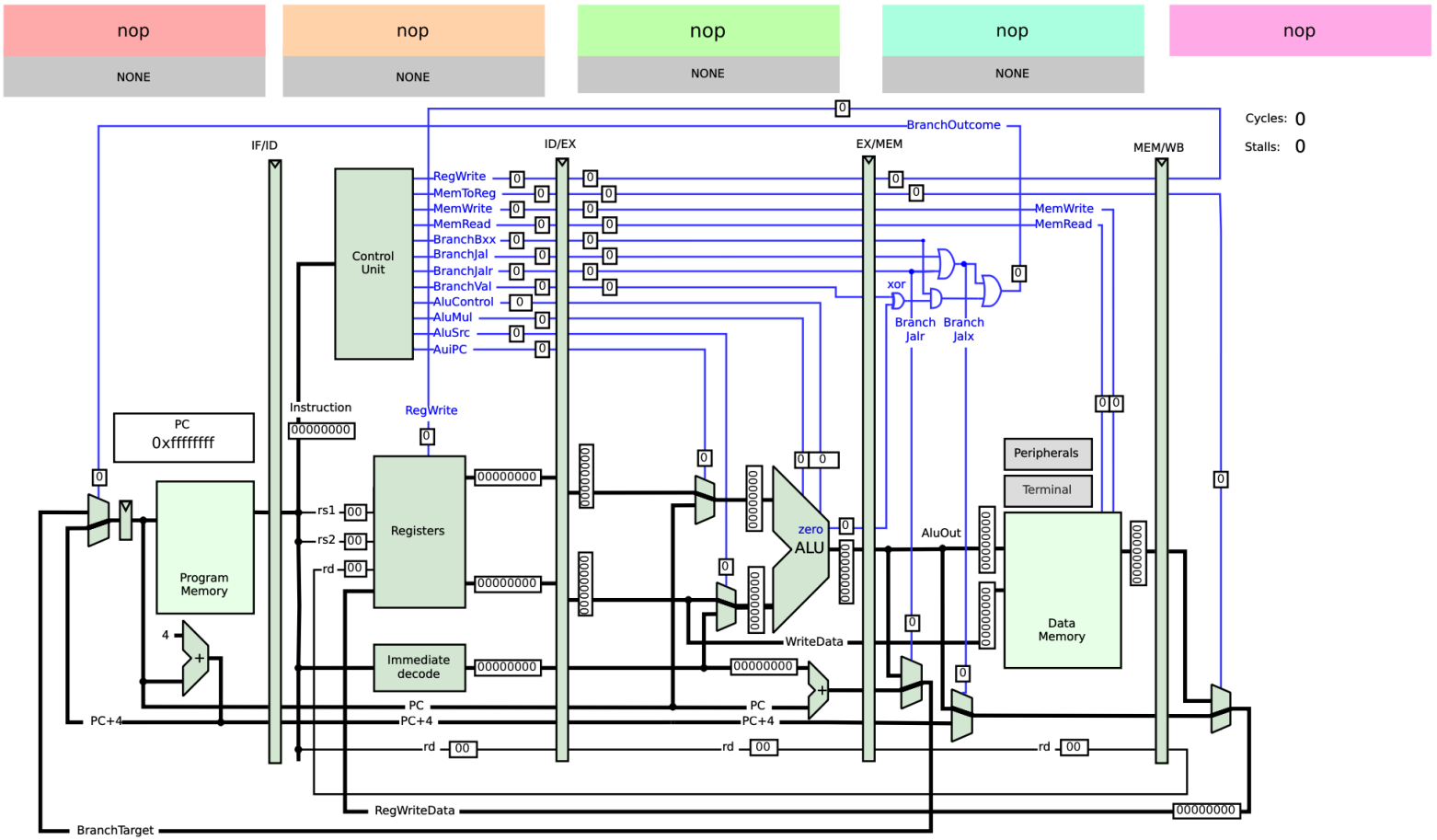


Abbildung 1: Datenpfad mit fünfstufiger Pipeline im **QtRVSim**

Fünfstufige Pipeline

Ausführungszeit

AUFGABE 1B

Wie lange dauert die Ausführung der gegebenen Instruktionen bei sequenzieller Bearbeitung? Welche Zeitersparnis ergibt die fünfstufige Pipeline ohne Hazardbehandlung?

```
addi a0, zero, 2
addi a1, a0, 1
sw a1, 0(a5)
addi a1, zero, 42
```

Fünfstufige Pipeline

Konflikte

AUFGABE 1C

Untersuchen Sie den gegebenen Code auf mögliche Hazards. Geben Sie jeweils die betroffenen Zeilen und Register an.

Hazards

Konflikte in Pipelines

Abhängigkeiten zwischen Instruktionen führen potenziell zu Konflikten (en. *Hazards*)[1]:

```
lui x1, 0x12345
addi x2, x1, 0x678
lui x1, 0xABCDE
```

-
-
-

[1] Beispiel von Werner Haas, entnommen aus SYSNAP-PMAP-M

Hazards

Konflikte in Pipelines

Abhängigkeiten zwischen Instruktionen führen potenziell zu Konflikten (en. *Hazards*)[2]:

```
lui x1, 0x12345
addi x2, x1, 0x678
lui x1, 0xABCDE
```

- **Flow Dependence**: eine Instruktion benötigt das Ergebnis einer vorherigen Instruktion (RAW, read-after-write, Z. 1/2)
-
-

[2] Beispiel von Werner Haas, entnommen aus SYSNAP-PMAP-M

Hazards

Konflikte in Pipelines

Abhängigkeiten zwischen Instruktionen führen potenziell zu Konflikten (en. *Hazards*)[3]:

```
lui x1, 0x12345
addi x2, x1, 0x678
lui x1, 0xABCDE
```

- **Flow Dependence**: eine Instruktion benötigt das Ergebnis einer vorherigen Instruktion (RAW, read-after-write, Z. 1/2)
- **Anti-Dependence**: das Tauschen zweier Instruktionen würde das Ergebnis ändern (WAR, write-after-read, Z. 2/3)
-

[3] Beispiel von Werner Haas, entnommen aus SYSNAP-PMAP-M

Hazards

Konflikte in Pipelines

Abhängigkeiten zwischen Instruktionen führen potenziell zu Konflikten (en. *Hazards*)[4]:

```
lui x1, 0x12345
addi x2, x1, 0x678
lui x1, 0xABCDE
```

- **Flow Dependence**: eine Instruktion benötigt das Ergebnis einer vorherigen Instruktion (RAW, read-after-write, Z. 1/2)
- **Anti-Dependence**: das Tauschen zweier Instruktionen würde das Ergebnis ändern (WAR, write-after-read, Z. 2/3)
- **Output Dependence**: an sich unabhängige Schreibzugriffe an derselben Stelle (WAW, write-after-write, Z. 1/3)

[4] Beispiel von Werner Haas, entnommen aus SYSNAP-PMAP-M

Fünfstufige Pipeline

Konflikte

AUFGABE 1D

Mit wie vielen NOP-Instruktionen könnte ein Compiler die Hazards der vorherigen Aufgabe auflösen? Wie wirkt sich dies auf die Ausführungszeit des Codes aus?

AUFGABE 1E

Kennen Sie eine passende Alternative zur Behandlung dieser Hazards mittels NOP-Instruktionen? Wie könnte dies in diesem Beispiel umgesetzt werden?

Input/Output

LIVE CODING

Datei- und Gerätezugriff in Unix

Relevante C-Funktionen zum Lesen von Dateien [1], [2]

```c
// inspect the file specified by the path
int stat(const char *restrict path, struct stat *restrict statbuf);

// open the file specified by the path
int open(const char *path, int flags);

// read bytes from a file into a buffer
ssize_t read(size_t count; int fd, void buf[count], size_t count);

// position the file offset of an open file
off_t lseek(int fd, off_t offset, int whence);

// close a file descriptor
int close(int fd);
```

Verteilte Zahlen einsammeln

Entwurf in Pseudocode

AUFGABE 2A

Betrachten Sie als gegeben:

> Dateien `1.txt`, `2.txt` … `10000.txt` mit je exakt einer positiven Ganzzahl

Gesucht ist die Summe der Zahlen in allen Textdateien. Formulieren Sie eine passende Lösung in Pseudocode.

Verteilte Zahlen einsammeln

Entwurf in Pseudocode

AUFGABE 2A

Betrachten Sie als gegeben:

> Dateien `1.txt`, `2.txt` … `10000.txt` mit je exakt einer positiven Ganzzahl

Gesucht ist die Summe der Zahlen in allen Textdateien. Formulieren Sie eine passende Lösung in Pseudocode.

```
sum = 0;
for (file_number in [1 .. 10000]) {
  file = file_number + ".txt"
  sum = sum + read_number(file);
}
print(sum);
```

Verteilte Zahlen einsammeln

Naive Umsetzung

AUFGABE 2B

Implementieren Sie den Algorithmus als C-Programm.

```
sum = 0;
for (file_number in [1 .. 10000]) {
  file = file_number + ".txt"
  sum = sum + read_number(file);
}
print(sum);
```

Verteilte Zahlen einsammeln

Naive Umsetzung

AUFGABE 2B

Implementieren Sie den Algorithmus als C-Programm.

AUFGABE 2C

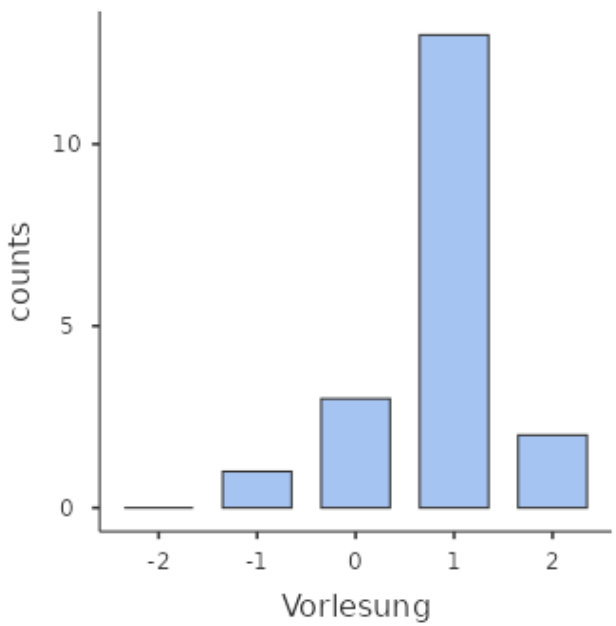
Welche realen Anwendungen können sich hinter diesem *Toy Example* verbergen?

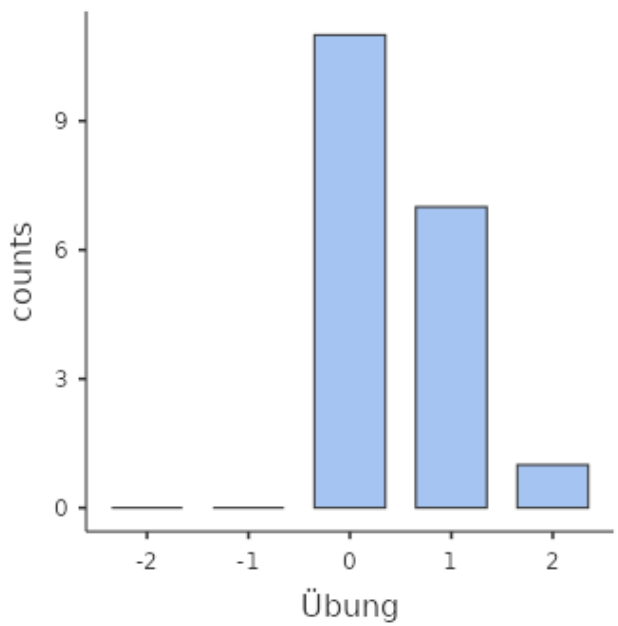
```
sum = 0;
for (file_number in [1 .. 10000]) {
  file = file_number + ".txt"
  sum = sum + read_number(file);
}
print(sum);
```

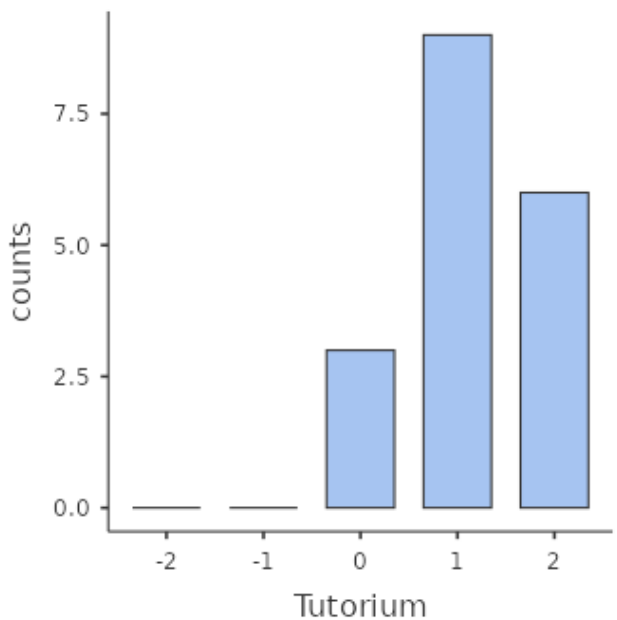
Organisatorisches

Auswertung zum Zwischenfeedback

Erhebungszeitraum: 7.—10. Mai 2026, N=19

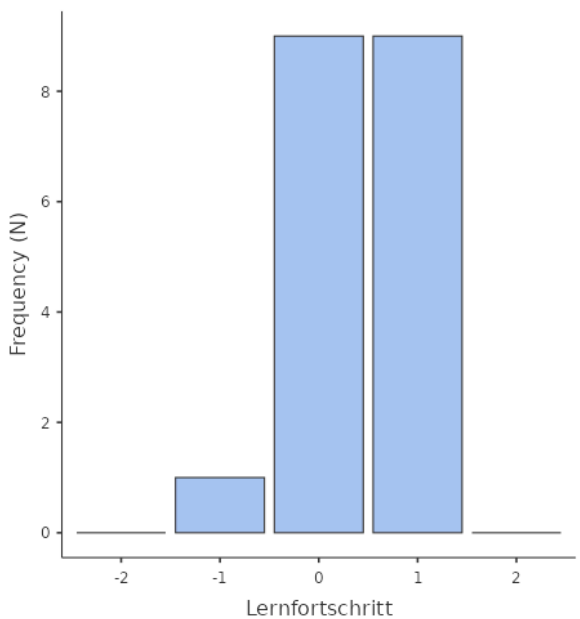


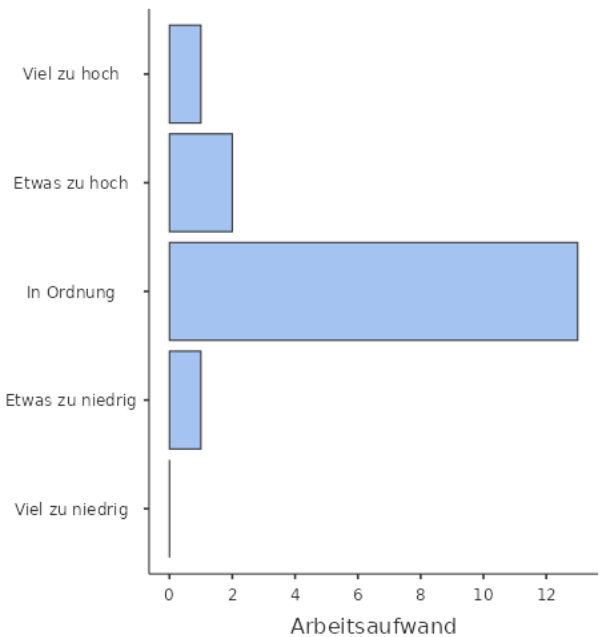


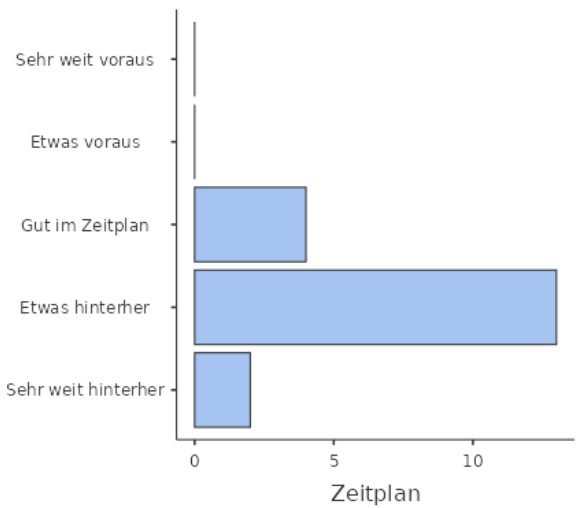


Auswertung zum Zwischenfeedback

Erhebungszeitraum: 7.—10. Mai 2026, N=19







Auswertung zum Zwischenfeedback

Erhebungszeitraum: 7.—10. Mai 2026, N=19

Hauptkritikpunkte

- Vorlesungsaufzeichnungen (2)
- Unverständnis bzgl. Inhalten oder Modus (2)
- zu viele Aufgaben im Tutorium
- Hervorhebung von Inhalten in der Vorlesung
- mehr Erklärungen von Zusammenhängen
- veraltete Vorlesungsfolien

Kursplattform

- Navigation und Übersichtlichkeit (3)
- Namensgebung (Übungsblätter vs. Übungsaufgaben)

Fragen?

Literatur

[1] J. Wolf und R. Krooß, *Systemnahe Programmierung mit C und Linux. Das umfassende Handbuch*. Rheinwerk, 2024.

[2] W. R. Stevens, *Advanced programming in the UNIX environment*. in The Addison-Wesley professional computing series. Addison-Wesley, 2013.

Lizenzhinweise

Kursmaterialien · **Quelltext**

Diese Präsentation wurde zuletzt am 29.05.2026 bearbeitet. Sie basiert auf den Folien der zugehörigen Vorlesung von Prof. Dr. Michael Engel. Sofern nicht anders angegeben, sind die Inhalte unter der **Lizenz CC BY-SA 4.0** verfügbar. Das Universitätslogo sowie die Schriftart UB Scala Sans sind Eigentum der Otto-Friedrich-Universität Bamberg.

Folgende **Open-Source-Komponenten** kommen in der Präsentation zum Einsatz: **circuiteria** (Apache 2.0), **finite** (MIT), **fontawesome** (SIL), **pgf-tikz** (GPL 2.0), **typst** (Apache 2.0), **typst-ccicons** (MIT), **typst-polylux** (MIT).