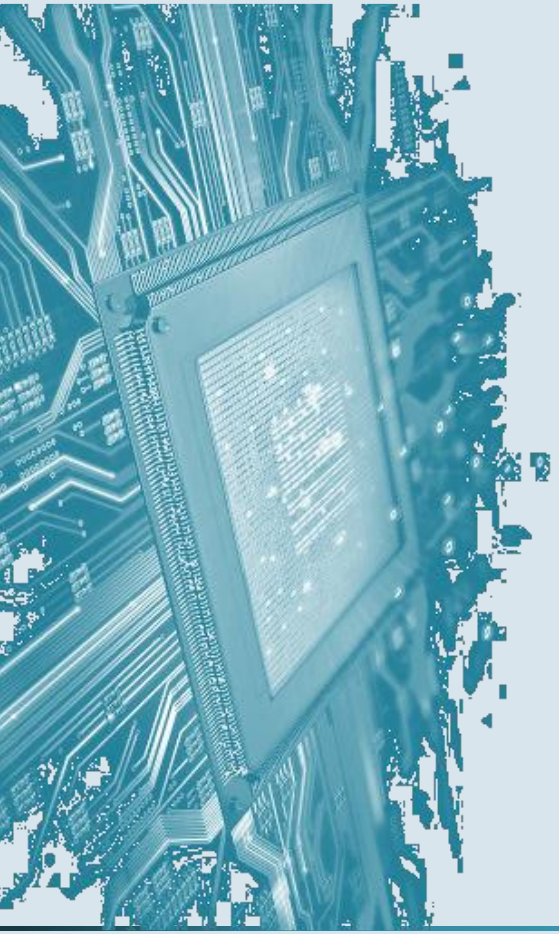


Rechnerarchitektur (AIN 2)

SoSe 2021

Kapitel 4

Multi-Cycle CPU – Pipeline-Architekturen

Prof. Dr.-Ing. Michael Blaich
mblaich@htwg-konstanz.de

Kapitel 4: Multi-Cycle CPU – Pipeline-Architekturen

4.1 Prinzip einer Pipeline

4.2 Datapath der MIPS Pipeline

4.3 Control der MIPS Pipeline

4.4 Hazards

4.4.1 Data Hazards

4.4.1.1 Forwarding

4.4.1.2 Stalling

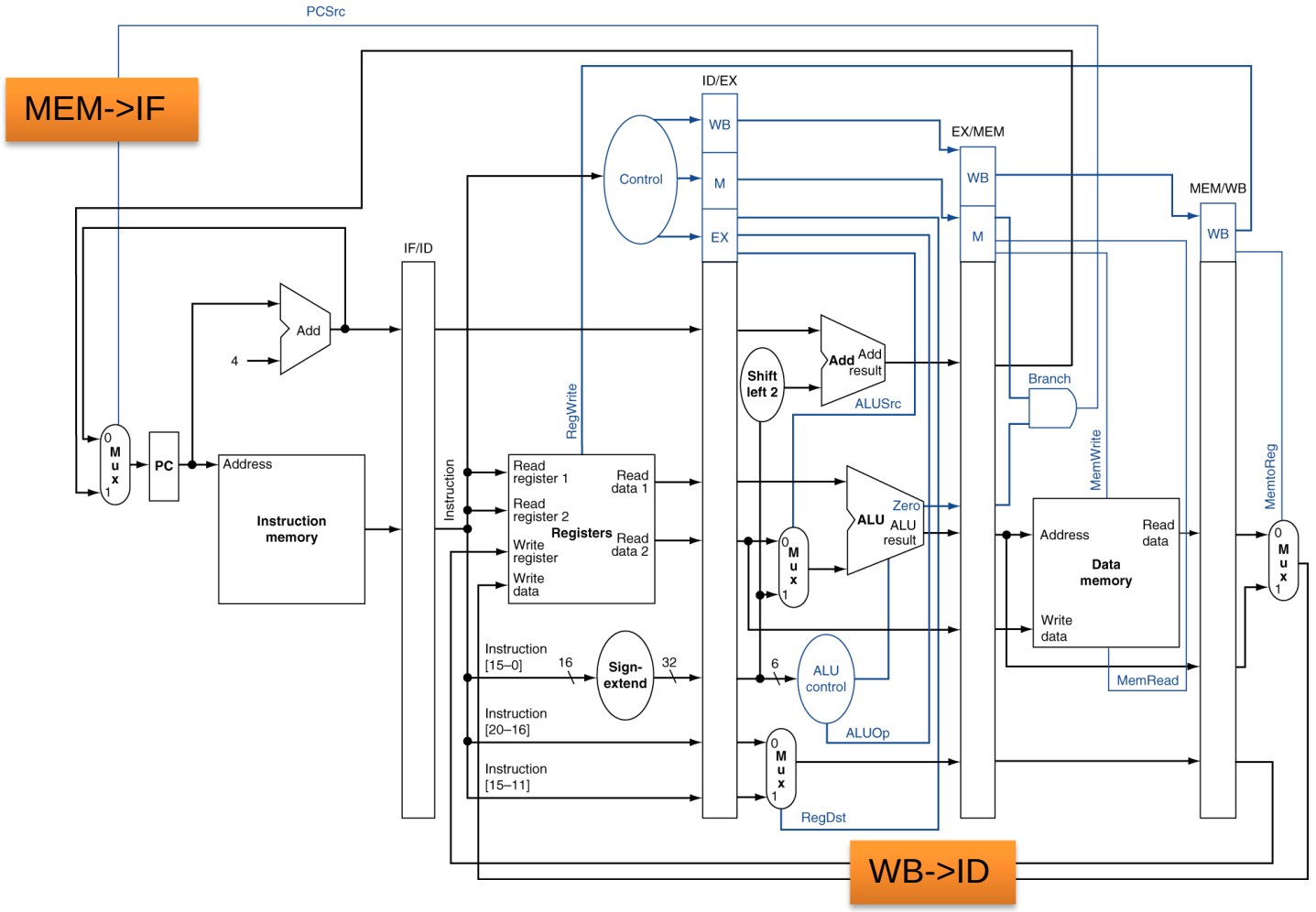
4.4.2 Control Hazards

4.5 Exceptions

Hazards in der MIPS Pipeline

Hazards treten auf, wenn Daten oder Steuersignale von rechts nach links in der Pipeline fließen

- Steuersignale verursachen Control-Hazards (zB. PCSrc von MEM nach IF)
- Daten verursachen Data-Hazard (z.B. zu schreibender Werte von WB nach ID)



Beispiel: Hazards in der MIPS Pipeline

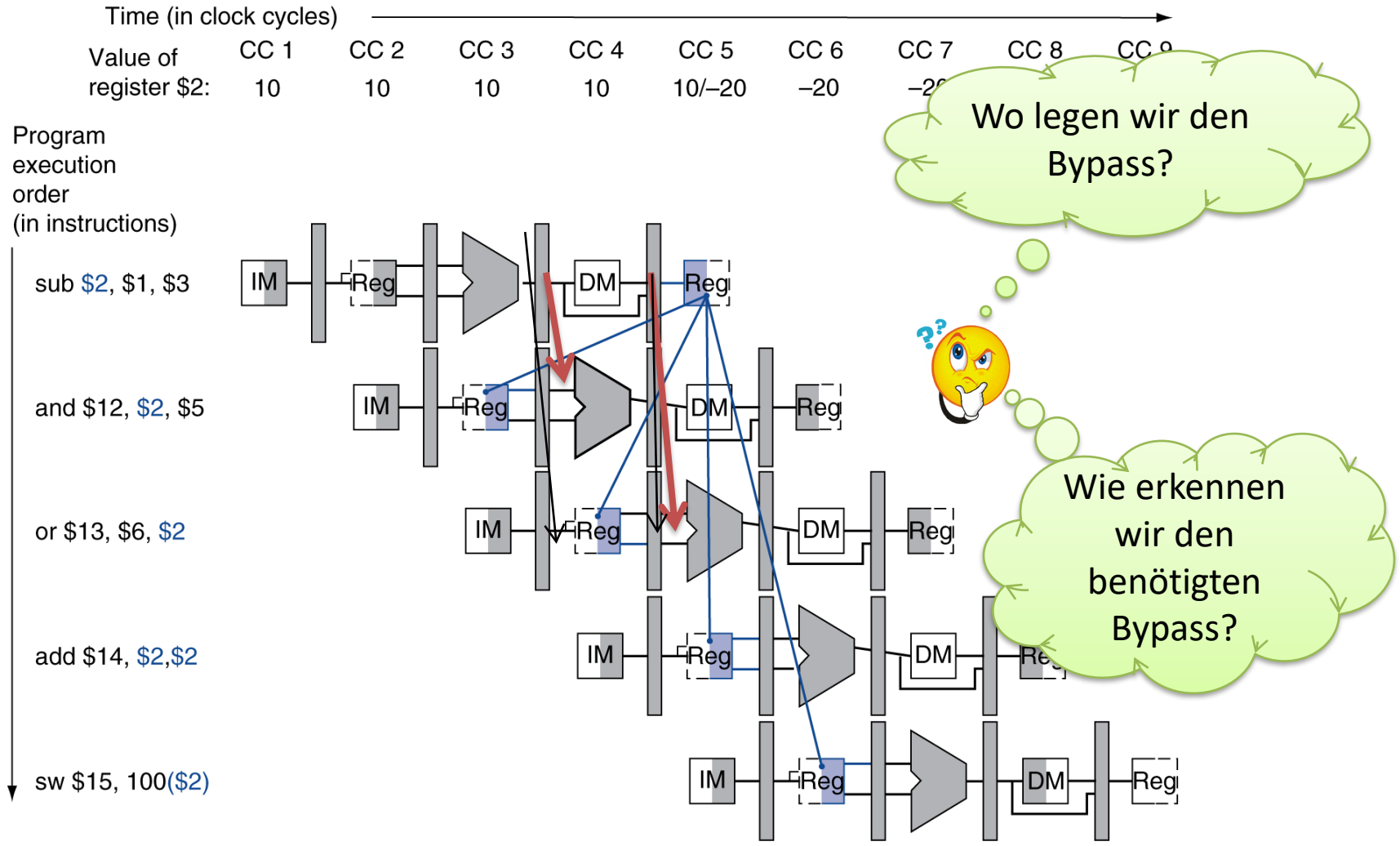
- Wir können Hazards einfach erkennen und wir wissen auch, welche Hazards wie durch Forwarding vermieden werden können.
- Aber wie wird das Erkennen von Hazards und Forwarding in Hardware realisiert?


```
1   sub $2, $1,$3
2   and $12,$2,$5
3   or  $13,$6,$2
4   add $14,$2,$2
5   sw  $15,100($2)
```

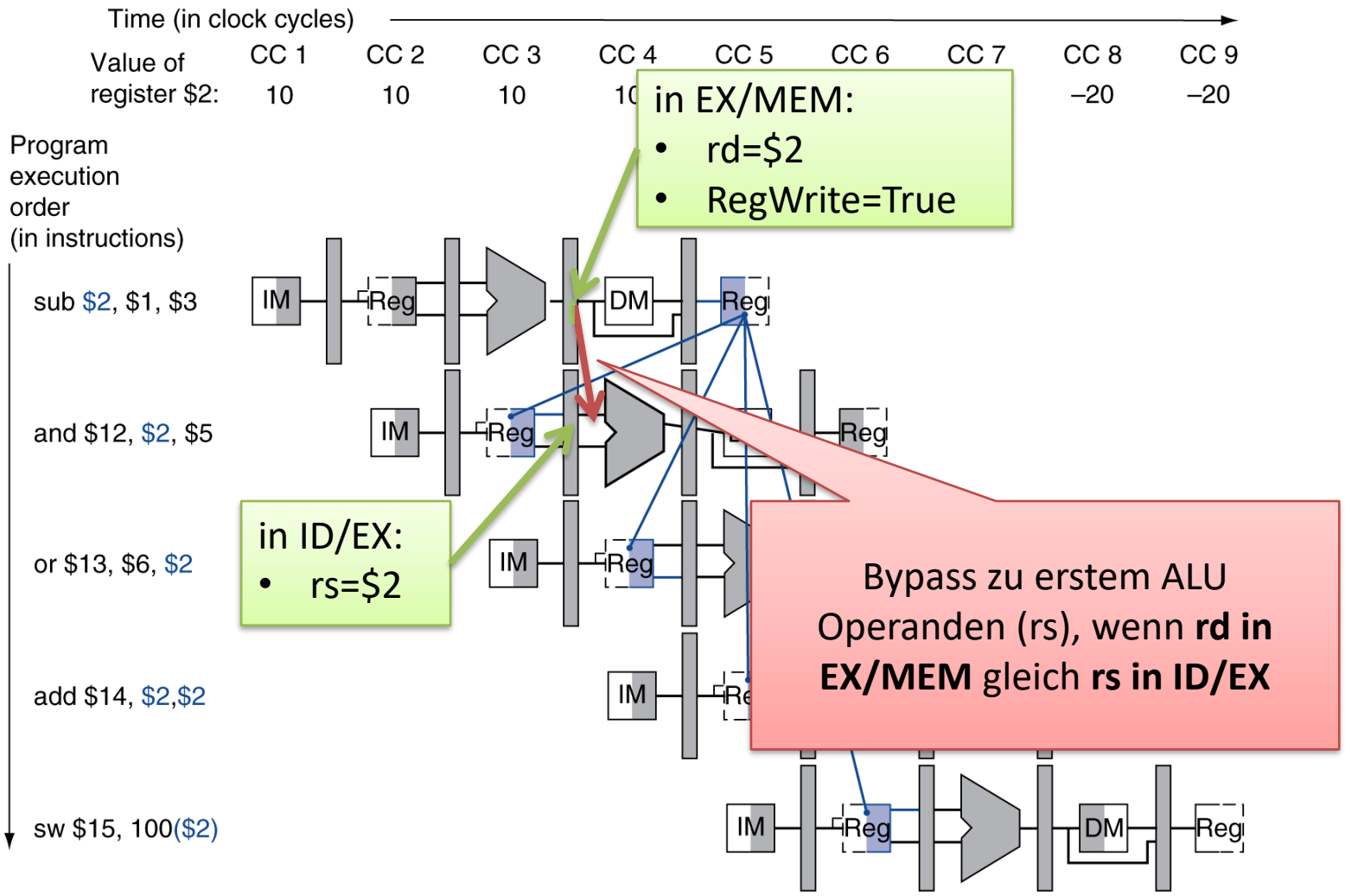
```
sub $2, $1,$3
and $12,$2,$5
or  $13,$6,$2
add $14,$2,$2
sw  $15,100($2)
```

Verwendung des Registers $2

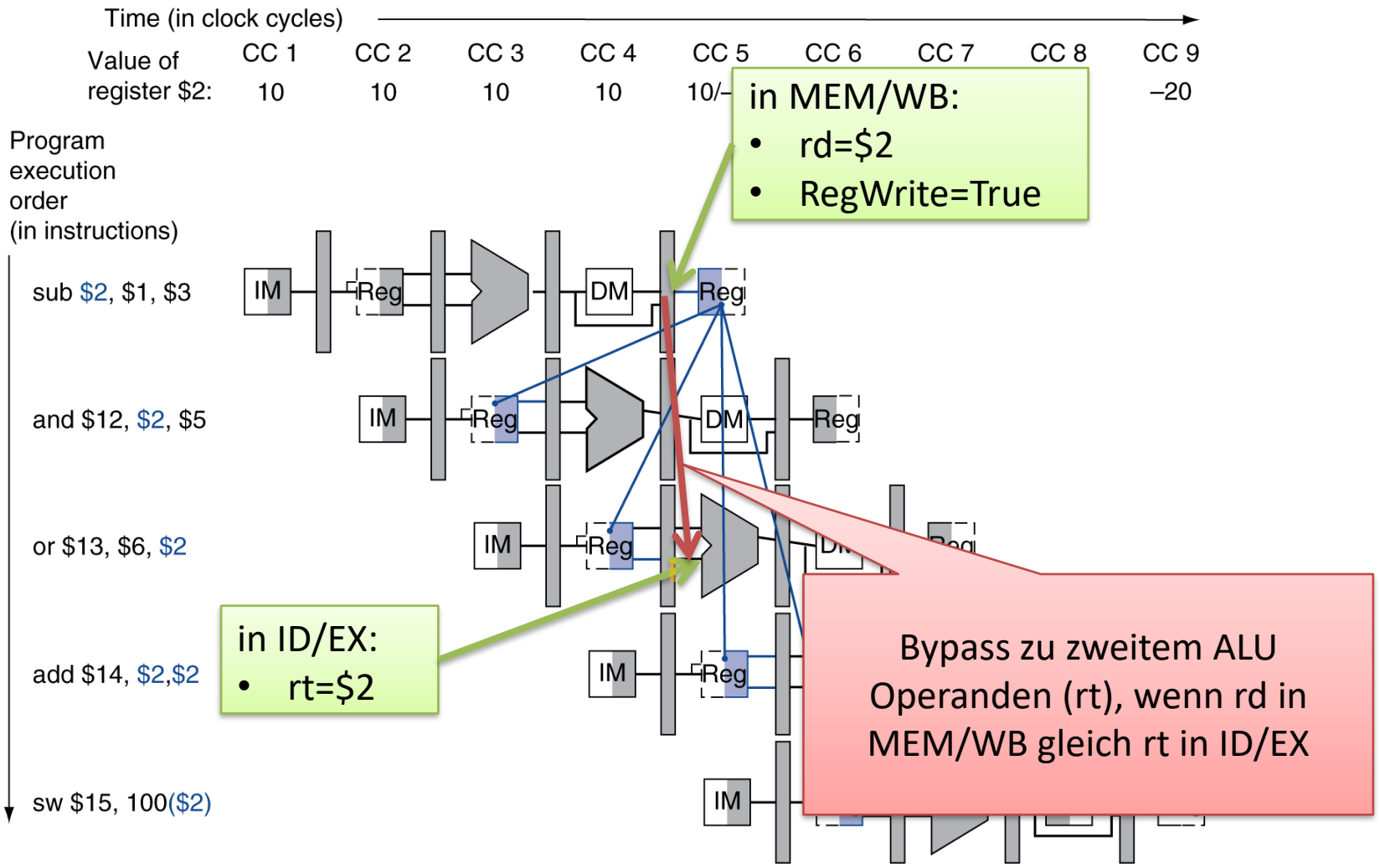
Das Register $2 tritt in aufeinanderfolgenden Befehlen als Zielregister und als Operandenregister auf.



Bypass-Erkennung: Beispiel 1



Bypass-Erkennung: Beispiel 2



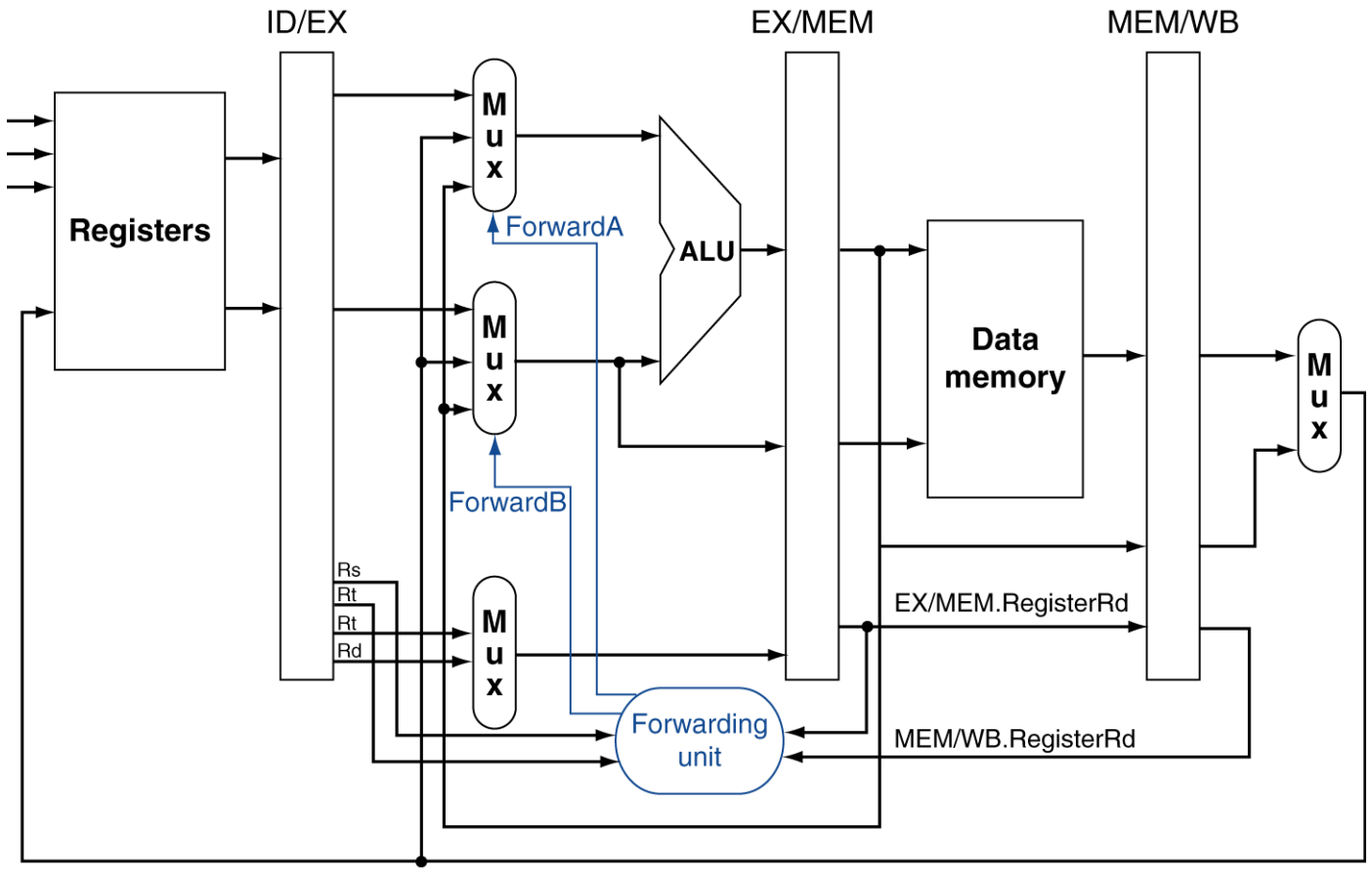
Notation und allgemeine Realisierung

- Notation:
  - <rd> in ID/EX: ID/EX.RegisterRd
  - <rt> in MEM/WB: MEM/WB.RegisterRt
  - Register **Rx** in Pipeline-Register **PRy**: **PRy**.Register**Rx**
- Bedingungen für Data-Hazard-Bypass:

von Stage nach Register	MEM	WB
rs (erster Operand)	ID/EX.RegisterRs == EX/MEM.RegisterRd	ID/EX.RegisterRs == MEM/WB.RegisterRd
rt (zweiter Operand)	ID/EX.RegisterRt == EX/MEM.RegisterRd	ID/EX.RegisterRt == MEM/WB.RegisterRd
rs und rt	EX/MEM.RegWrite==True	MEM/WB.RegWrite==True
	EX/MEM.Register.Rd!=0	MEM/WB.RegisterRd!=0

Forwarding in Hardware

Forwarding wir über eine Forwarding Unit realisiert, die über zwei Multiplexer in der EX-Stage bestimmt, ob die Operanden aus dem ID/EX-Register übernommen oder aus einem Pipeline-Register „geforwarded“ werden.



Forwarding

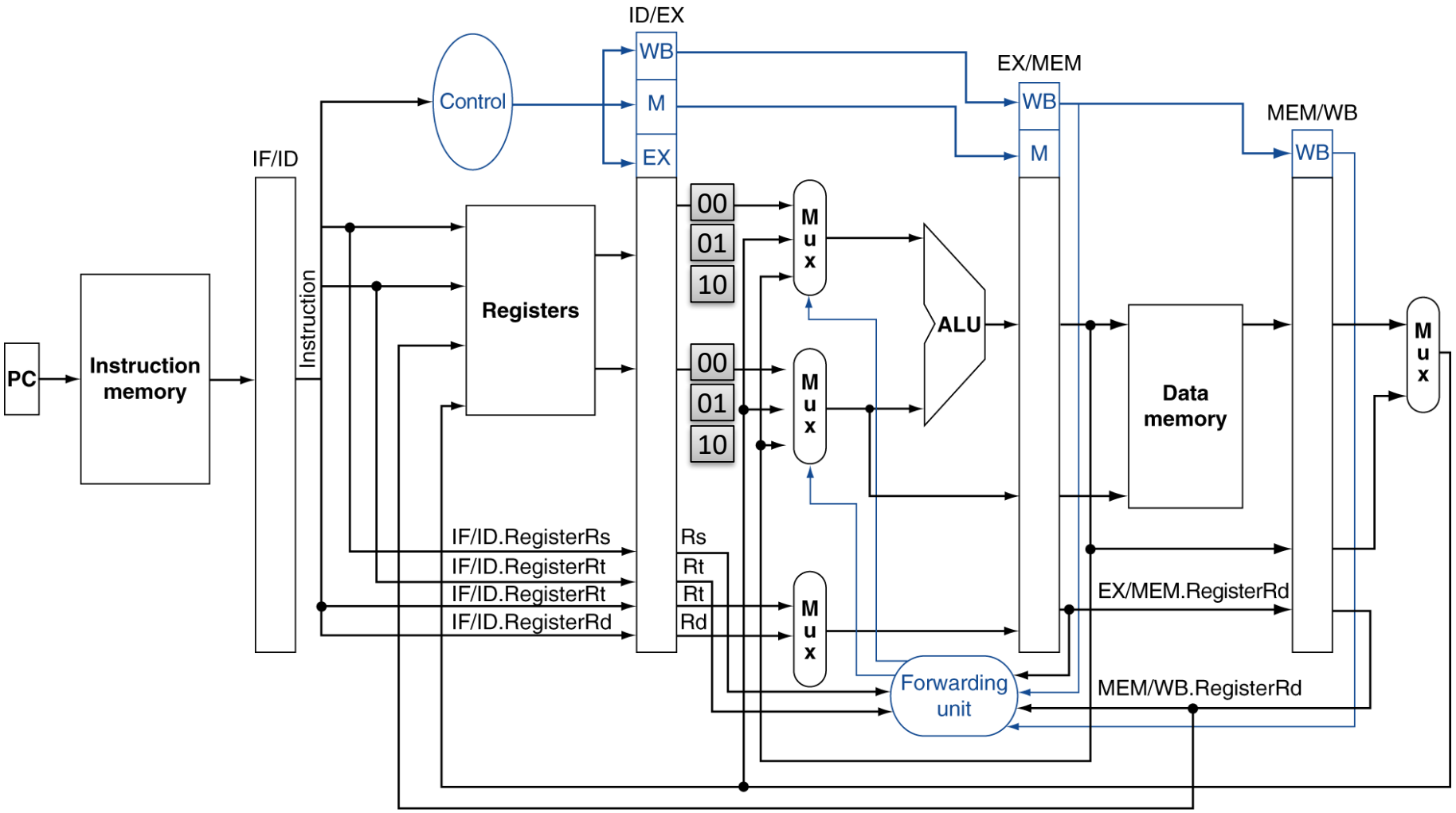
- Datapath:
  - jeder ALU Operand der Instruktion n wird über einen Multiplexer ausgewählt zwischen
  - dem "originärem" Register-Inhalt aus dem ID/EX-Register, der von Instruktion n bestimmt wurde
  - dem in Instruktion n-1 in der EX-Stage berechneten ALU-Ergebnis, das im EX/MEM-Register steht
  - dem in Instruktion n-2 in der EX-Stage berechneten ALU-Ergebnis, das im MEM/WB-Register steht
  - dem in Instruktion n-2 in der MEM-Stage geladenen Ergebnis, das im MEM/WB-Register steht
- Control:
  - Die Multiplexer werden von der Forwarding Unit gesteuert.
  - Eingänge der Forwarding Unit sind
    - die Zielregister der EX/MEM- und MEM/WB-Register (EX/MEM.RegisterRd bzw. MEM/WB.RegisterRd)
    - die Operanden-Register der Instruktion n in der ID-Stage, die im ID/EX-Register stehen (ID/EX.RegisterRs und ID/EX.RegisterRt)
    - die Steuersignale RegWrite der Instruktionen n-1 und n-2, die in den EX/MEM btw. MEM/WB-Registern stehen (EX/MEM.RegWrite bzw.) MEM/WB.RegWrite (nicht in der Grafik).
  - Aufgrund der Eingänge bestimmt die Forwarding Unit entsprechend der Logik auf der Folie - Forwarding-Bedingungen, welcher Eingang in den Multiplexern geschaltet wird.

Forwarding-Bedingungen

- EX hazard
  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd == ID/EX.RegisterRs)) ForwardA = 10
  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd == ID/EX.RegisterRt)) ForwardB = 10
- MEM hazard
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and (MEM/WB.RegisterRd == ID/EX.RegisterRs)) ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and (MEM/WB.RegisterRd == ID/EX.RegisterRt)) ForwardB = 01

Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file.
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result.
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result.
ForwardB = 00	ID/EX	The second ALU operand comes from the register file.
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result.
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result.

Datapath mit Forwarding



Doppelte Hazards

In diesem Beispiel sind für die dritte Instruktion die Bedingungen für Forwarding aus dem MEM/WB-Pipeline-Register als auch aus dem EX/MEM-Pipeline-Register erfüllt. In diesem Fall werden nur die Daten aus dem EX/MEM-Pipeline-Register „geforwardet“.

Einfacher Data-Hazard

```
add $1,$1,$2
sw  $15,100($2)
add $1,$1,$4
```

MEM/WB.RegisterRd == ID/EX.RegisterRs

Doppelter Data-Hazard

```
add $1,$1,$2
add $1,$1,$3
add $1,$1,$4
```

EX/MEM.RegisterRd == ID/EX.RegisterRs

MEM/WB.RegisterRd == ID/EX.RegisterRs

Forwarding-Bedingungen

- EX hazard
  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd == ID/EX.RegisterRs)) ForwardA = 10
  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd == ID/EX.RegisterRt)) ForwardB = 10
- MEM hazard
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd ≠ ID/EX.RegisterRs))
    and (MEM/WB.RegisterRd == ID/EX.RegisterRs)) ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd ≠ ID/EX.RegisterRt))
    and (MEM/WB.RegisterRd == ID/EX.RegisterRt)) ForwardB = 01

Kapitel 4 - Gliederung

Kapitel 4: Multi-Cycle CPU – Pipeline-Architekturen

4.1 Prinzip einer Pipeline

4.2 Datapath der MIPS Pipeline

4.3 Control der MIPS Pipeline

4.4 Hazards

4.4.1 Data Hazards

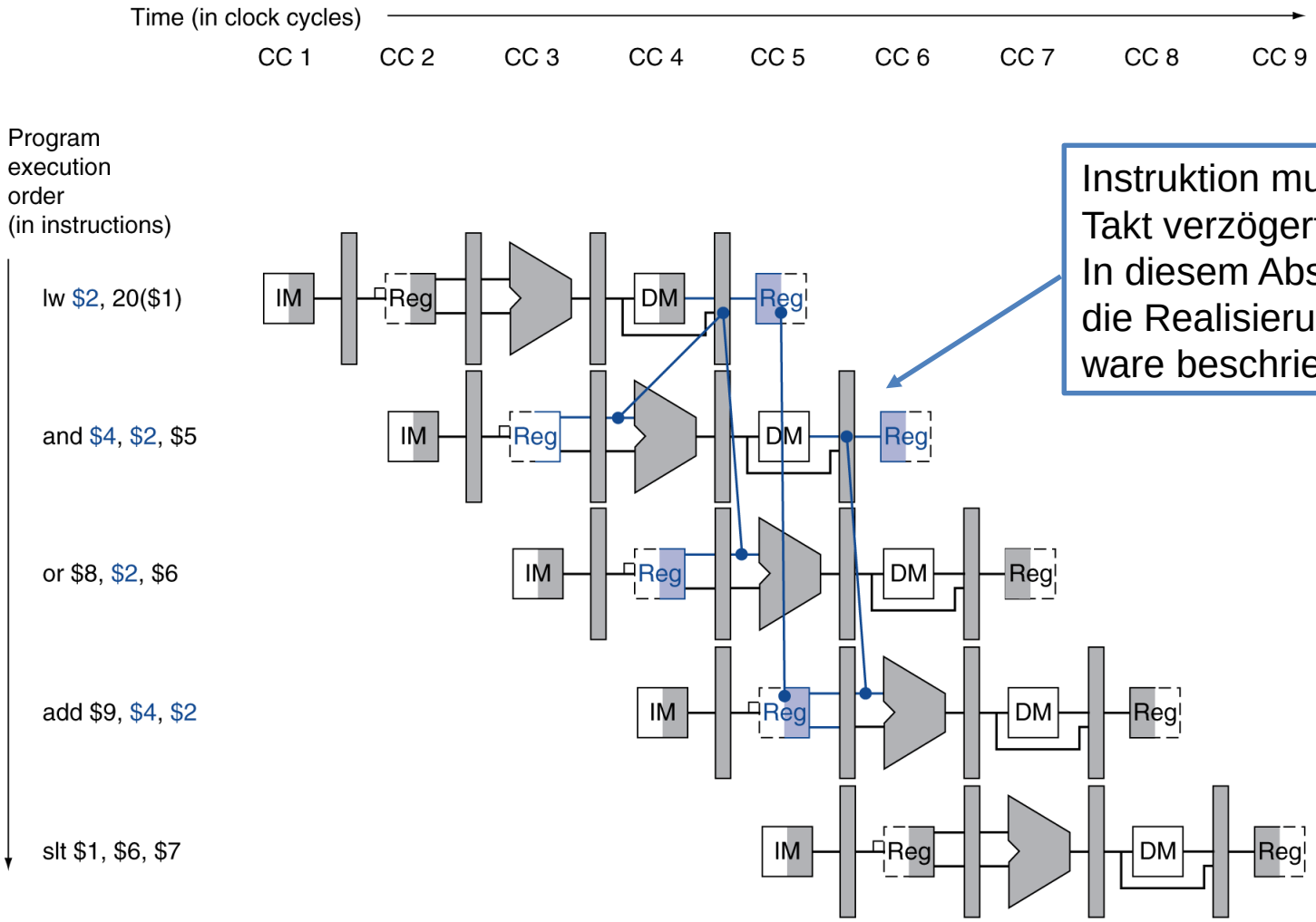
4.4.1.1 Forwarding

4.4.1.2 Stalling

4.4.2 Control Hazards

4.5 Exceptions

Load-Use-Hazards



Erkennung von Load-Use Hazards

- Bubbles werden so früh wie möglich erzeugt
  - Test beim Dekodieren der “Use”-Instruktion in der ID Stage

- Register der ALU Operanden (rs, rt) der „Use“ Instruktion stehen zu Beginn der ID Stage im Pipeline-Register IF/ID
  - IF/ID.RegisterRs, IF/ID.RegisterRt

- Ein Load-Use-Hazard liegt vor, wenn
  1. Die Instruktion n eine „Load“-Instruktion ist:

     `ID/EX.MemRead==True`
  2. Das Zielregister (rt) der „Load-Instruktion“ (Instruktion n in der EX-Stage) gleich einem der beiden ALU Operanden-Register (rs oder rt) der darauffolgenden Use-Instruktion (Instruktion n+1 in der ID-Stage) ist:

     `ID/EX.RegisterRt==IF/ID.RegisterRs`

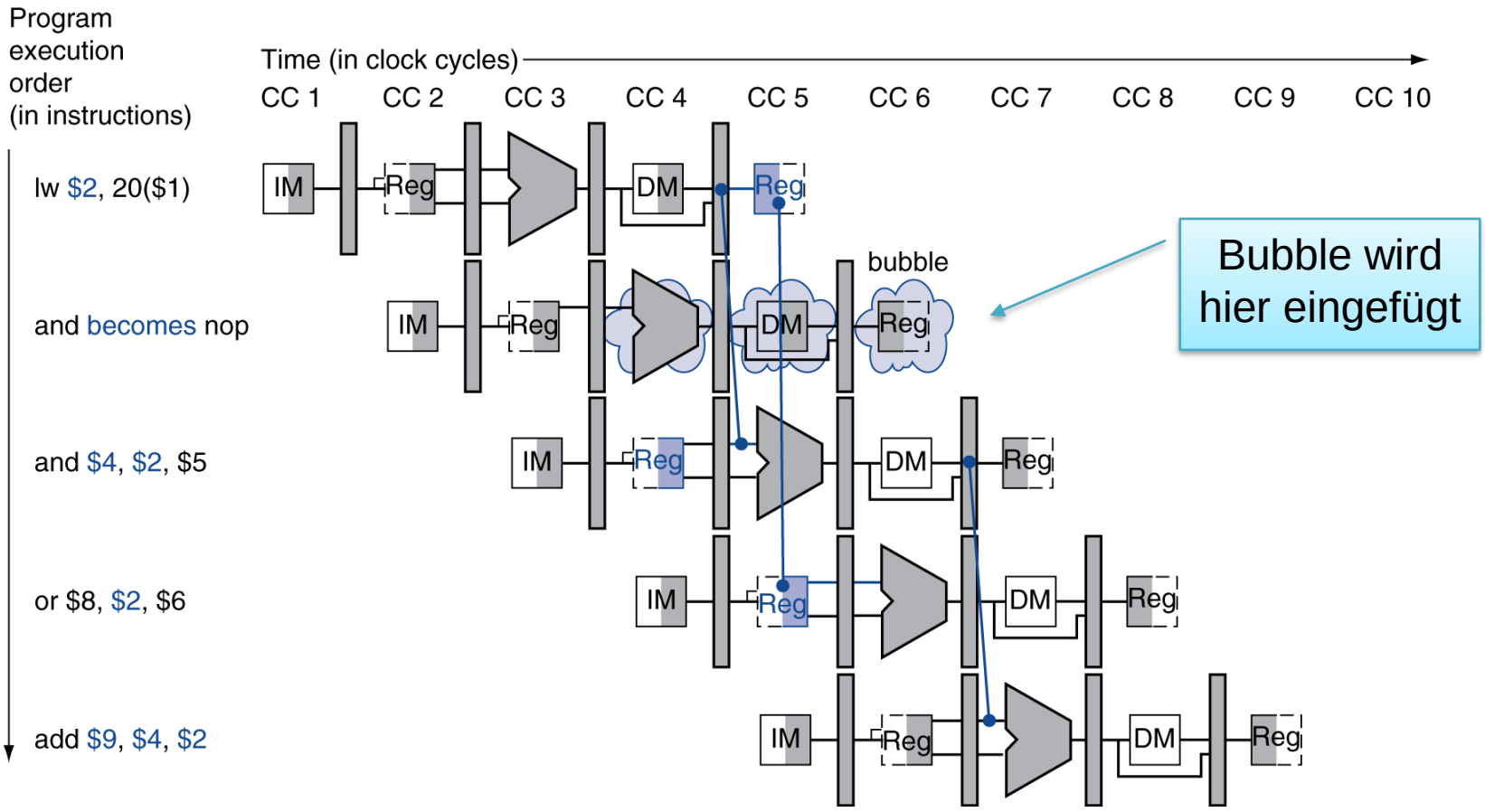
     oder `ID/EX.RegisterRt==IF/ID.RegisterRt`

Wie entsteht eine Bubble?

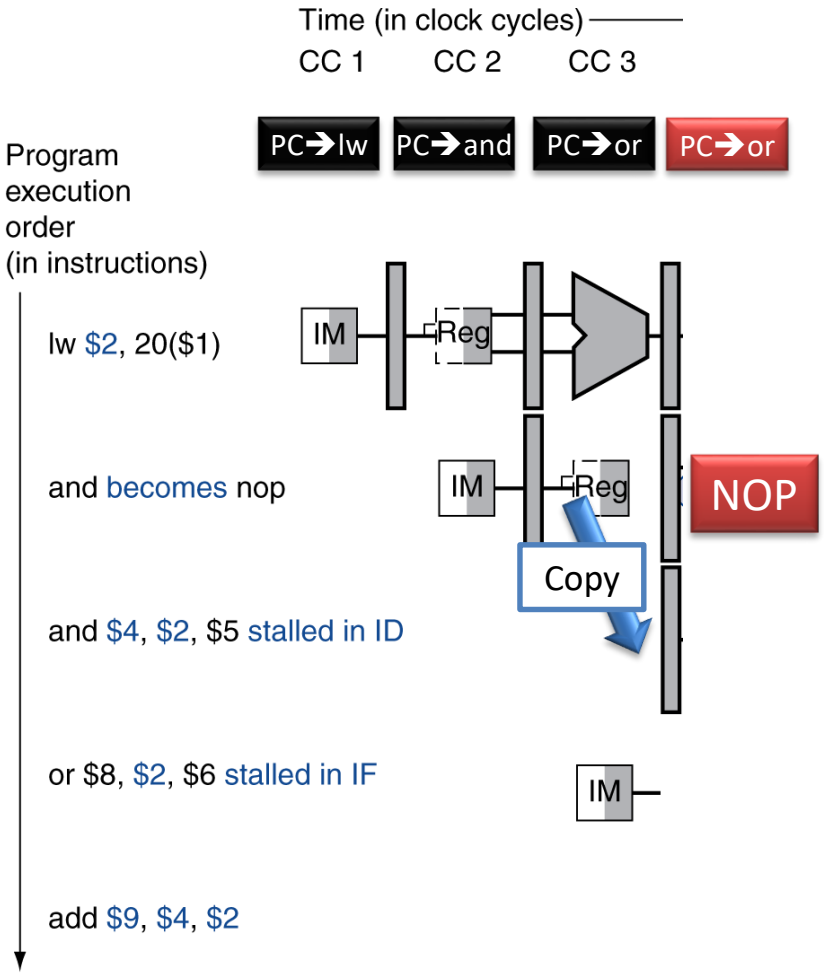
- Einträge in ID/EX-Register zurücksetzen
  - EX, MEM und WB führen keine Operation bzw. die Operation NOP (NO-Operation) aus
  - NOP wird ausgeführt, wenn alle Pipeline-Register-Einträge auf 0 gesetzt werden
- Update von PC und IF/ID Pipeline-Register muss verhindert werden
  - vorige Instruktion wird noch einmal dekodiert
  - nachfolgende Instruktion wird noch einmal geholt
  - Stalling für einen Takt ermöglicht es, in der MEM Stage die Daten für `lw` zu lesen
    - danach können die Daten in die EX Stage geforwarded werden

Beispiel: Erzeugen der Bubble

- Control erkennt in Takt 3 (CC 3, Load-Instruktion in EX), dass ein Load-Use-Hazard vorliegt, da ID/EX.RegisterRt=$2 und IF/ID.RegisterRs=$2
- eine Bubble wird eingefügt



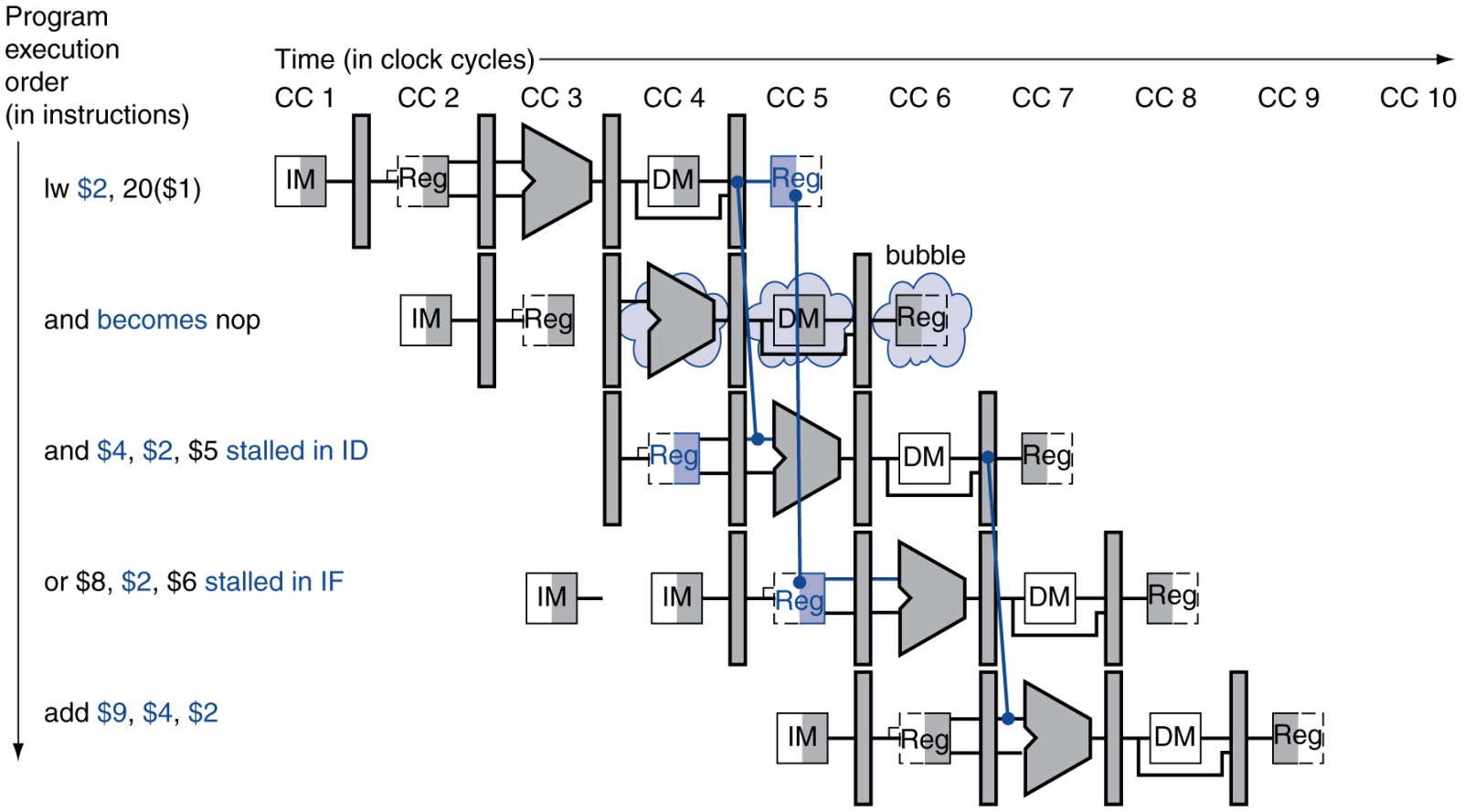
Ablauf in CC3



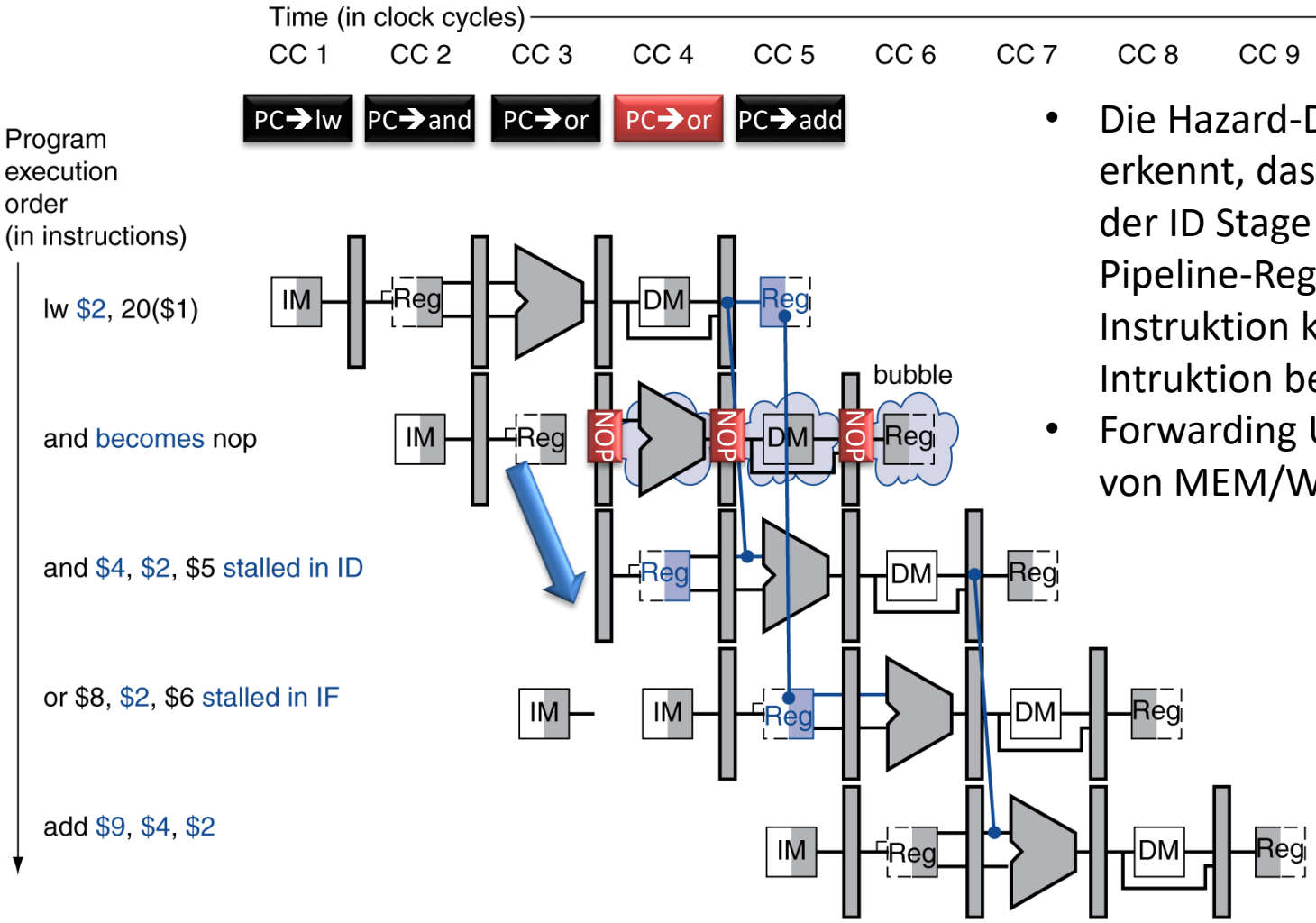
1. In Takt 3 erkennt die Hazard-Detection-Unit, dass ein Hazard vorliegt und die “and”-Instruktion um einen Takt verzögert werden muss.
2. Die Ergebnisse der AND-Instruktion in der ID-Stage werden ignoriert. Stattdessen wird eine NOP-Instruktion in das ID/EX-Register geschrieben. Diese Instruktion bewirkt, dass in den folgenden Stages EX, MEM und WB keine Daten verändert werden.
3. Die Ergebnisse der OR-Instruktion in der IF Stage werden ebenfalls ignoriert. Das IF/ID-Register bleibt unverändert, so dass im nächsten Takt die AND-Instruktion noch einmal in der ID Stage ausgeführt wird.
4. Auch der PC bleibt unverändert, so dass noch einmal die OR-Instruktion in der IF-Stage geladen wird.

Ablauf in CC3

1. Erkennen des Load-Use-Hazards
2. ID-Stage für AND ➔ NOP in ID/EX
3. IF-Stage für OR ➔ ID/EX bleibt unverändert und enthält Werte von AND
4. PC bleibt unverändert und zeigt auf OR



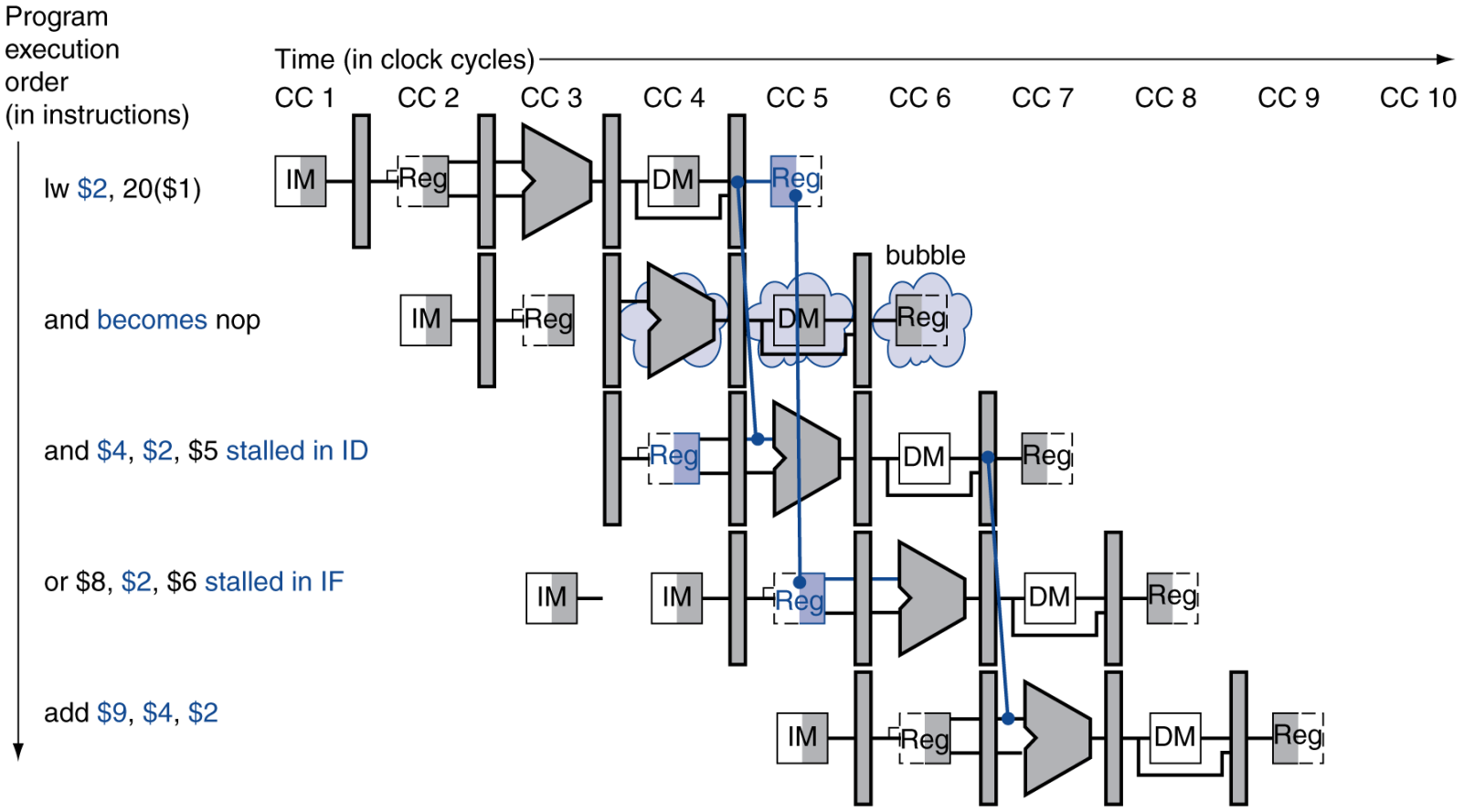
Ablauf in CC4



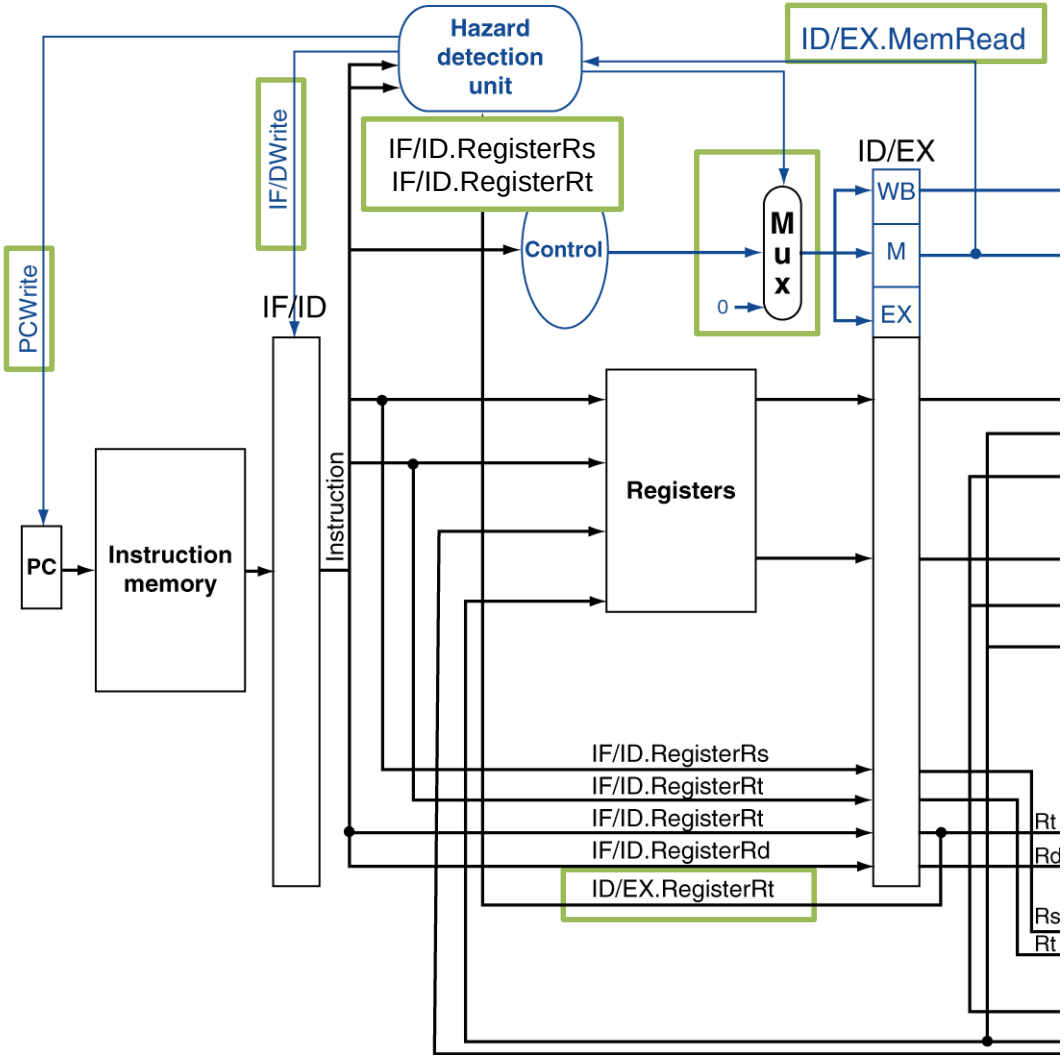
- Die Hazard-Detection-Unit erkennt, dass kein Hazard in der ID Stage vorliegt da ID/EX-Pipeline-Register eine NOP Instruktion keine LW Intruktion beinhalten
- Forwarding Unit leitet Daten von MEM/WB (lw) zu EX (and)

Ablauf in CC4

1. Hazard Detection Unit: kein Load-Use-Hazard
2. ID-Stage für AND ➔ Werte werden in ID/EX geschrieben
3. IF-Stage für OR ➔ Instruktion und nächster PC werden in IF/ID geschrieben
4. PC zeigt auf ADD



Hazard Detection Unit



Input:

- IF/ID.RegisterRs
- IF/ID.RegisterRt
- ID/EX.RegisterRt
- ID/EX.MemRead

Detection:

(IF/ID.RegisterRs==ID/EX.RegisterRt or IF/ID.RegisterRt== ID/EX.RegisterRt) and ID/EX.MemRead==True

Output (Bubble Erzeugung):

- PCWrite=False
- IF/IDWrite=False
- MUX-Input=0 (NOP)

Datapath mit Hazard Detection