Karlsruher Institut für Technologie

Rechnerorganisation

Prof. Dr. Wolfgang Karl

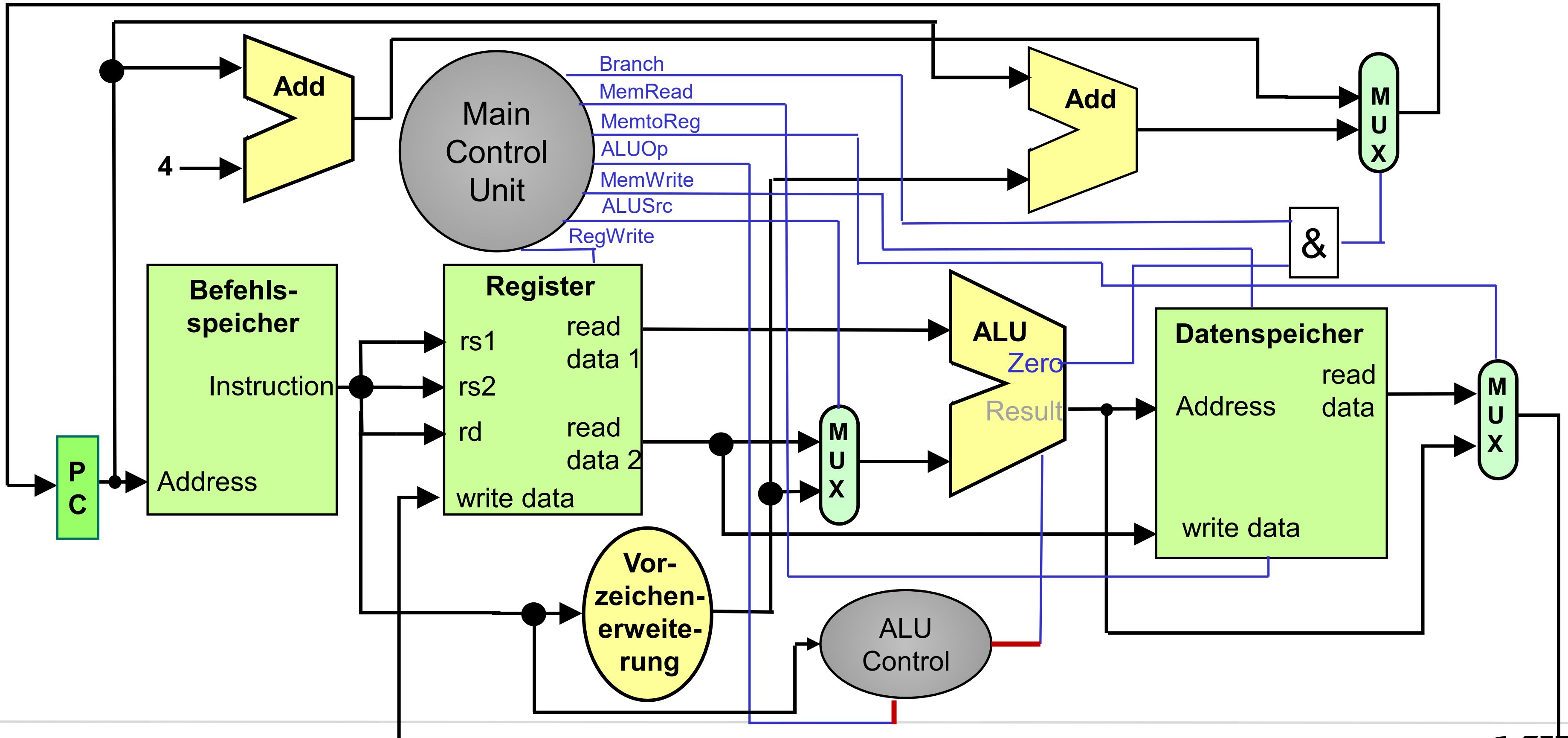
Vorlesung im Wintersemester 2025/2026 – Foliensatz: RO25-FS07

Kapitel 6
Prozessororganisation (Fortsetzung)

- **Grundlagen Pipelining des Maschinenbefehlszyklus**
- Pipeline-Hemmnisse, Pipeline-Konflikte
- Pipeline-Organisation RISC-V

 © Prof. Dr. Wolfgang Karl

Datenpfad für RISC-V: Einzyklen-Implementierung



© Prof. Dr. Wolfgang Karl

Datenpfad für RISC-V: Einzyklen-Implementierung

- **Ausführungsschritte für einen Maschinenbefehl**
  - **Beispiel RISC-V: `add x1,x2,x3`**
    1. Holen des Befehls und Inkrementieren des Inhalts des PCs um 4
    2. Lesen der Register **`x2`** und **`x3`**. Die Main Control Unit setzt die Steuersignale.
    3. Die ALU führt die Operation (**`add`**) auf den von der Registerdatei an den ALU-Eingängen bereitgestellten Operanden aus.
    4. Das Ergebnis wird in das Zielregister **`x1`** geschrieben

  - Die Ausführung der Schritte für einen Befehl erfolgt in einem Taktzyklus.

© Prof. Dr. Wolfgang Karl

Datenpfad für RISC-V: Einzyklen-Implementierung

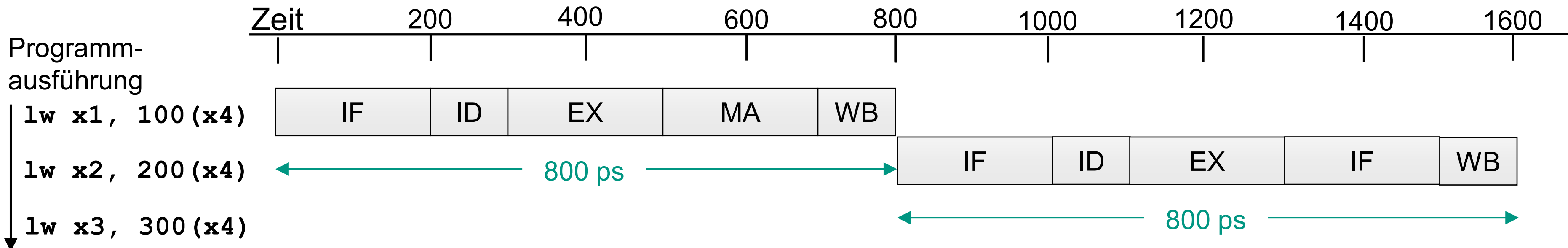
- **Ausführungsschritte für einen Maschinenbefehl**
  - **Beispiel RISC-V:** Ausführungszeiten für die Abarbeitung von Befehlen
    - Laden (`lw`), Speichern (`sw`)
    - Addition (`add`), Subtraktion (`sub`), logisches UND (`and`), logisches ODER (`or`)
    - Verzweigung (`beq`)
    - Ausführungszeiten der einzelnen Schritte in jeder Komponente und Gesamtausführungszeit für jeden Befehl:

Befehls-klasse	Befehl holen	Register lesen	ALU Operation	Datenzugriff	In Register schreiben	Gesamtzeit
`lw`	200 ps	100 ps	200 ps	200 ps	100 ps	**800 ps**
`sw`	200 ps	100 ps	200 ps	200 ps		**700 ps**
`R-Type`	200 ps	100 ps	200 ps		100 ps	**600 ps**
`beq`	200 ps	100 ps	200 ps			**500 ps**

© Prof. Dr. Wolfgang Karl

Datenpfad für RISC-V: Einzyklen-Implementierung

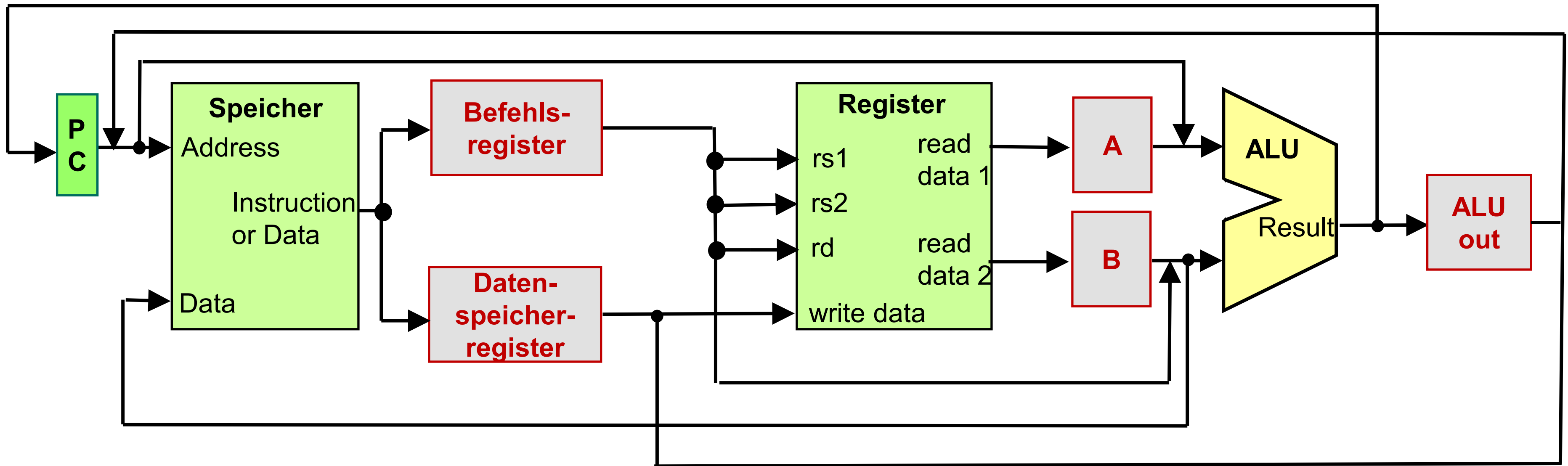
- **Ausführungsschritte für einen Maschinenbefehl**
  - **Beispiel RISC-V:** Ausführungszeiten für die Abarbeitung von Befehlen
    - Der Taktzyklus richtet sich nach dem längsten Befehl: 800ps



- $\mathbf{CPU - Zeit = IC * CPI * Zyklendauer = IC * 1 * 800ps}$
- Zeit zwischen dem 1. und 4. Befehl: $\mathbf{3 \times 800\boldsymbol{ps} = 2400\boldsymbol{ps}}$

© Prof. Dr. Wolfgang Karl

Datenpfad für RISC-V: Mehrzyklen-Implementierung



© Prof. Dr. Wolfgang Karl

Datenpfad für RISC-V: Mehrzyklen-Implementierung

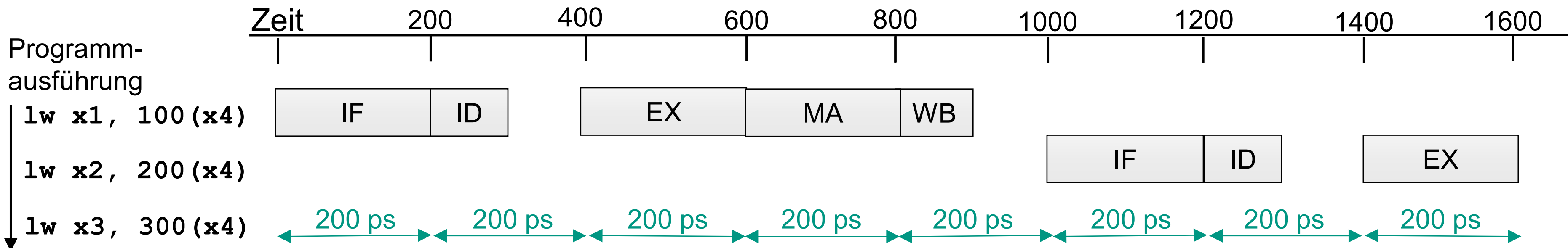
- **Ausführungsschritte für einen Maschinenbefehl**
  - **Beispiel RISC-V:** Ausführungszeiten für die Abarbeitung von Befehlen
    - Laden (`lw`), Speichern (`sw`)
    - Addtion (`add`), Subtraktion (`sub`), logisches UND (`and`), logisches ODER (`or`)
    - Verzweigung (`beq`)
    - Ausführungszeiten der einzelnen Schritte in jeder Komponente und Gesamtausführungszeit für jeden Befehl:

Befehls-klasse	Befehl holen	Register lesen	ALU Operation	Datenzugriff	In Register schreiben	Gesamtzeit
`lw`	200 ps	100 ps	200 ps	200 ps	100 ps	**800 ps**
`sw`	200 ps	100 ps	200 ps	200 ps		**700 ps**
`R-Type`	200 ps	100 ps	200 ps		100 ps	**600 ps**
`beq`	200 ps	100 ps	200 ps			**500 ps**

siehe Tabelle auf Folie 5

Datenpfad für RISC-V: Mehrzyklen-Implementierung

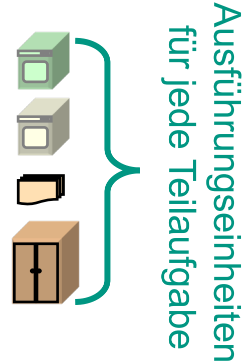
- **Ausführungsschritte für einen Maschinenbefehl**
  - **Beispiel RISC-V:** Ausführungszeiten für die Abarbeitung von Befehlen
    - Der Taktzyklus richtet sich nach dem längsten Arbeitsschritt: 200ps

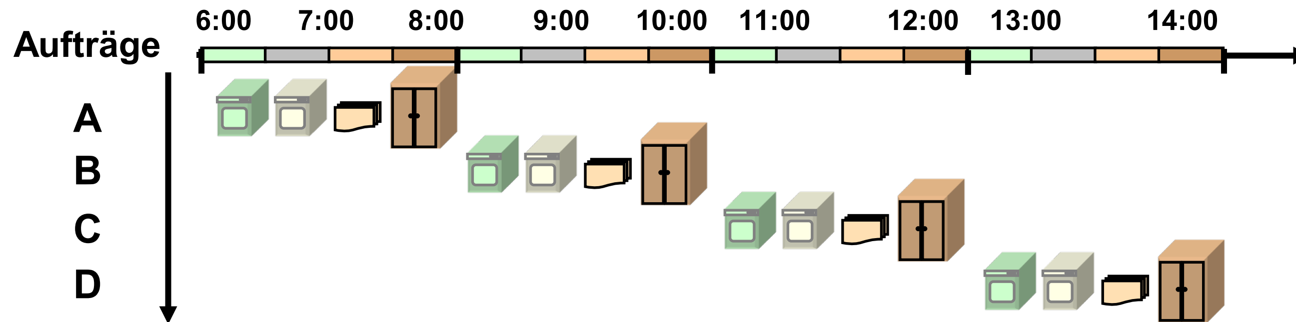


- $\mathbf{CPU - Zeit = IC * CPI * Zyklendauer = IC * 5 * 200ps}$
- Zeit zwischen dem 1. und 4. Befehl: $\mathbf{3 \times 1000\textit{ps} = 3000\textit{ps}}$

Pipelining

- **Beispiel: Waschvorgang**
  - Umfasst vier Teilaufgaben:
    - Schmutzige Wäsche in die Waschmaschine geben und waschen lassen
    - Nasse Wäsche in den Trockner geben und trocknen lassen
    - Bügeln, falten, ...
    - Saubere Wäsche in den Schrank aufräumen





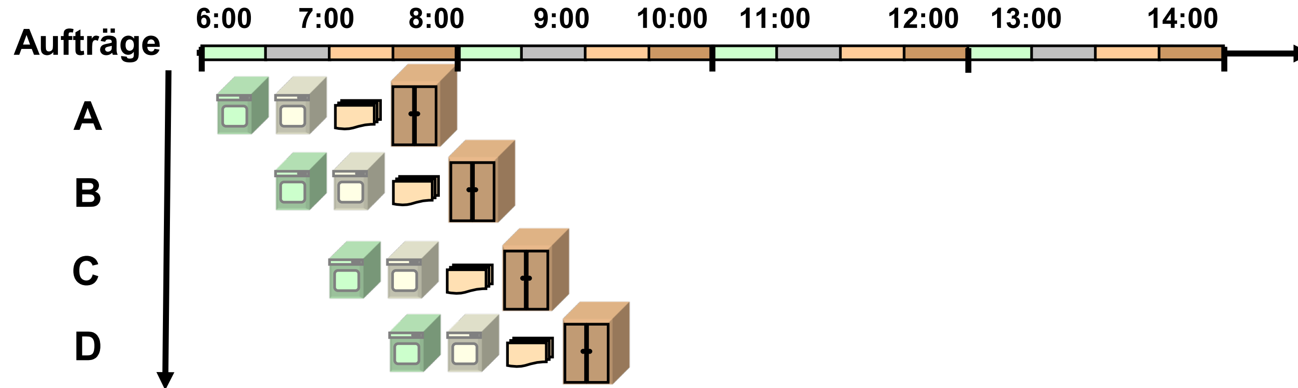
Jede Teilaufgabe benötigt 30 min.

1 Waschladung benötigt 2 Stunden.

4 Waschladungen benötigen 8 Stunden.

Pipelining

- **Beispiel: Waschvorgang**
  - Überlappte Verarbeitung



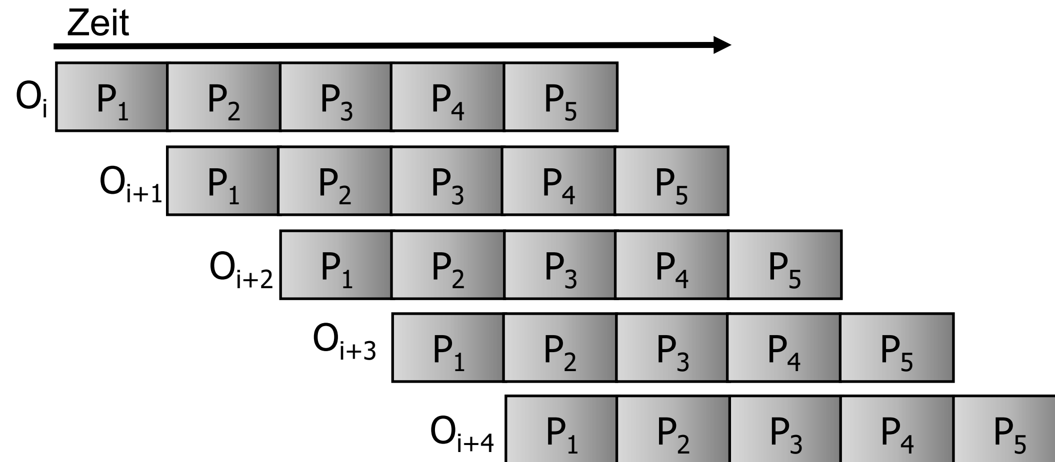
1 Waschladung benötigt 2 Stunden

4 Waschladungen benötigen 3,5 Stunden.

© Prof. Dr. Wolfgang Karl

Pipelining

- **Grundprinzip**
  - **Pipelining** auf einer Maschine liegt dann vor, wenn die Bearbeitung eines Objektes (O_i) in Teilschritte zerlegt und diese in einer sequentiellen Folge (Phasen der Pipeline, P_j) ausgeführt werden. Die Phasen der Pipeline können für verschiedene Objekte überlappt abgearbeitet werden.



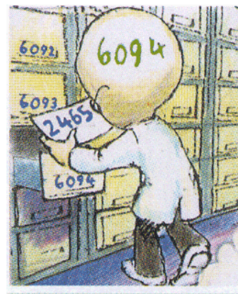
© Prof. Dr. Wolfgang Karl

Pipelining

- **Befehlsabarbeitung**
  - **Arbeitsschritte**

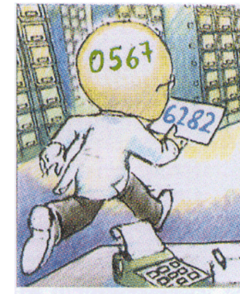
Befehl holen

Befehl dekodieren



Operanden bereitstellen

Operation ausführen



Ergebnis speichern

Alle Aufgaben müssen der Reihe nach bearbeitet werden.

© Prof. Dr. Wolfgang Karl

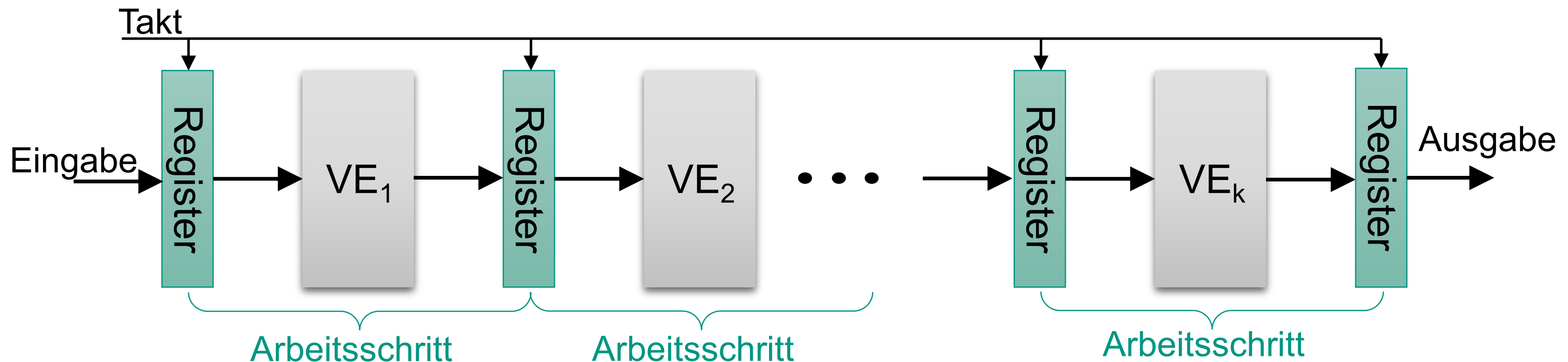
Pipelining

- **Befehlsabarbeitung**
  - **Arbeitsschritte, Phasen des Maschinenbefehlszyklus**
    1. Hole Befehl aus dem Speicher (Befehl holen, instruction fetch, **IF**)
    2. Dekodiere Befehl und lese Register (Befehl dekodieren, instruction decode, **ID**)
    3. Führe Operation aus oder berechne Adresse (Befehl ausführen, instruction execute, **EX**)
    4. Greife auf Datum im Speicher zu, falls notwendig (Speicherzugriff, memory accesss, **MA**)
    5. Schreibe Ergebnis in ein Register, falls notwendig (Ergebnis in Register schreiben, write back, **WB**)

© Prof. Dr. Wolfgang Karl

Pipelining

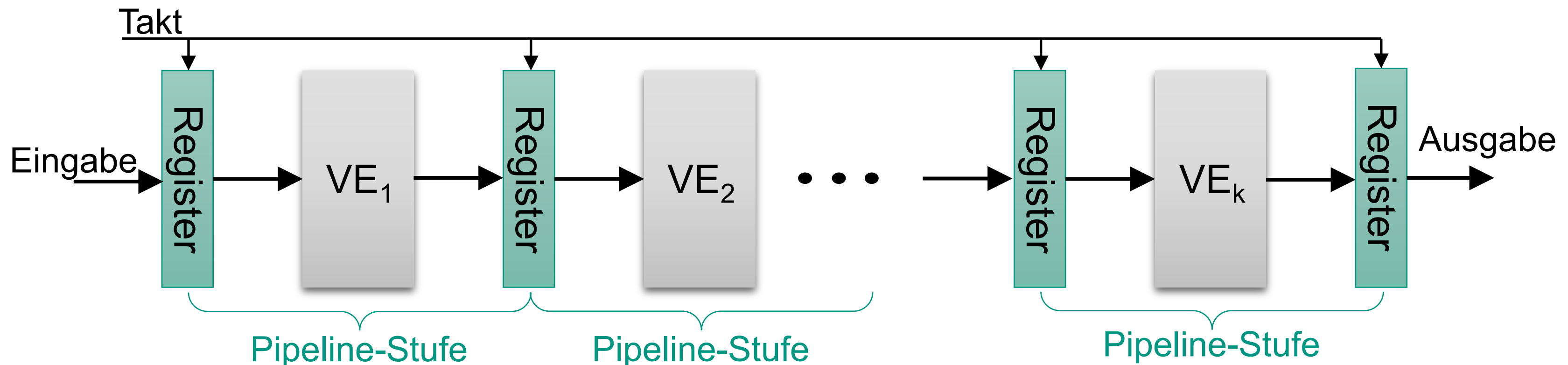
- **Befehlsabarbeitung**
  - **Datenpfad für RISC-V: Mehrzyklen-Implementierung**
    - Gesamtheit der Verarbeitungseinheiten VE_1, VE_2, …, VE_k
    - Arbeitsschritte sind durch Register getrennt: taktsynchrone Verarbeitung;



© Prof. Dr. Wolfgang Karl

Pipelining

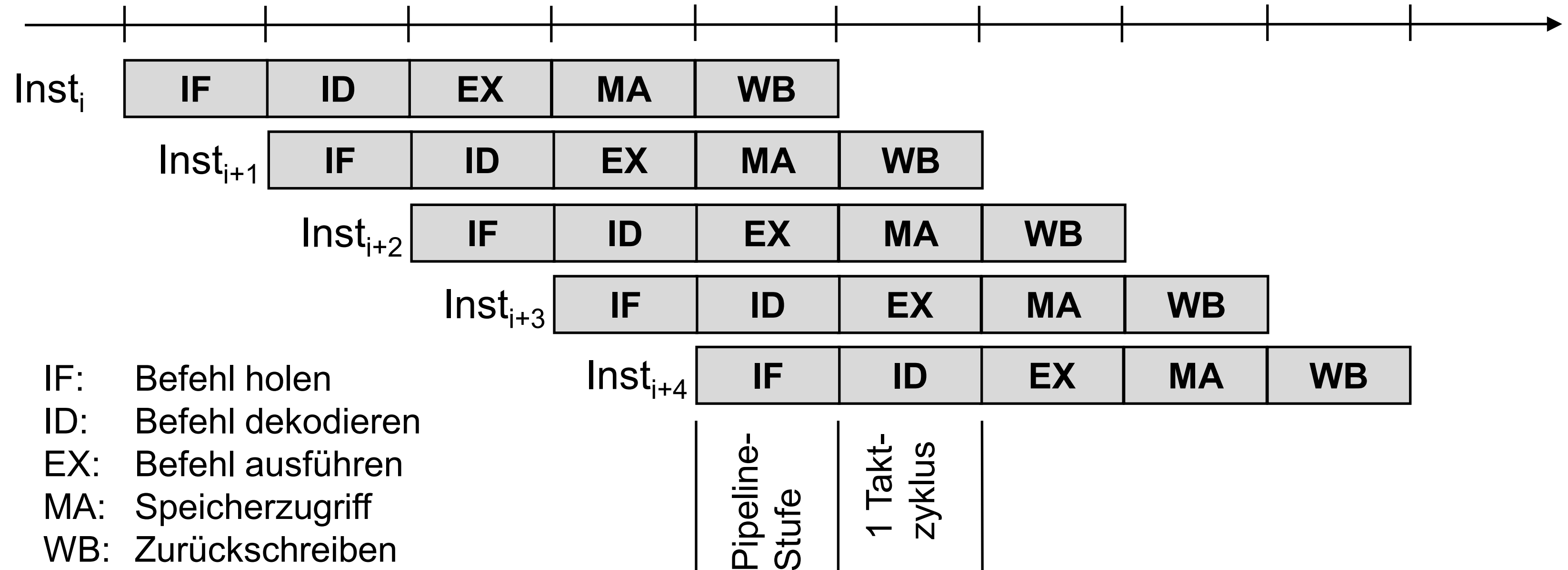
- **Befehlsabarbeitung**
  - **Pipelining des Maschinenbefehlszyklus (Instruction Pipelining)**
    - Gesamtheit der Verarbeitungseinheiten VE_1, VE_2, …, VE_k
    - Pipeline-Stufen sind durch Register getrennt: taktsynchrone Verarbeitung;
    - **k-stufige Pipeline:**



© Prof. Dr. Wolfgang Karl

Pipelining

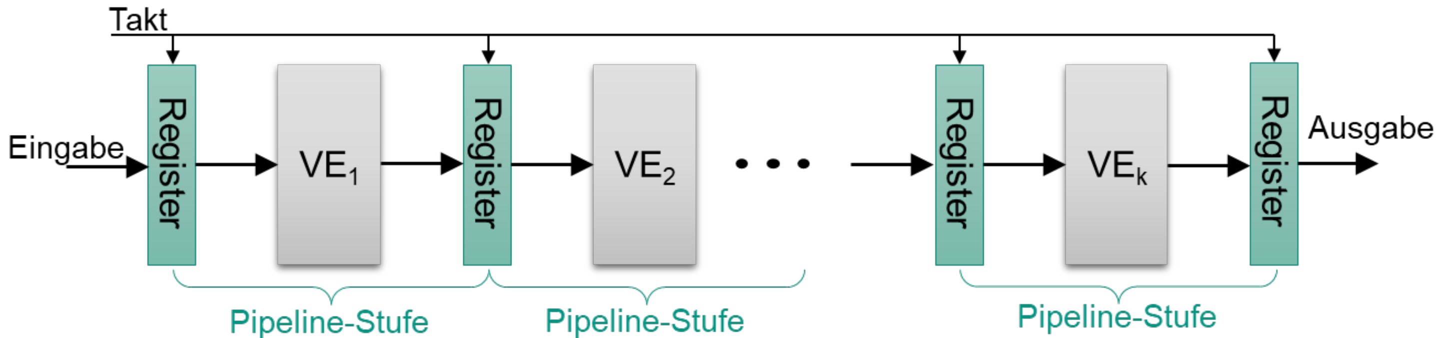
- **Pipelining des Maschinenbefehlszyklus (Instruction Pipelining)**
  - **k-stufige Pipeline (hier k=5)**



© Prof. Dr. Wolfgang Karl

Pipelining des Maschinenbefehlszyklus

- **Entwurfsaspekt: Dauer der Pipeline-Stufe**
  - **k-stufige Pipeline (hier k=5):**



- **Verzögerungszeiten**
  - der Verarbeitungseinheiten: $t_i \ (i = 1, 2, \dots, k)$
  - der Pipeline-Register: t_{reg}
  - ➢ Länge Taktzyklus: $t = max\{t_1, t_2, \dots, t_k\} + t_{reg}$

© Prof. Dr. Wolfgang Karl

Pipelining des Maschinenbefehlszyklus

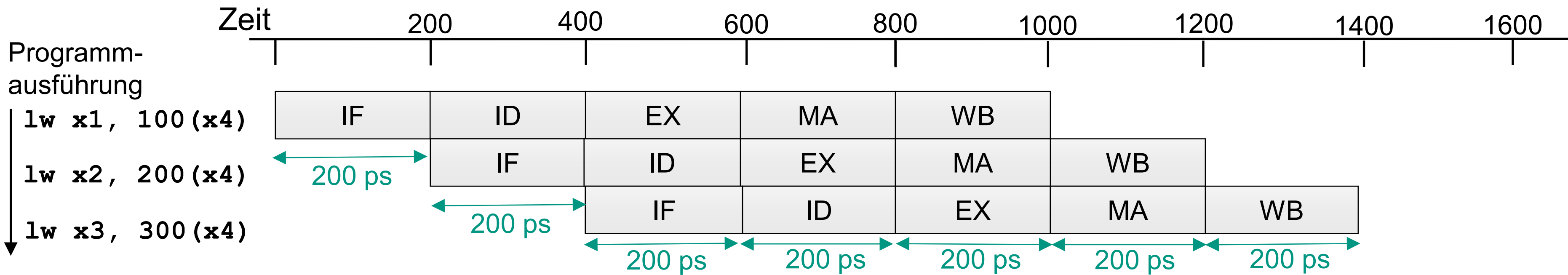
- **Entwurfsaspekt: Dauer der Pipeline-Stufe**
  - **Beispiel RISC-V:** Ausführungszeiten für die Abarbeitung von Befehlen
    - Laden (**`lw`**), Speichern (**`sw`**)
    - Addtion (**`add`**), Subtraktion (**`sub`**), logisches UND (**`and`**), logisches ODER (**`or`**)
    - Verzweigung (**`beq`**)
    - Ausführungszeiten in den Verarbeitungseinheiten:

Befehls-klasse	Befehls holen	Register lesen	ALU Operation	Datenzugriff	In Register schreiben	Gesamtzeit
`lw`	200 ps	100 ps	200 ps	200 ps	100 ps	**800 ps**
`sw`	200 ps	100 ps	200 ps	200 ps		**700 ps**
`R-Type`	200 ps	100 ps	200 ps		100 ps	**600 ps**
`beq`	200 ps	100 ps	200 ps			**500 ps**

siehe Tabelle auf Folie 5

Pipelining des Maschinenbefehlszyklus

- **Leistungsbetrachtung: Pipeline-Verarbeitung**
  - **Beispiel RISC-V: `lw` (load word)**
    - Taktzyklus richtet sich nach der langsamsten Pipeline-Stufe: $200ps$



- $\mathbf{CPU - Zeit = IC * CPI * Zyklendauer = IC * 1 * 200ps}$
- Zeit zwischen dem 1. und 4. Befehl: $3 \times 200ps = 600ps$

© Prof. Dr. Wolfgang Karl

Pipelining des Maschinenbefehlszyklus

- **Leistungsbetrachtung: Pipeline-Verarbeitung**
  - **Ausführungszeiten eines Programms mit *n* Befehlen**
    - **Sequentielle Ausführung:** $n \times k$ Taktzyklen (bei k Verarbeitungseinheiten)
    - **Pipeline-Verarbeitung auf Prozessor mit k-stufiger Pipeline:** $n + (k - 1)$ Taktzyklen
      - n Taktzyklen, um alle n Befehle der Pipeline zuzuführen
      - $(k - 1)$ zusätzliche Taktzyklen, um den letzten Befehl fertigzustellen (Abklingen der Pipeline)
    - **Beschleunigung (Speedup, S):** $S = \frac{n \times k}{n+(k-1)} \approx k$

Unter idealen Bedingungen und mit einer großen Anzahl von Befehlen entspricht die Beschleunigung in etwa der Anzahl der Pipeline-Stufen.

Pipelining des Maschinenbefehlszyklus

- **Kriterien für den Entwurf einer RISC Befehlssatzarchitektur (siehe Diskussion CISC / RISC Foliensatz: RO25-FS05, Folie 12):**
  - Einfache Maschinenbefehle
    - Einheitliches und festes Befehlsformat
  - Load/Store Architektur
    - Befehle arbeiten auf Registeroperanden
    - Lade- und Speicherbefehle greifen auf Speicher zu
  - Einzyklus-Maschinenbefehle
    - Einheitliches Zeitverhalten der Maschinenbefehle, wovon nur Lade- und Speicherbefehle sowie die Verzweigungsbefehle abweichen
  - Optimierende Compiler
    - Reduzierung der Befehle im Programm
  - ➢ **Effizientes Pipelining des Maschinenbefehlszyklus**

© Prof. Dr. Wolfgang Karl

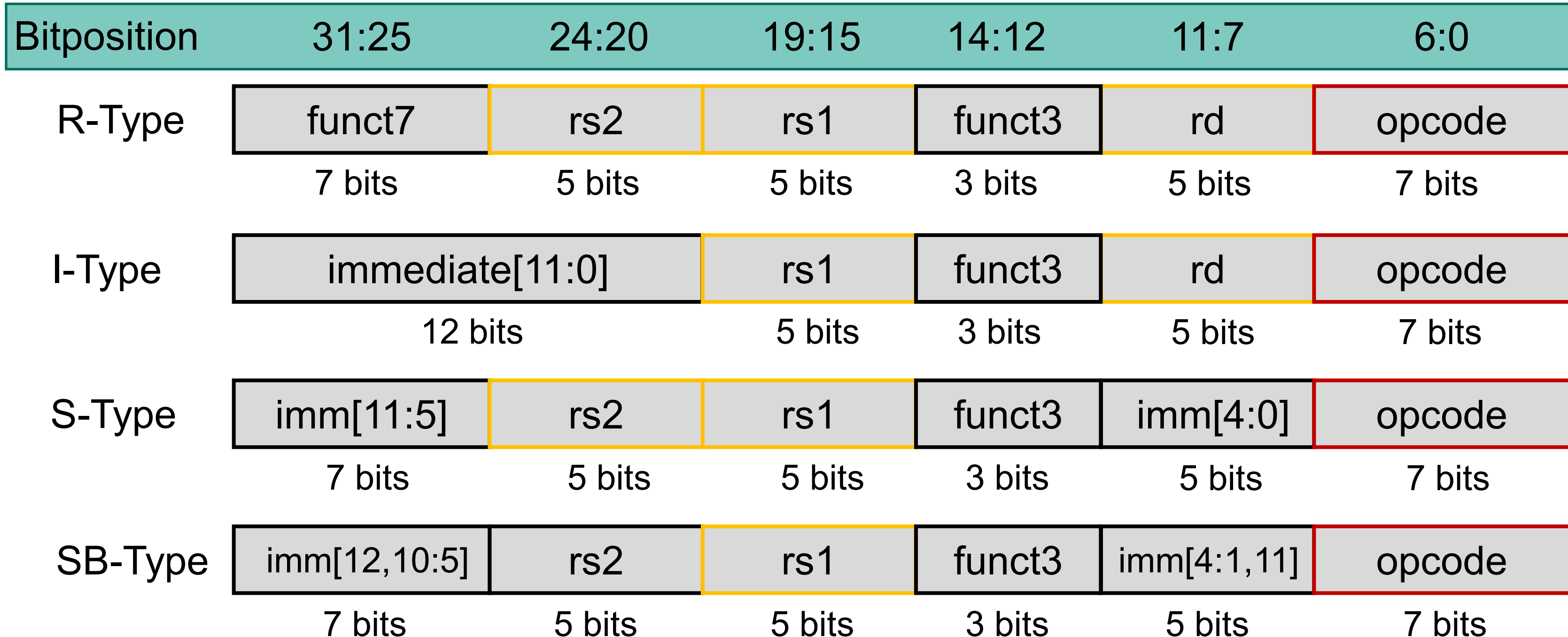
Pipelining des Maschinenbefehlszyklus

- **Entwurf der Befehlssatzarchitektur für RISC-V**
  - **Einfache Maschinenbefehle:**
    - **Einheitliches und festes Befehlsformat**
      - Alle Befehle sind 32 Bit lang.
      - ➢ Ein Befehl kann in einem Taktzyklus vollständig geholt werden (IF-Stufe).

      - Felder für Opcode, Registeradressen und Immediate-Werte sind in den verschiedenen Formaten in den gleichen Feldern (siehe nachfolgende Folie).
      - ➢ Vereinfachte Dekodierung;
      - ➢ Das Dekodieren des Befehls und das Lesen der Register kann in einem Taktzyklus erfolgen (ID-Stufe).

Pipelining des Maschinenbefehlszyklus

- **Entwurf der Befehlssatzarchitektur für RISC-V**
  - **Einheitliches und festes Befehlsformat**



Bitposition	31:25	24:20	19:15	14:12	11:7	6:0
R-Type	funct7	rs2	rs1	funct3	rd	opcode
	7 bits	5 bits	5 bits	3 bits	5 bits	7 bits
I-Type	immediate[11:0]		rs1	funct3	rd	opcode
	12 bits		5 bits	3 bits	5 bits	7 bits
S-Type	imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
	7 bits	5 bits	5 bits	3 bits	5 bits	7 bits
SB-Type	imm[12,10:5]	rs2	rs1	funct3	imm[4:1,11]	opcode
	7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

Pipelining des Maschinenbefehlszyklus

- **Entwurf der Befehlssatzarchitektur für RISC-V**
  - **Einzyklus-Maschinenbefehle**
    - Einheitliches Zeitverhalten der Maschinenbefehle, wovon nur Lade- und Speicherbefehle sowie die Verzweigungsbefehle abweichen;
    - Befehlsausführung für einfache Befehle dauert 1 Takltzyklus (EX-Stufe)
    - Mit jedem Taktzyklus beendet ein Befehl die Bearbeitung in der Pipeline (nach dem Einschwingen der Pipeline und unter idealen Bedingungen)

© Prof. Dr. Wolfgang Karl

Pipelining des Maschinenbefehlszyklus

- **Entwurf der Befehlssatzarchitektur für RISC-V**
  - **Load/Store Architektur**
    - Befehle arbeiten auf nur auf Operanden in Registern;
    - Nur ausgezeichnete Lade- und Speicherbefehle greifen auf den Speicher zu.
    - Berechnung der Speicheradresse in EX-Stufe
    - Speicherzugriff in der nächsten Stufe (MA-Stufe)

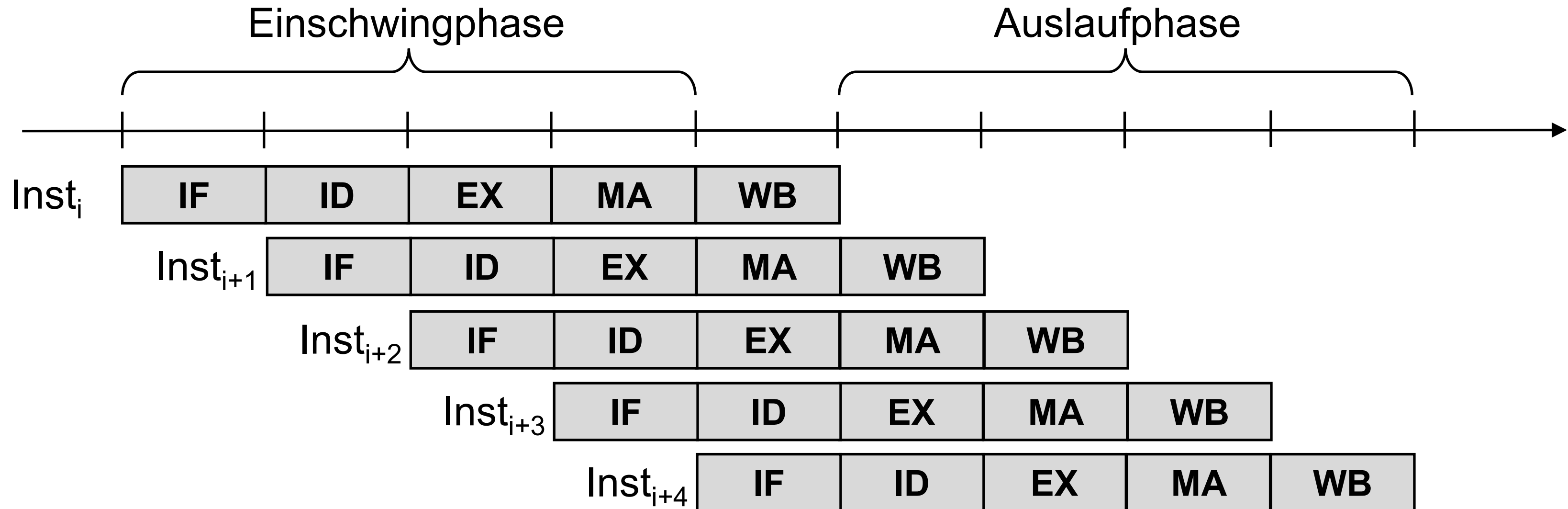
Pipelining des Maschinenbefehlszyklus

- **Entwurf der Befehlssatzarchitektur für RISC-V**
  - **Ziel. Effizientes Pipelining des Maschinenbefehlszyklus**
    - **Cycles per Instruction: $CPI \approx 1$**

© Prof. Dr. Wolfgang Karl

Pipelining des Maschinenbefehlszyklus

- **Effizientes Pipelining des Maschinenbefehlszyklus**
  - **Warum Cycles per Instruction $CPI \approx 1$**
    - Einschwing- und Auslaufphase:



© Prof. Dr. Wolfgang Karl

Pipelining des Maschinenbefehlszyklus

- **Effizientes Pipelining des Maschinenbefehlszyklus**
  - **Warum Cycles per Instruction $CPI \approx 1$**
    - Pipeline-Konflikte, Pipeline-Hemmnisse

© Prof. Dr. Wolfgang Karl

Kapitel 6
Prozessororganisation (Fortsetzung)

- Grundlagen Pipelining des Maschinenbefehlszyklus
- **Pipeline-Hemmnisse, Pipeline-Konflikte**
- Pipeline-Organisation RISC-V

© Prof. Dr. Wolfgang Karl

Pipelining des Maschinenbefehlszyklus

- **Pipeline-Konflikte (Hazards, Pipeline-Hemmnisse)**
  - Situationen in der Pipeline, in denen die nächste Instruktion nicht im folgenden Taktzyklus weiter verarbeitet werden kann.
  - Können aufgrund der überlappten Abarbeitung von Befehlen in einer Pipeline auftreten.

  - **3 Arten von Konflikten:**
    - **Strukturkonflikte**
    - **Datenkonflikte**
    - **Steuerkonflikte**

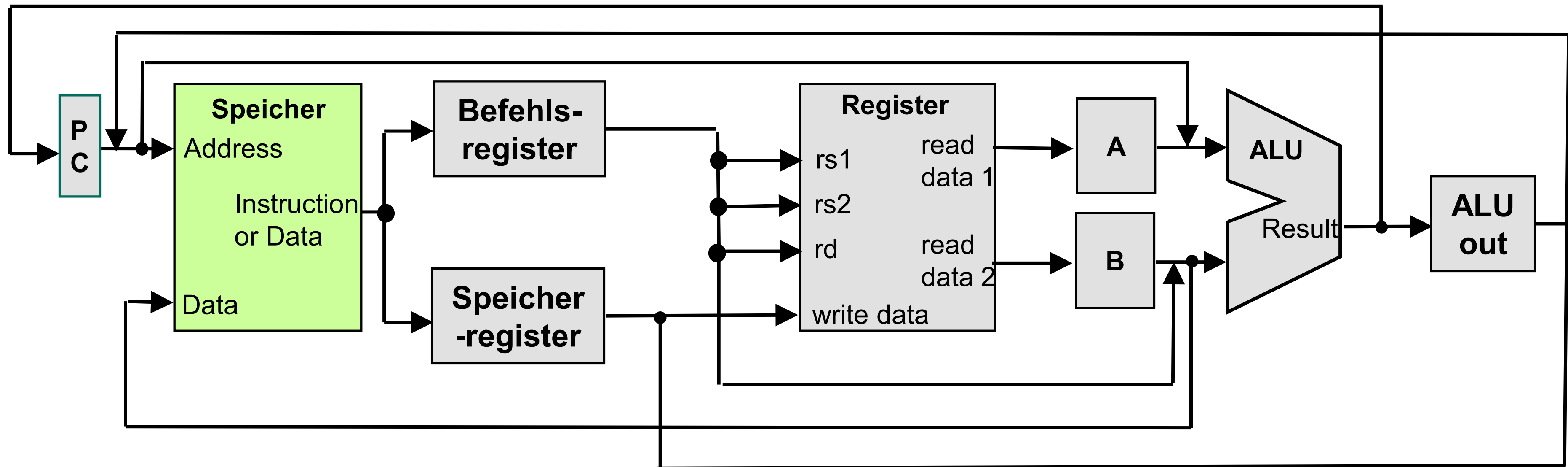
Pipeline-Konflikte

- **Strukturkonflikte oder Ressourcenkonflikte**
  - Eine Instruktion kann nicht im vorgesehenen Taktzyklus ausgeführt werden, da die Hardware die überlappte Bearbeitung von zwei oder mehr Befehlen in diesem Takt nicht unterstützt.
  - **Ursache:**
    - Zwei oder mehr Instruktionen in unterschiedlichen Stufen benötigen im selben Taktzyklus dieselbe Ressource, auf die aber nur einmal zugegriffen werden kann.
  - **Ziel beim Pipeline-Entwurf**
    - Ressourcenkonflikte möglichst vermeiden!

© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

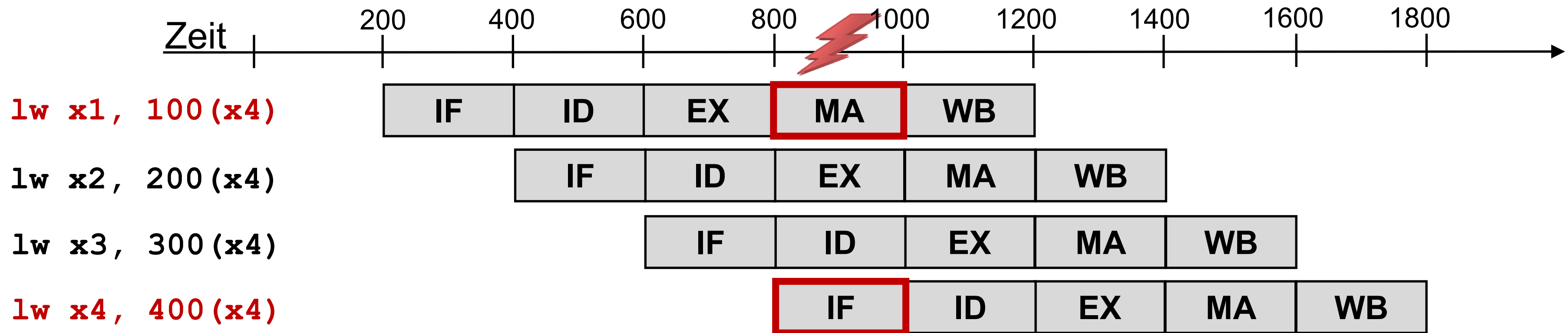
- **Strukturkonflikte oder Ressourcenkonflikte**
  - Beispiel: Datenpfad für RISC-V für Mehrzyklen-Implementierung
    - Ein Speicher für Befehle und Daten:



KIT

Pipeline-Konflikte

- **Strukturkonflikte oder Ressourcenkonflikte**
  - Beispiel: Datenpfad für RISC-V für Mehrzyklen-Implementierung
    - Ein Speicher für Befehle und Daten:



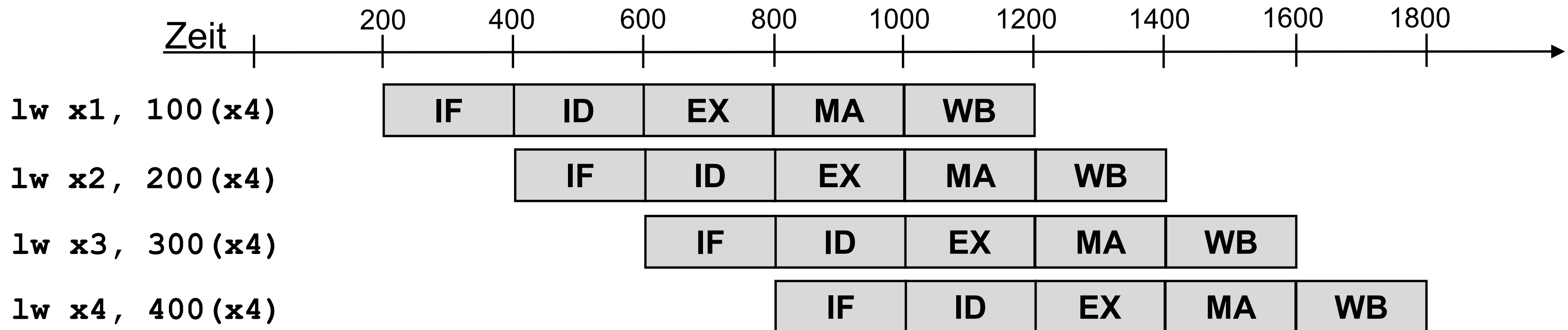
Strukturkonflikt:

Speicherzugriff des 1. Befehls und Holen des 4. Befehls aus dem Speicher im selben Taktzyklus

© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

- **Strukturkonflikte oder Ressourcenkonflikte**
  - Beispiel: Datenpfad für RISC-V für Mehrzyklen-Implementierung
    - Ein Speicher für Befehle und Daten:



Lösung beim Entwurf:
Getrennte Speicher für Befehle und für Daten

Pipeline-Konflikte

- **Datenkonflikte**
  - können zwischen zwei Befehlen in der Pipeline auftreten.
  - **Ursache: Echte Datenabhängigkeiten**
    - **Definition:**
      - Zwischen zwei Befehlen `i1` und `i2`, wobei `i2` dem Befehl `i1` im Programm folgt, besteht eine **echte Datenabhängigkeit (true dependence)** von `i1` zu `i2`, wenn `i1` seine Ausgabe in ein Register (oder Speicherstelle) schreibt, das von `i2` als Eingabe gelesen wird.
    - **Beispiel:**

```
i1: add x19,x0,x1   # x19 := x0+x1
 :
i2: sub x2,x19,x3   # x2  := x19-x3
```

echte Datenabhängigkeit

© Prof. Dr. Wolfgang Karl

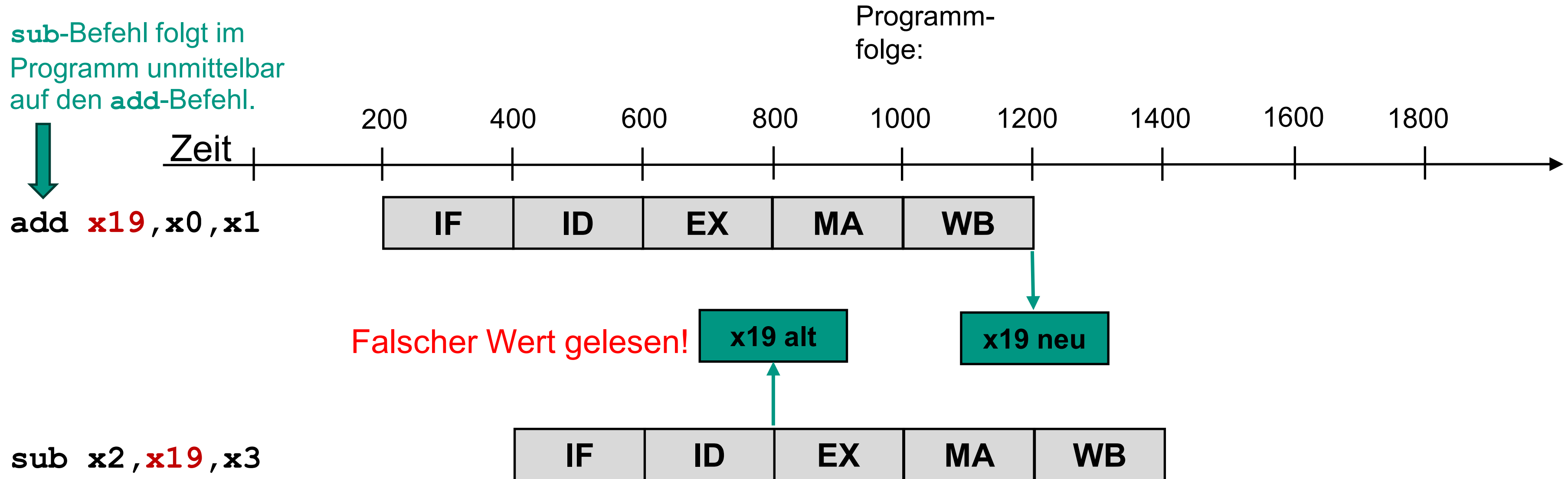
Pipeline-Konflikte

- **Datenkonflikte**
  - **Echte Datenabhängigkeiten**
    - Sind Eigenschaften des Programms!
    - Zeigen die Möglichkeit eines Konflikts in einer Pipeline an!

© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

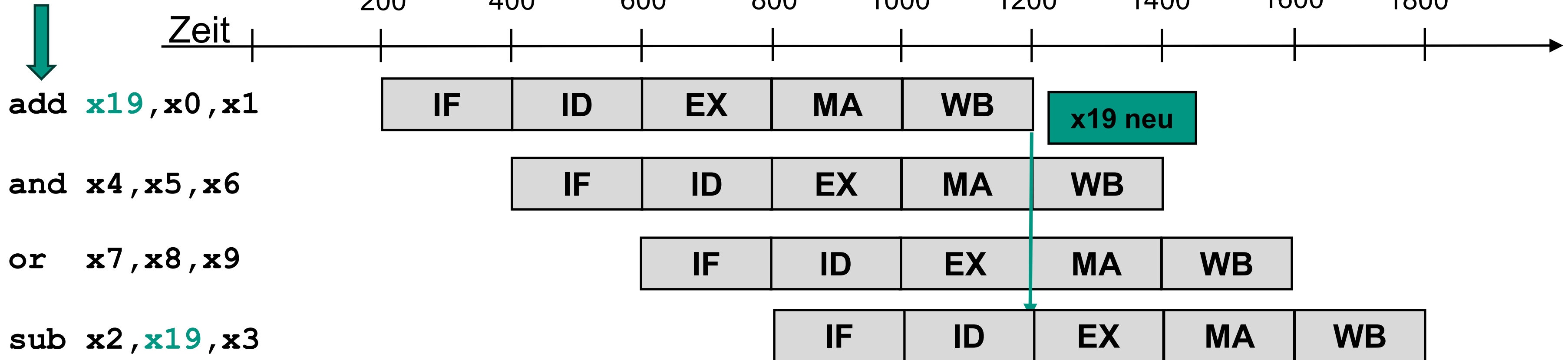
- **Datenkonflikte**
  - **Lese-nach-Schreibe-Konflikt (read-after-write, RAW)**
    - Verursacht aufgrund der echten Datenabhängigkeit zwischen `i1` und `i2`.



© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

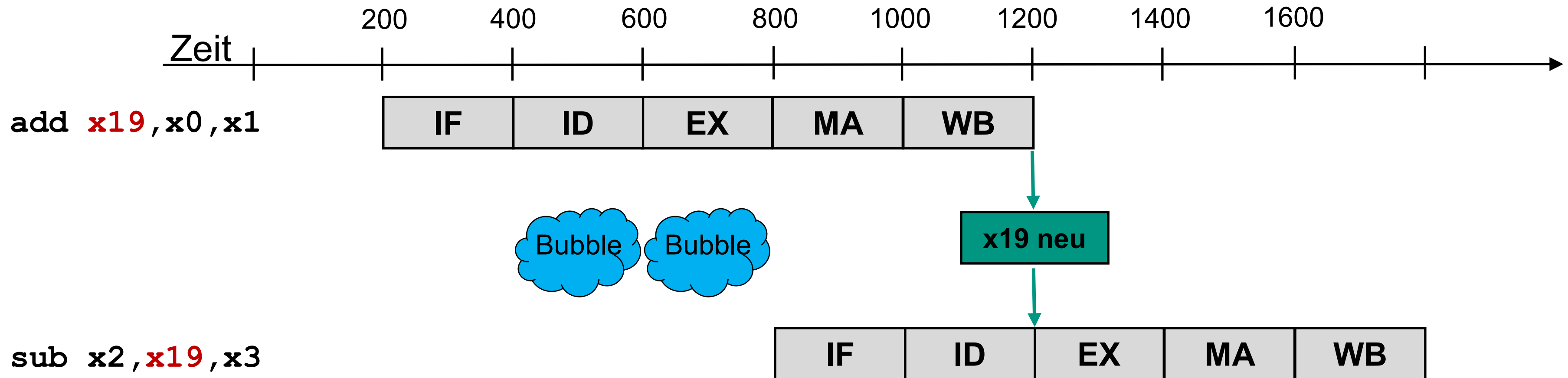
- **Datenkonflikte**
  - **Kein Lese-nach-Schreibe-Konflikt (RAW)**
    - Echte Datenabhängigkeit zwischen `i1` und `i2` weiterhin gegeben.



© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

- Datenkonflikte
  - Auflösen eines Lese-nach-Schreibe-Konflikts (RAW)
    - Verzögerungen (delay slots)

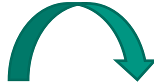


© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

- **Datenkonflikte**
  - **Erkennen und Auflösen eines Lese-nach-Schreibe-Konflikts (RAW)**
    - **Verzögerung (delay slots): Software-Lösung durch Compiler**
    - Aufgabe des Compilers:
      1. Datenabhängigkeitsanalyse: Erkennen von Datenkonflikten;
      2. Einfügen von Leeroperationen (Nulloperation, `nop`) nach jedem Befehl, der einen Konflikt verursacht oder verursachen kann;

```
add  x19,x0,x1
sub  x2,x19,x3
add  x4,x5,x6
add  x7,x8,x9
```



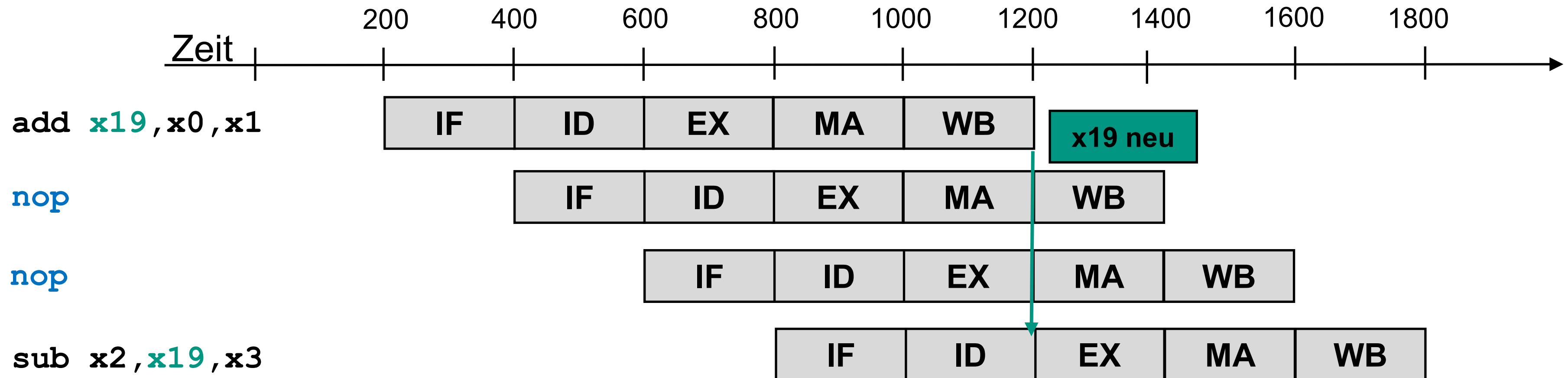
```
add  x19,x0,x1
nop
nop
sub  x2,x19,x3
add  x4,x5,x6
add  x7,x8,x9
```

Eine Leeroperation (`nop`) verändert den Zustand des Prozessors nicht. Beispiel RISC-V: `addi x0,x0,0`

© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

- Datenkonflikte
  - Erkennen und Auflösen eines Lese-nach-Schreibe-Konflikts (RAW)
    - Verzögerung (delay slots): Software-Lösung durch Compiler



 © Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

- **Datenkonflikte**
  - **Erkennen und Auflösen eines Lese-nach-Schreibe-Konflikts (RAW)**
    - **Verzögerung (delay slots): Software-Lösung durch Compiler**
    - Aufgabe des Compilers:
      1. Erkennen von Datenkonflikten
      2. Einfügen von Leeroperationen (nop)
      3. Umordnen von Befehlen des Programms, um Leeroperationen zu eliminieren (Instruction Scheduling, Pipeline Scheduling, Code-Optimierung)

```
add  x19,x0,x1
nop
nop
sub  x2,x19,x3
add  x4,x5,x6
add  x7,x8,x9
```

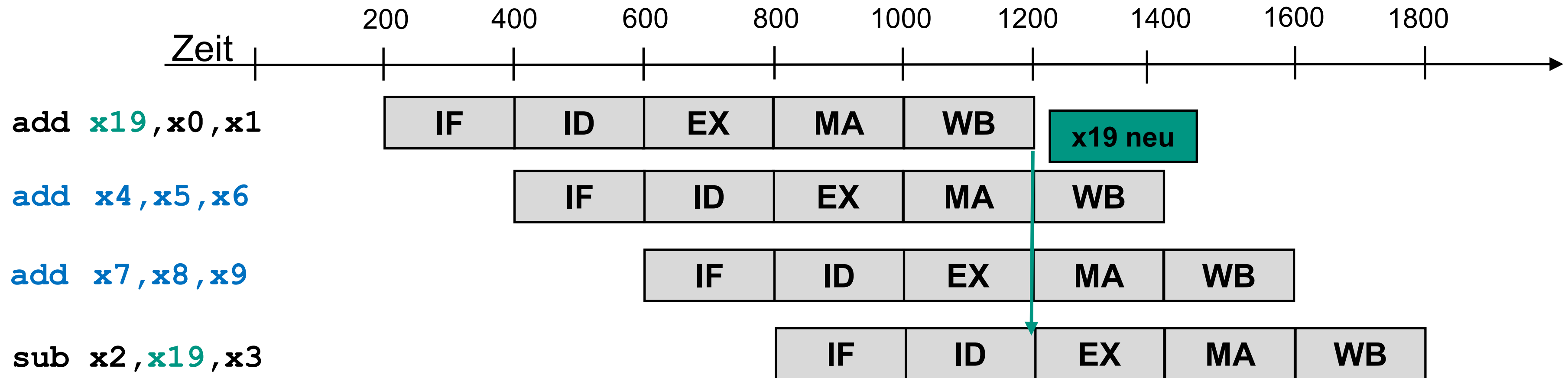
```
add  x19,x0,x1
add  x4,x5,x6
add  x7,x8,x9
sub  x2,x19,x3
```

- Programmsemantik muss erhalten bleiben.
- Nur unabhängige Befehle dürfen umgeordnet werden.

© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

- Datenkonflikte
  - Erkennen und Auflösen eines Lese-nach-Schreibe-Konflikts (RAW)
    - Verzögerung (delay slots): Software-Lösung durch Compiler



KIT

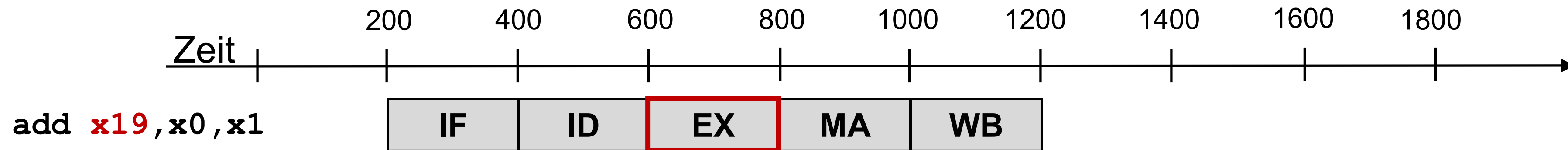
Pipeline-Konflikte

- **Datenkonflikte**
  - **Erkennen und Auflösen eines Lese-nach-Schreibe-Konflikts (RAW)**
    - **Verzögerung: Hardware-Lösungen – Pipeline-Stall (Leerlauf), Interlocking (Anhalten)**
      - Konflikt muss in der HW erkannt werden:
        - In der ID-Stufe erkennt die HW, ob der Befehl in der ID-Stufe ein Register liest, in das der Befehl in der EX-Stufe das Ergebnis schreiben will. In diesem Fall wird die Instruktion sowie die Instruktion in der IF-Phase angehalten.

© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

- **Datenkonflikte**
  - **Erkennen und Auflösen eines Lese-nach-Schreibe-Konflikts (RAW)**
    - **Verzögerung: Hardware-Lösungen – Pipeline-Stall (Leerlauf), Interlocking (Anhalten)**
      - Konflikt muss in der HW erkannt werden:

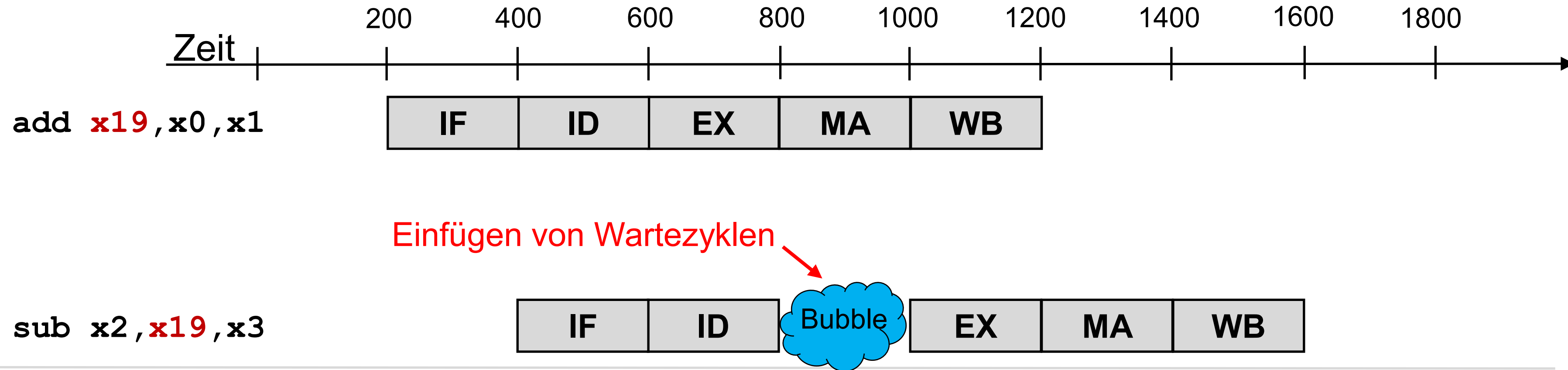


Erkennen eines RAW-Konflikts

© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

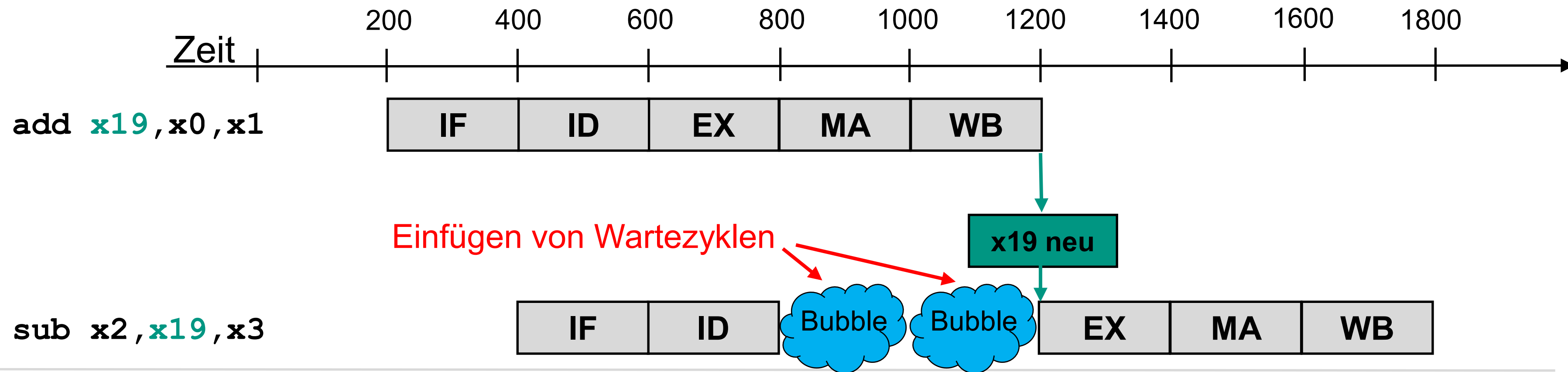
- **Datenkonflikte**
  - **Erkennen und Auflösen eines Lese-nach-Schreibe-Konflikts (RAW)**
    - **Verzögerung: Hardware-Lösungen – Pipeline-Stall (Leerlauf), Interlocking (Anhalten)**
      - Konflikt muss in der HW aufgelöst werden:
        - Pipeline-Stall, Anhalten der Pipeline



© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

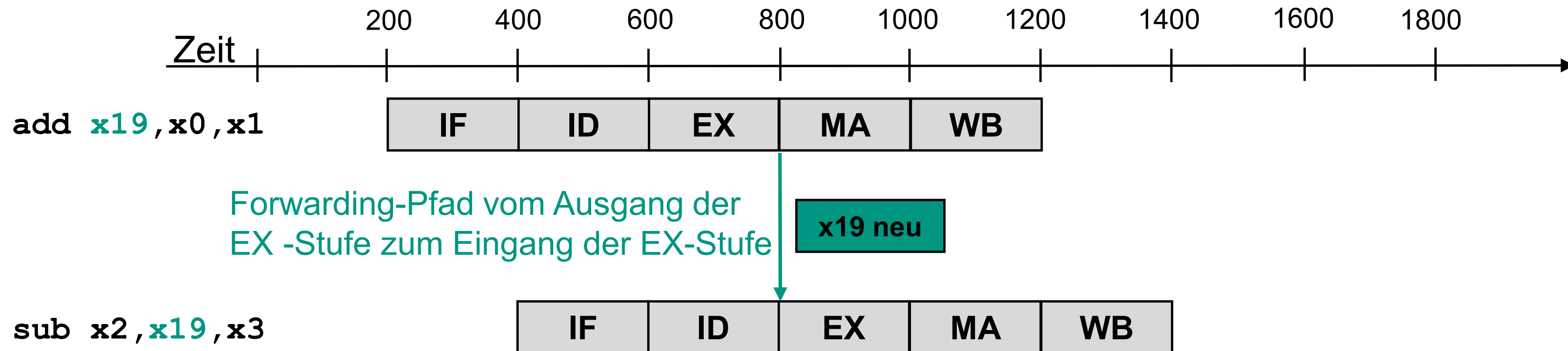
- **Datenkonflikte**
  - **Erkennen und Auflösen eines Lese-nach-Schreibe-Konflikts (RAW)**
    - **Verzögerung: Hardware-Lösungen – Pipeline-Stall (Leerlauf), Interlocking (Anhalten)**
      - Konflikt muss in der HW aufgelöst werden:
        - Pipeline-Stall, Anhalten der Pipeline



© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

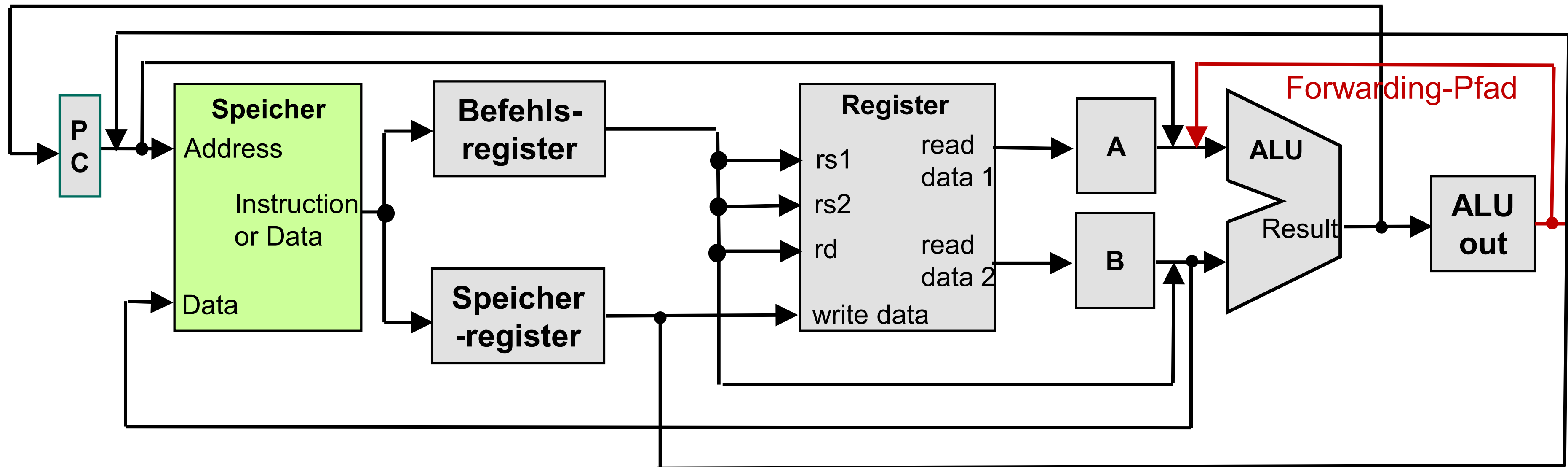
- **Datenkonflikte**
  - **Erkennen und Auflösen eines Lese-nach-Schreibe-Konflikts (RAW)**
    - **Verzögerung: Hardware-Lösungen – Forwarding (Result-Forwarding):**
      - Konflikt muss in der HW erkannt werden.
      - Auflösung eines RAW-Konflikts mit Hilfe einer Leitung vom Ausgang der EX-Stufe zum Eingang der EX-Stufe. Der Operand wird nicht vom Register bereitgestellt.



 © Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

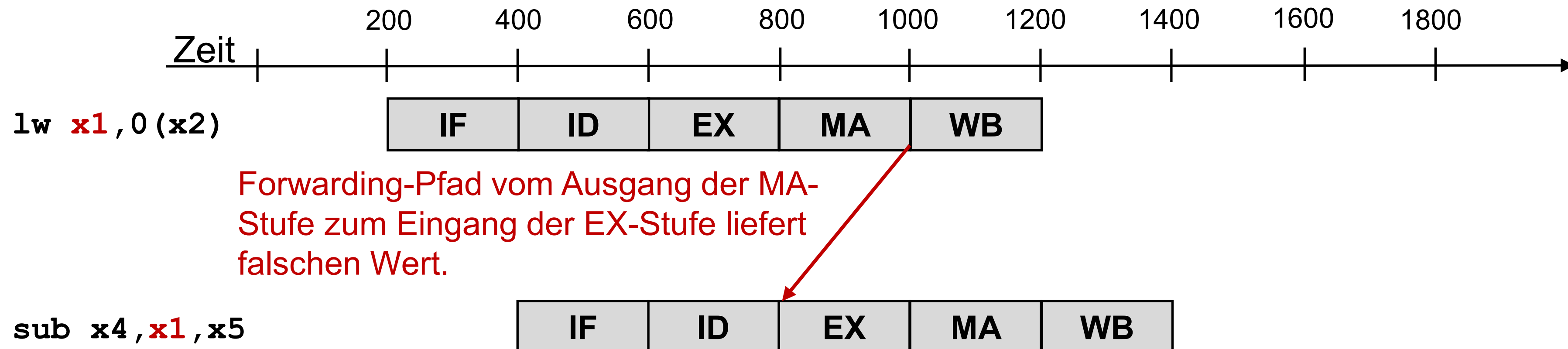
- Datenkonflikte
  - Erkennen und Auflösen eines Lese-nach-Schreibe-Konflikts (RAW)
    - Verzögerung: Hardware-Lösungen – Forwarding (Result-Forwarding):



 © Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

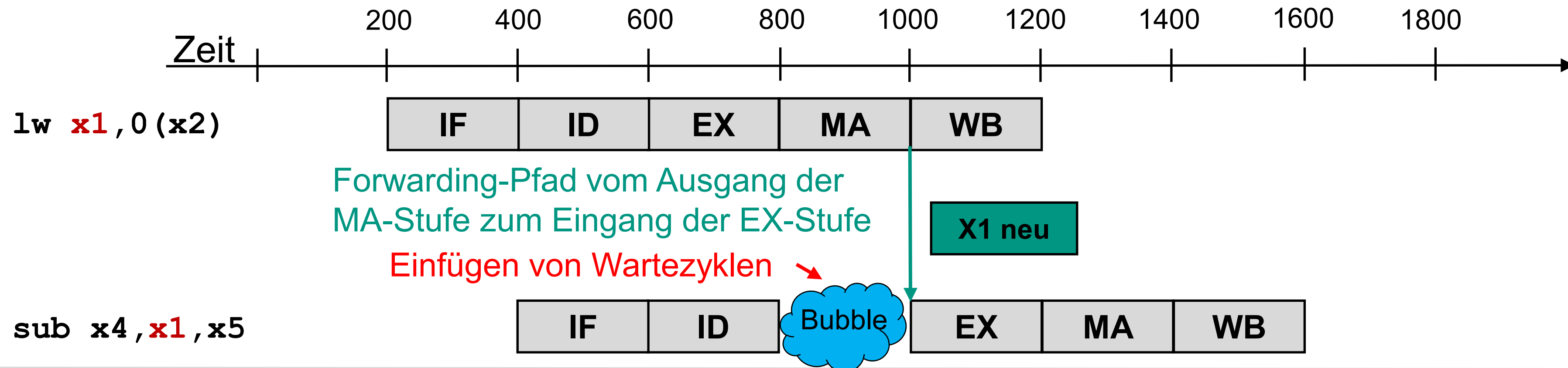
- **Datenkonflikte**
  - **Erkennen und Auflösen eines Lese-nach-Schreibe-Konflikts (RAW)**
    - **Verzögerung: Hardware-Lösungen – Forwarding with Interlocking:**
      - Problem bei Lade-Befehlen:
        - Ein Forwarding-Pfad vom Ausgang der MA-Stufe zum Eingang der EX-Stufe liefert ungültigen Operanden für unmittelbar nachfolgenden Befehl.



© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

- **Datenkonflikte**
  - **Erkennen und Auflösen eines Lese-nach-Schreibe-Konflikts (RAW)**
    - **Verzögerung: Hardware-Lösung – Forwarding with Interlocking:**
      - Lösung: Einfügen von Wartezyklen



 © Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

- **Weitere Datenabhängigkeiten**
  - **Namensabhängigkeiten**
    - Zwischen zwei Befehlen `i1` und `i2`, wobei `i2` dem Befehl `i1` im Programm folgt, besteht eine **Gegenabhängigkeit (antidependence)** zwischen `i1` und `i2`, falls `i1` Daten von einem Register (oder Speicherstelle) liest, das anschließend von `i2` überschrieben wird.
    - Beispiel:

```
add x5,x19,x1   # x5 := x19 + x1
sub x19,x0,x3   # x19 := x0 + x3
```


© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

- **Weitere Datenabhängigkeiten**
  - **Namensabhängigkeiten**
    - Zwischen zwei Befehlen `i1` und `i2`, wobei `i2` dem Befehl `i1` im Programm folgt, besteht eine **Ausgabeabhängigkeit (output dependence)** zwischen `i1` und `i2`, wenn beide in das gleiche Register (oder Speicherstelle) schreiben
    - Beispiel:

```
add x19,x4,x1   # x19 := x4 + x1
sub x19,x0,x3   # x19 := x0 + x3
```


© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

- **Steuerflusskonflikte**
  - Hervorgerufen durch **Programmsteuerbefehle:**
    - unbedingte Sprungbefehle und Verzweigungen (bedingte Sprungbefehle),
    - Unterprogrammaufruf- und –rücksprungbefehle,
    - die Unterbrechungsbefehle.

© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

- **Steuerflusskonflikte**
  - Beispiel: Ausschnitt aus Programm `sort` [1]

```
        addi    x21, x10, 0     # kopiert Parameter x10 nach x21
        addi    x22, x11, 0     # kopiert Parameter x11 nach x22
        addi    x19, x0, 0      # i = 0
for1tst:
        bge x19, x22, exit1     # go to exit1 if 1 >= n
        addi    x20, x19, -1    # j = i - 1
for2tst:
        blt     x20, x0, exit2  # go to exit2 if x20 < 0
        slli    x5, x20, 2      # reg x5 = j*4
        add     x5, x10, x5     # reg x5 = v + (j*4)
        lw      x6, 0(x5)       # reg x6 = v[j]
        lw      x7, 4(x5)       # reg x7 = v[j+1]
        …
```

[1] Siehe Foliensatz RO25-FS05

© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

- **Steuerflusskonflikte**
  - Beispiel: Ausschnitt aus Programm **`sort`** [1]

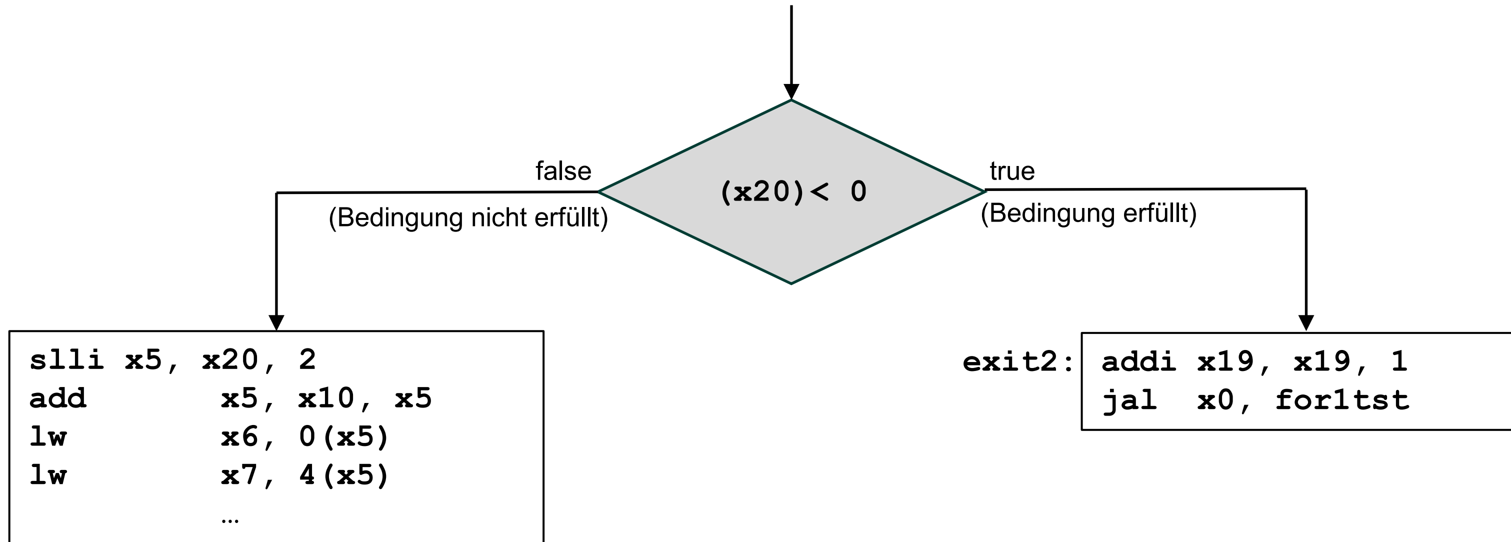
```
exit2:  addi    x19, x19, 1     # i +=1
        jal x0, for1tst         # go to for1tst
```

[1] Siehe Foliensatz RO25-FS05

© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

- **Steuerflusskonflikte**
  - Beispiel: Ausschnitt aus Programm `sort` [1]

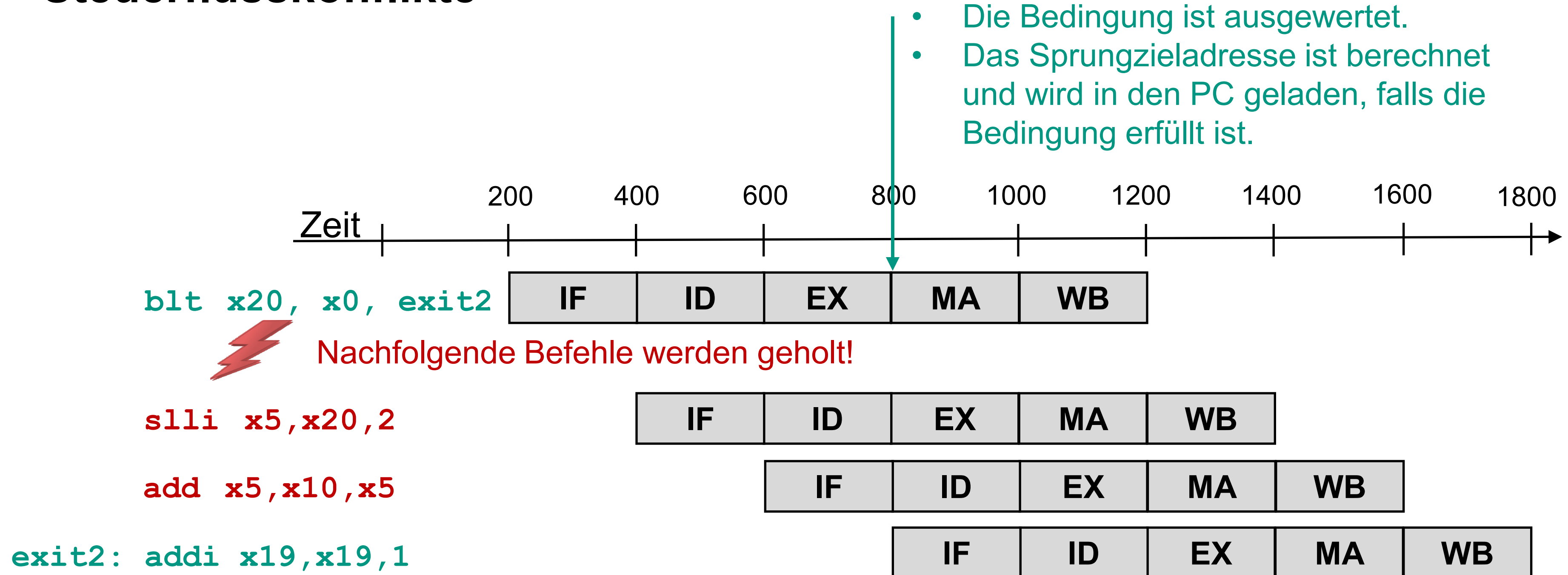


[1] Siehe Foliensatz RO25-FS05

© Prof. Dr. Wolfgang Karl
KIT

Pipeline-Konflikte

Steuerflusskonflikte



© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

Steuerflusskonflikte

Probleme:

1. Ein Befehl wird in der ID-Stufe dekodiert. D. h. die Verzweigung `blt x20,x0,exit2` wird erst in der ID-Stufe als Verzweigung erkannt. Mit der Dekodierung der Verzweigung wird gleichzeitig der im Programm nachfolgende Befehl (`slli x5,x20,2`) geladen. Falls die Bedingung erfüllt ist, ist ein Befehl auf dem Programmpfad false, d. h. ein Befehl, der nicht ausgeführt werden soll, in der Pipeline.
2. Die Sprungzieladresse wird erst in der EX-Stufe berechnet. Ebenso wird die Bedingung ausgewertet. Weitere Befehle auf dem Programmpfades false kommen in die Pipeline, obwohl die Bedingung erfüllt ist.
3. Der PC wird am Ende der EX-Stufe mit der berechneten Sprungzieladresse geladen, wenn die Bedingung erfüllt ist.

© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

- **Steuerflusskonflikte**
  - **Lösungsansatz für die Probleme:**
    2. Die Sprungzieladresse wird erst in der EX-Stufe berechnet. Ebenso wird die Bedingung ausgewertet. Weitere Befehle auf dem Programmpfades false kommen in die Pipeline, obwohl die Bedingung erfüllt ist.
    3. Der PC wird am Ende der EX-Stufe mit der berechneten Sprungzieladresse geladen, wenn die Bedingung erfüllt ist.

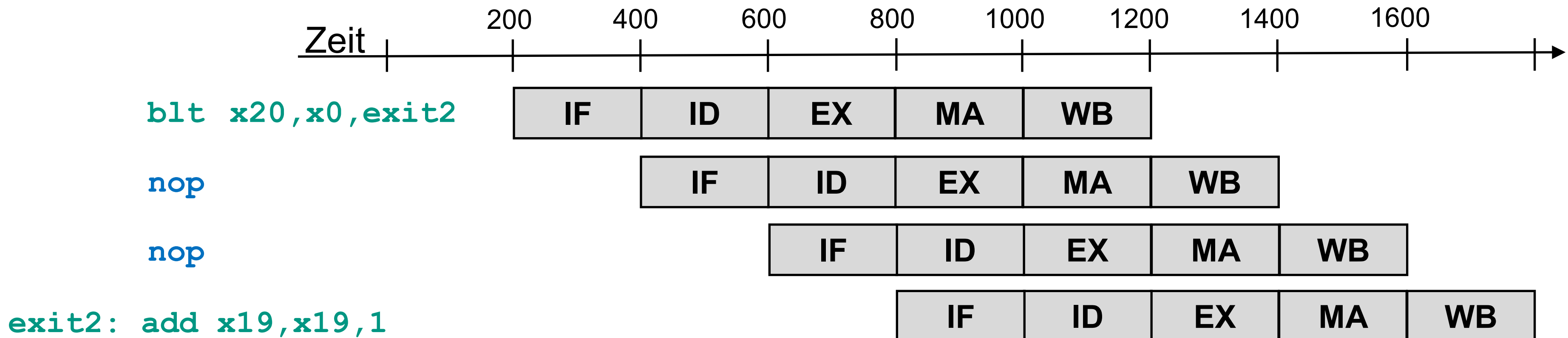
    - **Änderung des Datenpfades beim Entwurf:**
      - Verlegen der Adressberechnung und der Auswertung der Bedingung in die ID-Stufe
      - Laden des PCs am Ende der ID-Stufe

    - **Problem 1. besteht weiterhin!**

© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

- **Steuerflusskonflikte**
  - **Auflösung mit Hilfe von SW-Lösungen (Compiler)**
    - **Verzögerte Sprungtechnik (delayed branch): Einfügen von Leeroperationen**
      - Anzahl der Leeroperationen (nop) hängt von der Organisation der Pipeline ab.



© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

- **Steuerflusskonflikte**
  - **Auflösung mit Hilfe von SW-Lösungen (Compiler)**
    - **Verzögerte Sprungtechnik (delayed branch): Einfügen von Leeroperationen**
      - Umordnen von Befehlen des Programms (Instruction Scheduling, Optimierung)
        - Befehle, die in der in der statischen Programmreihenfolge vor dem Sprungbefehl liegen, können die Leeroperationen ersetzen.
        - Verzögerungstakte (delay slots) werden sinnvoll genutzt.

```
       add x6,x17,x5
       add x4,x19,x3
       beq x1,x2,Marke
       nop
       nop
Marke: or  x1,x8,x9
```

```
       beq x1,x2,Marke
       add x6,x17,x5
       add x4,x19,x3
Marke: or  x1,x8,x9
```


© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

- **Steuerflusskonflikte**
  - **Auflösung mit Hilfe von SW-Lösungen (Compiler)**
    - **Verzögerte Sprungtechnik (delayed branch): Einfügen von Leeroperationen**
      - Umordnen von Befehlen des Programms (Instruction Scheduling, Optimierung)
        - Nur dann möglich, wenn die verschobenen Befehle keinen Einfluss auf die Sprungrichtung haben!
        - Gibt es keine Befehle, die in die Verzögerungszeitschlitze verschoben werden können, müssen Leerbefehle bleiben.
        - Siehe Beispiel `sort` auf nachfolgender Folie.

© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

- **Steuerflusskonflikte**
  - **Auflösung mit Hilfe von SW-Lösungen (Compiler)**
    - **Verzögerte Sprungtechnik (delayed branch): Einfügen von Leeroperationen**
      - Beispiel: Ausschnitt aus Programm `sort`

echte Datenabhängigkeit

```
        addi    x20, x19, -1    # j = i - 1
for2tst:
        blt     x20, x0, exit2 # go to exit2 if x20 < 0
        nop
        nop
        slli    x5, x20, 2      # reg x5 = j*4
        add     x5, x10, x5     # reg x5 = v + (j*4)
                …
```

Umordnen nicht möglich!

© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

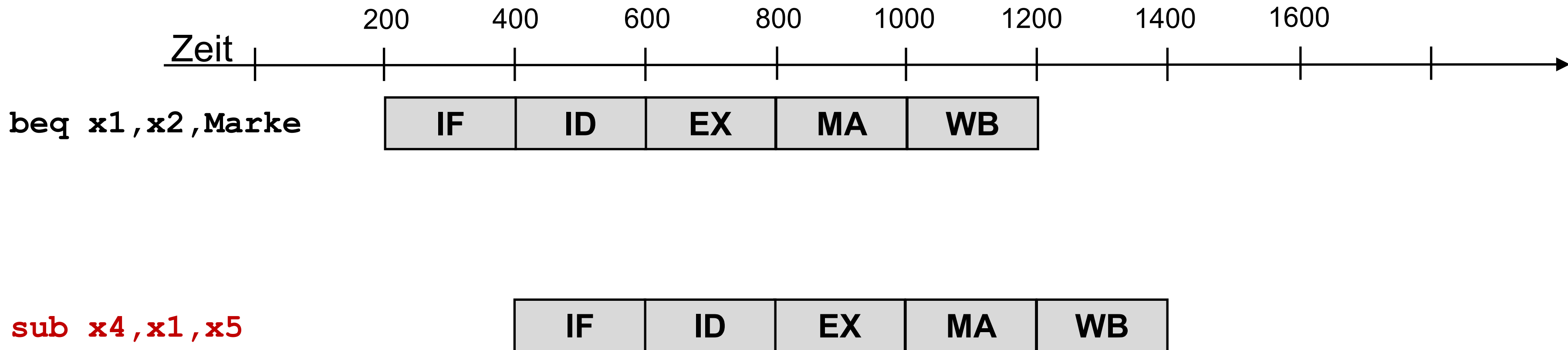
- **Steuerflusskonflikte**
  - **Auflösung mit Hilfe von HW-Lösungen**
    - **Pipeline-Stall (Leerlauf)**
      - Konflikt muss in der HW erkannt werden:
        - Die HW erkennt in der ID-Stufe, ob ein Programmsteuerbefehl vorliegt.
        - Es werden keine weiteren Befehle in die Pipeline geladen, bis die Sprungzieladresse berechnet ist und bei Verzweigungen die Bedingung ausgewertet ist.
        - Der eine Befehl, der bereits ins Pipeline-Register der IF-Stufe geladen worden ist, muss gelöscht werden.

      - Einfachste und ineffizienteste Methode

© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

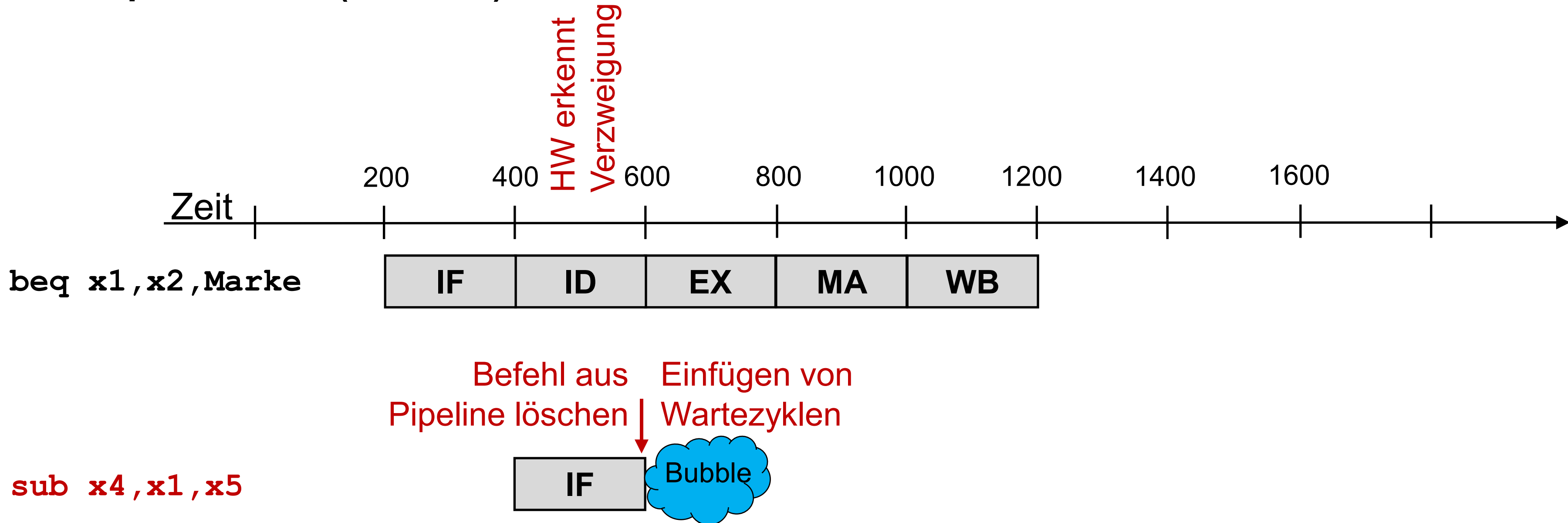
- Steuerflusskonflikte
  - Auflösung mit Hilfe von HW-Lösungen
    - Pipeline-Stall (Leerlauf)



© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

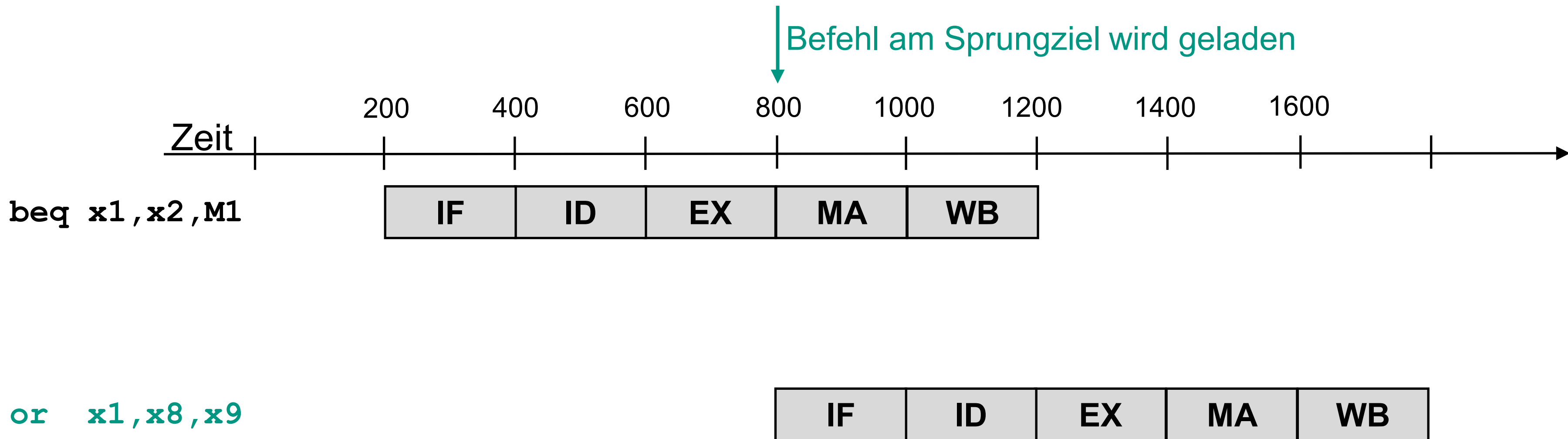
- Steuerflusskonflikte
  - Auflösung mit Hilfe von HW-Lösungen
    - Pipeline-Stall (Leerlauf)



© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

- Steuerflusskonflikte
  - Auflösung mit Hilfe von HW-Lösungen
    - Pipeline-Stall (Leerlauf)



Pipeline-Konflikte

- **Steuerflusskonflikte**
  - Beispiel: Ausschnitte aus Programm `sort` [1]

```
for2tst:
        blt     x20, x0, exit2 # go to exit2 if x20 < 0 (j < 0)
        slli    x5, x20, 2     # reg x5 = j*4
        add     x5, x10, x5    # reg x5 = v + (j*4)
        lw      x6, 0(x5)      # reg x6 = v[j]
        lw      x7, 4(x5)      # reg x7 = v[j+1]
        …

exit2:  addi    x19, x19, 1# i +=1
        jal x0, for1tst        # go to for1tst
```

[1] Siehe Foliensatz RO25-FS05

© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

- **Steuerflusskonflikte**
  - Beispiel: Ausschnitte aus Programm `sort` [1]
    - 1. Fall: Bedingung der Verzweigung `x20 < 0` ist erfüllt:

```
for2tst:
        blt     x20, x0, exit2 # go to exit2 if x20 < 0 (j < 0)
        slli    x5, x20, 2     # reg x5 = j*4
        add     x5, x10, x5    # reg x5 = v + (j*4)
        lw      x6, 0(x5)      # reg x6 = v[j]
        lw      x7, 4(x5)      # reg x7 = v[j+1]
        ...

exit2:  addi    x19, x19, 1# i +=1
        jal x0, for1tst      # go to for1tst
```

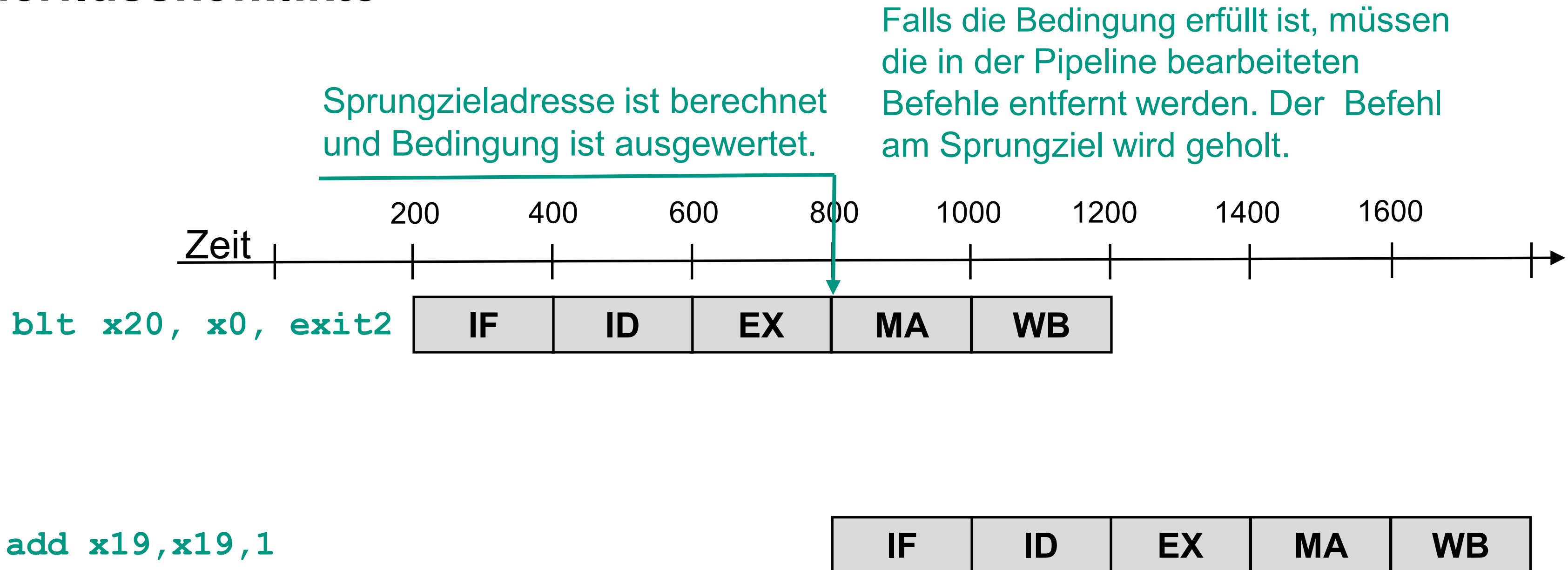
Sprung

[1] Siehe Foliensatz RO25-FS05

© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

- **Steuerflusskonflikte**



© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

- **Steuerflusskonflikte**
  - Beispiel: Ausschnitte aus Programm `sort` [1]
    - 2. Fall: Bedingung der Verzweigung `x20 < 0` ist nicht erfüllt:

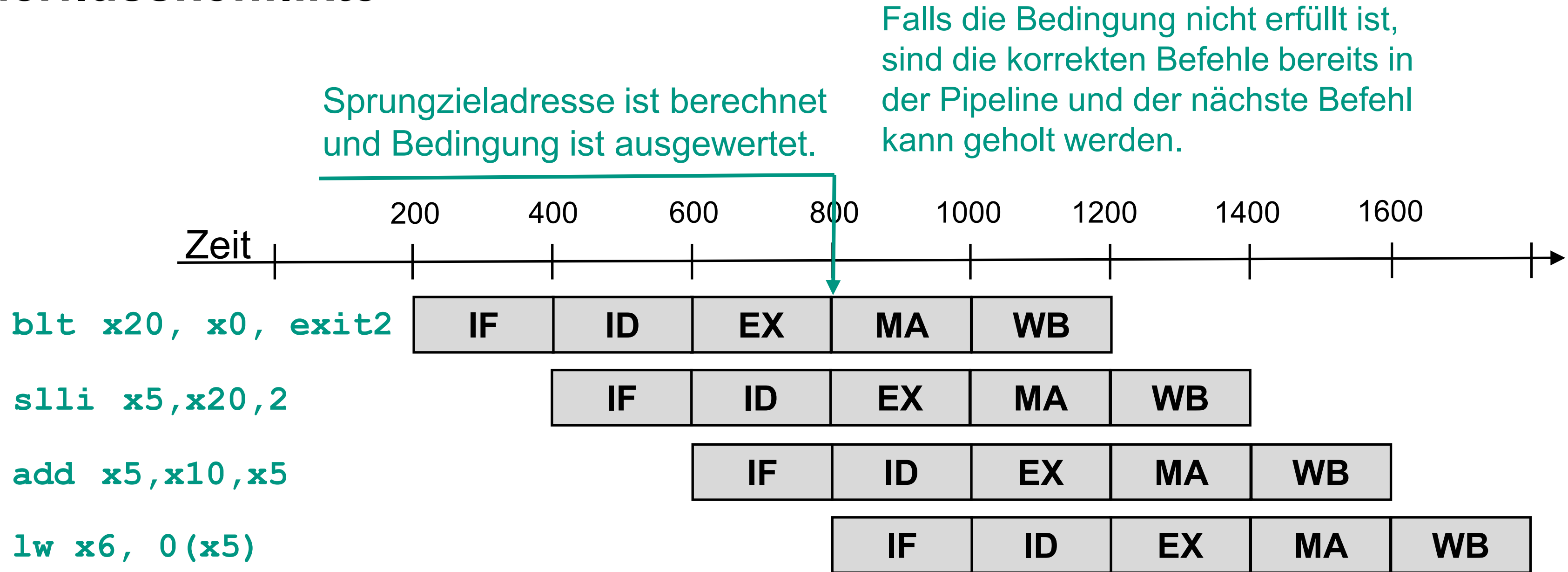
```
for2tst:
        blt     x20, x0, exit2 # go to exit2 if x20 < 0 (j < 0)
        slli    x5, x20, 2     # reg x5 = j*4
        add     x5, x10, x5    # reg x5 = v + (j*4)
        lw      x6, 0(x5)      # reg x6 = v[j]
        lw      x7, 4(x5)      # reg x7 = v[j+1]
        …

exit2:  addi    x19, x19, 1# i +=1
        jal x0, for1tst      # go to for1tst
```

[1] Siehe Foliensatz RO25-FS05

Pipeline-Konflikte

Steuerflusskonflikte



© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

- **Steuerflusskonflikte**
  - **Sprungvorhersage (branch prediction)**
    - Beim Auftreten einer Verzweigung wird eine Vorhersage des Sprungverhaltens getroffen.
    - Annahme, dass die Bedingung ist erfüllt ist:
      - Die Verzögerungsphasen werden **spekulativ** mit Befehlen gefüllt, die am Sprungziel stehen.
    - Annahme, dass die Bedingung nicht erfüllt ist:
      - Die Verzögerungsphasen werden **spekulativ** mit Befehlen gefüllt, die dem Sprung folgen.
    - Nach Auswertung der Sprungbedingung:
      - Bei korrekter Vorhersage: Fortfahren mit der Ausführung ohne Verzögerung
      - Bei falscher Vorhersage: Verwerfen der geholten Befehle

Pipeline-Konflikte

- **Steuerflusskonflikte**
  - **Statische Sprungvorhersage (branch prediction):**
    - Die Vorhersage ist für jede Verzweigung fest vorgegeben.
    - **Treffe bei jeder Verzweigung die Vorhersage „Verzweigung wird nicht genommen“ (predict branch not taken)**
      - Die der Verzweigung folgenden Befehle werden weiter in der Pipeline bearbeitet.
      - Falls sich die Vorhersage als richtig erweist, dann kann ohne Zeitverlust weitergearbeitet werden.
      - Falls sich die Vorhersage als falsch erweist, dann müssen die in der Pipeline bereits bearbeiteten Befehle gelöscht werden und die am Sprungziel stehenden Befehle geholt werden.
        - Nur in diesem Fall gibt es einen Zeitverlust!

Pipeline-Konflikte

- **Steuerflusskonflikte**
  - **Statische Sprungvorhersage (branch prediction):**
    - Die Vorhersage ist für jede Verzweigung fest vorgegeben.
    - **Treffe bei jeder Verzweigung die Vorhersage „Verzweigung wird genommen“ (predict branch taken)**
      - Die Befehle am Sprungziel werden spekulativ geholt.
      - Problem: auch Sprungziel muss vorhergesagt werden.
      - Die Pipeline muss so organisiert werden, dass Berechnung des Sprungziels möglichst früh in der Pipeline erfolgt.
      - Falls die Vorhersage sich als falsch erweist, dann müssen die in der Pipeline bereits bearbeiteten Befehle gelöscht werden. Und es müssen die der Verzeigung folgenden Befehle geholt werden.

Pipeline-Konflikte

- **Steuerflusskonflikte**
  - **Statische Sprungvorhersage (branch prediction):**
    - In HW festverdrahtete Implementierung sehr starr und die Wahrscheinlichkeit für eine korrekte Vorhersage ist nicht sehr hoch.
    - **Compiler-Unterstützung**
      - Verzweigungsbefehle enthalten ein Bit, in das der Compiler eine Vorhersage kodiert.
      - Die Befehle werden spekulativ von der vorhergesagten Sprungrichtung geholt.
      - Eine Vorhersage wird auf der Grundlage einer Programmanalyse oder mit Hilfe von Profiling getroffen.
    - In modernen Mikroprozessoren mit Pipeline-Organisation nicht implementiert.

© Prof. Dr. Wolfgang Karl

Pipeline-Konflikte

- **Steuerflusskonflikte**
  - **Dynamische Sprungvorhersage**
    - Vorhersage des Verhaltens von Verzweigungen zur Laufzeit mit Hilfe von zur Laufzeit gesammelten Informationen.
      - Aufzeichnen der Vorgeschichte für jede ausgeführte Verzweigung im Programm;
        - Es wird festgehalten, ob ein Sprung genommen worden ist oder nicht.
      - Die Vorhersage für eine Verzweigung berücksichtigt das Verhalten dieser Verzweigung in der Vergangenheit.
      - Komplexere Verfahren berücksichtigen auch das Verhalten benachbarter Verzweigungen in der Vergangenheit.
      - Genauere Vorhersage möglich: > 90%
      - Hoher Hardware-Aufwand!

© Prof. Dr. Wolfgang Karl