5.2 Pipelining des Maschinenbefehlszyklus

- **Pipelining – Maschinenbefehlszyklus (Instruction Pipelining)**
  - Zerlegung der Ausführung einer Maschinenoperation in Teilphasen, die dann von hintereinander geschalteten Verarbeitungseinheiten taktsynchron bearbeitet werden, wobei jede Einheit genau eine spezielle Teiloperation ausführt.

  - **Pipeline: logische Phasen**
    - Gesamtheit der Verarbeitungseinheiten

Teiloperationen:
IF: Befehl holen
ID: Befehl dekodieren / Operanden bereitstellen
EX: Befehl ausführen
MA: Speicherzugriff
WB: Zurückschreiben

5.2 Pipelining des Maschinenbefehlszyklus

- **Pipelining – Maschinenbefehlszyklus (Instruction Pipelining)**
  - **1. Phase: Befehlsbereitstellungs-Phase (IF-Phase: Instruction Fetch)**
    - Der durch den Befehlszähler adressierte Befehl wird aus dem Arbeitsspeicher (bzw. dem Befehlscache) in einen Befehlspuffer geladen
    - Der Befehlszähler wird weitergeschaltet
  - **2. Phase: Dekodier- und Operandenbereitstellungsphase (ID/RF-Phase: Instruction Decode & Register Fetch)**
    - Aus dem Operationscode des Maschinenbefehls werden prozessorinterne Steuersignale erzeugt
    - Die Operanden werden aus dem Registerfile bereit gestellt (2. Takthälfte)

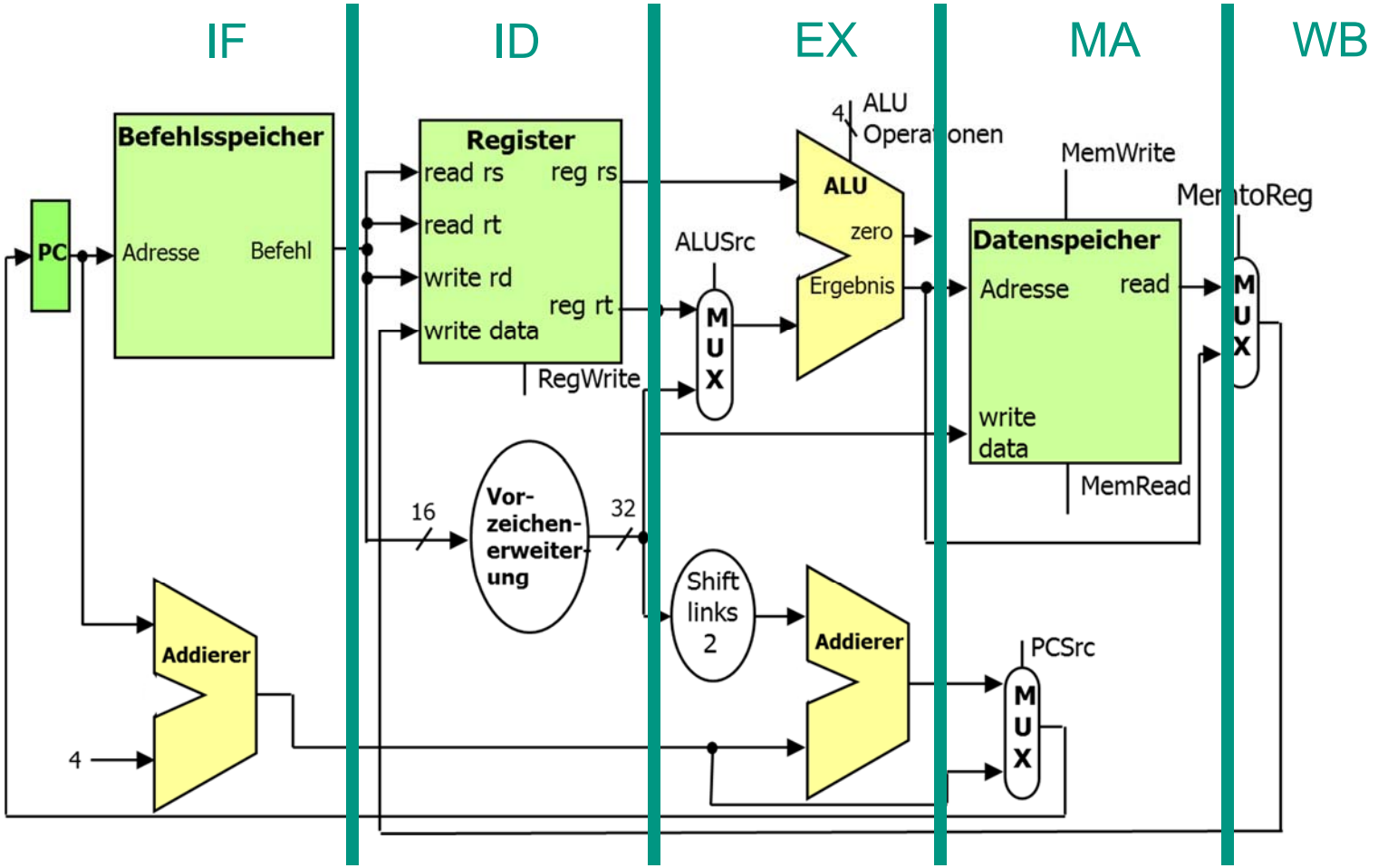
5.2 Pipelining des Maschinenbefehlszyklus

- **Pipelining – Maschinenbefehlszyklus (Instruction Pipelining)**
  - **3. Phase: Ausführungsphase / Berechnung der effektiven Adresse (EX-Phase: Execution/Effective Address Calculation)**
    - Die Operation wird auf den Operanden ausgeführt
    - Bei Lade- und Speicherbefehlen oder Verzweigungen berechnet die ALU die effektive Adresse
  - **4. Phase: Speicherzugriffsphase (MA-Phase: memory access)**
    - Der Speicherzugriff (bei Lade- und Speicherbefehlen) wird durchgeführt
  - **5. Phase: Resultatspeicherphase (WB-Phase: write back)**
    - Das Ergebnis wird in das Registerfile geschrieben (1. Takthälfte)
    - Befehle ohne Ergebnis durchlaufen diese Phase passiv

5.2 Pipelining des Maschinenbefehlszyklus

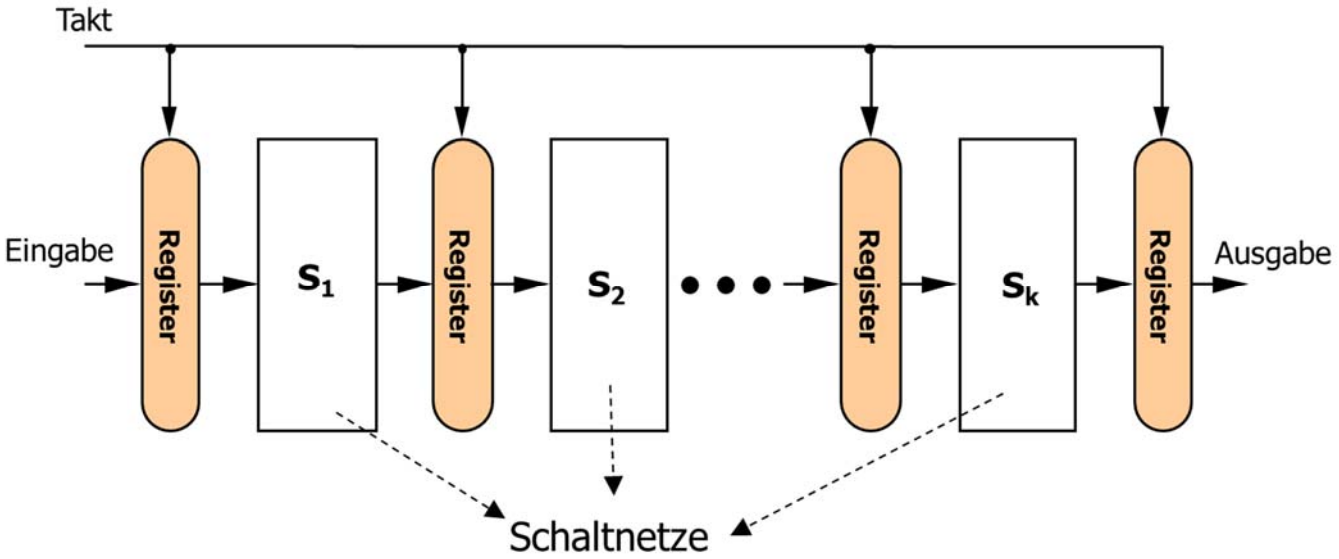
Maschinenbefehlszyklus der DLX-Pipeline

- Pipeline: Gesamtheit der Verarbeitungseinheiten



5.2 Pipelining des Maschinenbefehlszyklus

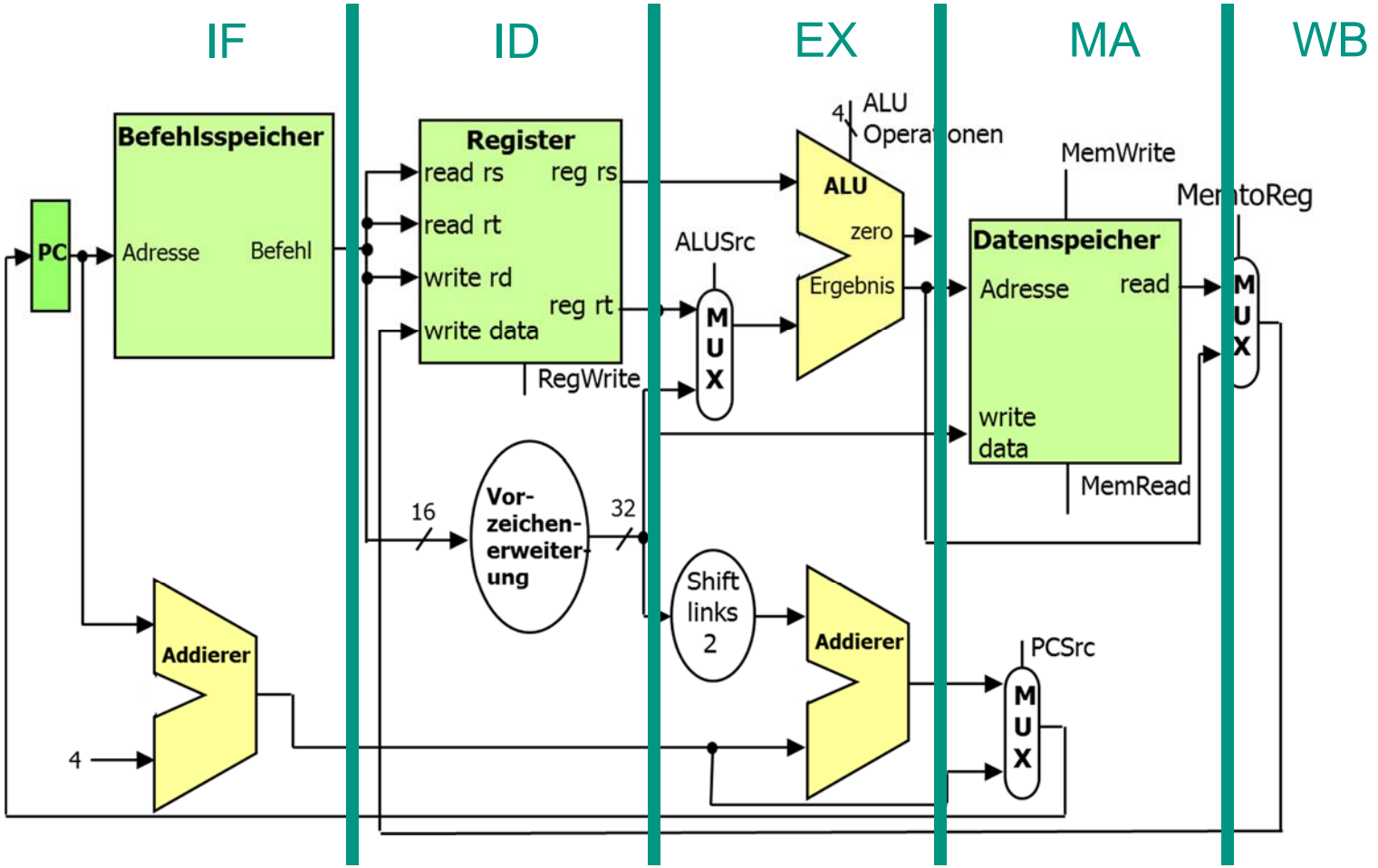
- **Pipelining – Maschinenbefehlszyklus (Instruction Pipelining)**
  - k-stufige Befehlspipeline



  - Pipeline-Stufen durch Pipelineregister getrennt: Taktsynchrone Abarbeitung
  - Verzögerungszeiten:
    - der Schaltnetze: $t_i \quad (i = 1,2, \dots k)$
    - der Pipeline-Register: t_{reg}
  - Länge des Taktzyklus: $t = \max\{t_1, t_2, \dots, t_k\} + t_{reg}$

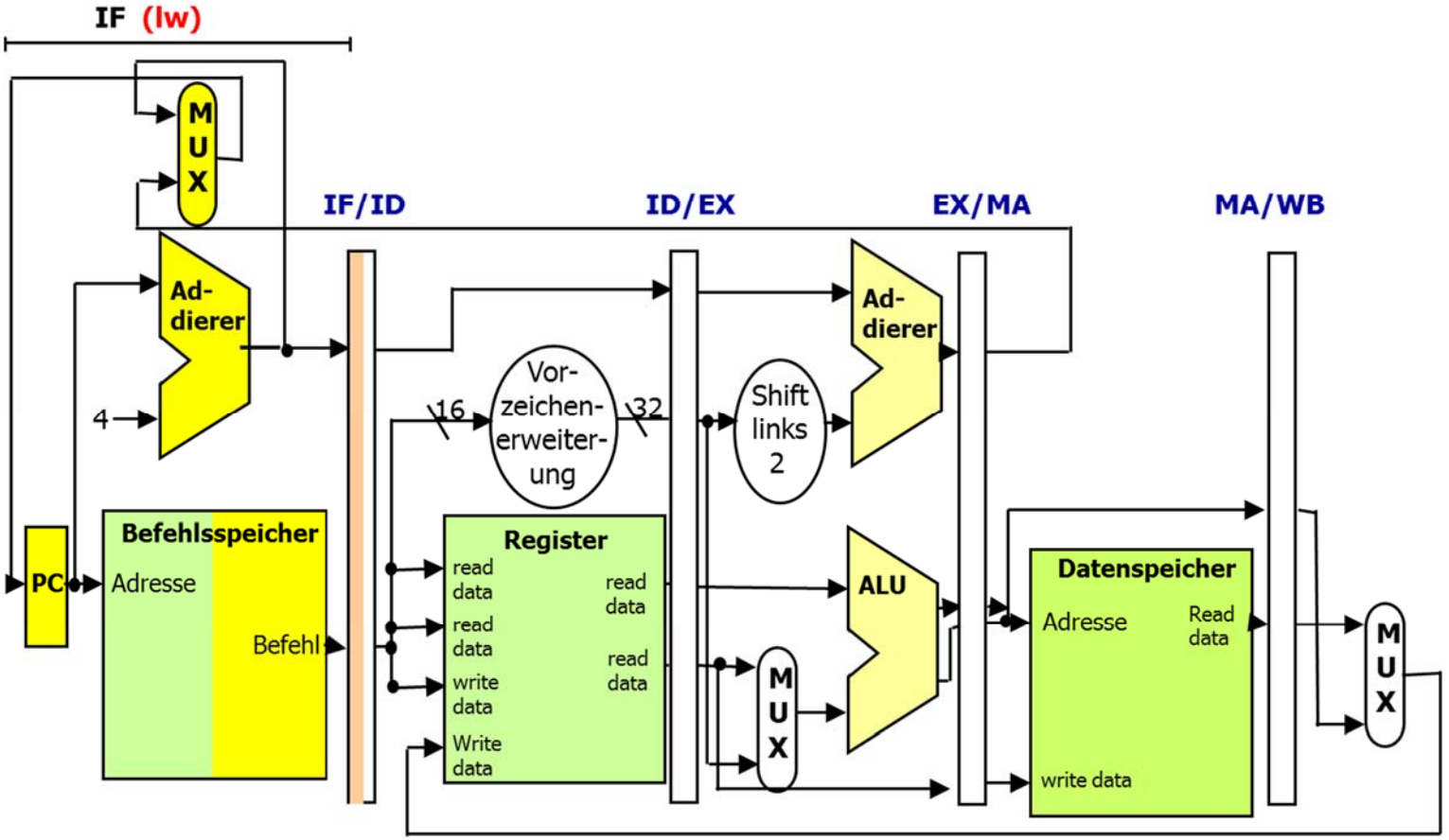
5.2 Pipelining des Maschinenbefehlszyklus

- **Pipelining DLX- (MIPS-)Prozessor**
  - Pipeline: Gesamtheit der Verarbeitungseinheiten



5.2 Pipelining DLX- (MIPS-)Prozessor

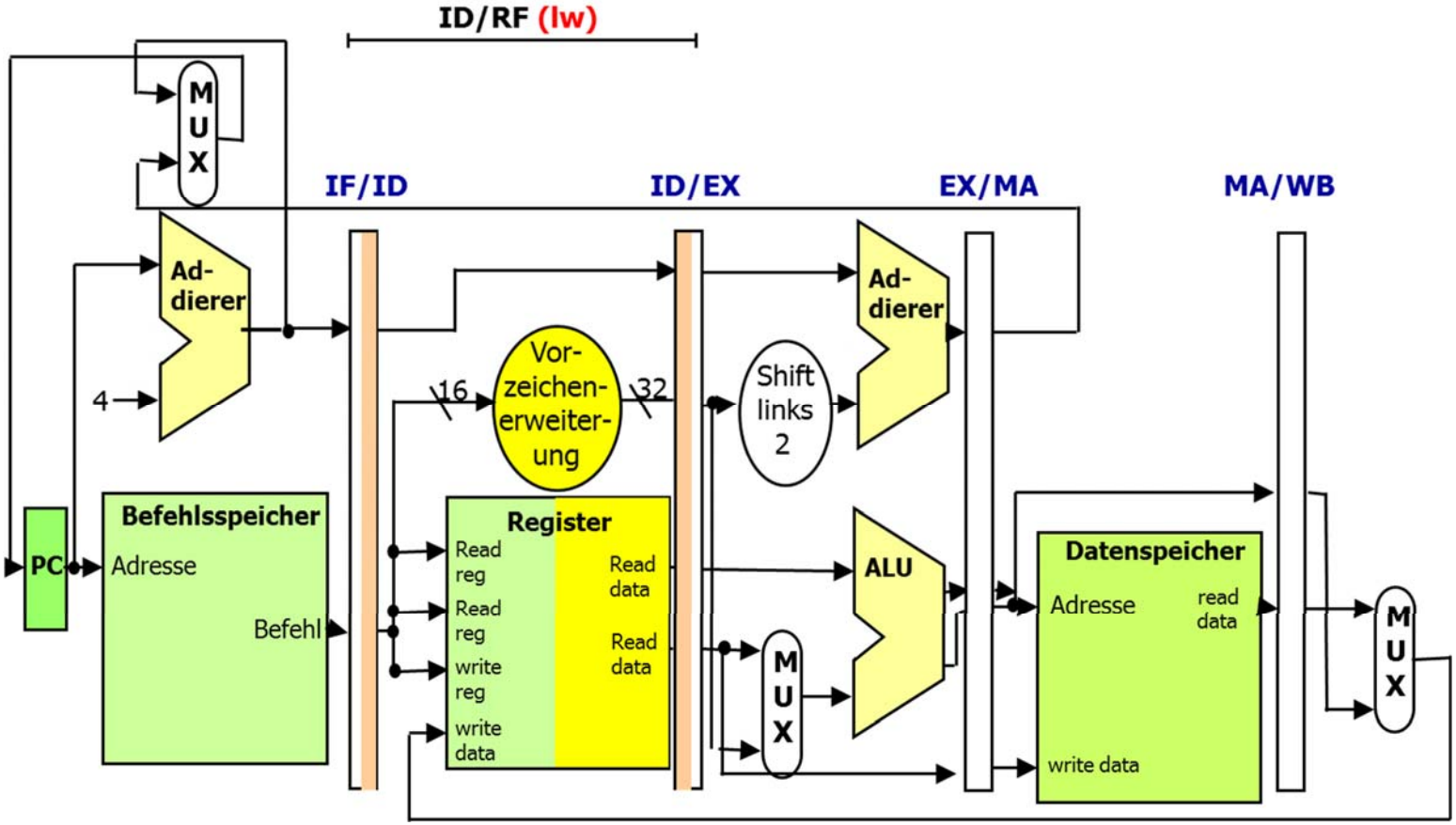
DLX-Pipeline-Stufen: 1. Stufe (IF)



- Der durch den Befehlszähler adressierte Befehl wird aus dem Arbeitsspeicher (bzw. dem Befehlscache) in einen Befehlspuffer geladen.
- Der Befehlszähler wird weitergeschaltet.

5.2 Pipelining DLX- (MIPS-)Prozessor

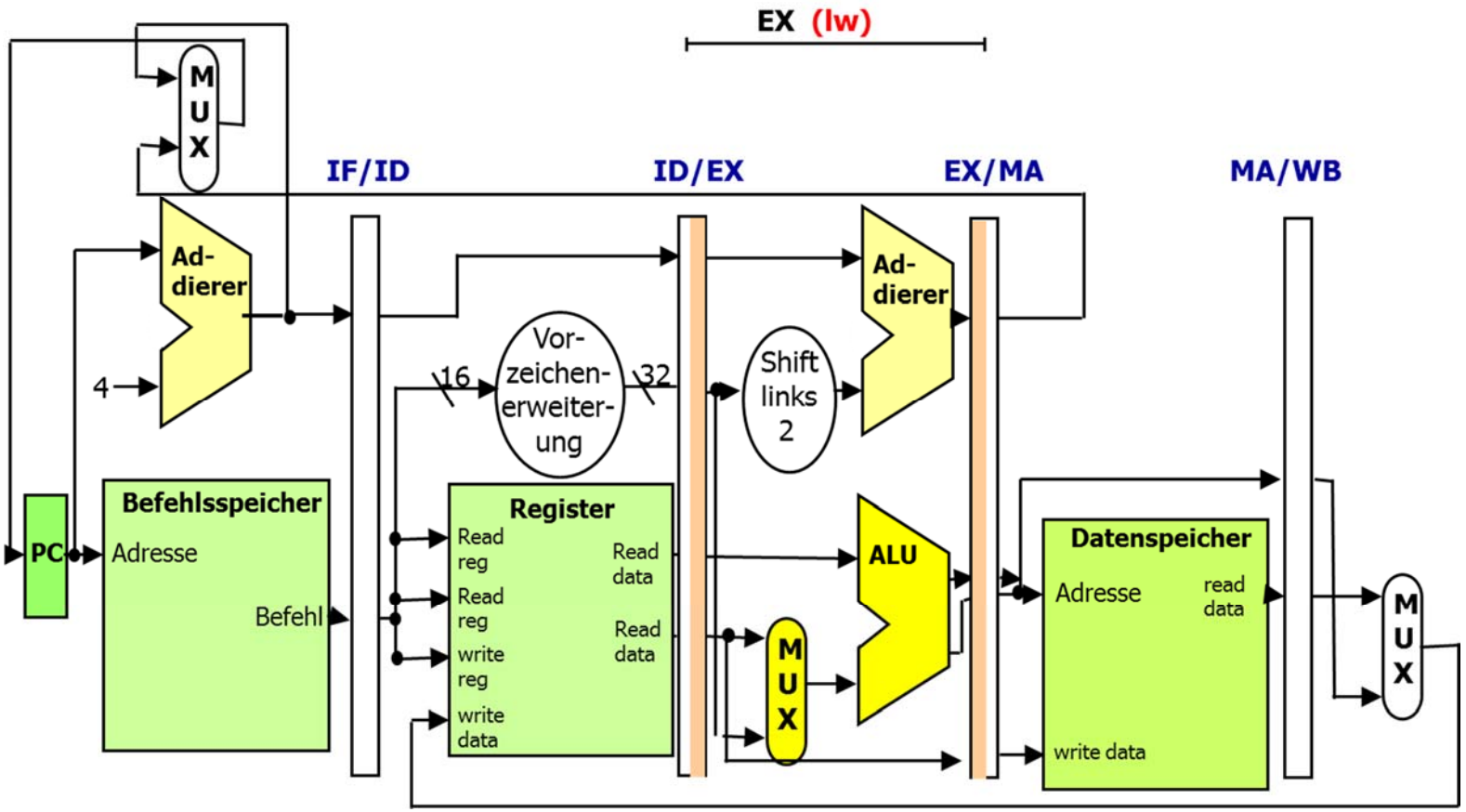
DLX-Pipeline-Stufen: 2. Stufe (ID)



- Aus dem Operationscode des Maschinenbefehls werden prozessorinterne Steuersignale erzeugt.
- Die Operanden werden aus (Universal)-Register bereit gestellt (2. Takthälfte).

5.2 Pipelining DLX- (MIPS-)Prozessor

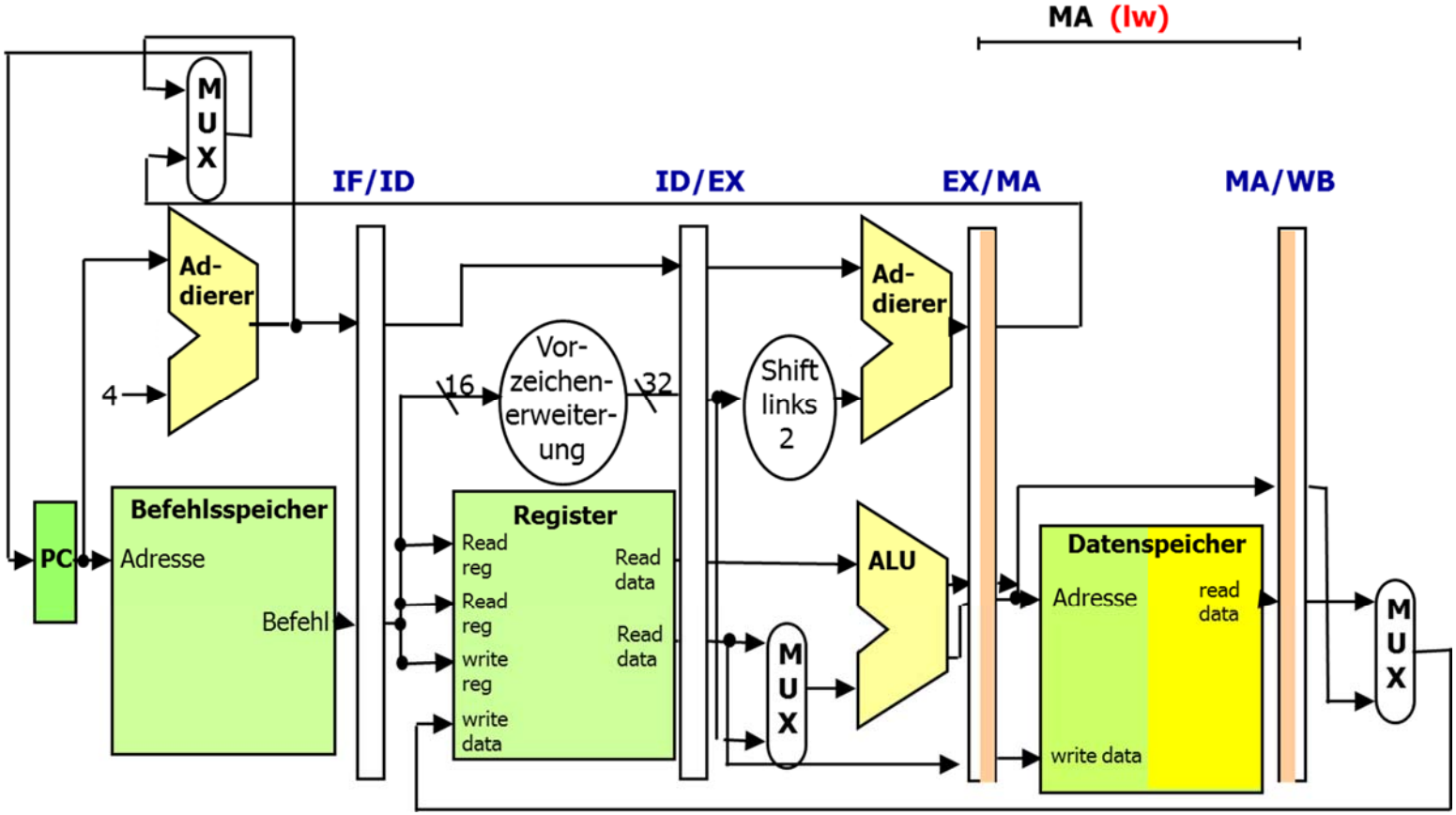
DLX-Pipeline-Stufen: 3. Stufe (EX)



- Die Operation wird auf den Operanden ausgeführt.
- Bei Lade- und Speicherbefehlen oder Verzweigungen berechnet die ALU die effektive Adresse

5.2 Pipelining DLX- (MIPS-)Prozessor

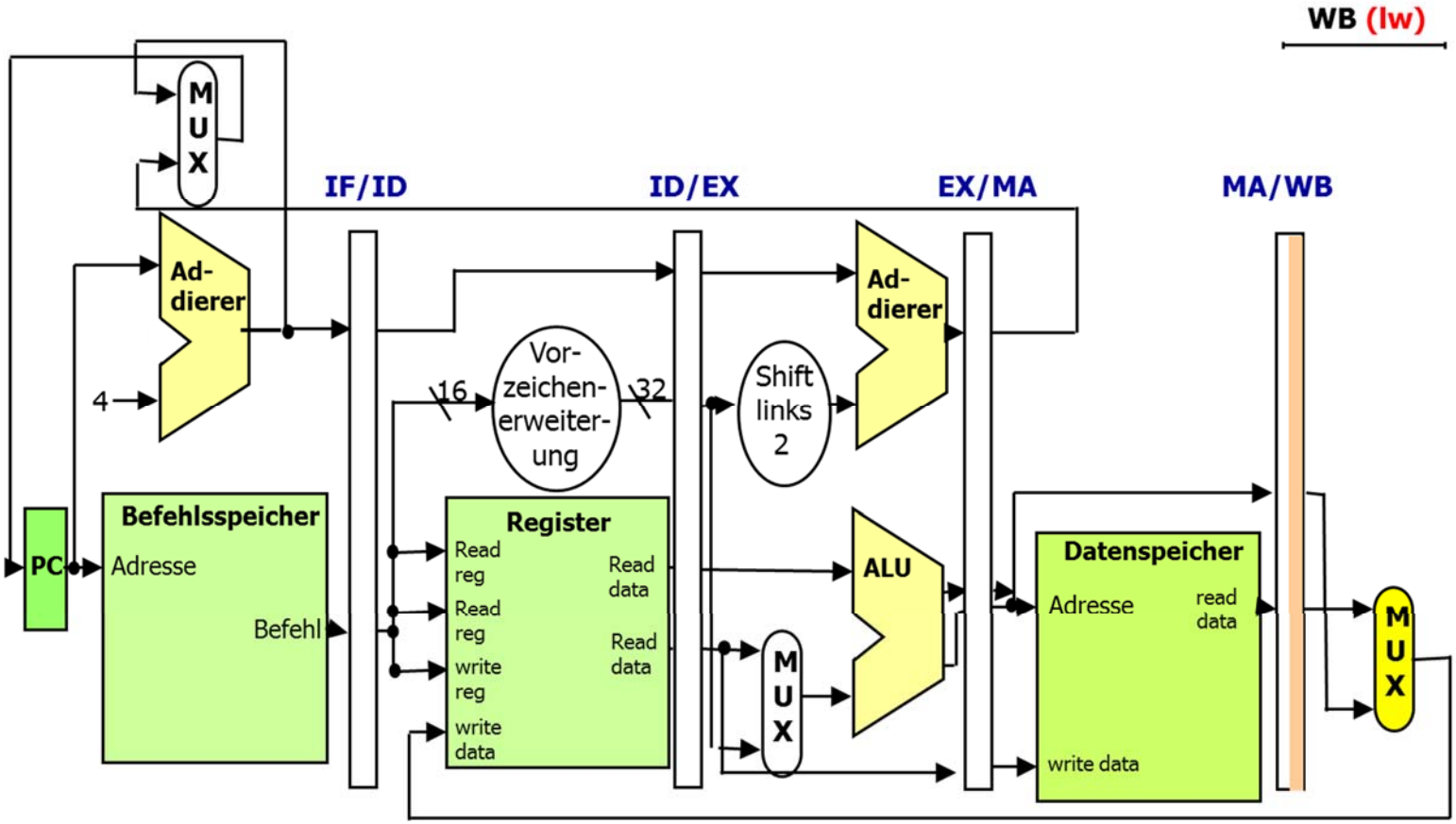
DLX-Pipeline-Stufen: 4. Stufe (MA)



- Der Speicherzugriff (bei Lade- und Speicherbefehlen) wird durchgeführt

5.2 Pipelining DLX- (MIPS-)Prozessor

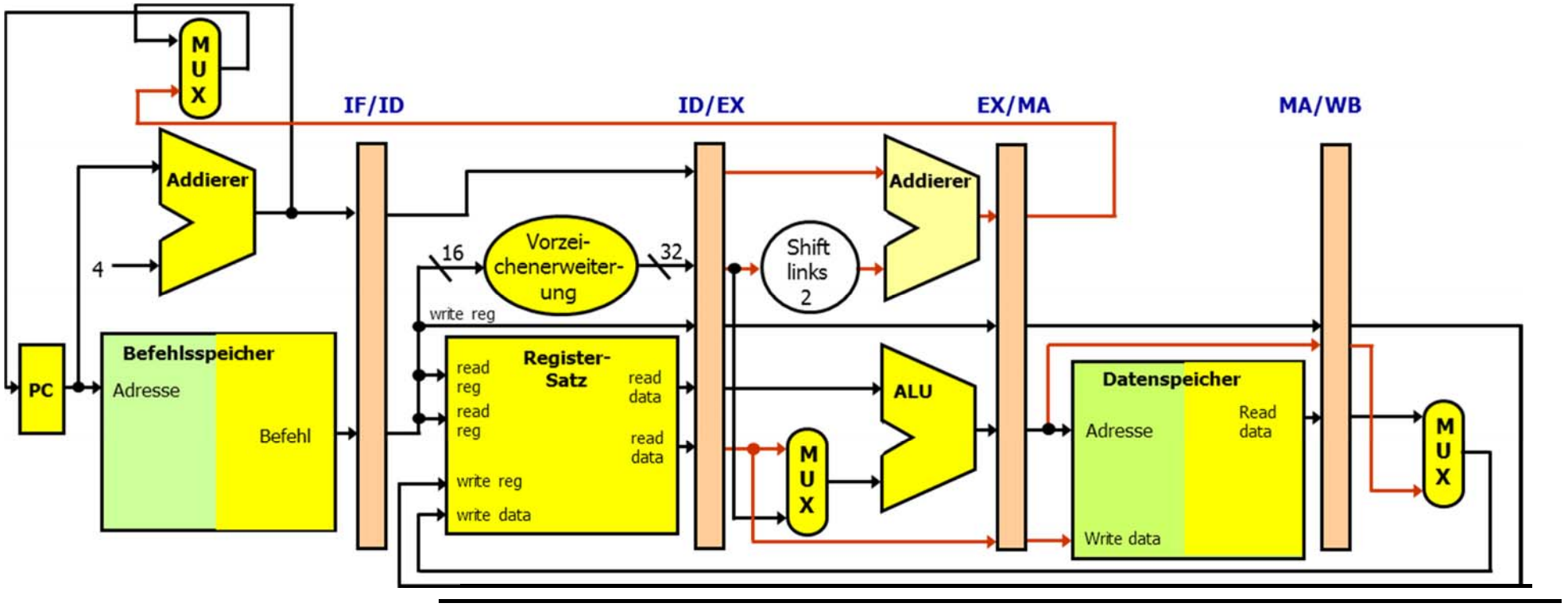
DLX-Pipeline-Stufen: 5. Stufe (WB)



- Das Ergebnis wird in ein (Universal)-Register geschrieben (1. Takthälfte).
- Befehle ohne Ergebnis durchlaufen diese Phase passiv.

5.2 Pipelining DLX- (MIPS-)Prozessor

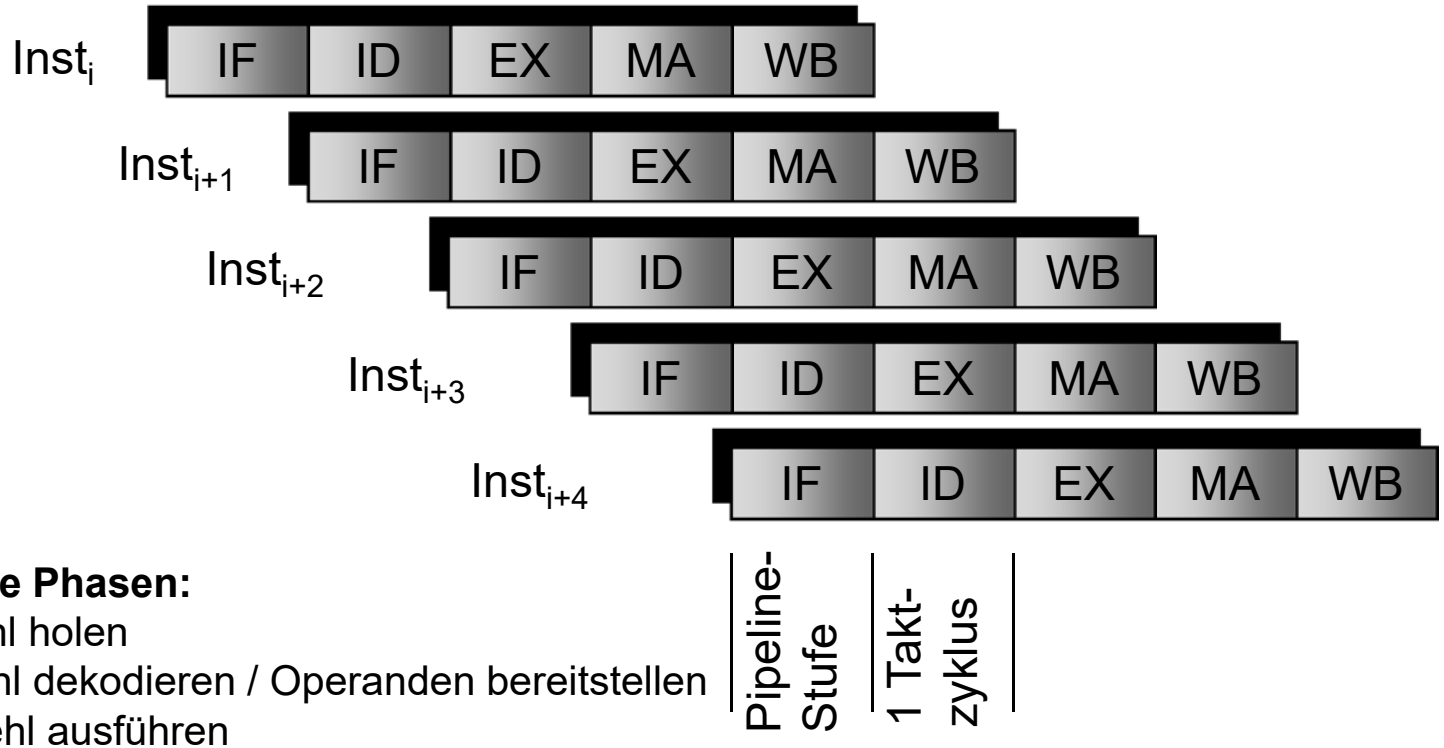
- **DLX-Pipeline-Stufen: 5 Stufen**



Zyklus 6				
Befehl 6	Befehl 5	Befehl 4	Befehl 3	Befehl 2

5.2 Pipelining des Maschinenbefehlszyklus

- **Pipelining – Maschinenbefehlszyklus (Instruction Pipelining)**
  - k-stufige Befehlspipeline (k=5)



Logische Phasen:
IF: Befehl holen
ID: Befehl dekodieren / Operanden bereitstellen
EX: Befehl ausführen
MA: Speicherzugriff
WB: Zurückschreiben

5.2 Pipelining des Maschinenbefehlszyklus

- **RISC ISAPipelining – Maschinenbefehlszyklus (Instruction Pipelining)**
  - Effizientes Pipelining des Maschinenbefehlszyklus
    - Alle Pipelinestufen benützen unterschiedliche Ressourcen
    - Siehe DLX-Pipeline

    - Pipelining erhöht den Durchsatz
      - Mit jedem Takt wird unter Annahme idealer Verhältnisse ein Befehl geholt bzw. beendet.
      - Im eingeschwungenen Zustand der Pipeline: Durchsatz = 1 Befehl / Taktzyklus (CPI ~ 1)
      - Aber, reduziert nicht die Ausführungszeit einer individuellen Instruktion

    - Zykluszeit, Taktzyklus:
      - Abhängig vom kritischen Pfad, der langsamsten Pipelinestufe

5.2 Pipelining des Maschinenbefehlszyklus

- **Pipelining – Maschinenbefehlszyklus (Instruction Pipelining)**
- Leistungsbetrachtung
  - Ausführung eines Programms mit *n* Befehlen
    - Sequentielle Ausführung: $n * k$ Taktzyklen (bei k Ausführungsstufen)
    - Pipelineverarbeitung auf Prozessor mit *k*-stufiger Pipeline: $n + (k - 1)$
      - n Taktzyklen, um alle n Befehle der ersten Pipelinestufe zuzuführen
      - $k - 1$ zusätzliche Taktzyklen, um den letzten Befehl fertigzustellen
    - Speedup *S* (Leistungssteigerung):

$$S = \frac{n * k}{n + (k - 1)}$$

5.2 Pipelining des Maschinenbefehlszyklus

- **Pipelining – Maschinenbefehlszyklus (Instruction Pipelining)**
  - Effizientes Pipelining des Maschinenbefehlszyklus
    - Alle Pipelinestufen benützen unterschiedliche Ressourcen
  - Pipelining erhöht den Durchsatz
    - Mit jedem Takt wird unter Annahme idealer Verhältnisse ein Befehl geholt bzw. beendet.
    - Im eingeschwungenen Zustand der Pipeline: Durchsatz = 1 Befehl / Taktzyklus (CPI ~ 1)
    - Aber, reduziert nicht die Ausführungszeit einer individuellen Instruktion
  - Zykluszeit, Taktzyklus:
  - Abhängig vom kritischen Pfad, der langsamsten Pipelinestufe
    - der Schaltnetze: $t_i \quad (i = 1,2, \dots k)$
    - der Pipeline-Register: t_{reg}
    - Länge des Taktzyklus: $t = \max\{t_1, t_2, \dots, t_k\} + t_{reg}$

5.2 Pipelining des Maschinenbefehlszyklus

- **Pipeline-Konflikte (Pipeline Hazards, Pipeline-Hemmnisse)**
  - **Strukturkonflikte**
    - Ergeben sich aus Ressourcenkonflikten
    - Die Hardware kann nicht alle möglichen Kombinationen von Befehlen unterstützen, die sich in der Pipeline befinden können
    - Beispiel:
      - Gleichzeitiger Schreibzugriff zweier Befehle auf eine Registerdatei mit nur einem Schreibeingang
  - **Datenkonflikte**
    - Ergeben sich aus Datenabhängigkeiten zwischen Befehlen im Programm
    - Beispiel:
      - Instruktion benötigt das Ergebnis einer vorangehenden und noch nicht abgeschlossenen Instruktion in der Pipeline
      - D.h. ein Operand ist noch nicht verfügbar
  - **Steuerkonflikte**
    - Treten bei Verzweigungsbefehlen und anderen Instruktionen auf, die den Befehlszähler verändern

5.3 Pipeline-Konflikte

- **Datenabhängigkeiten und Konflikte:**
  - **Datenabhängigkeiten zwischen Befehlen im Programm**
    - sind Eigenschaften des Programms!
    - zeigen die Möglichkeit eines Konflikts an!
    - legen die Programmordnung fest, d.h. die Reihenfolge in der die Ergebnisse berechnet werden müssen.
    - legen eine obere Grenze für den Grad des Parallelismus fest, der ausgenützt werden kann
  - **Konflikte:**
    - Es hängt von der Pipeline-Organisation ab, ob eine gegebene Anhängigkeit zu einem Konflikt führt und ob Konflikte zu einem Anhalten der Pipeline führen!

5.3 Pipeline-Konflikte

- **Datenabhängigkeiten und Konflikte:**
  - Zwischen zwei aufeinander folgenden Befehle $\texttt{Inst}_1$ und $\texttt{Inst}_2$ besteht eine echte Datenabhängigkeit (true dependence) δ^t von $\texttt{Inst}_1$ zu $\texttt{Inst}_2$, wenn $\texttt{Inst}_1$ seine Ausgabe in ein Register (oder Speicherstelle) schreibt, das von $\texttt{Inst}_2$ als Eingabe gelesen wird
  - Zwischen zwei aufeinander folgenden Befehle $\texttt{Inst}_1$ und $\texttt{Inst}_2$ besteht eine Gegenabhängigkeit (antidependence) δ^a von $\texttt{Inst}_1$ zu $\texttt{Inst}_2$, falls $\texttt{Inst}_1$ Daten von einem Register (oder Speicherstelle) liest, das anschließend von $\texttt{Inst}_2$ überschrieben wird
  - Zwischen zwei aufeinander folgenden Befehle $\texttt{Inst}_1$ und $\texttt{Inst}_2$ besteht eine Ausgabeabhängigkeit (output dependence) δ^o von $\texttt{Inst}_2$ zu $\texttt{Inst}_1$, wenn beide in das gleiche Register (oder Speicherstelle) schreiben

5.3 Pipeline-Konflikte

- **Datenabhängigkeiten und Konflikte:**
  - **Beispiel:**

```
S1:     add     r1,r2,2         #       r1 := r2 + 2
S2:     add     r4,r1,r3        #       r4 := r1 + r3
S3:     mul     r3,r5,3         #       r3 := r5 * 3
S4:     mul     r3,r6,3         #       r3 := r6 * 3
```

5.3 Pipeline-Konflikte

- **Datenabhängigkeiten und Konflikte:**
  - **Beispiel:**

```
S1:     add     r1,r2,2         #       r1 := r2 + 2

S2:     add     r4,r1,r3        #       r4 := r1 + r3

S3:     mul     r3,r5,3         #       r3 := r5 * 3

S4:     mul     r3,r6,3         #       r3 := r6 * 3
```

Echte Datenabhängigkeit d^t

5.3 Pipeline-Konflikte

- **Datenabhängigkeiten und Konflikte:**
  - **Beispiel:**

```
S1:     add     r1,r2,2         #       r1 := r2 + 2

S2:     add     r4,r1,r3        #       r4 := r1 + r3

S3:     mul     r3,r5,3         #       r3 := r5 * 3

S4:     mul     r3,r6,3         #       r3 := r6 * 3
```

Gegenabhängigkeit d^a

5.3 Pipeline-Konflikte

- **Datenabhängigkeiten und Konflikte:**
  - **Beispiel:**

```
S1:     add     r1,r2,2         #       r1 := r2 + 2

S2:     add     r4,r1,r3        #       r4 := r1 + r3

S3:     mul     r3 r5,3         #       r3 := r5 * 3

S4:     mul     r3 r6,3         #       r3 := r6 * 3
```

Ausgabeabhängigkeit d^o

5.3 Pipeline-Konflikte

- **Datenabhängigkeiten und Konflikte:**
  - Namensabhängigkeiten
    - Treten auf, wenn zwei Instruktionen dasselbe Register dieselbe Speicherzelle (den Namen) verwenden, aber kein Datenfluss zwischen den Befehlen mit dem Namen verbunden ist.
      - Gegenabhängigkeit
      - Ausgabeabhängigkeit

5.3 Pipeline-Konflikte

- **Datenabhängigkeiten und Konflikte:**
  - **Konflikte**
    - **Lese-nach-Schreibe-Konflikt (read after write, RAW)**
      - Wird durch echte Abhängigkeit verursacht
    - **Schreibe-nach-Lese-Konflikt (write after read, WAR)**
      - Wird durch Gegenabhängigkeit verursacht
      - Tritt z.B. dann auf, wenn in einer Pipeline die Schreibestufe der Lesestufe vorangeht
    - **Schreibe-nach-Schreibe-Konflikt (write after write, WAW)**
      - Wird durch Ausgabeabhängigkeit verursacht
      - Tritt in Pipelines auf, die in mehr als einer Stufe schreiben, oder es erlauben, dass die Ausführung eines Befehls fortgesetzt werden darf, wenn ein vorhergehender Befehl angehalten worden ist

5.3 Pipeline-Konflikte

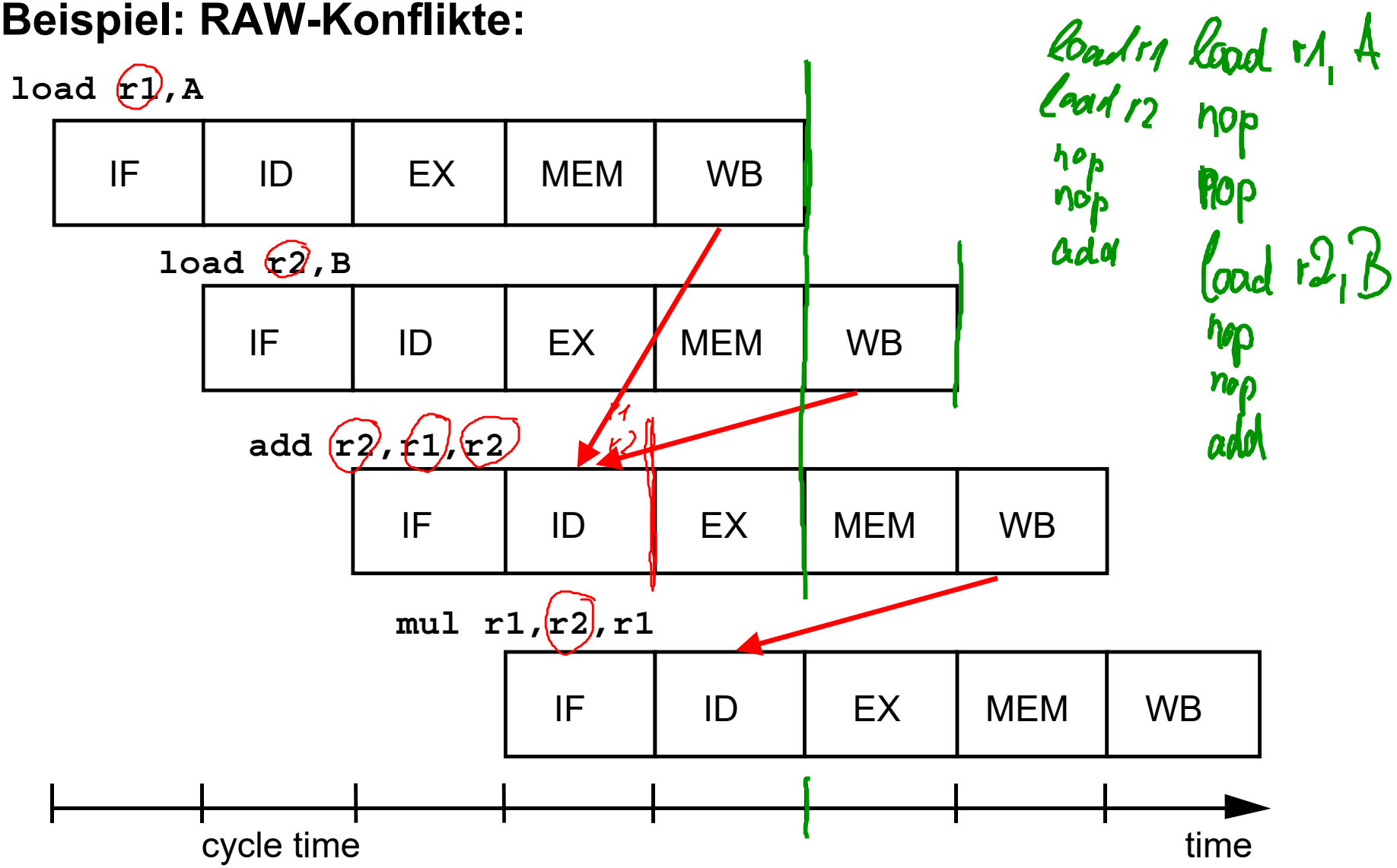
- **Datenabhängigkeiten und Konflikte:**
  - **Konflikte**
    - Können WAR und WAW in der fünfstufigen DLX-Pipeline auftreten?

5.3 Pipeline-Konflikte

- **Datenabhängigkeiten und Konflikte:**
  - **Konflikte**
    - Können WAR und WAW in der fünfstufigen DLX-Pipeline auftreten?
    - Nein, weil:
      - Alle Befehle gehen durch alle 5 Stufen (kein Überholen),
      - Lesen aus Registern immer in Stufe 2, und
      - Schreiben in Register immer in Stufe 5
      - WAR und WAW treten bei “komplexeren” Pipelines auf

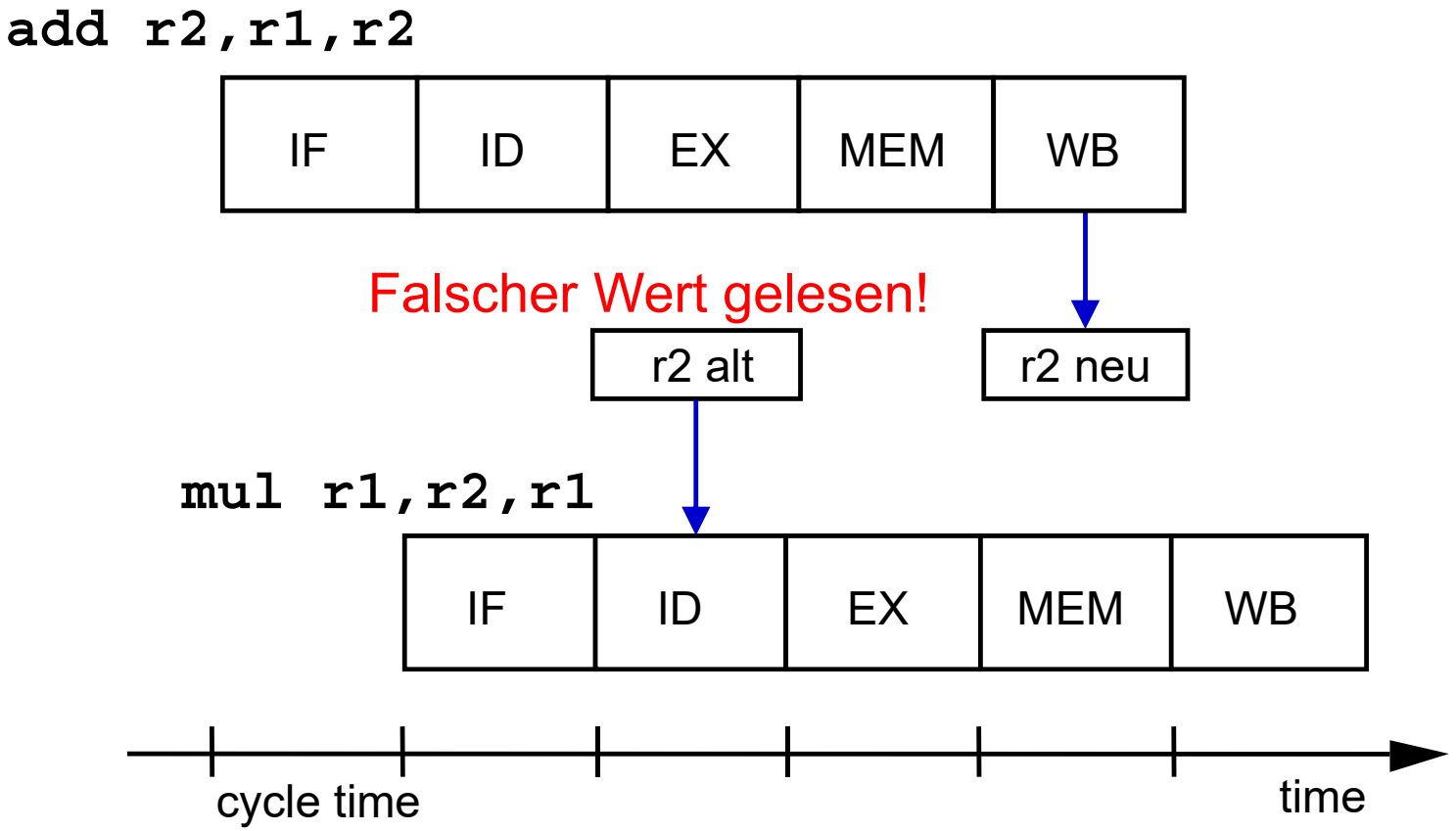
5.3 Pipeline-Konflikte

- **Beispiel: RAW-Konflikte:**



5.3 Pipeline-Konflikte

- **Beispiel: RAW-Konflikte:**
  - **Fehlzuweisung durch Datenkonflikt**

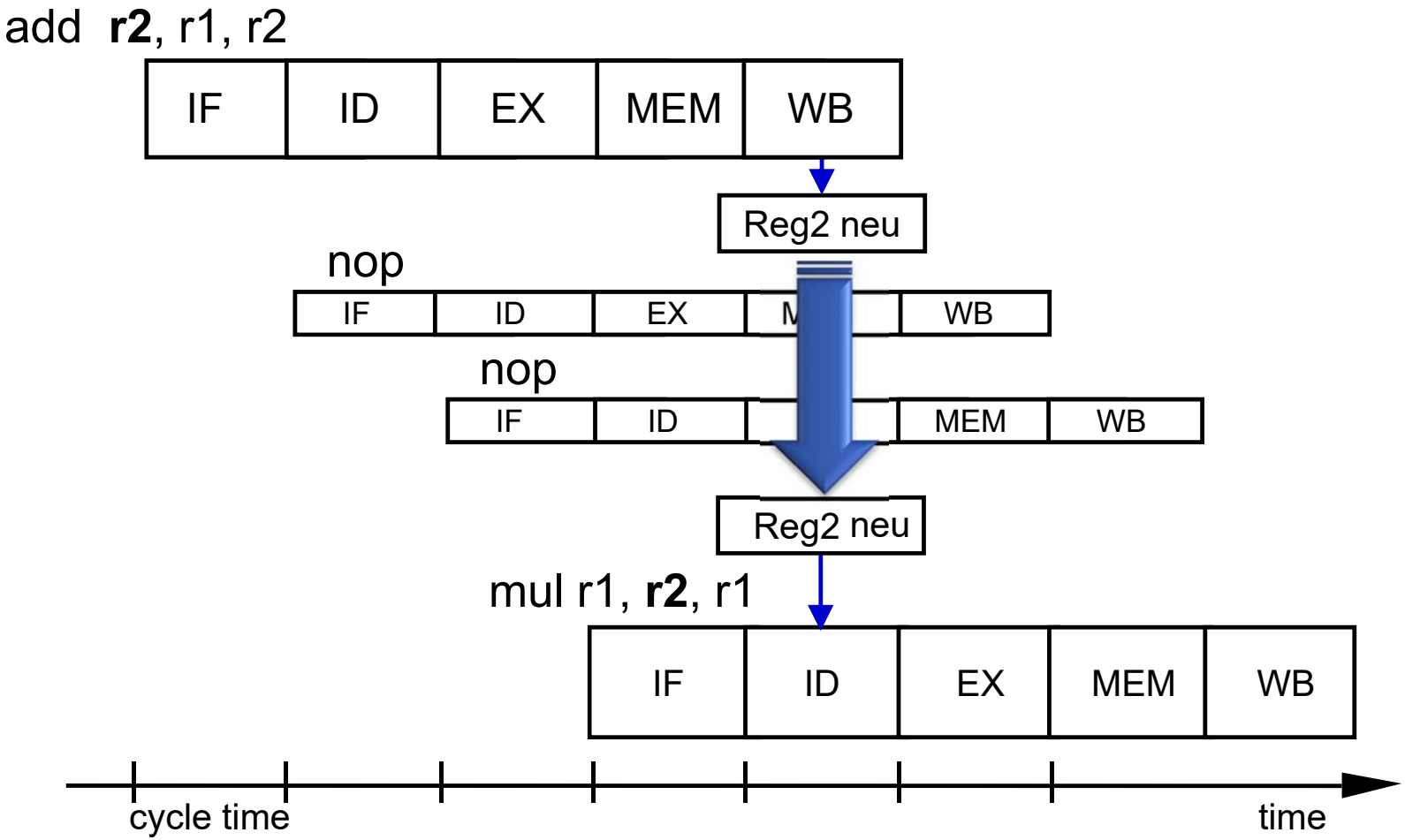


5.3 Pipeline-Konflikte

- **Auflösen von Pipeline-Konflikten**
  - **Software-Lösung durch Compiler**
    - Aufgabe des Compilers:
      - Erkennen von Datenkonflikten
      - Einfügen von Leeroperationen (nop) nach jedem Befehl, der einen Konflikt verursacht oder verursachen kann
    - Statische Befehlsanordnung:
      - Instruction Scheduling, Pipeline Scheduling
      - Umordnen der Befehle des Programms (Code-Optimierung), um Leeroperationen zu eliminieren

5.3 Pipeline-Konflikte

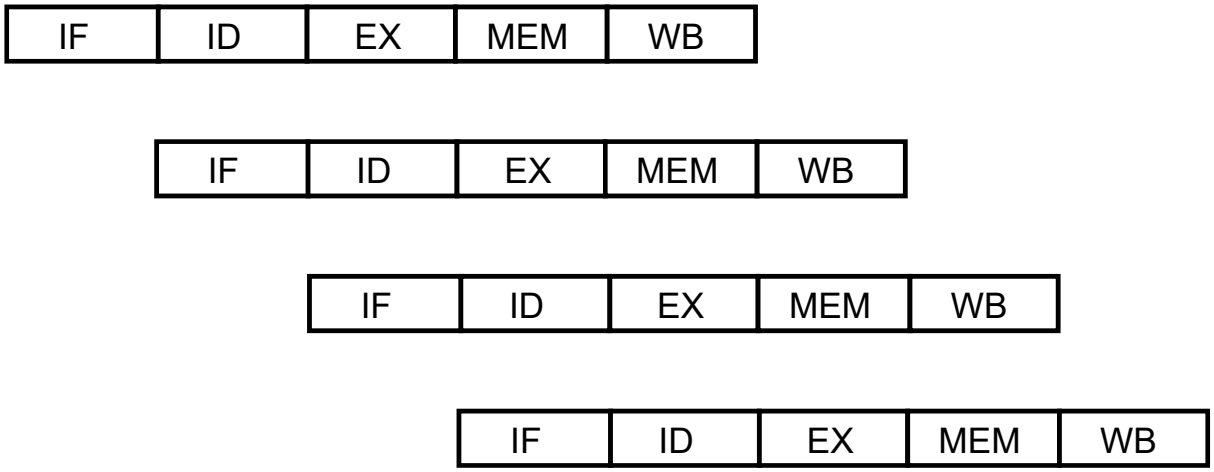
- **Auflösen von Pipeline-Konflikten**
  - **Software-Lösung durch Compiler: Einfügen von nop**



5.3 Pipeline-Konflikte

- **Auflösen von Pipeline-Konflikten**
  - **Software-Lösung durch Compiler**
    - Befehlsanordnung (instruction scheduling):
    - Befehlsumordnungen, um nops zu entfernen

```
add r0, r1, r2
sub r3, r0, r4
mul r5, r6, r7
mul r8, r9, r10
```

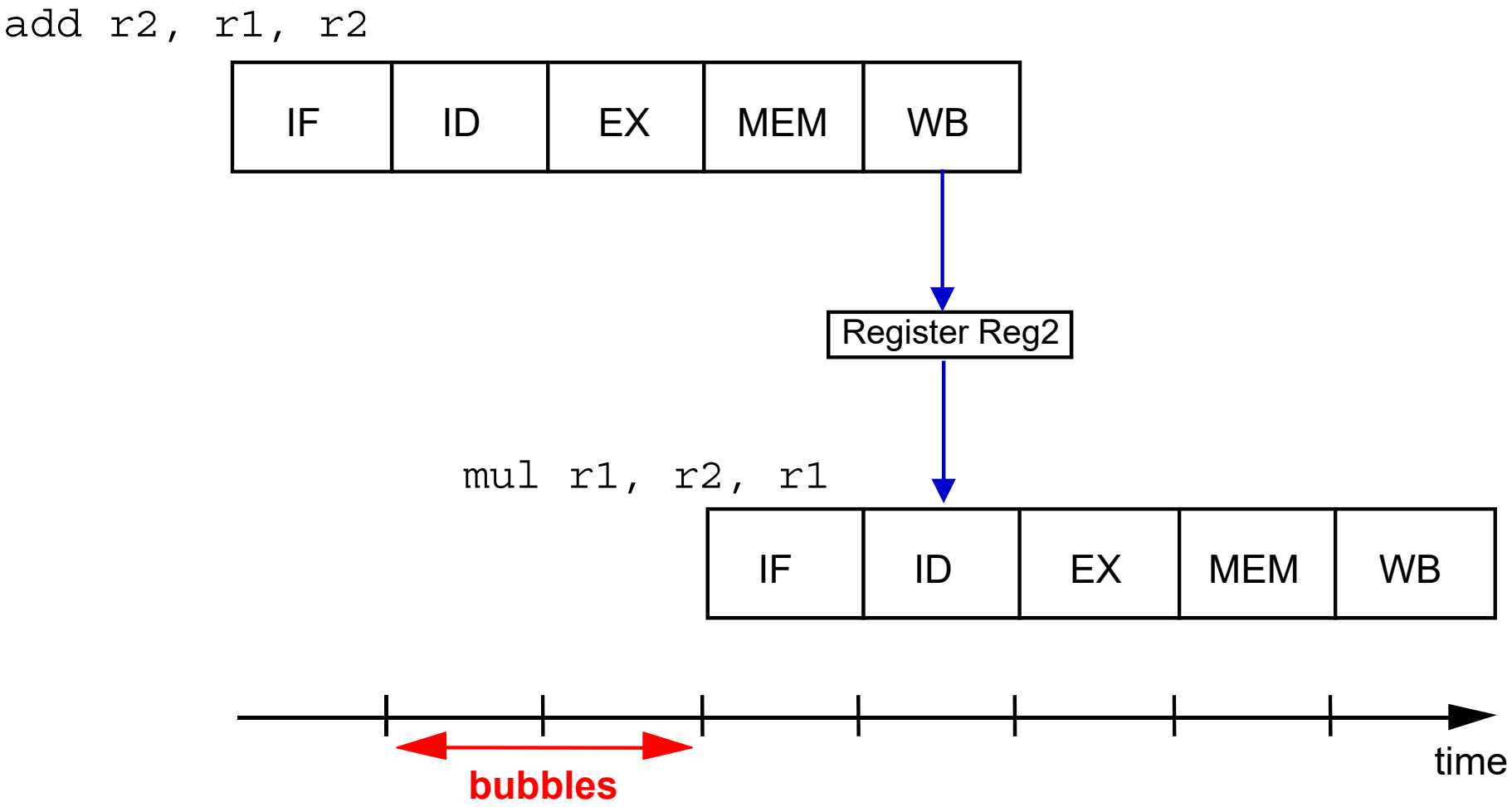


5.3 Pipeline-Konflikte

- **Auflösen von Pipeline-Konflikten**
  - **Hardware-Lösungen**
    - Konflikt muss per HW entdeckt und aufgelöst werden!
    - **Leerlauf der Pipeline:**
      - Die einfachste Art, mit solchen Datenkonflikten umzugehen ist es, Inst2 in der Pipeline für zwei Takte anzuhalten.
      - Auch als Pipeline-Sperrung (interlocking) oder Pipeline-Leerlauf (stalling) bezeichnet
    - **Forwarding:**
      - Wenn ein Datenkonflikt erkannt wird, so sorgt eine Hardware-Schaltung dafür, dass der betreffende Operand nicht aus dem Universalregister, sondern direkt aus dem ALU-Ausgaberegister der vorherigen Operation in das ALU-Eingaberegister übertragen wird
    - **Forwarding with interlocking:**
      - Forwarding löst nicht alle möglichen Datenkonflikte auf

5.3 Pipeline-Konflikte

- **Auflösen von Pipeline-Konflikten**
  - **Hardware-Lösungen: Interlocking**

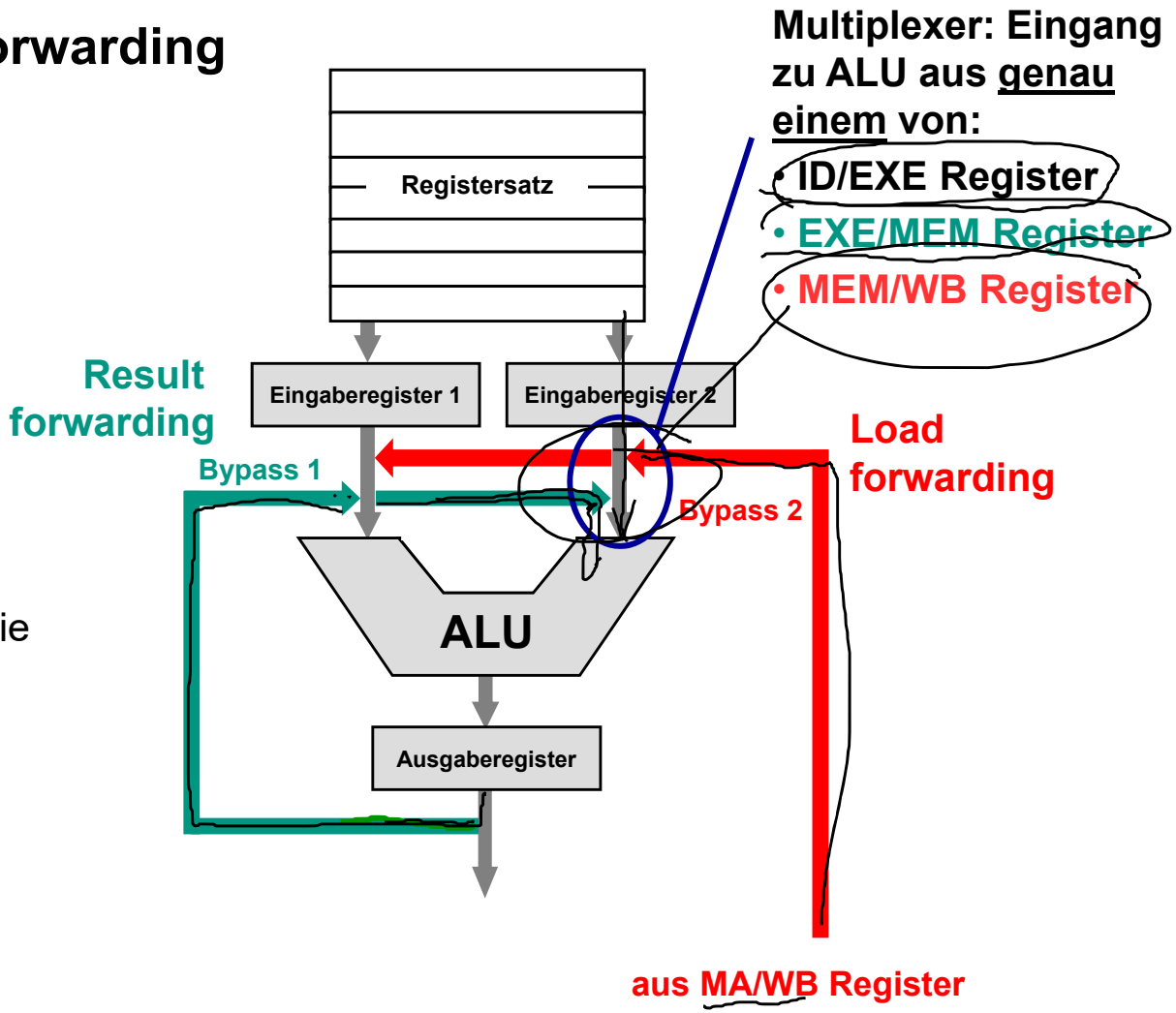


5.3 Pipeline-Konflikte

- **Auflösen von Pipeline-Konflikten**
  - **Hardware-Lösungen: Forwarding**

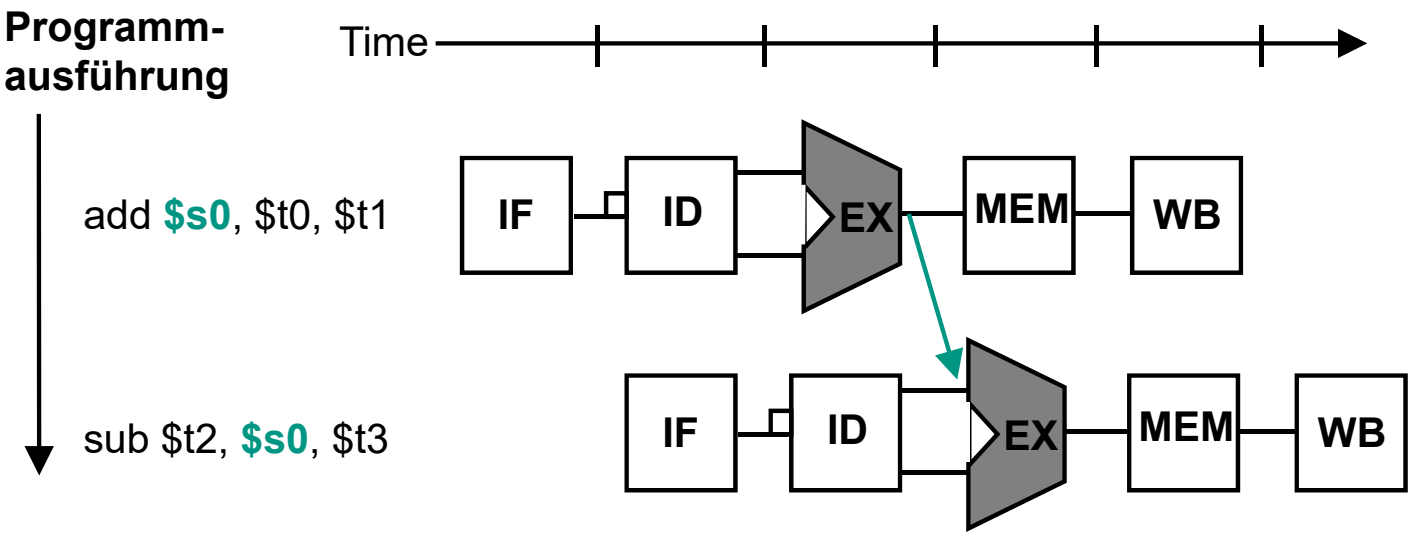
Rückführung des ALU-Ausgaberegister (result forwarding) bzw. des Ladewertregisters (load forwarding) auf die ALU-Eingaberegister

Erhöhter Hardware- und Steuerungs-aufwand



5.3 Pipeline-Konflikte

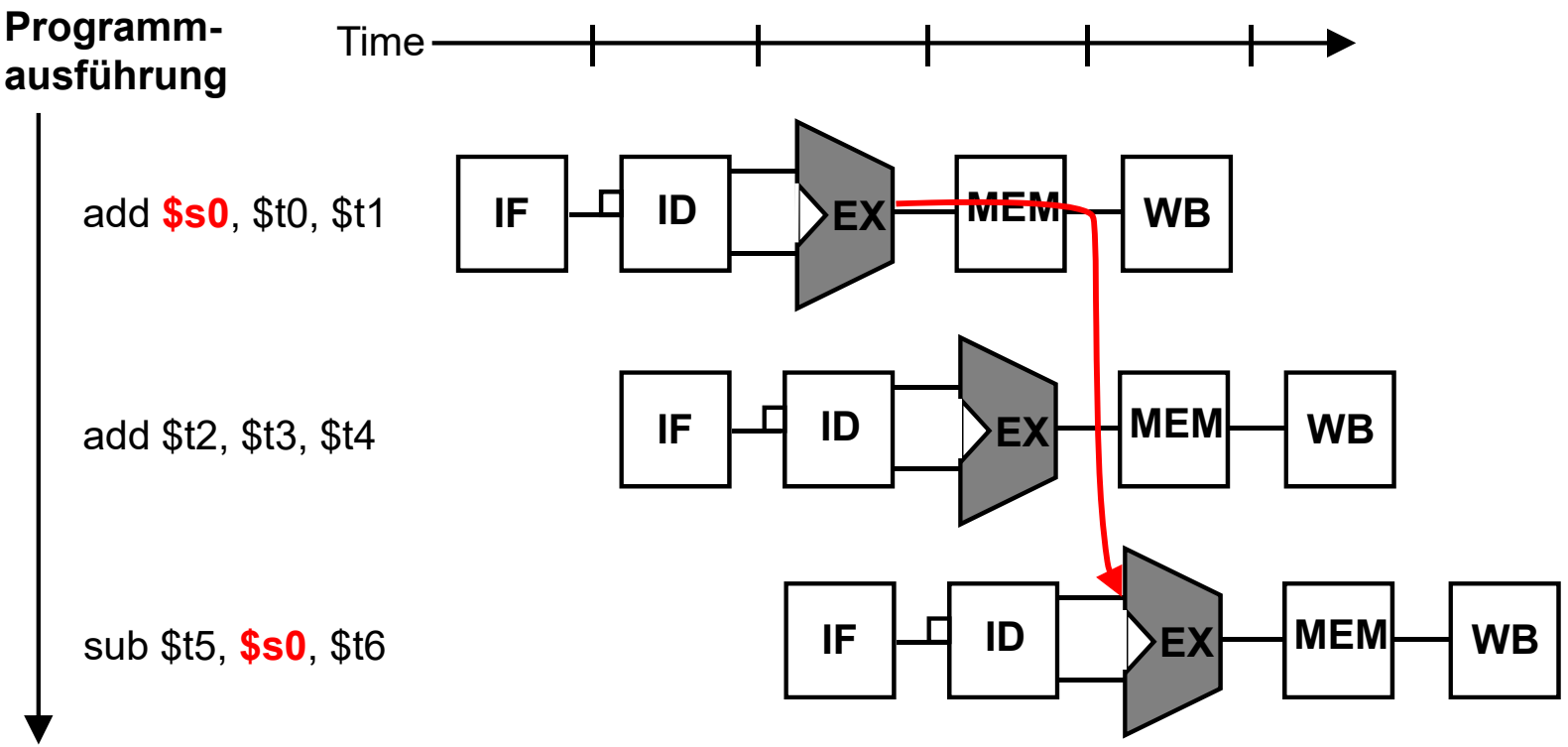
- **Auflösen von Pipeline-Konflikten**
  - **Hardware-Lösungen: Result-Forwarding**



Erhöhter Hardware- und Steuerungsaufwand für Forwarding-Logik und zusätzliche Datenpfade

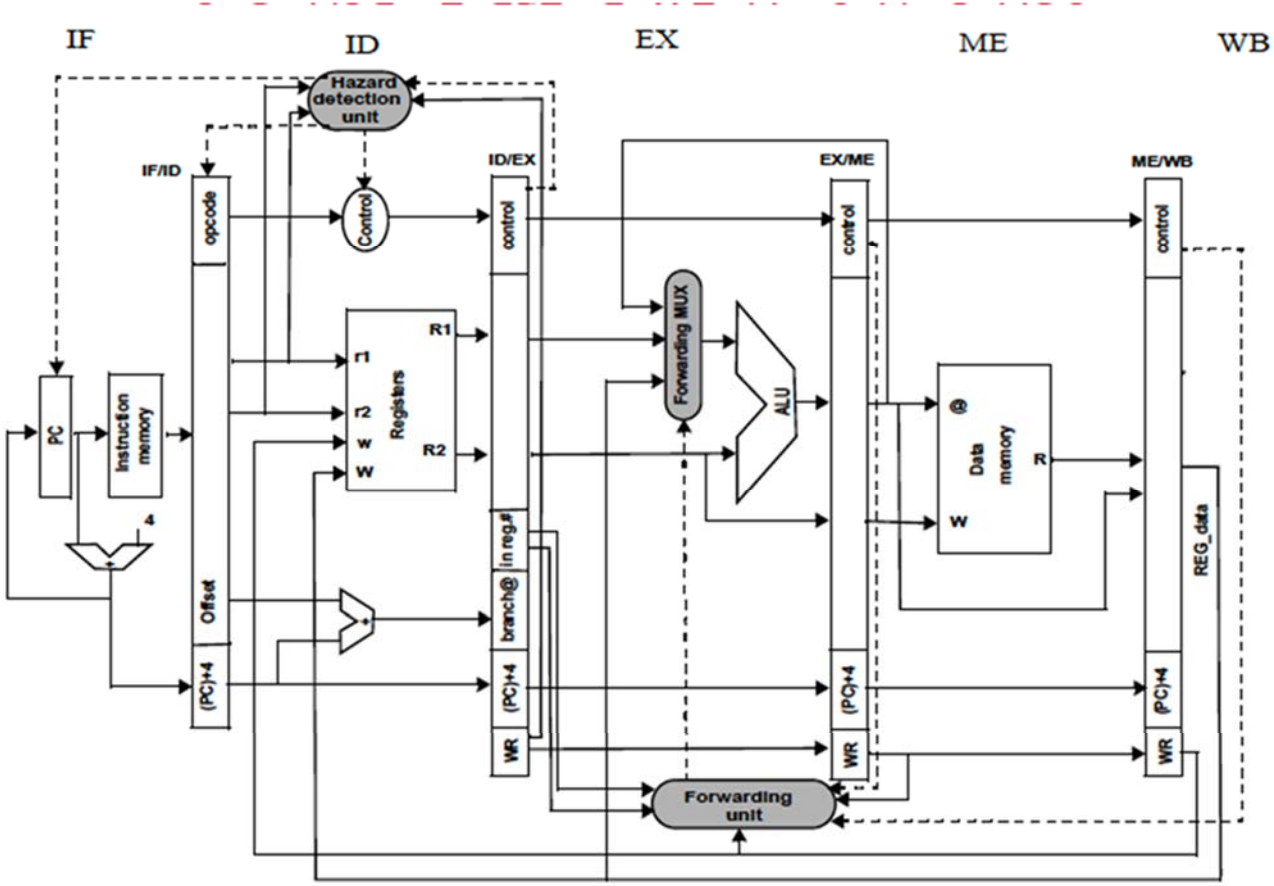
5.3 Pipeline-Konflikte

- **Auflösen von Pipeline-Konflikten**
  - **Hardware-Lösungen: Load-Forwarding**



5.3 Pipeline-Konflikte

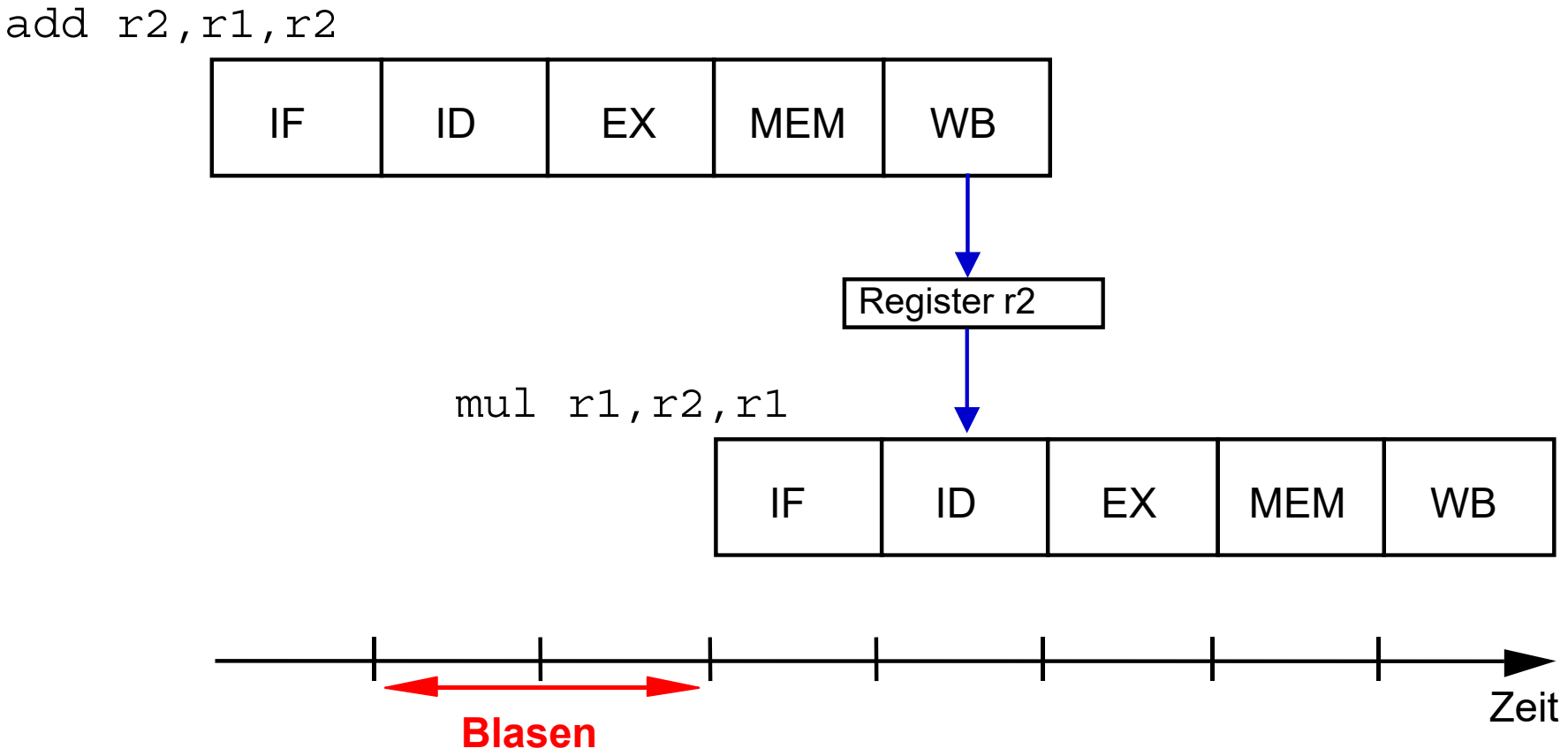
- **Auflösen von Pipeline-Konflikten**
  - **Hardware-Lösungen**



Quelle: Michel Dubois, Murali Annavaram, Per Stenström: Parallel Computer Organization and Design. Cambridge University Press, 2012

5.3 Pipeline-Konflikte

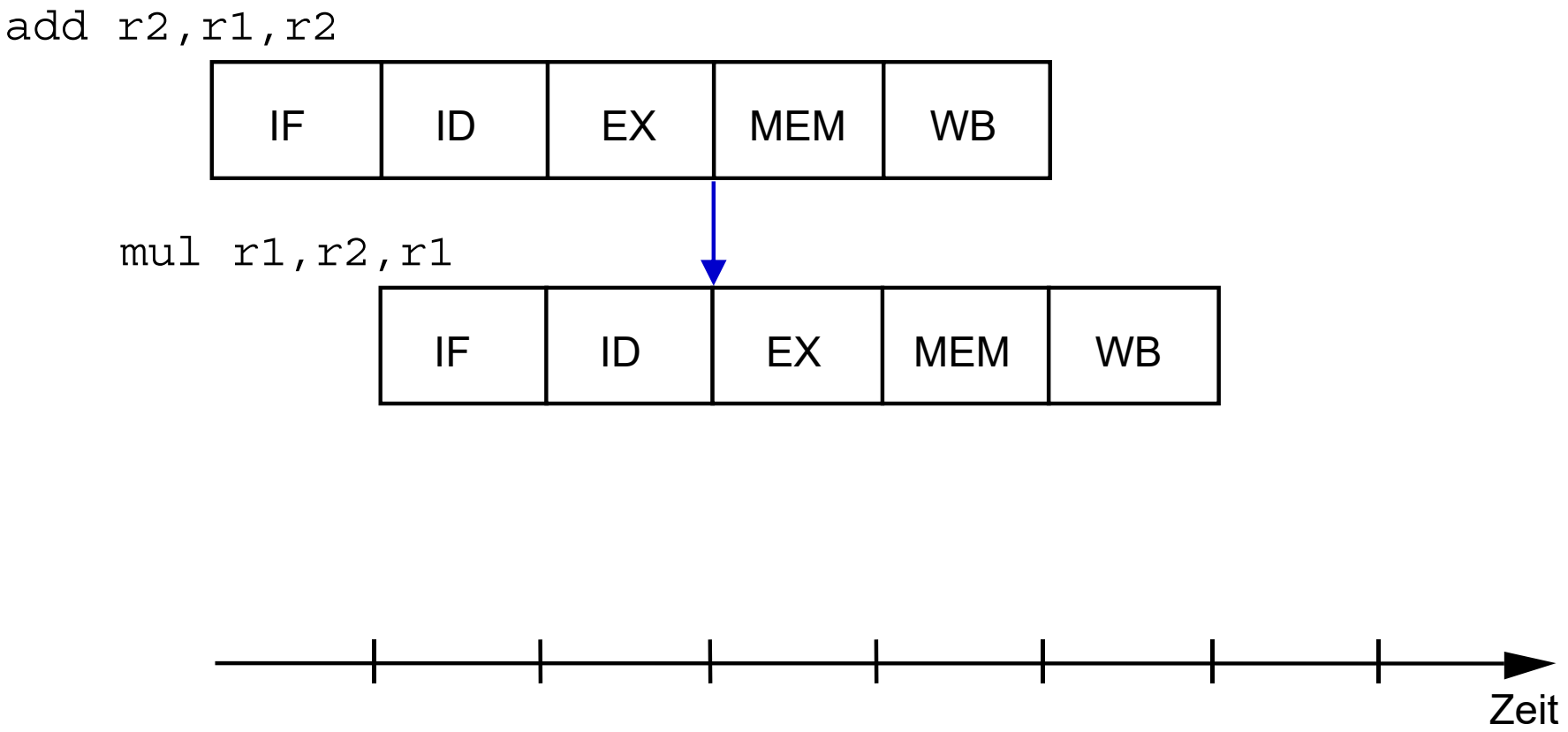
- **Auflösen von Pipeline-Konflikten**
  - **Hardware-Lösungen: Vergleich Interlocking vs. Forwarding**



Anhalten des `mul`-Befehls für zwei Takte

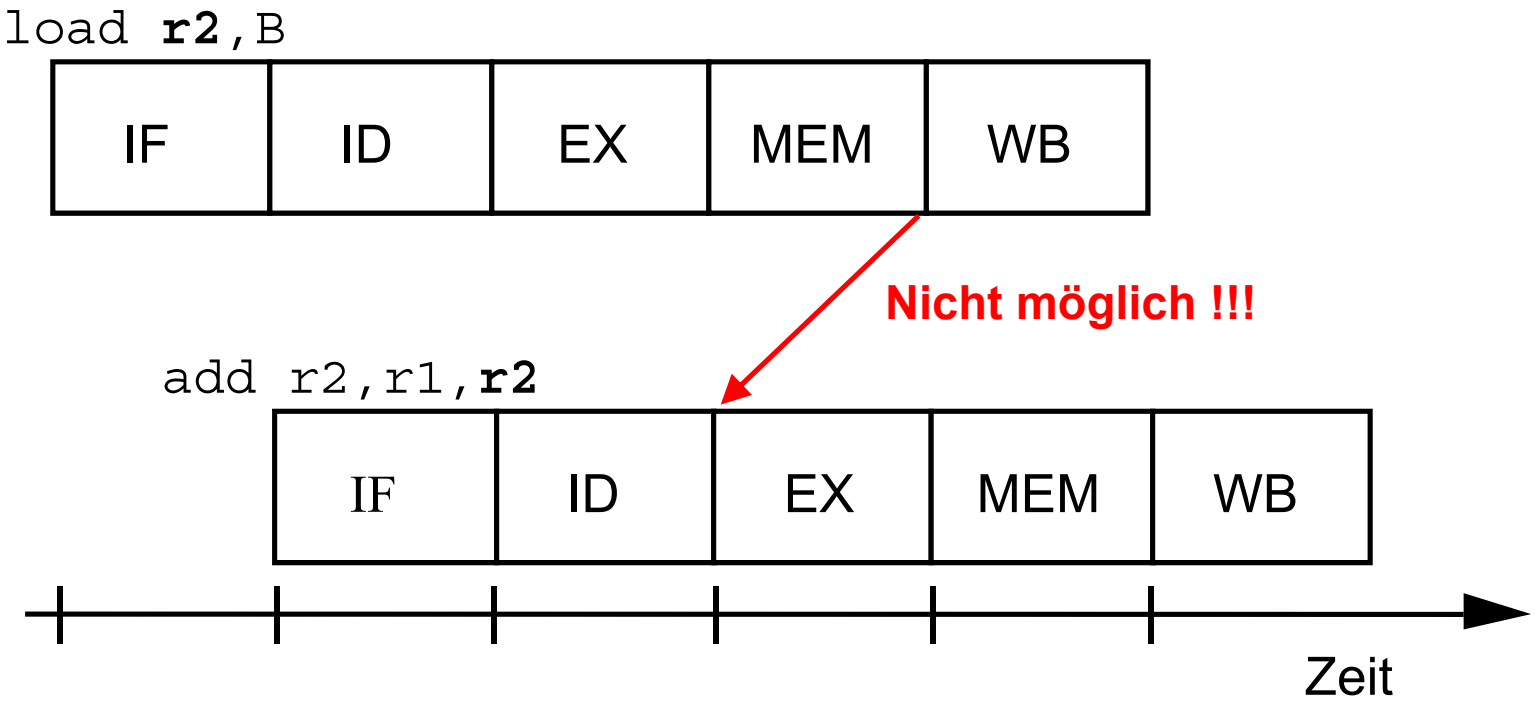
5.3 Pipeline-Konflikte

- **Auflösen von Pipeline-Konflikten**
  - **Hardware-Lösungen: Vergleich Interlocking vs. Forwarding**



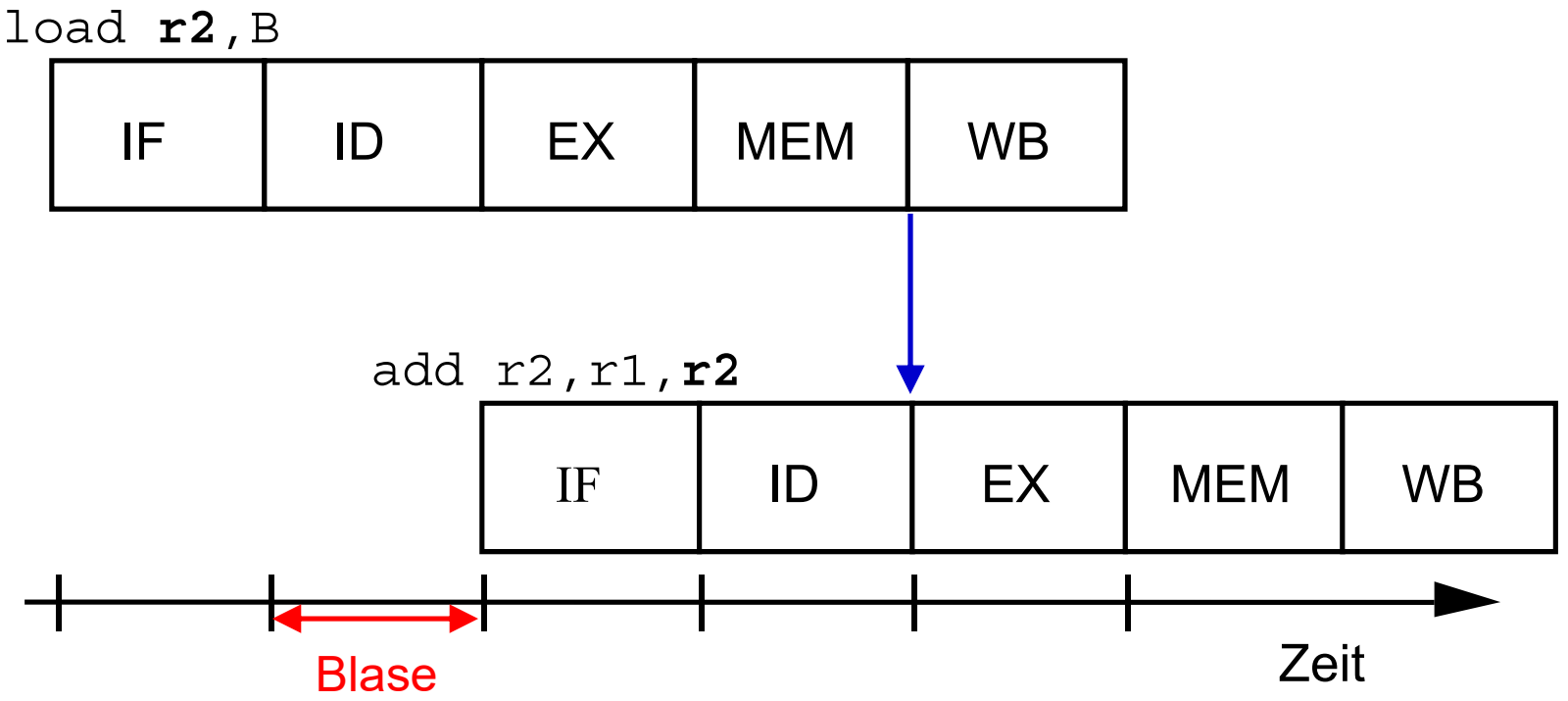
5.3 Pipeline-Konflikte

- **Auflösen von Pipeline-Konflikten**
  - **Hardware-Lösungen: Forwarding mit Interlocking**
    - Beispiel: Speicherzugriff (z.B. Ladebefehl): EX-Stufe berechnet die effektive Adresse, MEM-Stufe lädt Daten



5.3 Pipeline-Konflikte

- **Auflösen von Pipeline-Konflikten**
  - **Hardware-Lösungen: Forwarding mit Interlocking**
    - Lösung: Der **`add`** Befehl muss angehalten werden, bis die geladenen Daten im Ladewertregister der MEM-Stufe verfügbar sind

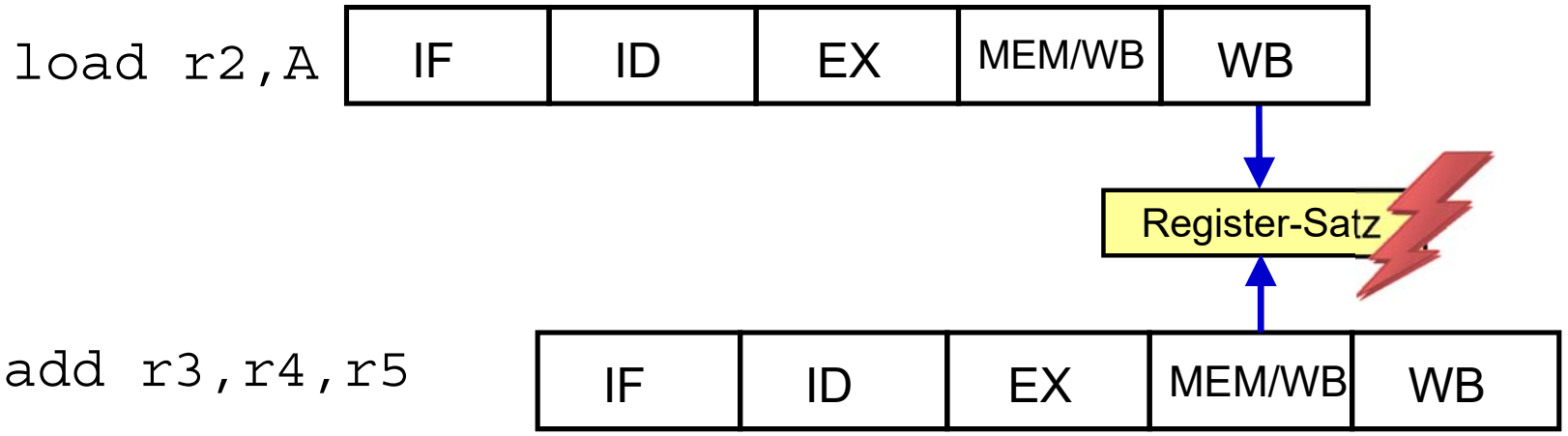


5.3 Pipeline-Konflikte

- **Resourcen- oder Strukturkonflikte**
  - Treten auf, wenn zwei oder mehr Befehle gleichzeitig dieselbe Ressource benötigen, auf die aber nur einmal zugegriffen werden kann
  - Treten bei einer einfachen Pipeline, wie der DLX-Pipeline, nicht auf
  - Ziel beim Pipeline-Entwurf: Ressourcenkonflikte möglichst zu vermeiden und dort, wo sie nicht vermeidbar sind, zu erkennen und behandeln

5.3 Pipeline-Konflikte

- **Resourcen- oder Strukturkonflikte**
  - **Beispiel: Prozessor mit veränderter DLX-Pipeline**
    - Die MEM-Stufe sei in der Lage, bei einem Register-Register-Befehl die Ausgabe der ALU durch einen Schreibkanal auf den Registersatz zurückzuschreiben
    - Im Beispiel greifen zwei Befehle gleichzeitig auf einen nur einfach vorhandenen Schreibeingang zu



5.3 Pipeline-Konflikte

- **Resourcen- oder Strukturkonflikte**
  - **Lösungen (Hardware):**
    - Arbitrierung mit Interlocking:
      - Arbitrierungslogik hält den Befehl an, der den Konflikt verursacht Verzögerung durch Leerzyklen
    - Übertaktung:
      - Die Ressource, die den Konflikt hervorruft schneller takten als die übrigen Pipeline-Stufen
    - Ressourcenreplizierung:
      - Vervielfachung der Ressourcen
      - Beispiel: Registersatz mit mehreren Schreibkanälen

5.3 Pipeline-Konflikte

- **Steuerflusskonflikte**
  - Programmsteuerbefehle:
    - die bedingten und die unbedingten Sprungbefehle,
    - die Unterprogrammaufruf- und -rückkehrbefehle sowie
    - die Unterbrechungsbefehle
  - Steuerflussabhängigkeiten verursachen Steuerflusskonflikte in der DLX-Pipeline, da
    - der Programmsteuerbefehl erst in der ID-Stufe als solcher erkannt und damit bereits ein Befehl des falschen Programmpfades in die Pipeline geladen wurde
    - die Sprungzieladresse von der ALU erst in der EX-Stufe berechnet wird und
    - der PC am Ende der MEM-Stufe ersetzt wird