Rechnerarchitektur (RA)

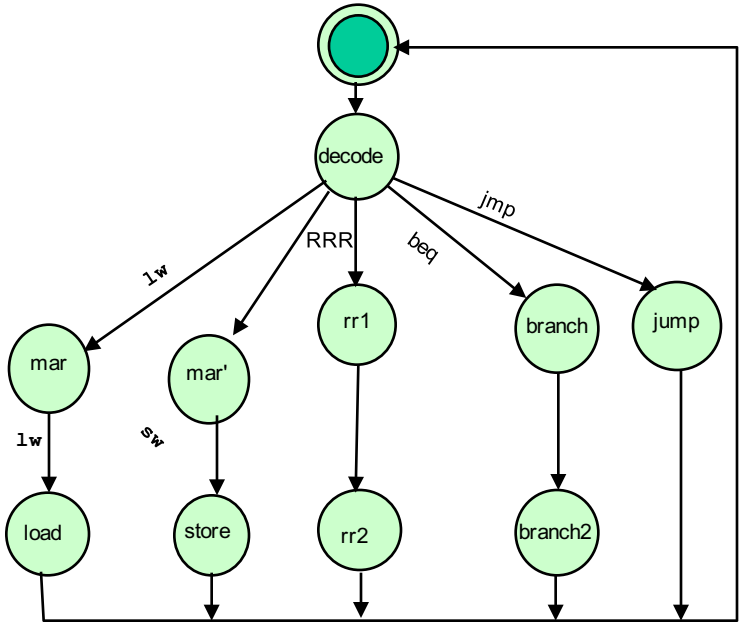
Sommersemester 2017

Pipelines

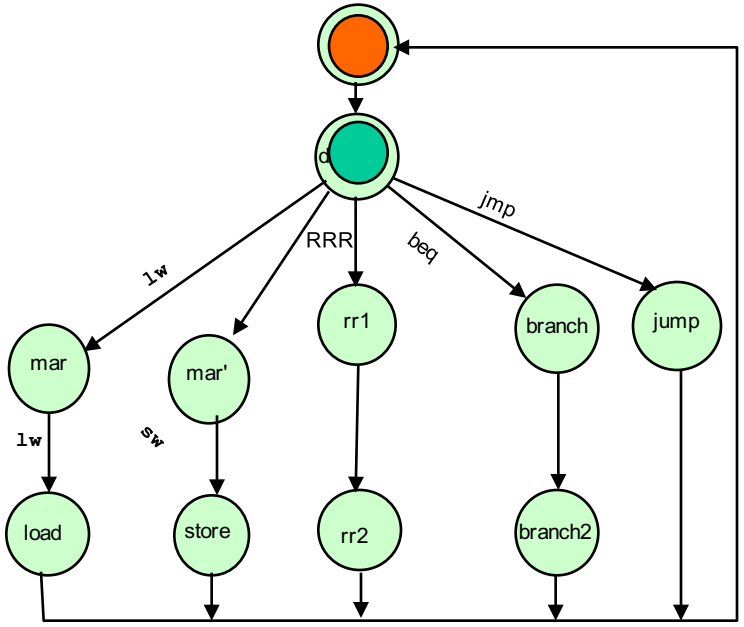
Jian-Jia Chen
Informatik 12
http://ls12-www.cs.tu-dortmund.de/daes/

2017/05/30

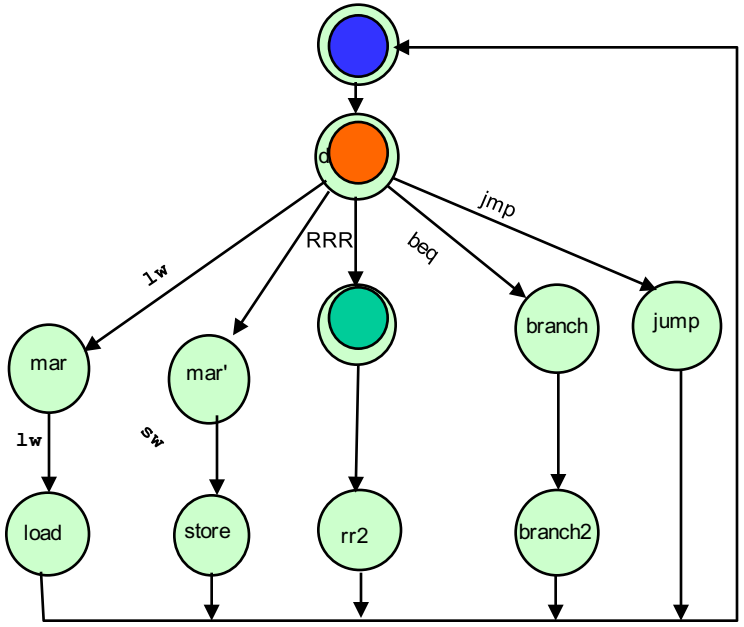
Mikroprogrammierung → Fließbandverarbeitung



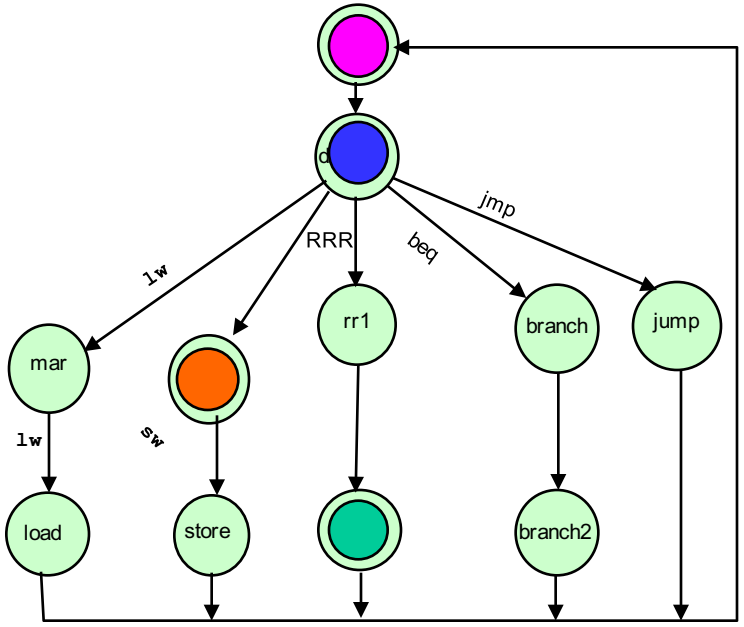
Mikroprogrammierung → Fließbandverarbeitung



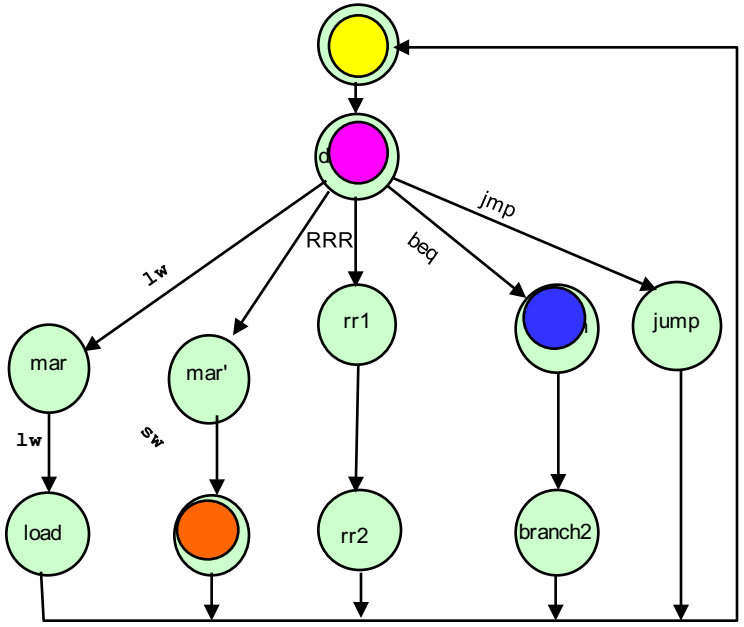
Mikroprogrammierung → Fließbandverarbeitung

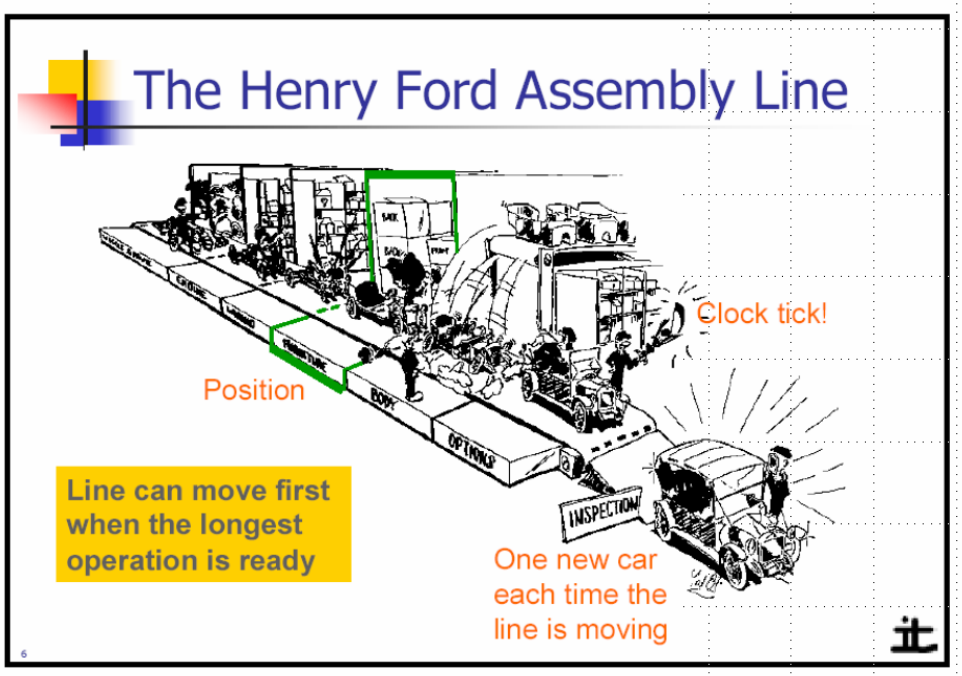


Mikroprogrammierung → Fließbandverarbeitung



Mikroprogrammierung → Fließbandverarbeitung

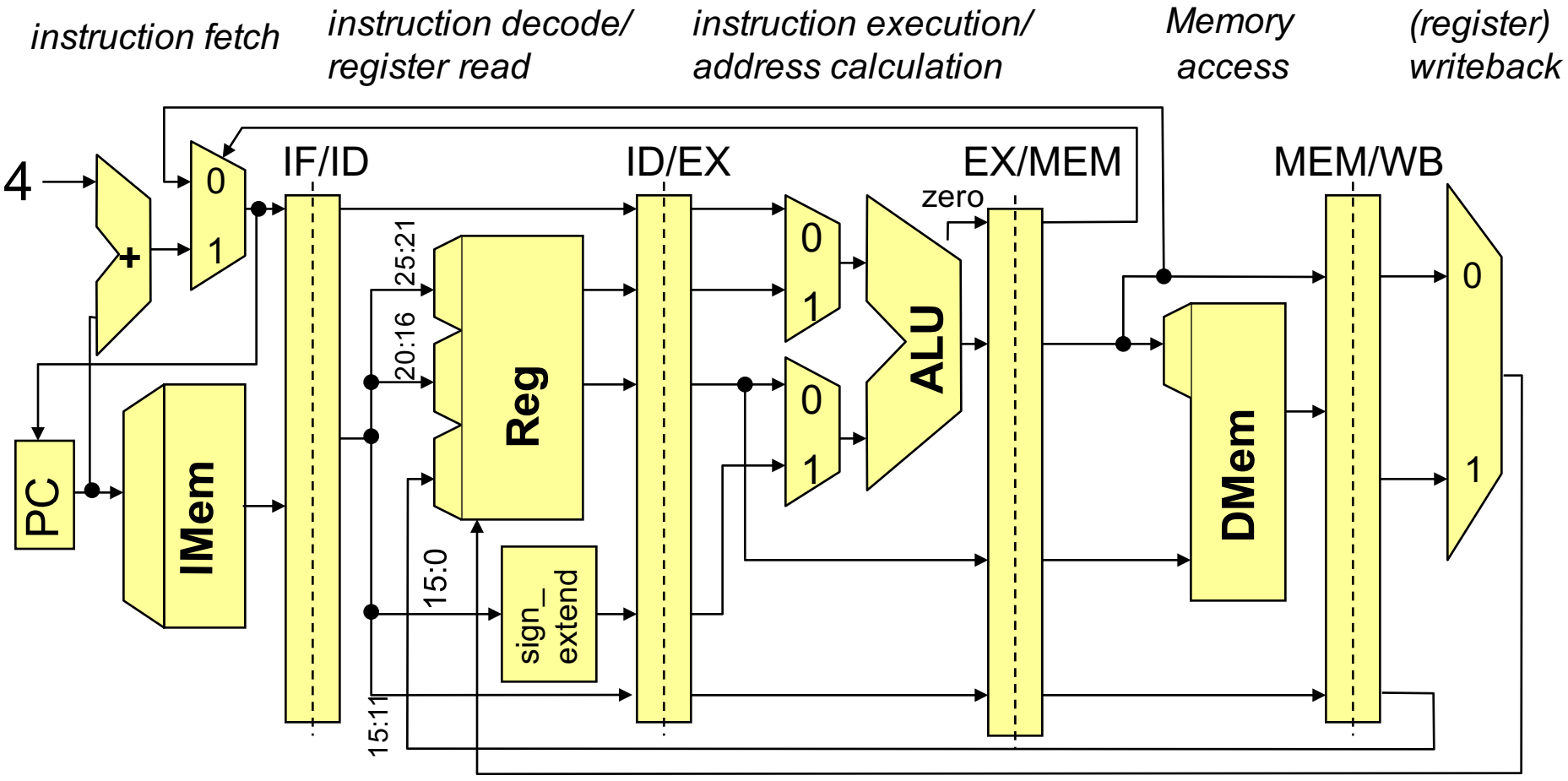




www.it.lth.se/courses/dsi/material/Lectures/Lecture6.pdf

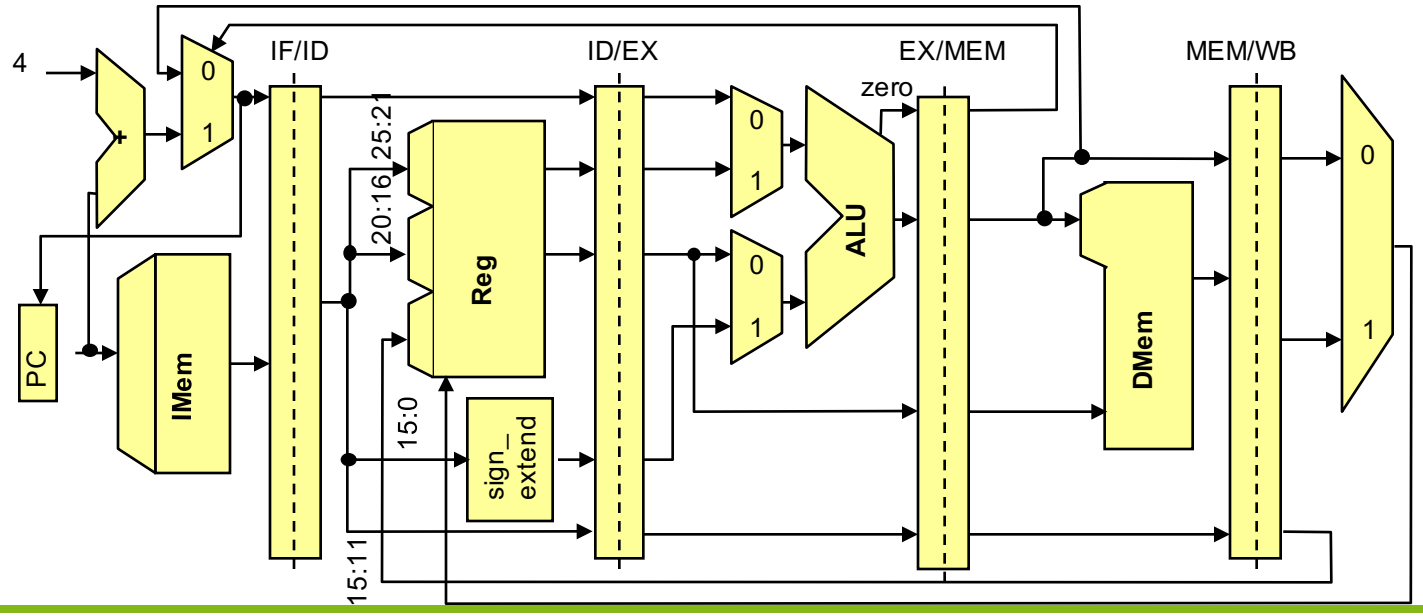
2.3.2 Fließbandverarbeitung

Fließband-Architektur (engl. *pipeline architecture*): Bearbeitung mehrerer Befehle gleichzeitig, analog zu Fertigungsfließbändern. Beispiel MIPS:



Änderungen gegenüber der Struktur ohne Fließband

- Separater Addierer für Programm-Folgeadressen.
- Konzeptuelle Aufteilung des Speichers in Daten- und Befehlsspeicher.
- Aufteilung des Rechenwerks in Fließbandstufen, Trennung durch Pufferregister, **`T`** und **`Befehlsregister`** werden Pufferregistern.

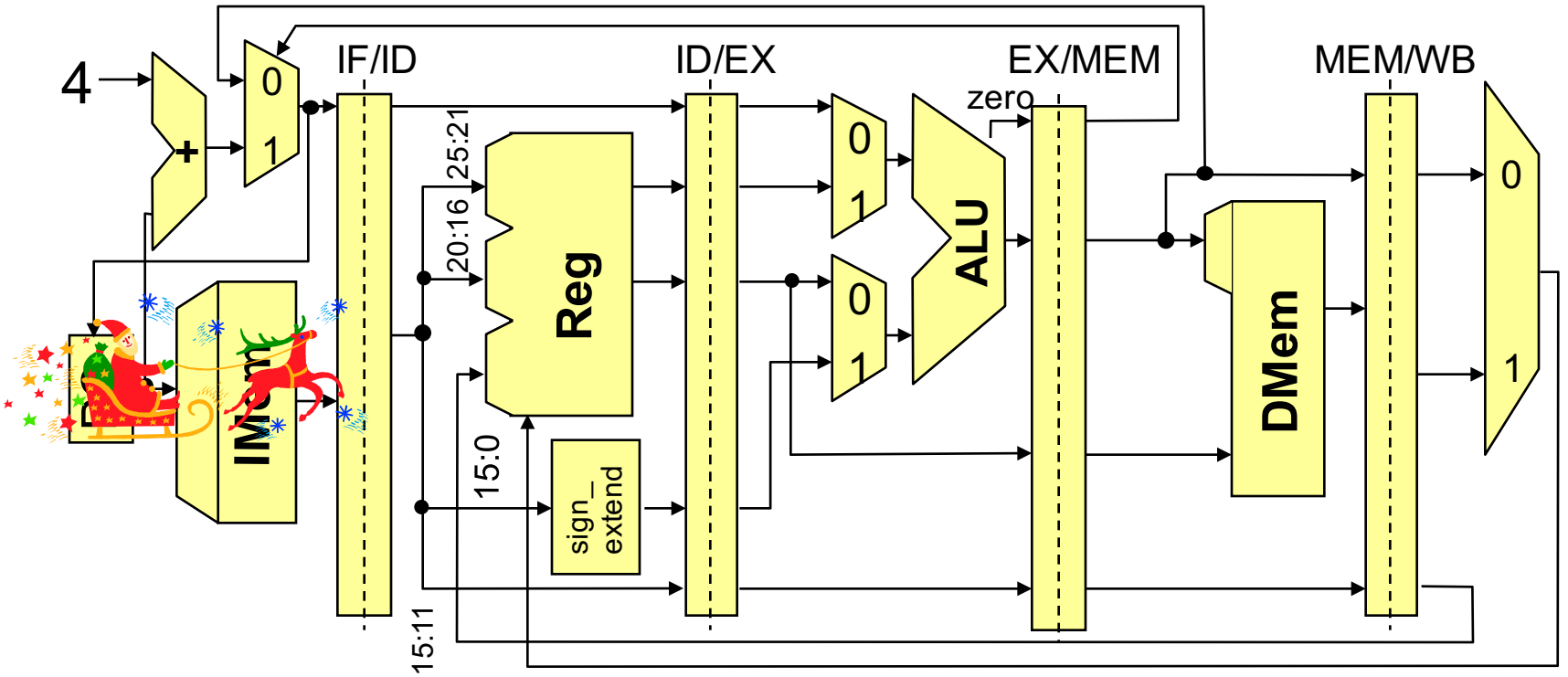


Steuer-werk nicht dar-gestellt

Aufgaben der einzelnen Phasen bzw. Stufen

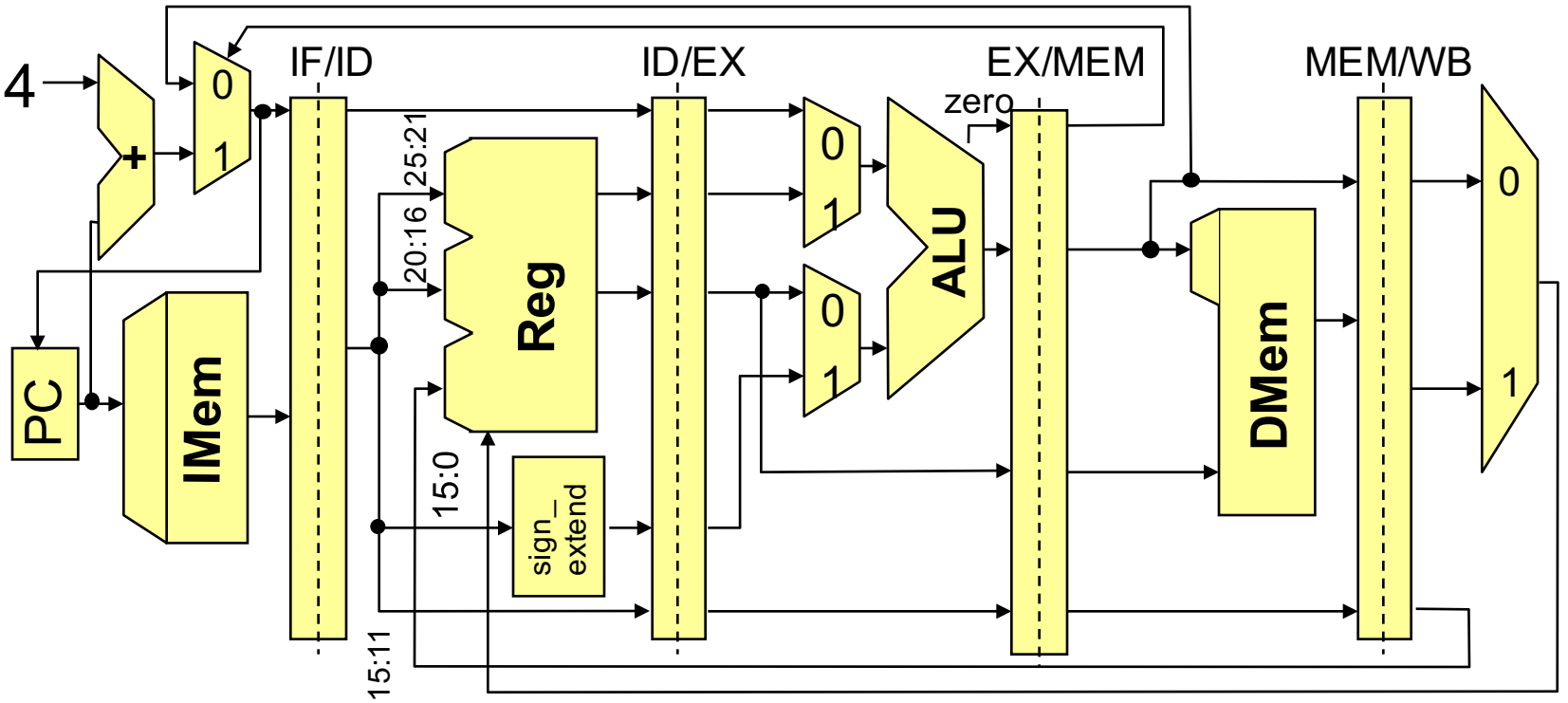
- **Befehlsholphase (IF)**
  Lesen des aktuellen Befehls; separater Speicher, zur Vermeidung von Konflikten mit Datenzugriffen (☞ Cache).
- **Dekodier- und Register-Lese-Phase (ID)**
  Lesen der Register möglich wegen fester Plätze für Nr.
- **Ausführungs- und Adressberechungsphase (EX)**
  Berechnung arithmetischer Funktion bzw. Adresse für Speicherzugriff.
- **Speicherzugriffsphase (MEM)**
  Wird nur bei Lade- und Speicherbefehlen benötigt.
- **Abspeicherungsphase (WB)**
  Speichern in Register, bei Speicherbefehlen nicht benötigt.

Idealer Fließbanddurchlauf



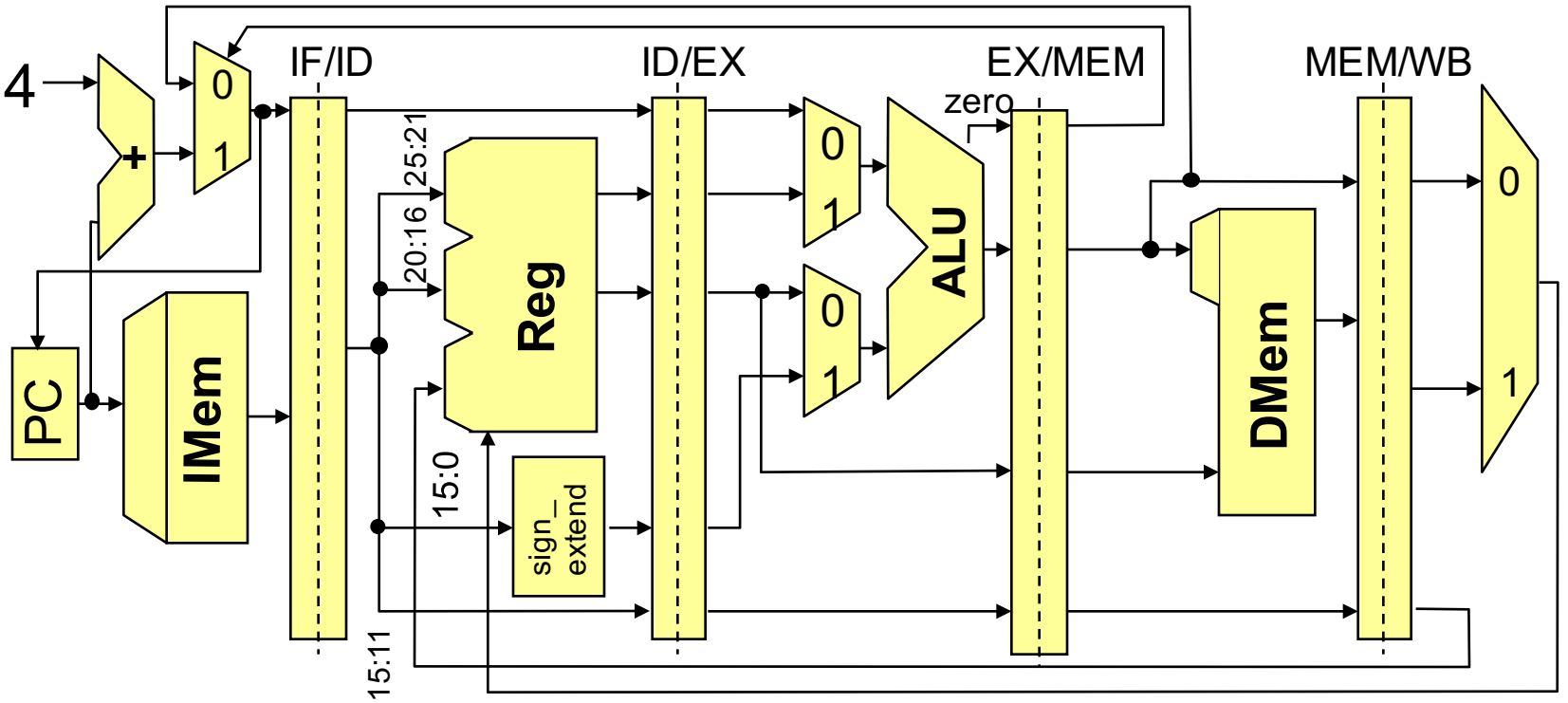
Zyklus 1				
Befehl 1				

Idealer Fließbanddurchlauf



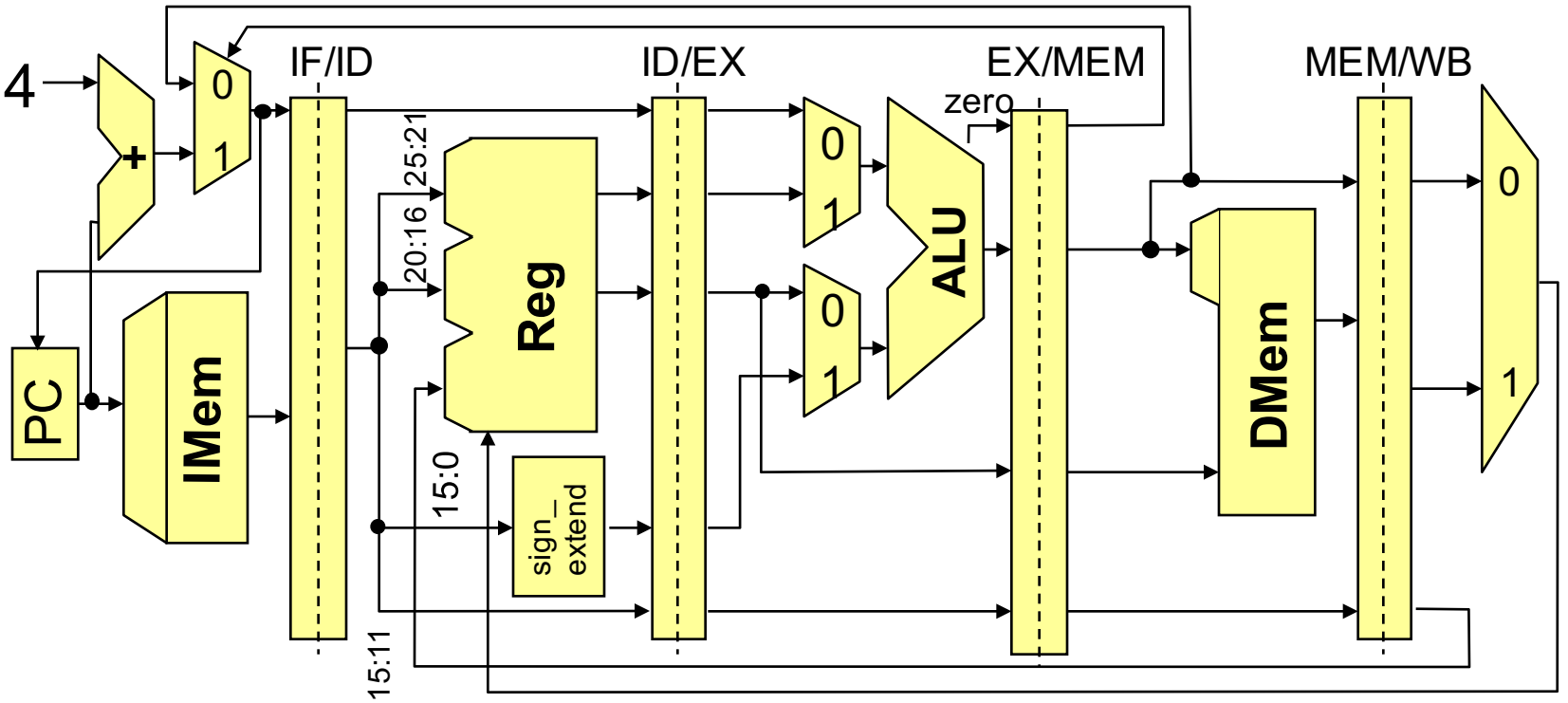
Zyklus 2				
Befehl 2	**Befehl 1**			

Idealer Fließbanddurchlauf



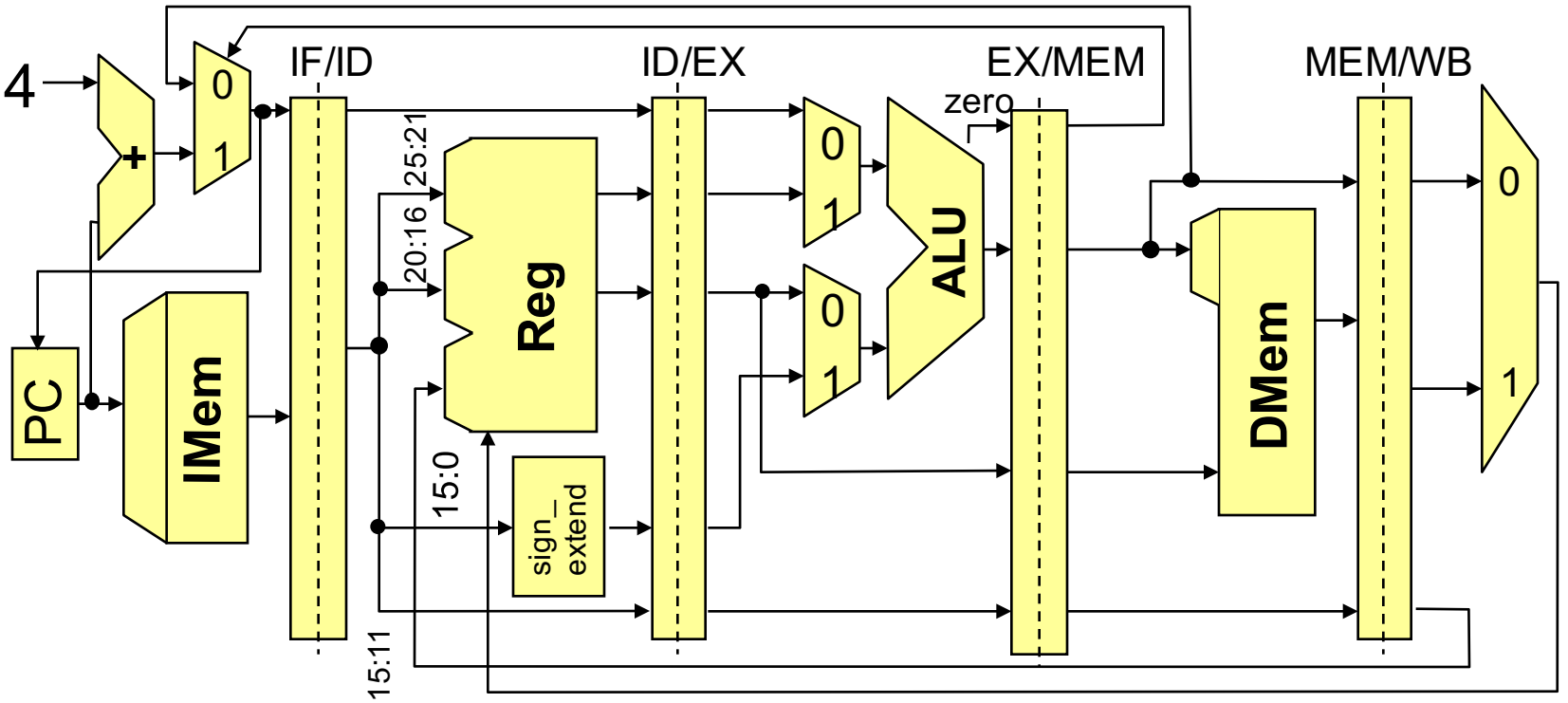
Zyklus 3				
Befehl 3	**Befehl 2**	**Befehl 1**		

Idealer Fließbanddurchlauf



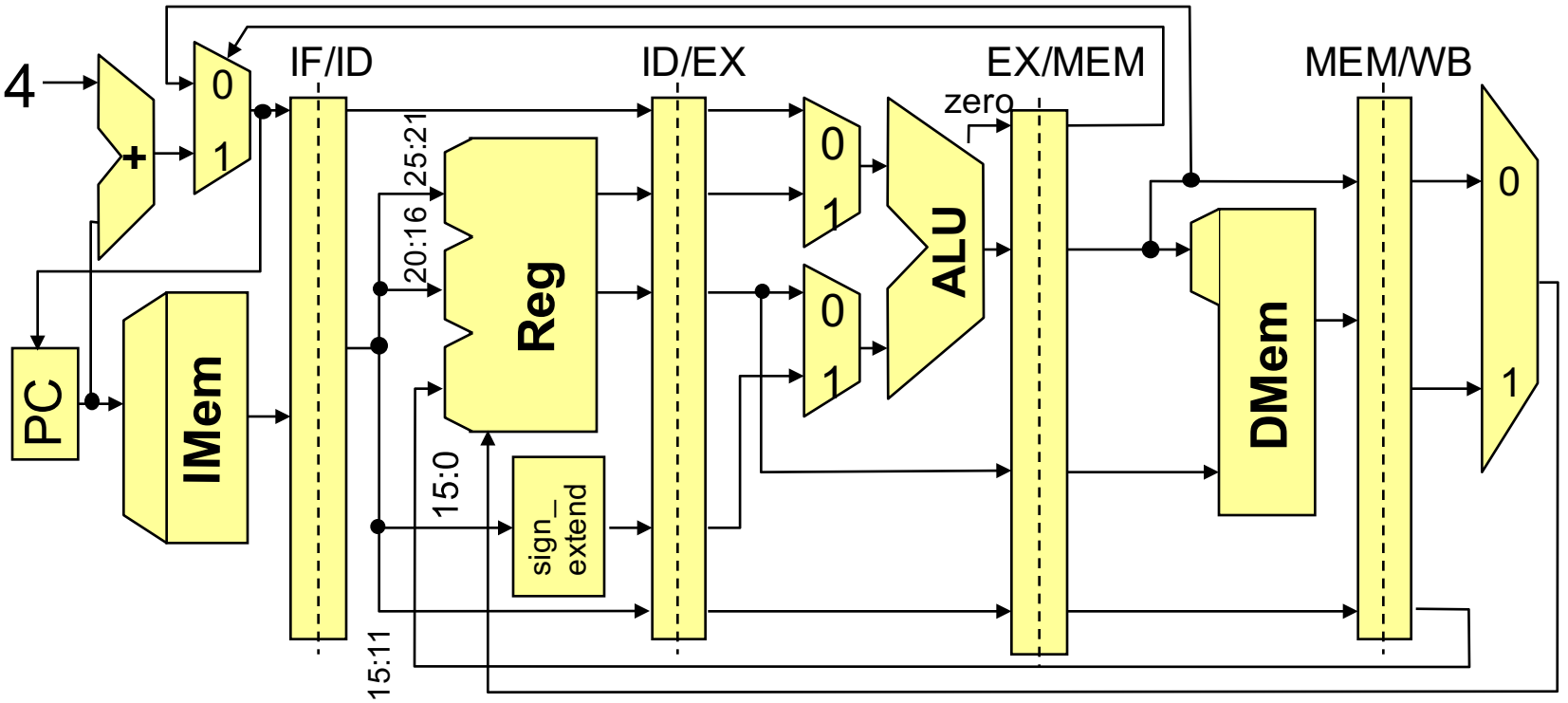
Zyklus 4				
Befehl 4	**Befehl 3**	**Befehl 2**	**Befehl 1**	

Idealer Fließbanddurchlauf



Zyklus 5				
Befehl 5	**Befehl 4**	**Befehl 3**	**Befehl 2**	**Befehl 1**

Idealer Fließbanddurchlauf



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

Pipeline-Hazards

Structural hazards
(deutsch: strukturelle Abhängigkeiten oder Gefährdungen).

Verschiedene Fließbandstufen müssen auf dieselbe Hardware-Komponente zugreifen, weil diese nur sehr aufwändig oder überhaupt nicht zu duplizieren ist.

Beispiele:

Speicherzugriffe, sofern für Daten und Befehle nicht über separate Pufferspeicher (*caches*) eine weitgehende Unabhängigkeit erreicht wird.

Bei Gleitkommaeinheiten lässt sich häufig nicht mit jedem Takt eine neue Operation starten (zu teuer).

☞ Eventuell Anhalten des Fließbandes (*pipeline stall)* nötig.

Datenabhängigkeiten (1)

Gegeben sei eine Folge von Maschinen-Befehlen.

Def.: Ein Befehl j heißt von einem vorausgehenden Befehl i **datenabhängig**, wenn i Daten bereitstellt, die j benötigt.

Beispiel:

```
add $12,$2,$3
sub $4,$5,$12
and $6,$12,$7
or  $8,$12,$9
xor $10,$12,$11
```

Diese 4 Befehle sind vom **add**-Befehl wegen **$12** datenabhängig

Diese Art der Abhängigkeit heißt (bei Hennessy und anderen) *read after write*- (oder RAW-) Abhängigkeit.

Datenabhängigkeiten (2)

Gegeben sei wieder eine Folge von Maschinen-Befehlen.

Def.: Ein Befehl i heißt von einem **nachfolgenden** Befehl j **antidatenabhängig**, falls j eine Speicherzelle beschreibt, die von i noch gelesen werden müsste.

Beispiel:

```
add $12,$2,$3
sub $4,$5,$12
and $6,$12,$7
or  $12,$12,$9
xor $10,$12,$11
```

Diese 2 Befehle sind vom **or**-Befehl wegen **$12** antidatenabhängig

Diese Art der Abhängigkeit heißt (bei Hennessy und anderen) *write after read* - (oder WAR-) Abhängigkeit.

Datenabhängigkeiten (3)

Gegeben sei (wieder) eine Folge von Maschinen-Befehlen.

Def.: Zwei Befehle i und j heißen voneinander **Ausgabe-abhängig**, falls i und j dieselbe Speicherzelle beschreiben.

Beispiel:

```
add $12,$2,$3
sub $4,$5,$12
and $6,$12,$7
or  $12,$12,$9
xor $10,$12,$11
```

Voneinander ausgabeabhängig.

Diese Art der Abhängigkeit heißt (bei Hennessy und anderen) *write after write* - (oder WAW-) Abhängigkeit.

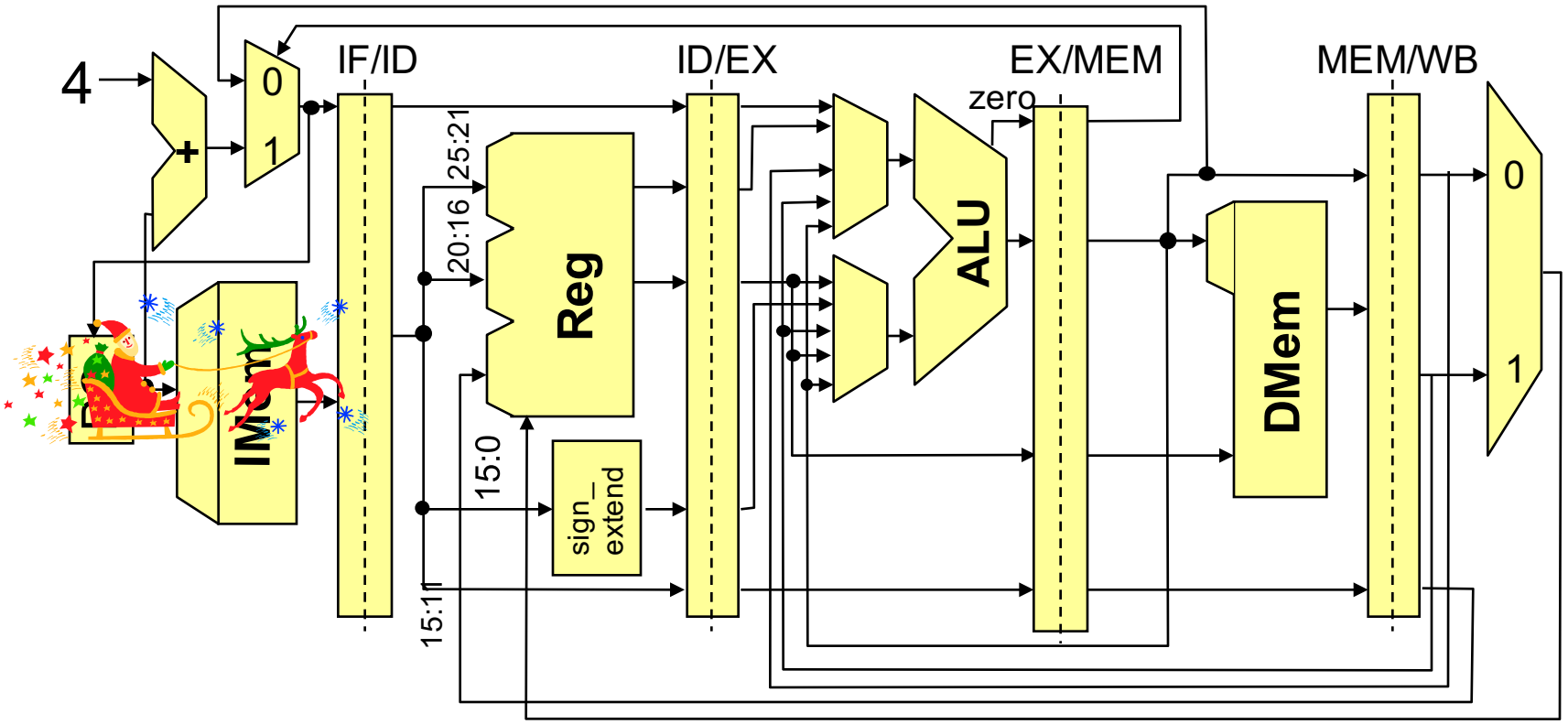
Quiz

Data Hazard

- Is WAR an issue in the previous pipeline structure?
- Is WAW an issue in the previous pipeline structure?

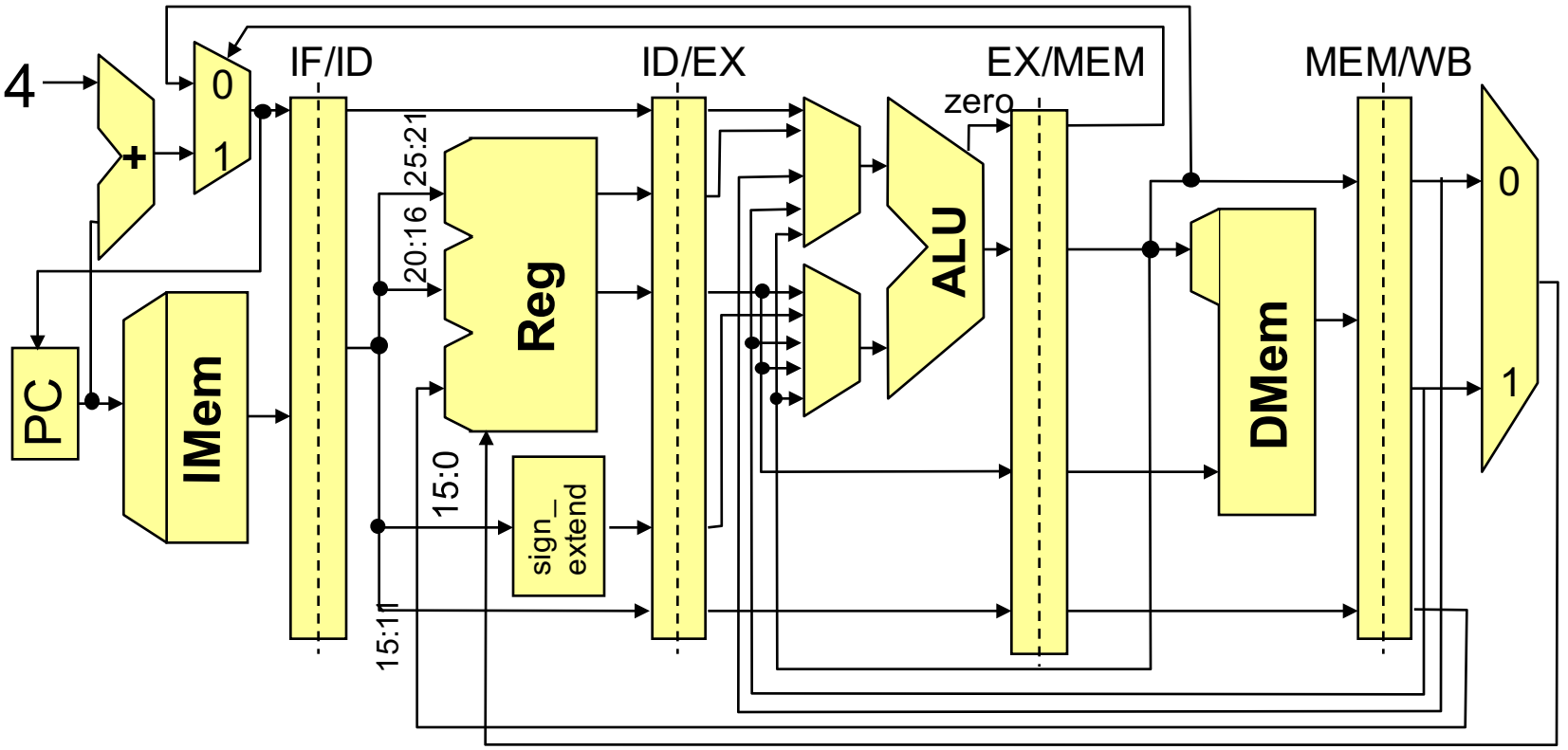
Why do we need IMem and DMem separately?

Bypässe, *forwarding*: Behandlung des *data hazards* bei `and` und `sub`



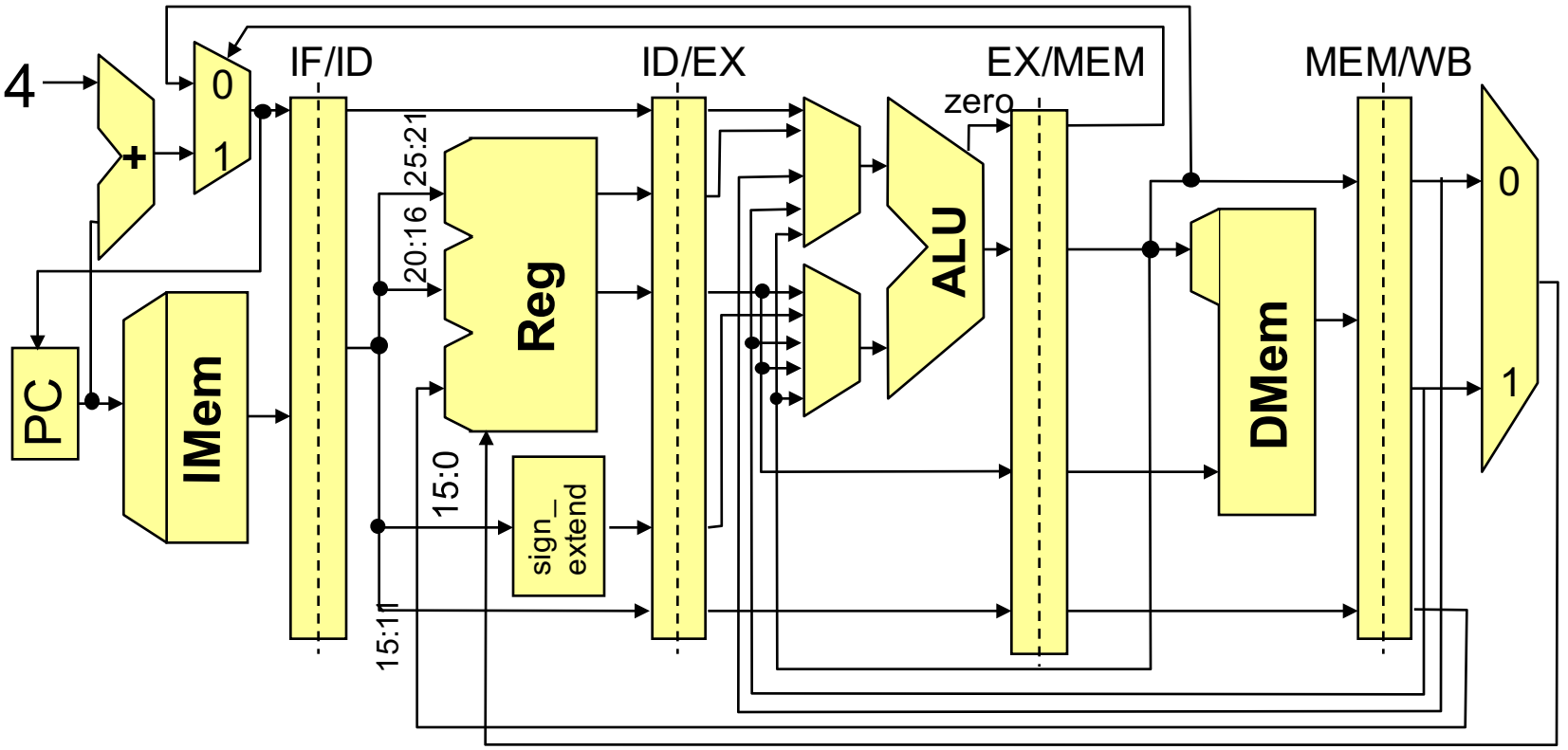
Zyklus 1				
add $1,$2,$3				

Bypässe, *forwarding*: Behandlung des *data hazards* bei `and` und `sub`



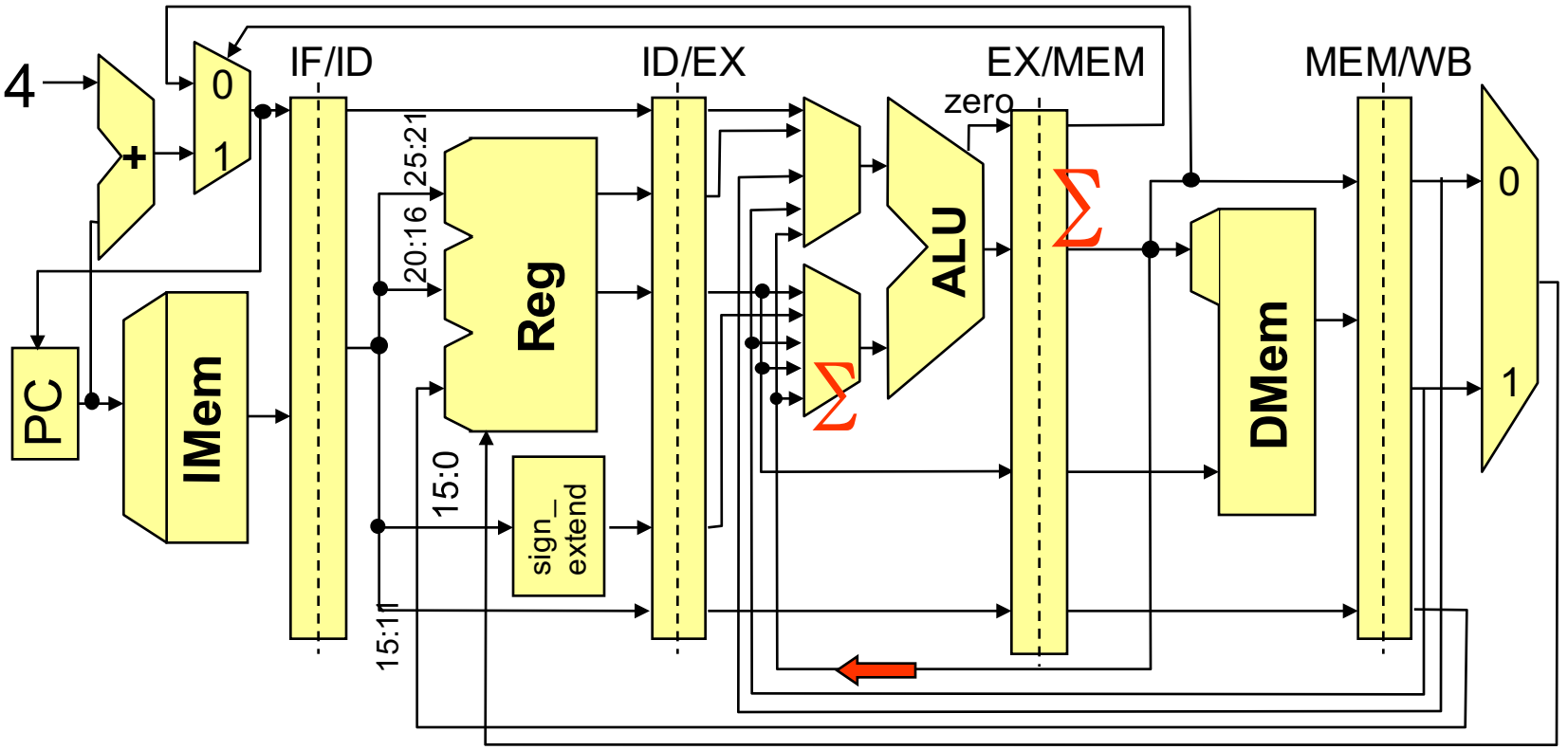
Zyklus 2				
sub $4,$5,$1	**add $1,$2,$3**			

Bypässe, *forwarding*: Behandlung des *data hazards* bei `and` und `sub`



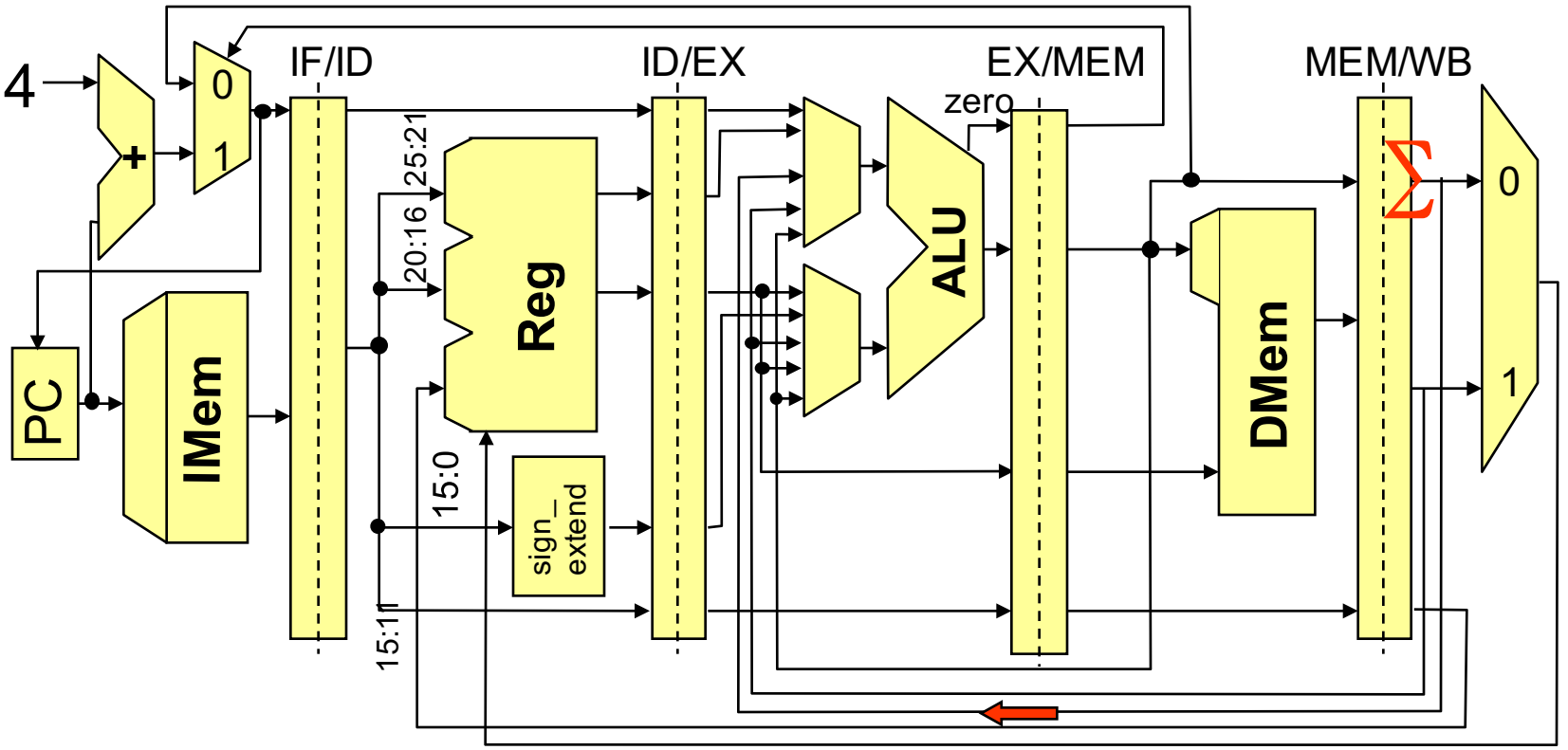
Zyklus 3				
and $6,$1,$7	**sub $4,$5,$1**	**add $1,$2,$3**		

Bypässe, *forwarding*: Behandlung des *data hazards* bei `and` und `sub`



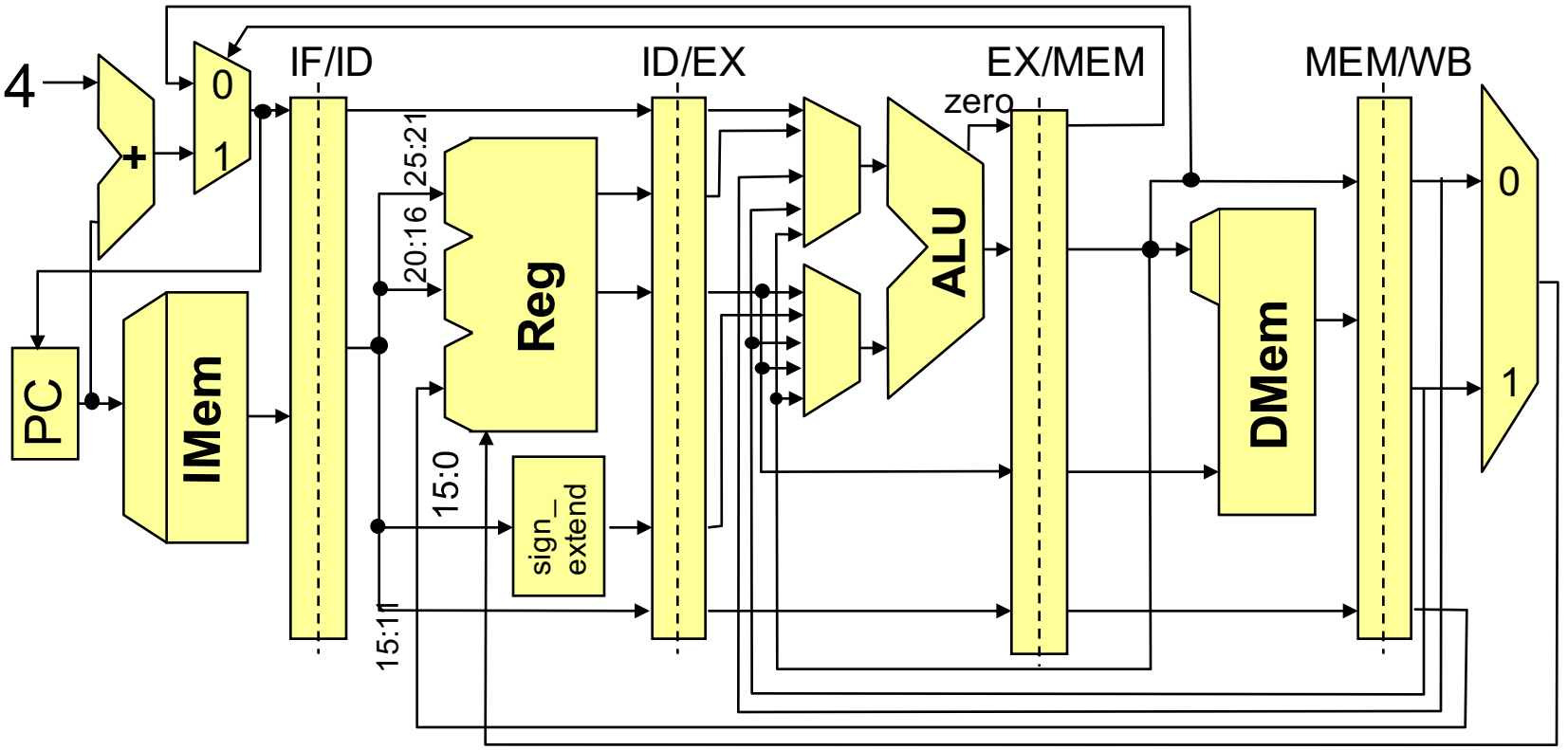
Zyklus 4				
or $8,$1,$9	and $6,$1,$7	sub $4,$5,$1	add $1,$2,$3	

Bypässe, *forwarding*: Behandlung des *data hazards* bei `and` und `sub`



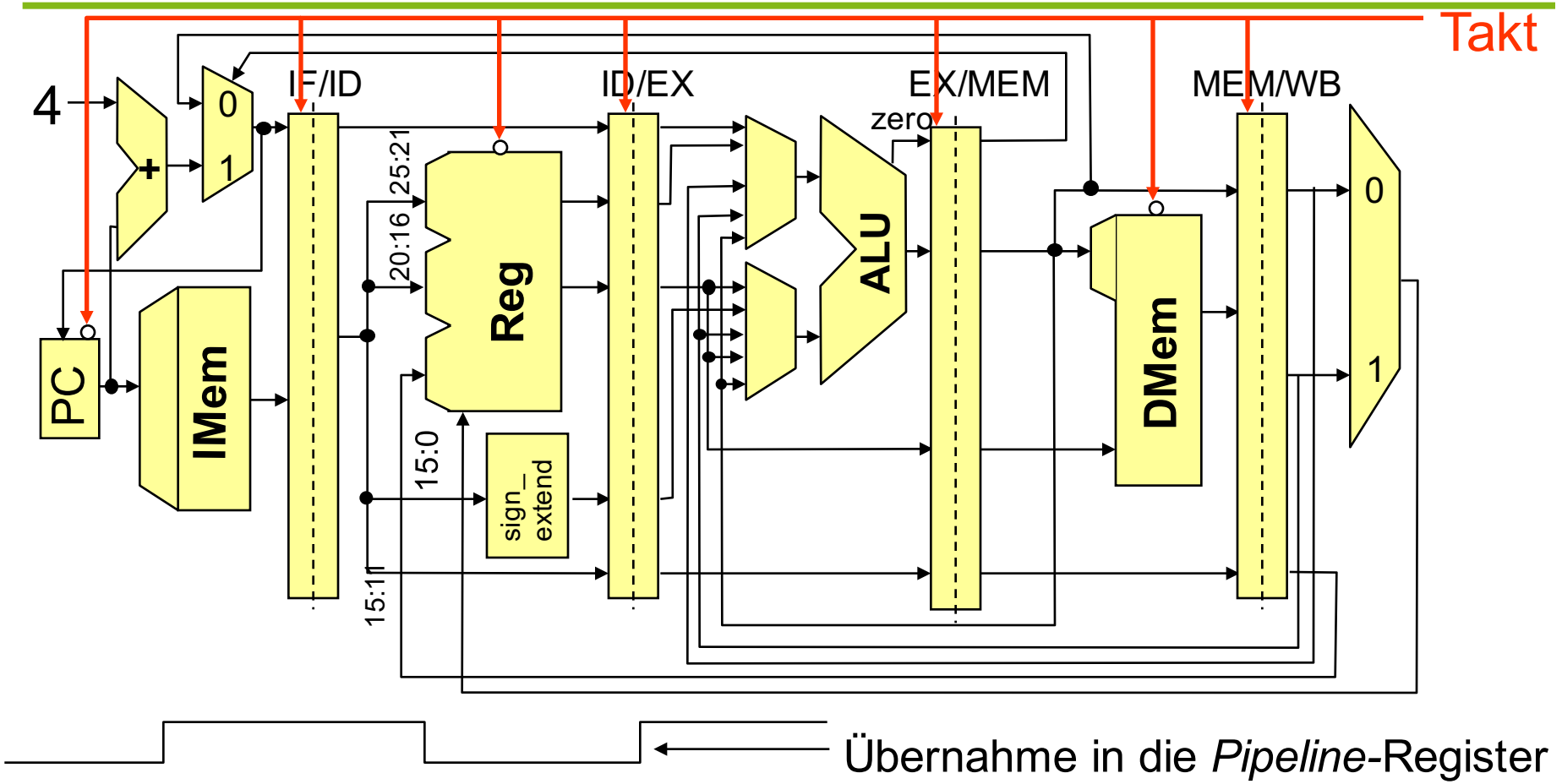
Zyklus 5				
xor $10,$1,$11	or $8,$1,$9	and $6,$1,$7	sub $4,$5,$1	add $1,$2,$3

Bypässe, *forwarding*: Behandlung des *data hazards* bei `and` und `sub`



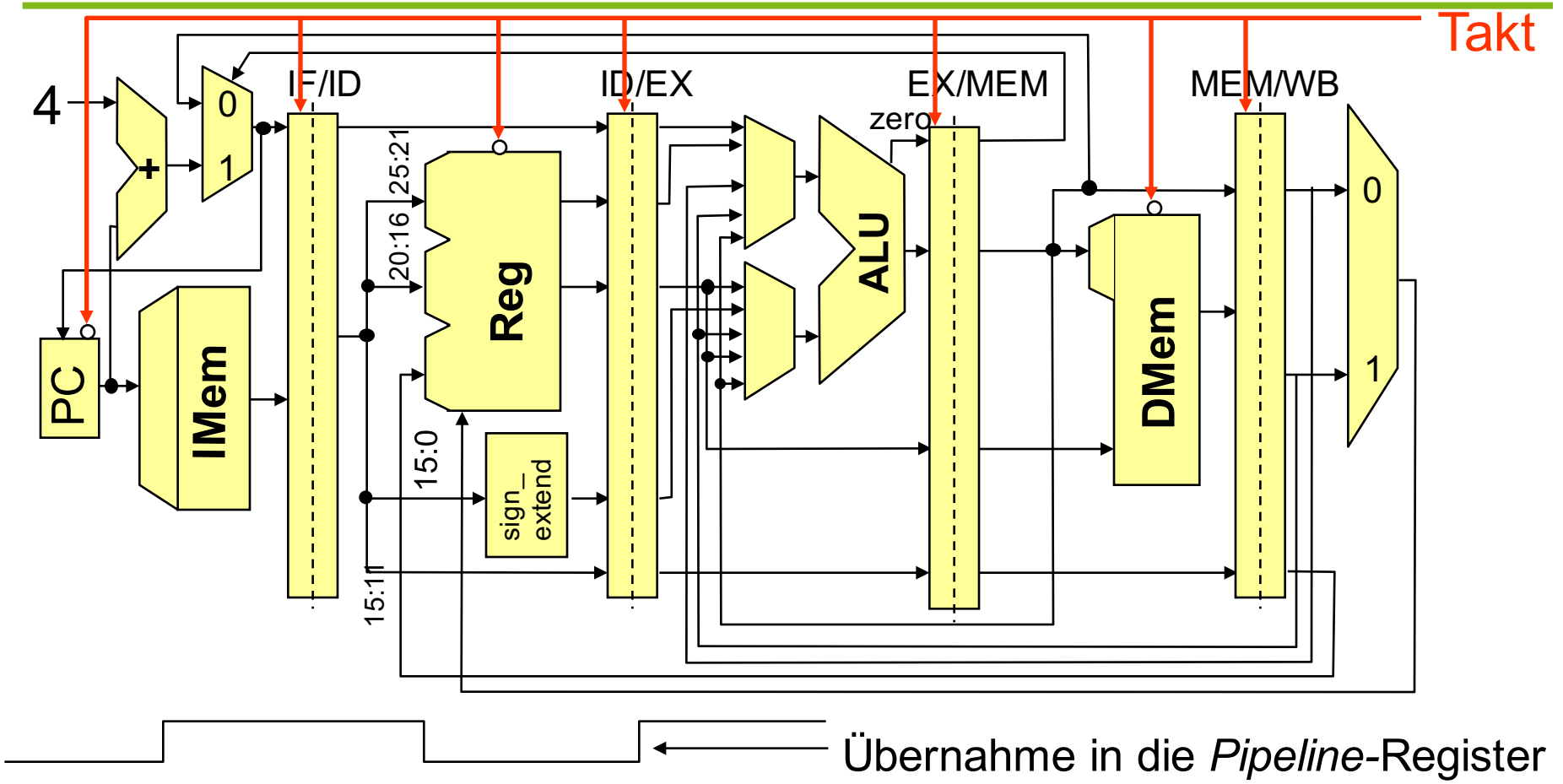
Zyklus 6				
?	xor $10,$1,$11	or $8,$1,$9	and $6,$1,$7	sub $4,$5,$1

Taktung zur Behandlung des *data hazards* bei `or`



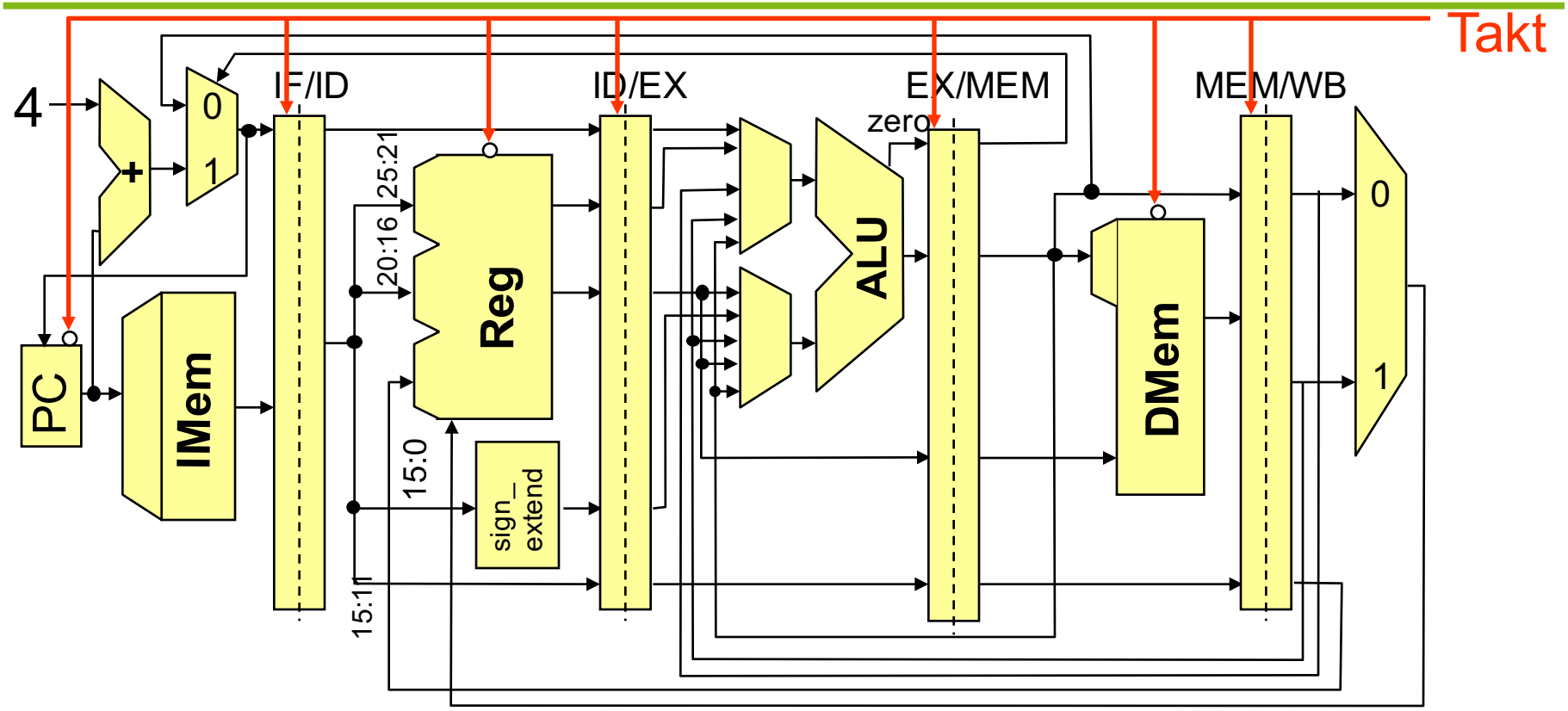
Zyklus 1				
add $1,$2,$3				

Taktung zur Behandlung des *data hazards* bei `or`



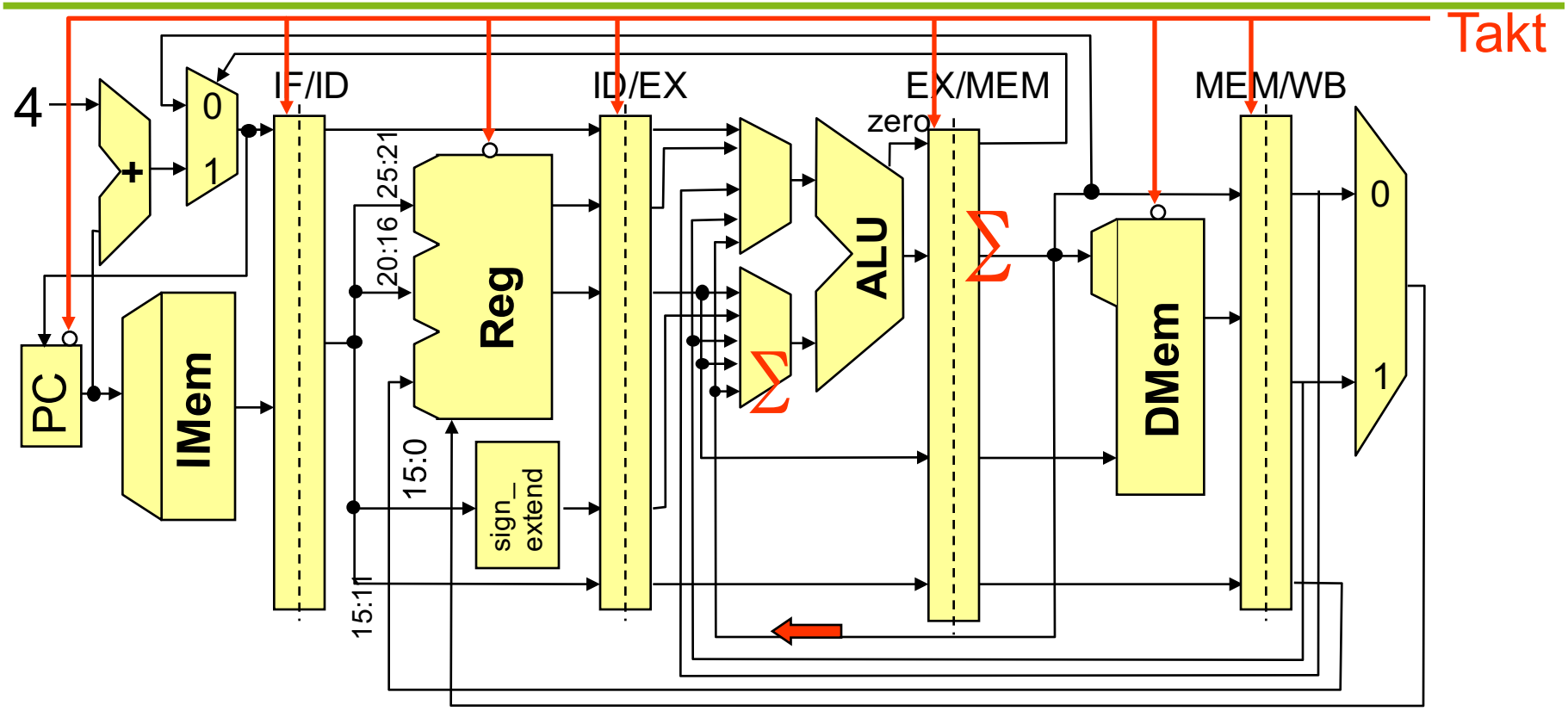
Zyklus 2				
sub $4,$5,$1	**add $1,$2,$3**			

Taktung zur Behandlung des *data hazards* bei `or`



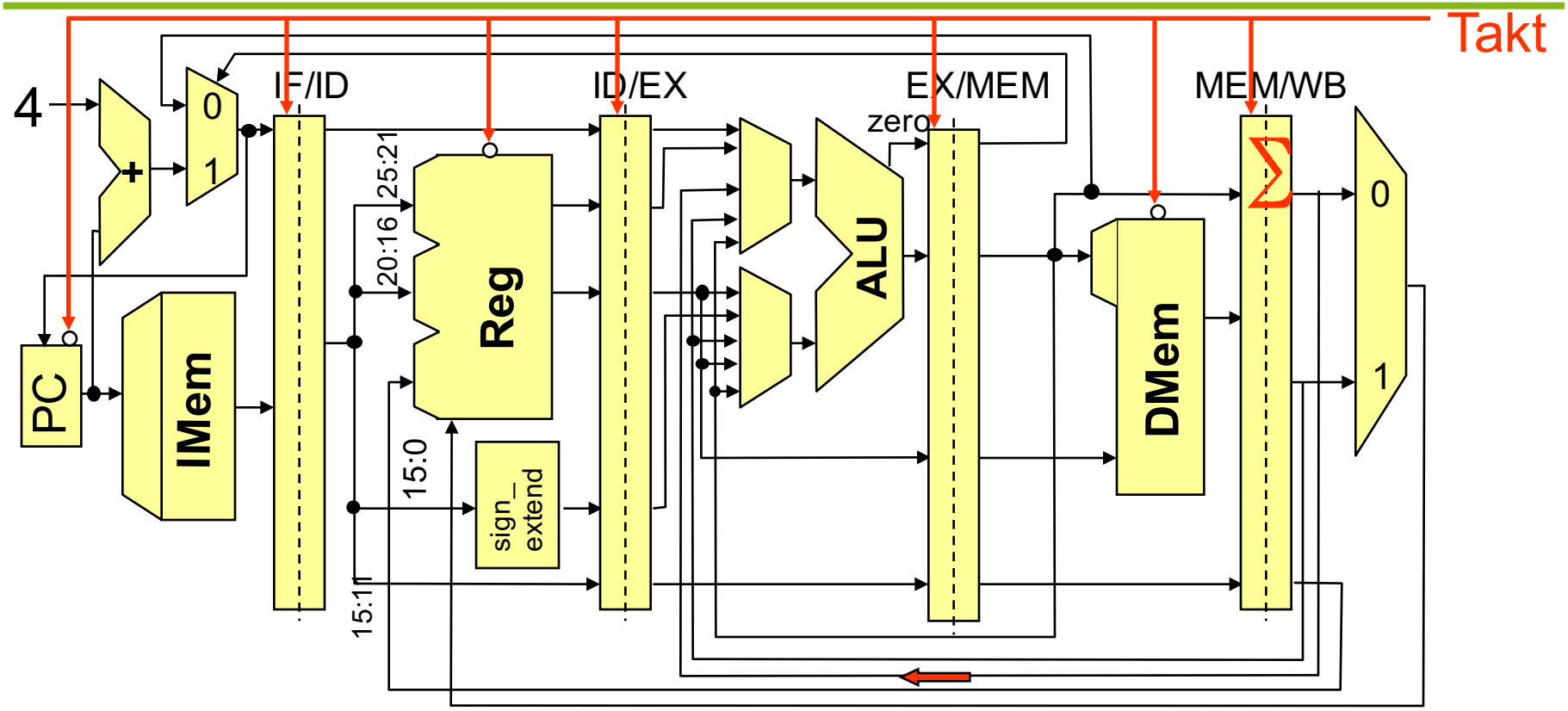
Zyklus 3				
and $6,$1,$7	**sub $4,$5,$1**	**add $1,$2,$3**		

Taktung zur Behandlung des *data hazards* bei `or`



Zyklus 4				
or $8,$1,$9	and $6,$1,$7	sub $4,$5,$1	add $1,$2,$3	

Taktung zur Behandlung des *data hazards* bei `or`

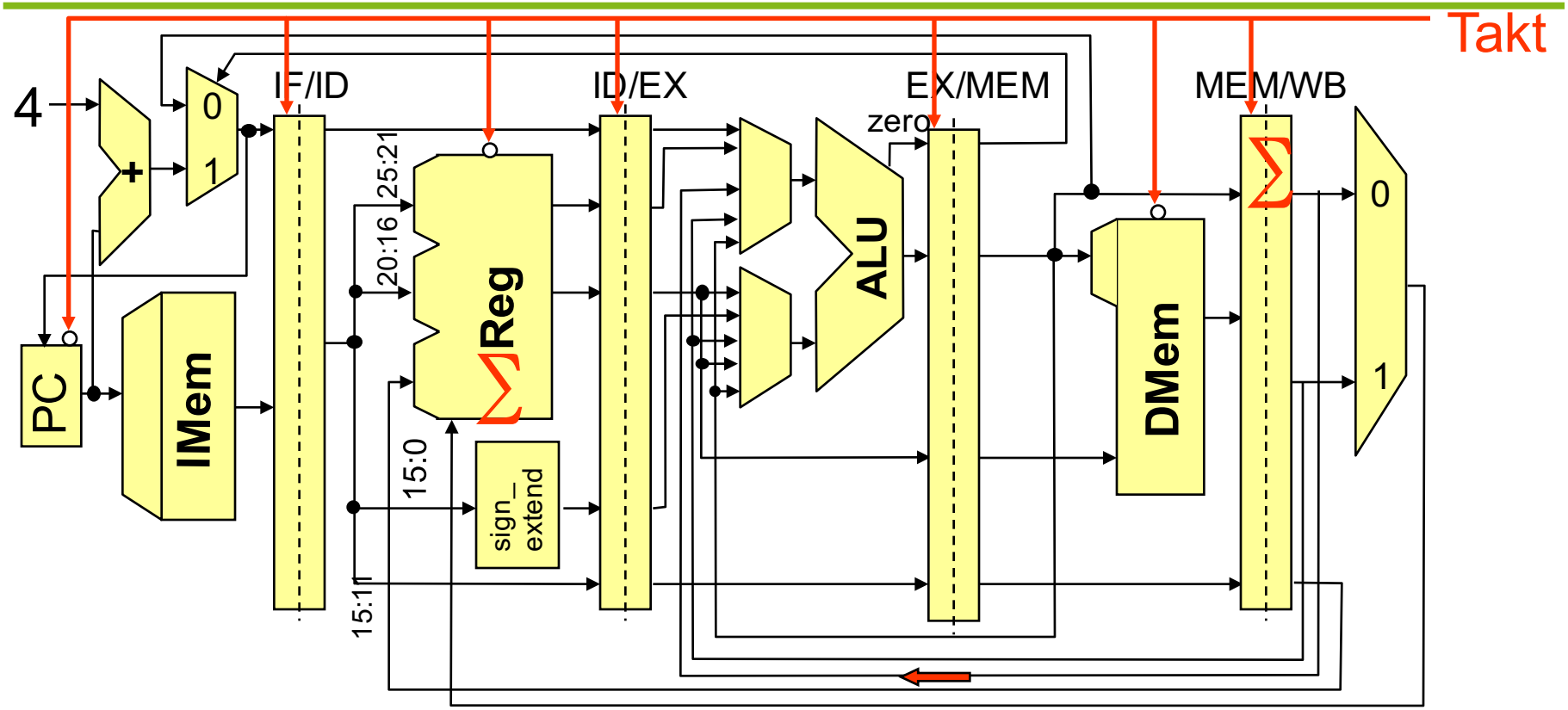


Zyklus 4

Übernahme in die *Pipeline*-Register

Zyklus 5				
xor $10,$1,$11	**or $8,$1,$9**	**and $6,$1,$7**	**sub $4,$5,$1**	**add $1,$2,$3**

Taktung zur Behandlung des *data hazards* bei `or`

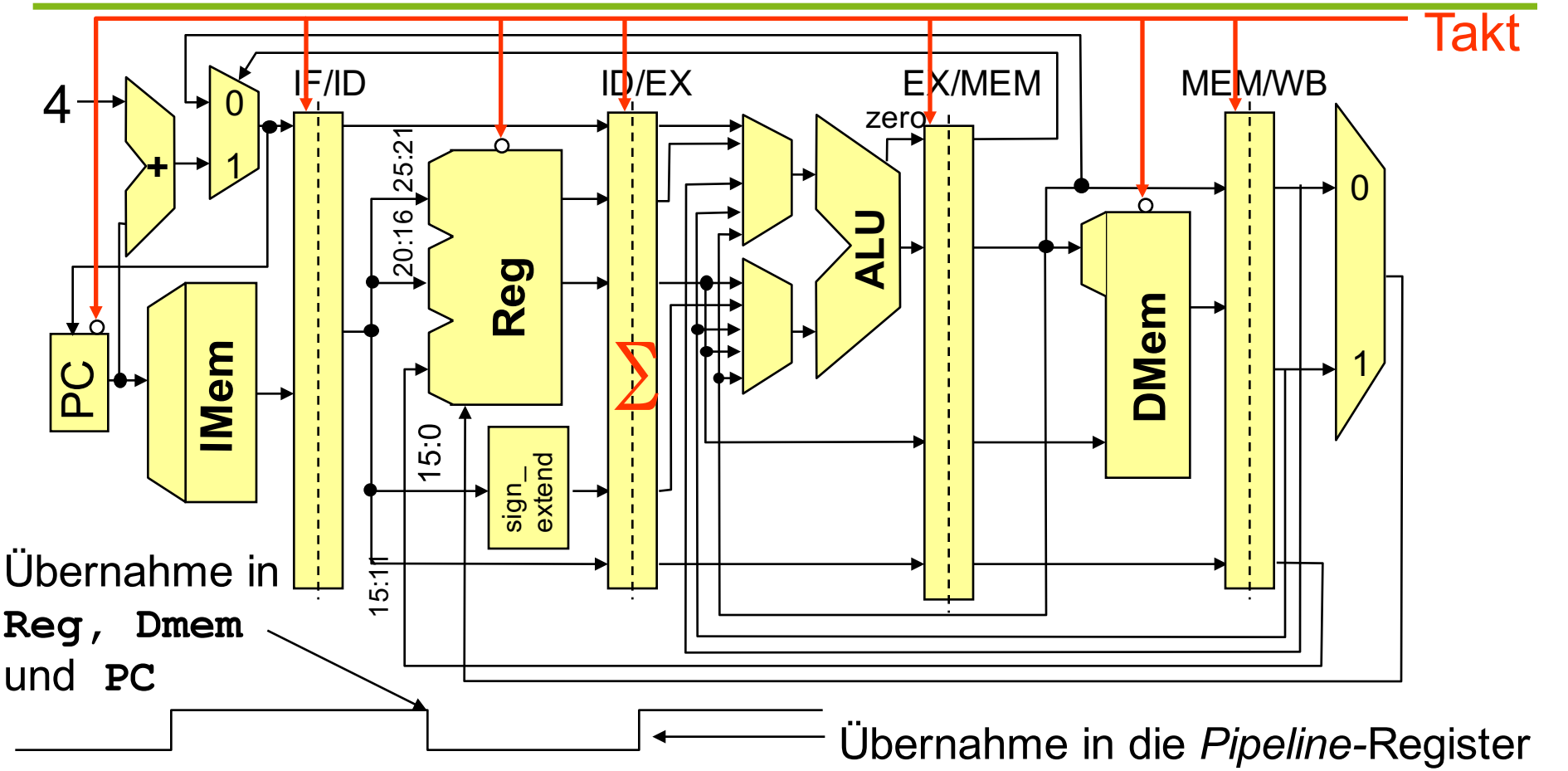


Zyklus 4

Übernahme in die *Pipeline*-Register

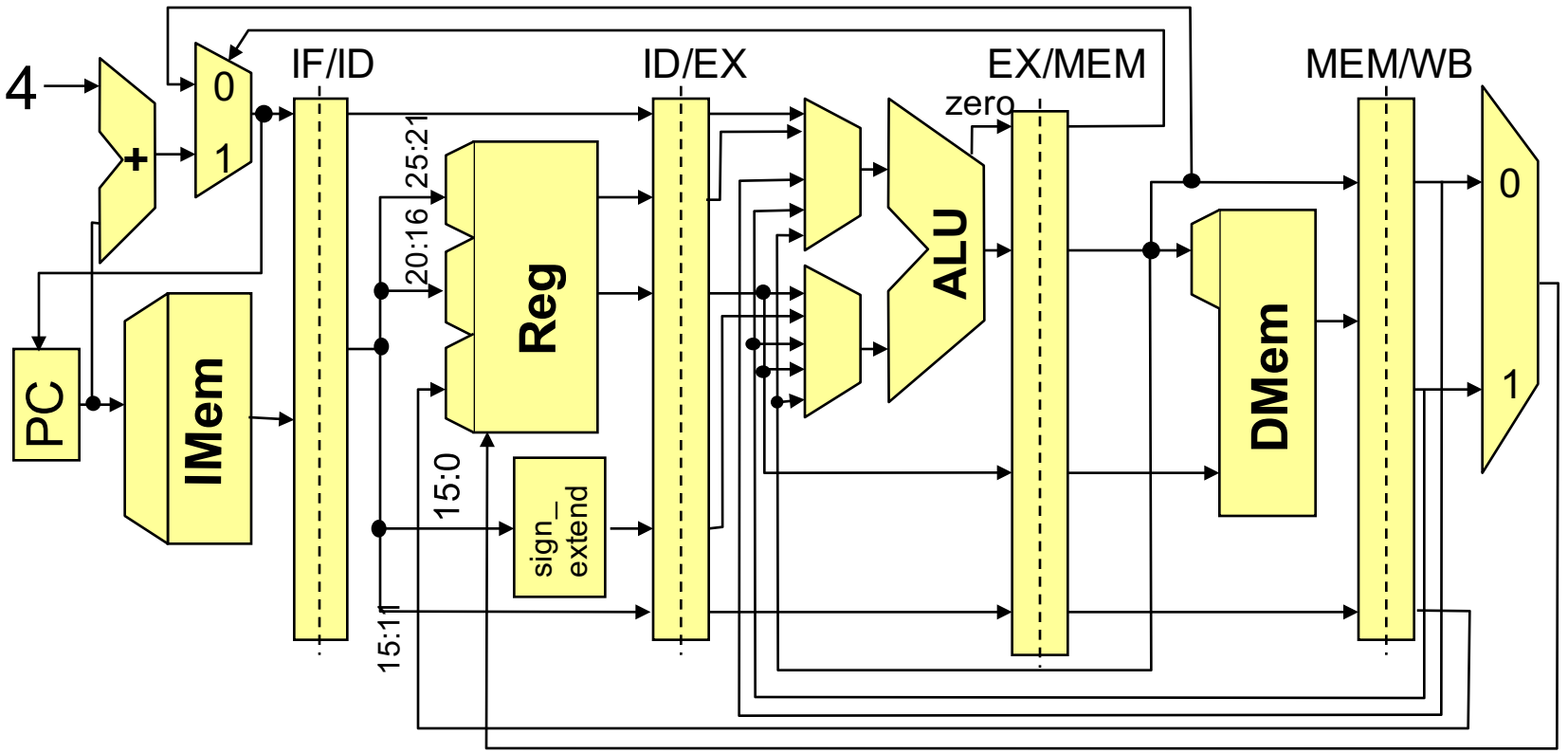
Zyklus 5				
xor $10,$1,$11	**or $8,$1,$9**	**and $6,$1,$7**	**sub $4,$5,$1**	**add $1,$2,$3**

Taktung zur Behandlung des *data hazards* bei `or`



Zyklus 6				
?	xor $10,$1,$11	or $8,$1,$9	and $6,$1,$7	sub $4,$5,$1

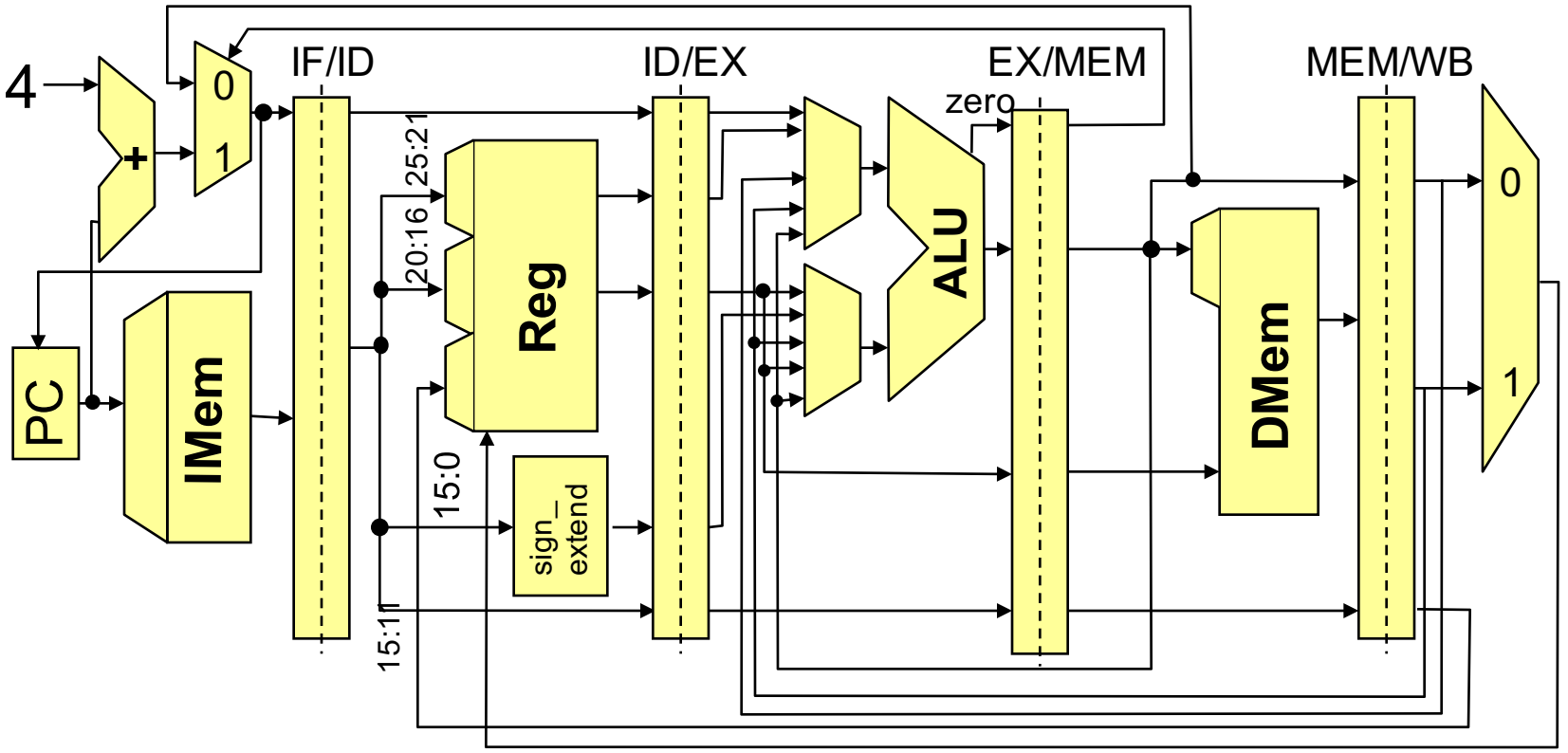
Alle *data hazards* durch Bypässe behandelbar?



Zyklus 6				
?	xor $10,$1,$11	or $8,$1,$9	and $6,$1,$7	sub $4,$5,$1

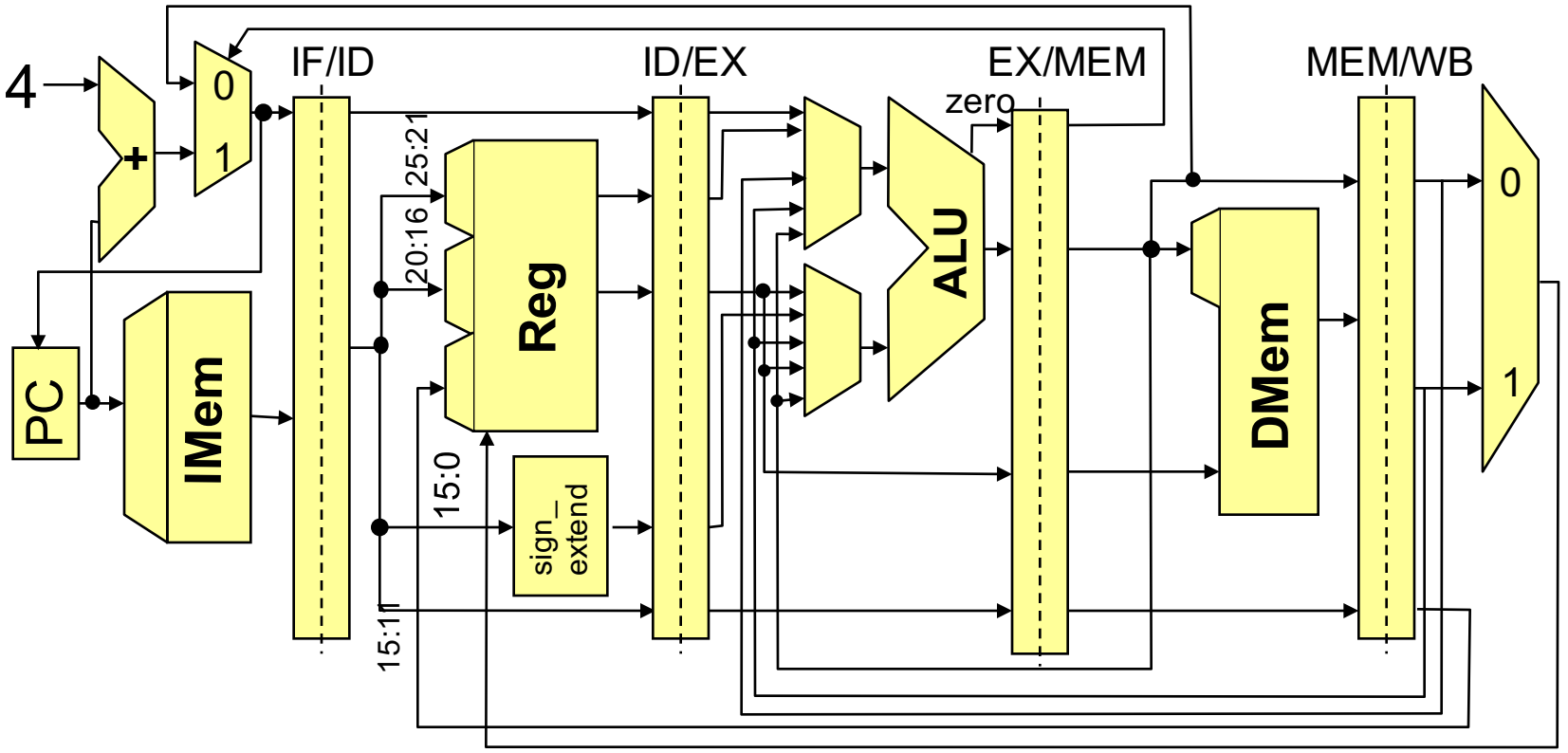
Speicherwort wird am Ende des Zyklus 4 gespeichert, steht für Differenz noch nicht bereit.

Lösung durch Anhalten des Fließbandes (*pipeline stall, hardware interlocking, bubbles*)



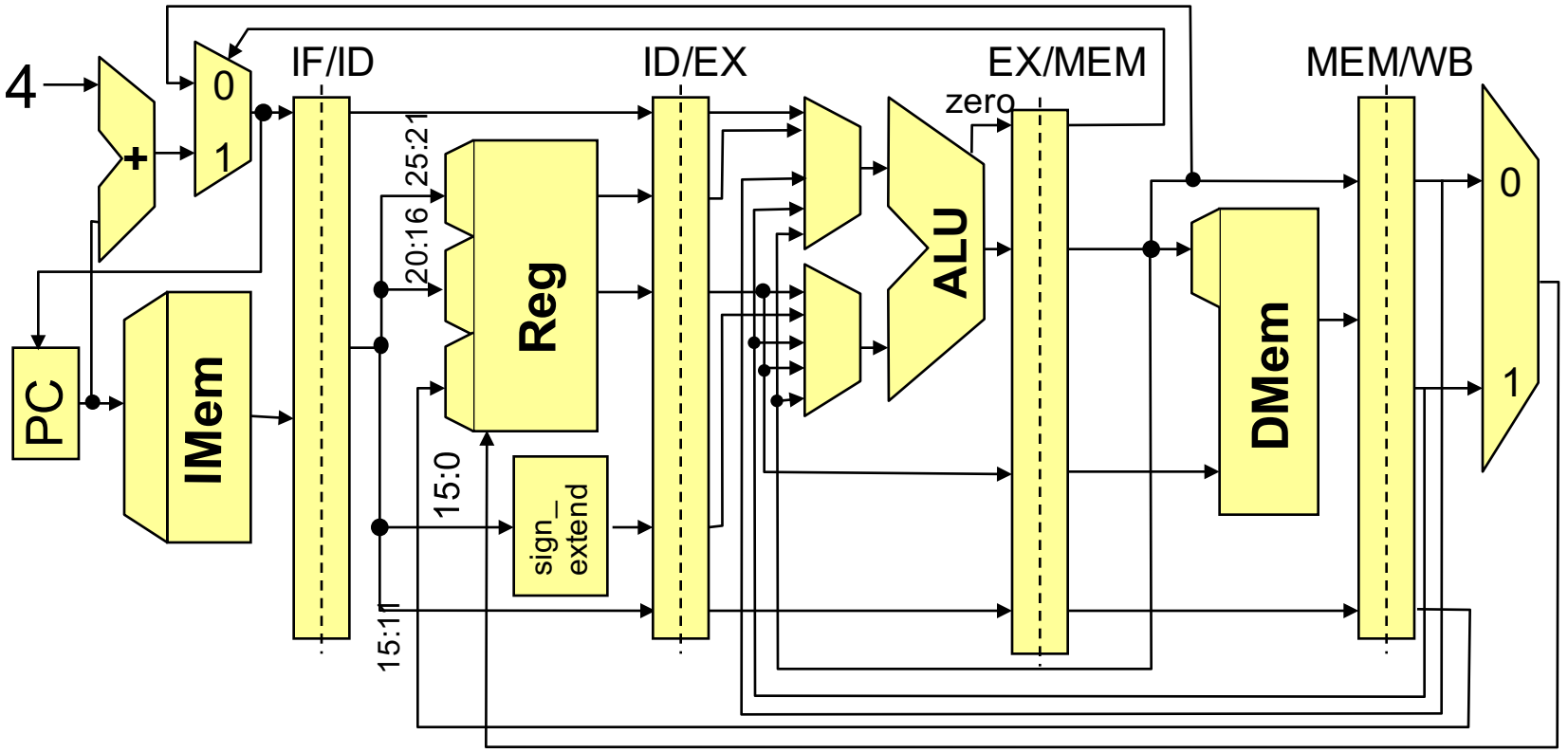
Zyklus 1				
lw $1,0($2)				

Lösung durch Anhalten des Fließbandes (*pipeline stall, hardware interlocking, bubbles*)



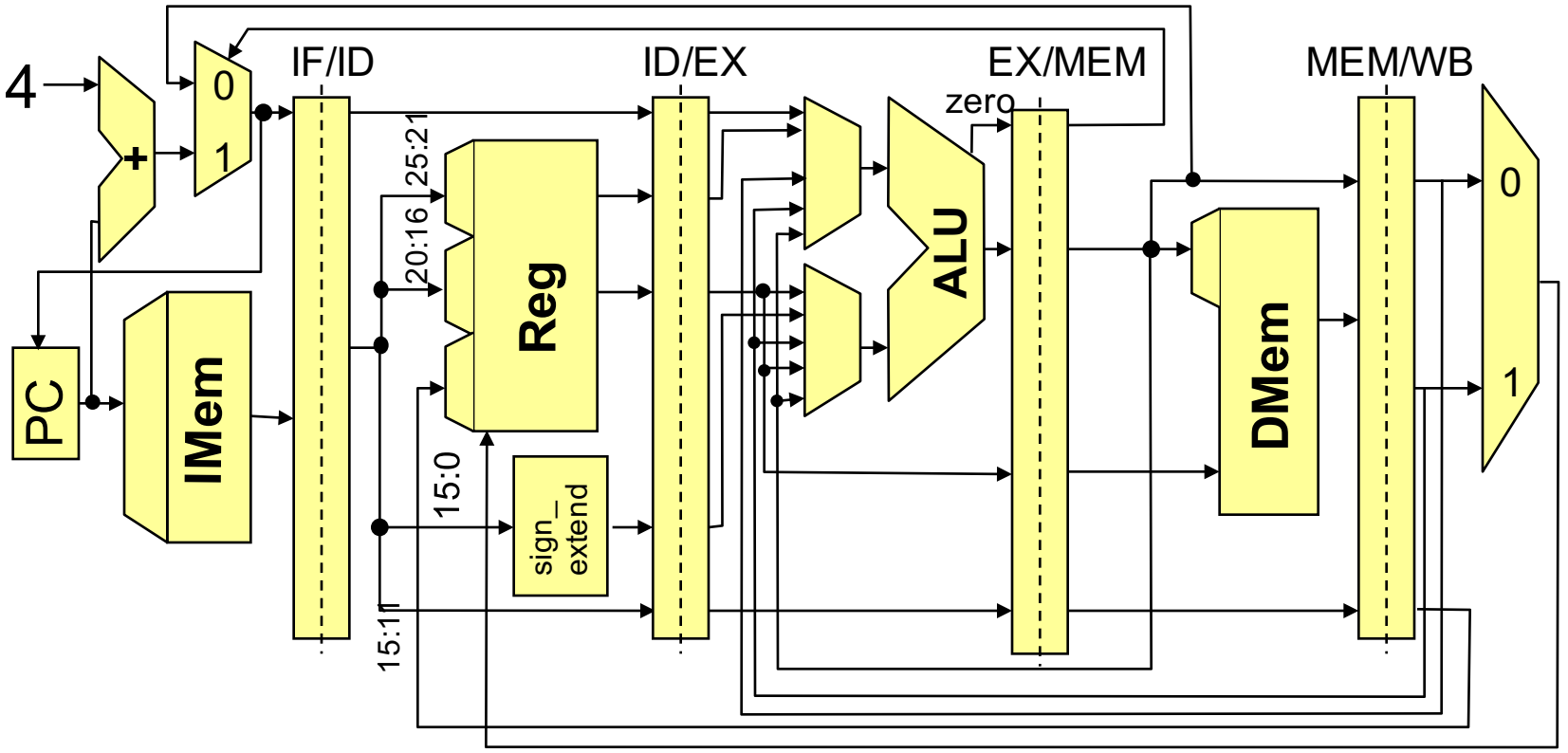
Zyklus 2				
sub $4,$5,$1	**lw $1,0($2)**			

Lösung durch Anhalten des Fließbandes (*pipeline stall, hardware interlocking, bubbles*)



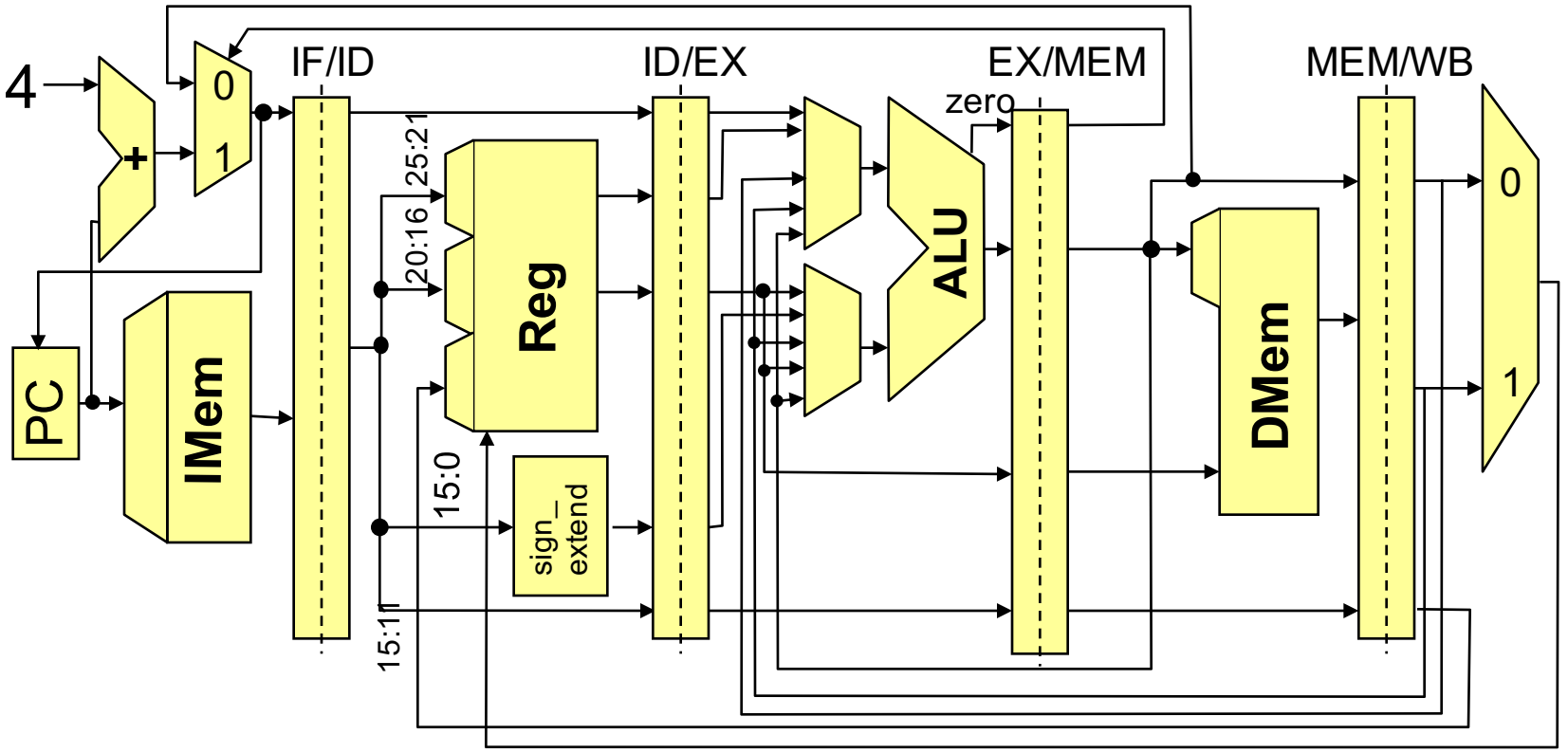
Zyklus 3				
and $6,$1,$7	**sub $4,$5,$1**	**lw $1,0($2)**		

Lösung durch Anhalten des Fließbandes (*pipeline stall, hardware interlocking, bubbles*)



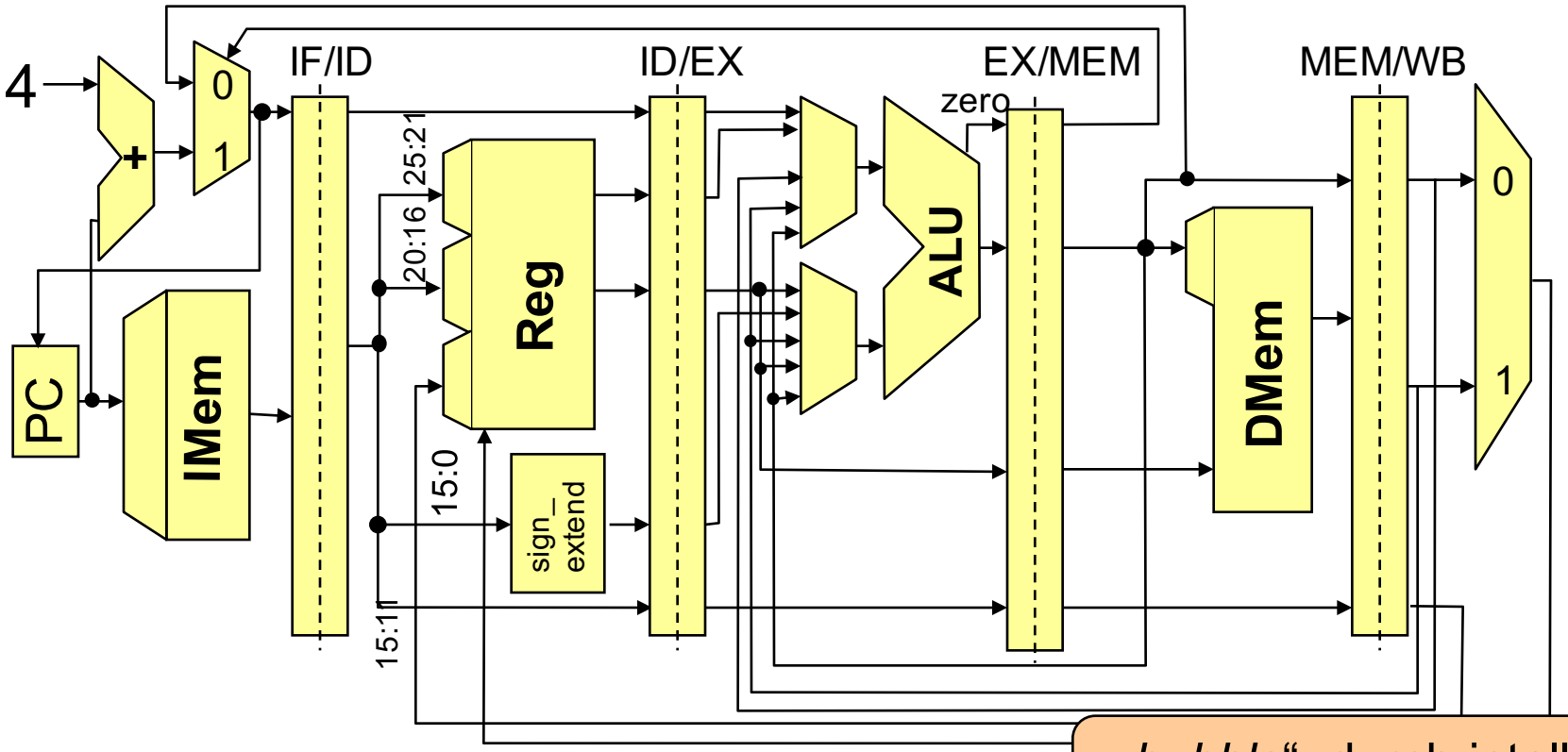
Zyklus 4				
NOOP	NOOP	NOOP	lw $1,0($2)	

Lösung durch Anhalten des Fließbandes (*pipeline stall, hardware interlocking, bubbles*)



Zyklus 5				
or $8,$1,$9	and $6,$1,$7	sub $4,$5,$1	NOOP	lw $1,0($2)

Lösung durch Anhalten des Fließbandes (*pipeline stall, hardware interlocking, bubbles*)



„*bubble*“, durch intelligente Compiler vermeiden!

Zyklus 6				
xor $10,$1,$11	or $8,$1,$9	and $6,$1,$7	sub $4,$5,$1	NOOP

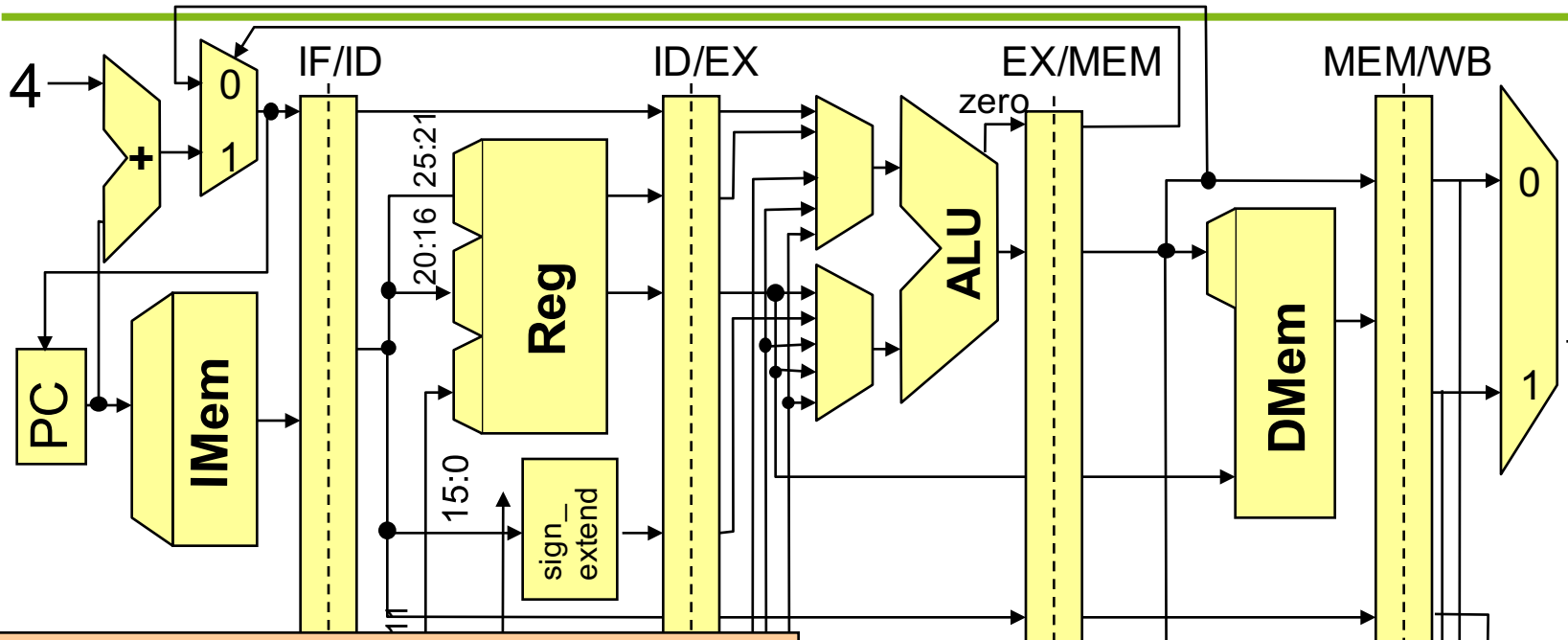
Kontrollfluss-Abhängigkeiten oder - Gefährdungen, *control hazards* (1)

Beispielprogramm:

```
   beq $12,$2,t   -- springe zur Marke t, falls Reg[12]=Reg[2]
   sub  ...
   ....
t:add ..
```

Wir versuchen zunächst, durch Einfügen von NOOPs, die intuitive Bedeutung des Programms zu realisieren...

Kontrollfluss-Abhängigkeiten oder - Gefährdungen, *control hazards* (2)



Takt 2: Sprung wird erkannt, deshalb werden zwei NOOP-Befehle eingefügt.

Takt 4: Mit fallender Flanke wird PC getaktet, mit steigender IF/ID-Register.

Takt 3: ALU müsste sowohl Vergleich wie auch das Sprungziel ausrechnen können

Zyklus 6				
...	...	**sub oder add**	**NOOP**	**NOOP**

Kontrollfluss-Abhängigkeiten oder - Gefährdungen, *control hazards* (3)

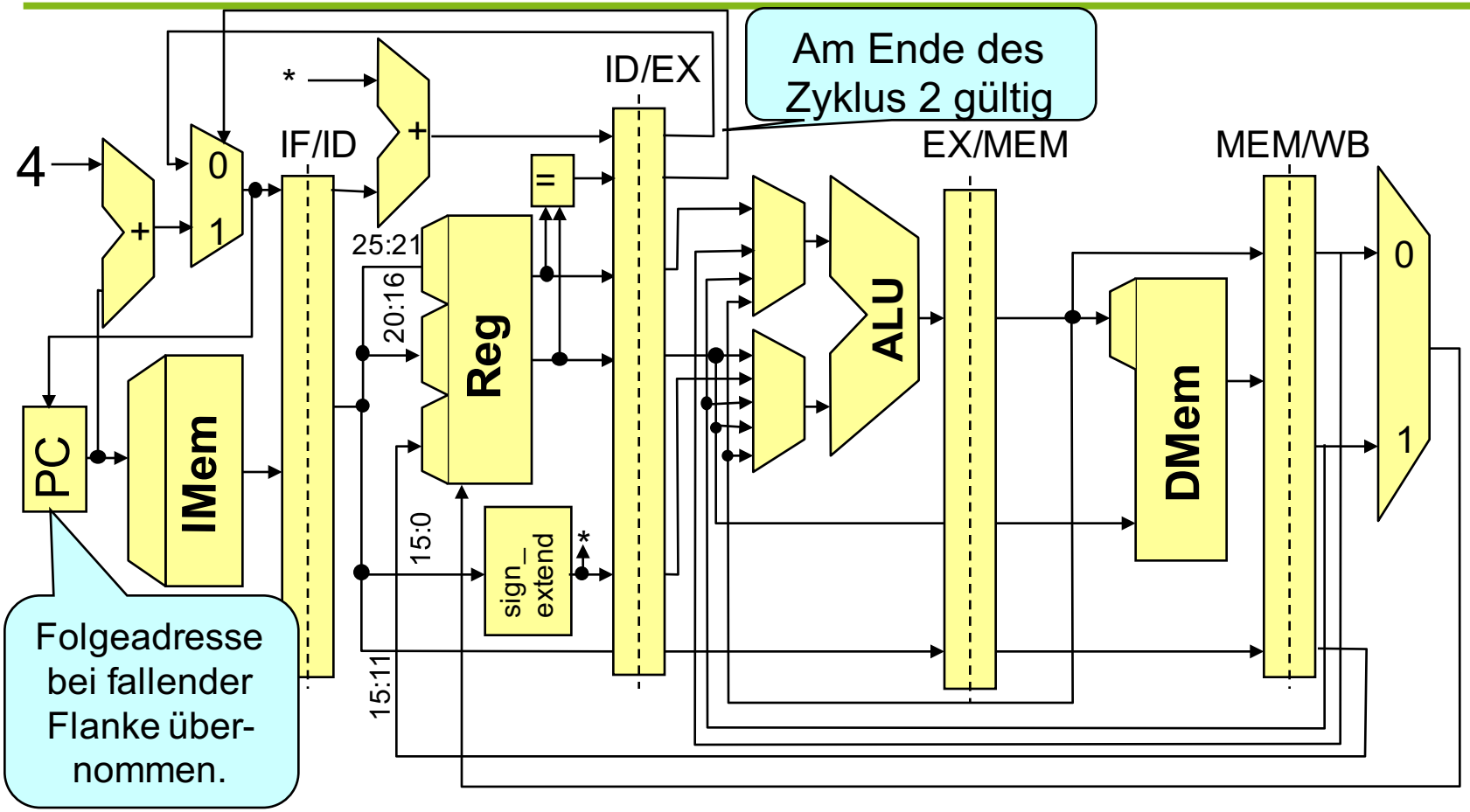
Probleme beim gezeigten Ansatz:

- Leistungsverlust durch 2 NOOPs (*branch delay penalty)*.
- ALU/Multiplexer in der gezeigten Form nicht ausreichend, um Test und Sprungzielberechnung in einem Takt auszuführen.

Lösungsansatz:

- Gleichheit der Register wird schon in der *instruction decode*-Stufe geprüft.
- Sprungziel wird in separatem Adressaddierer ebenfalls bereits in der *instruction decode*-Stufe berechnet.
- Sofern weiterhin noch Verzögerungen auftreten:
  - nächsten Befehl einfach ausführen (*delayed branch*).
  - oder weiterhin NOOP(s) einfügen (*stall*).

Reduktion der *branch delay penalty; delayed branch*



Zyklus 6				
...	...		**add**	**sub**

Delayed Branches, verzögerte Sprünge

Beim gezeigten Beispiel wird der auf den Sprungbefehl folgende Befehl immer noch ausgeführt.

```
        beq $12,$2,t
        sub  ...   # wird immer noch ausgeführt
        ....
t:      add ..
```

„It‘s not a bug, it‘s a feature“

Einen Platz für die Aufnahme eines solchen Befehls nennt man ***delay slot,*** die Sprünge ***delayed branches.***
Manche Maschinen haben mehrere *delay slots*.
Delay slots sollten von Übersetzern mit nützlichen Befehlen gefüllt werden. Nur notfalls sollte es ein NOOP sein.
Die MIPS-Maschine hat ein *delay slot*, welches aber vom Assembler verdeckt wird.

Typen von Fließband-Gefährdungen (*hazards*)

- **Strukturelle Abhängigkeiten/Gefährdungen (*structural hazards*)**
- **Datenfluß- Abhängigkeiten/Gefährdungen (*data hazards*)**
  - **aufgrund von Datenabhängigkeiten (RAW)**

    ☞ *forwarding, pipeline stalls*
  - **aufgrund von Antidatenabhängigkeiten (WAR)**
    (erst bei komplizierteren Systemen wichtig)
  - **aufgrund von Ausgabeabhängigkeiten (WAW)**
    (erst bei komplizierteren Systemen wichtig)
- **Kontrollfluß-Abhängigkeiten/Gefährdungen (*control hazards*)**

  ☞ *delayed branches, pipeline stalls,* spekulative Ausführung, Sprungvorhersage