MIXED SIGNAL
MSC
CIRCUIT DESIGN

Technische Universität Berlin

Rechnerorganisation

Einstein-Prof. Dr.-Ing. Friedel Gerfers

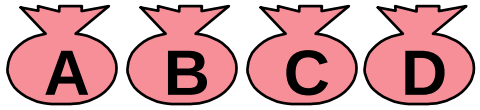
Kapitel 7: Pipelining

Ziele

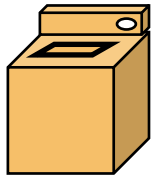
- Nach dieser Vorlesung sollten Sie in der Lage sein…
  - Die Grundlagen von Pipelining zu erklären
  - Pipeliningkonflikte und mögliche Lösungen zu erläutern (Datenkonflikte, Steuerkonflikte, Forwarding)
  - MIPS-Code zu modifizieren, um die Leistung für Prozessoren mit Pipelining zu erhöhen
  - Die Veränderungen zu benennen, die notwendig sind, um einen Befehl zum Instruktionssatz des Pipeline-Prozessors hinzuzufügen
  - Die Steuerung des Pipeline-Prozessors zu beschreiben
  - Zu erläutern, was Sprung-Vorhersage (*branch prediction*) und Superskalar-Prozessoren sind

Pipelining ist alltäglich

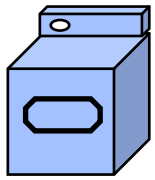
- Beispiel: Wäscheproblem von Ann, Brian, Cathy und Dave
- Jeder hat eine Ladung Kleidung
- zu waschen, zu trocknen und zu falten.



- Waschmaschine benötigt 30 Minuten

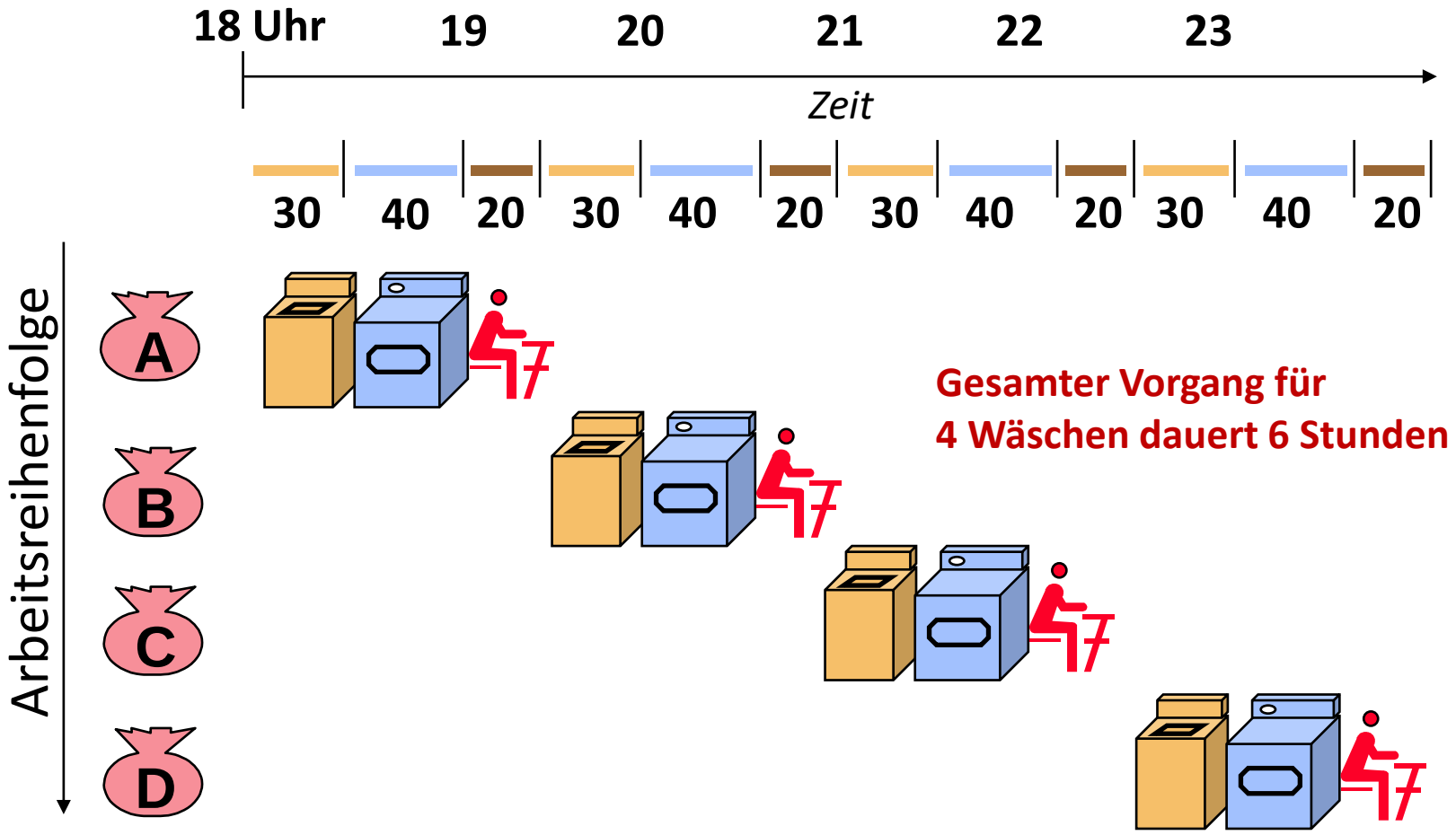


- Trockner ist nach 40 Minuten fertig

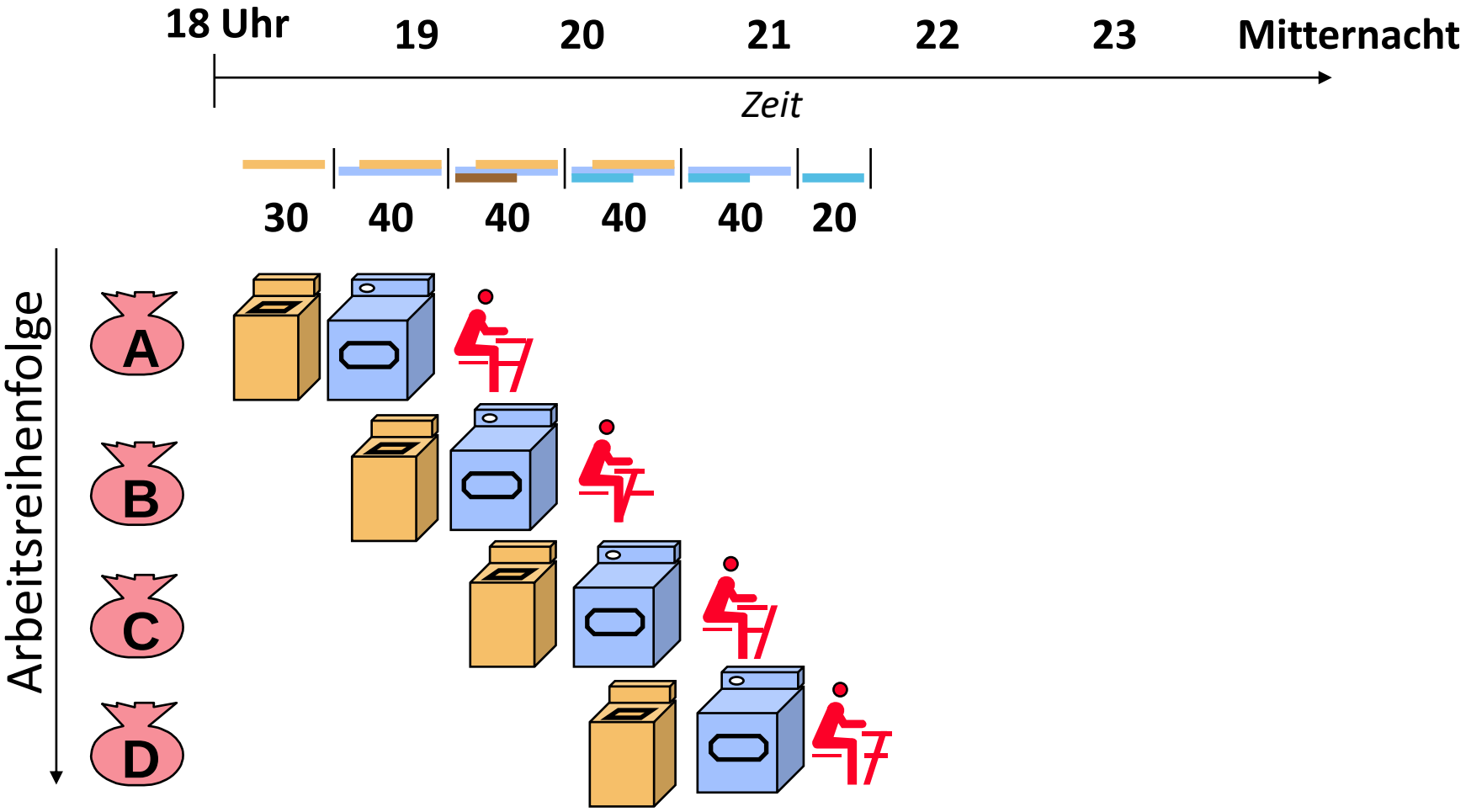


- Falten benötigt 20 Minuten

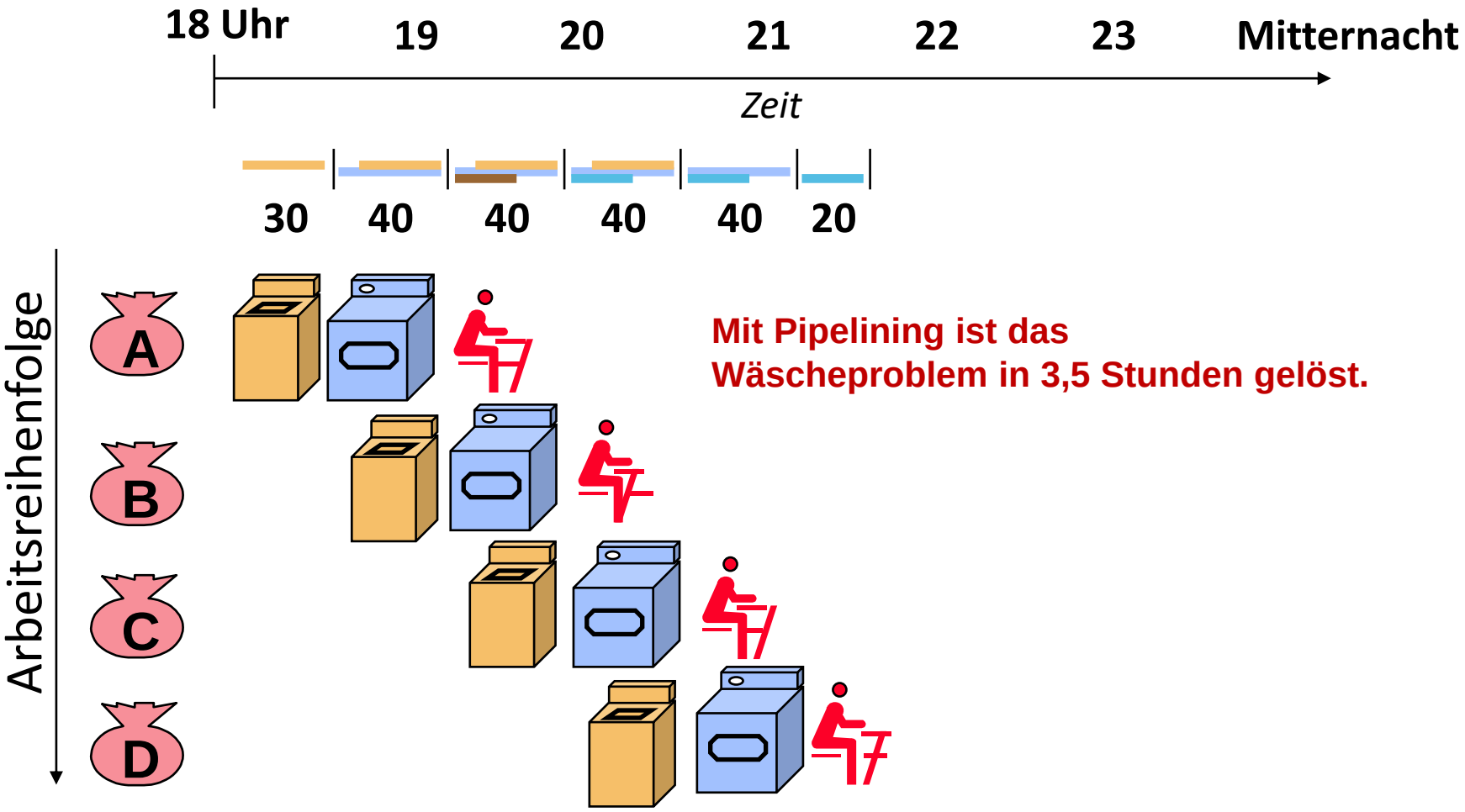
Wäscheproblem (sequentiell)



Wäscheproblem (pipelined) – Starte die Arbeit sobald wie möglich



Wäscheproblem (pipelined) – Speed-Up



Mit Pipelining ist das Wäscheproblem in 3,5 Stunden gelöst.

Computer Pipeline / 1

- MIPS-Befehle haben typischerweise 5 Ausführungsschritte:
  1. Befehlsholschritt - Instruction Fetch (IF)
  2. Befehlsdekodierung- und Registerholschritt - Instruction Decode (ID)
  3. Ausführung oder Adressberechnung - Execution (EX)
  4. Speicherzugriff – Memory Access (MEM)
  5. Rückschreiben - Write Back (WB)

- Nicht alle Befehle brauchen alle Stufen, aber sie müssen diese Stufen durchlaufen, da die Pipeline synchron bleiben muss.

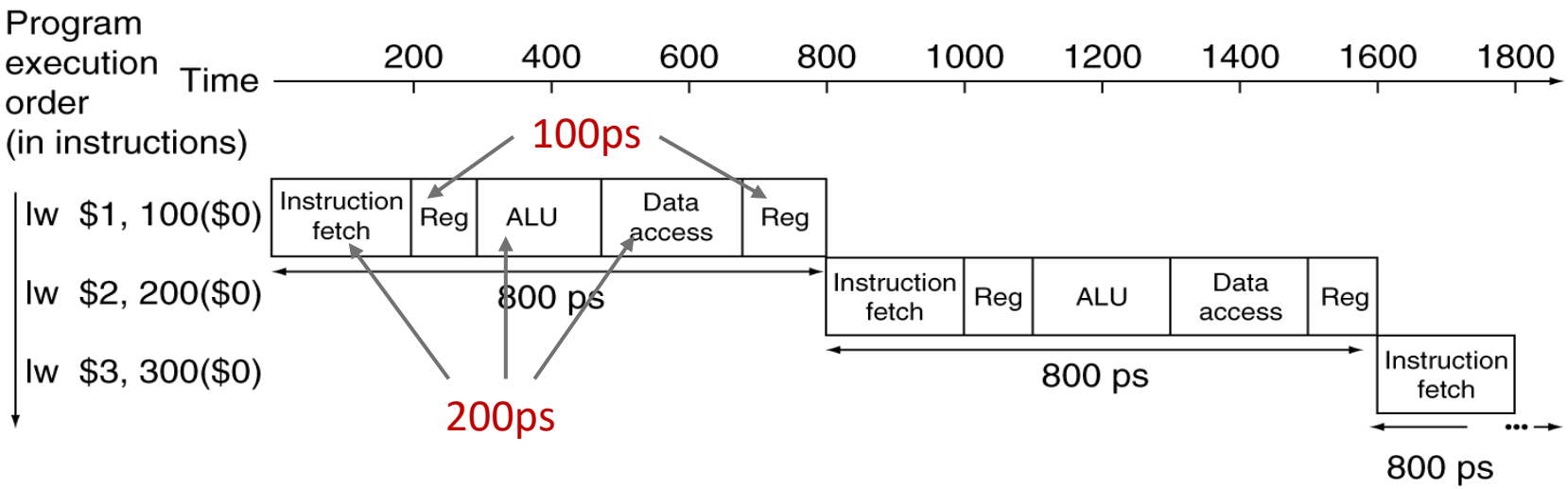
Computer Pipeline / 2

Befehl	Zyklus								
	1	2	3	4	5	6	7	8	9
Befehl 1	IF	ID	EX	MEM	WB				
Befehl 2		IF	ID	EX	MEM	WB			
Befehl 3			IF	ID	EX	MEM	WB		
Befehl 4				IF	ID	EX	MEM	WB	
Befehl 5					IF	ID	EX	MEM	WB

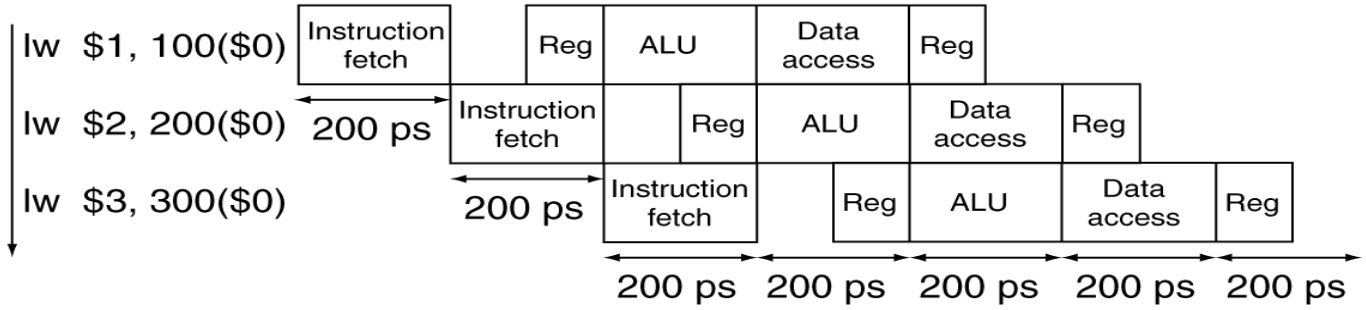
- Pipelining verbessert die Leistung durch das Erhöhen des Durchsatzes, **nicht** durch Reduzierung der Ausführungszeit der einzelnen Befehle.

Leistung (mit und ohne Pipelining)

- **Ohne Pipelining (T_{cycle}= 800ps)**



- **Mit Pipelining (T_{cycle}= 200ps)**



Best möglicher Speedup

- Best möglicher Speedup gegenüber Eintakt-Implementierung = Anzahl der Stufen

- Wird dieser Wert erreicht?

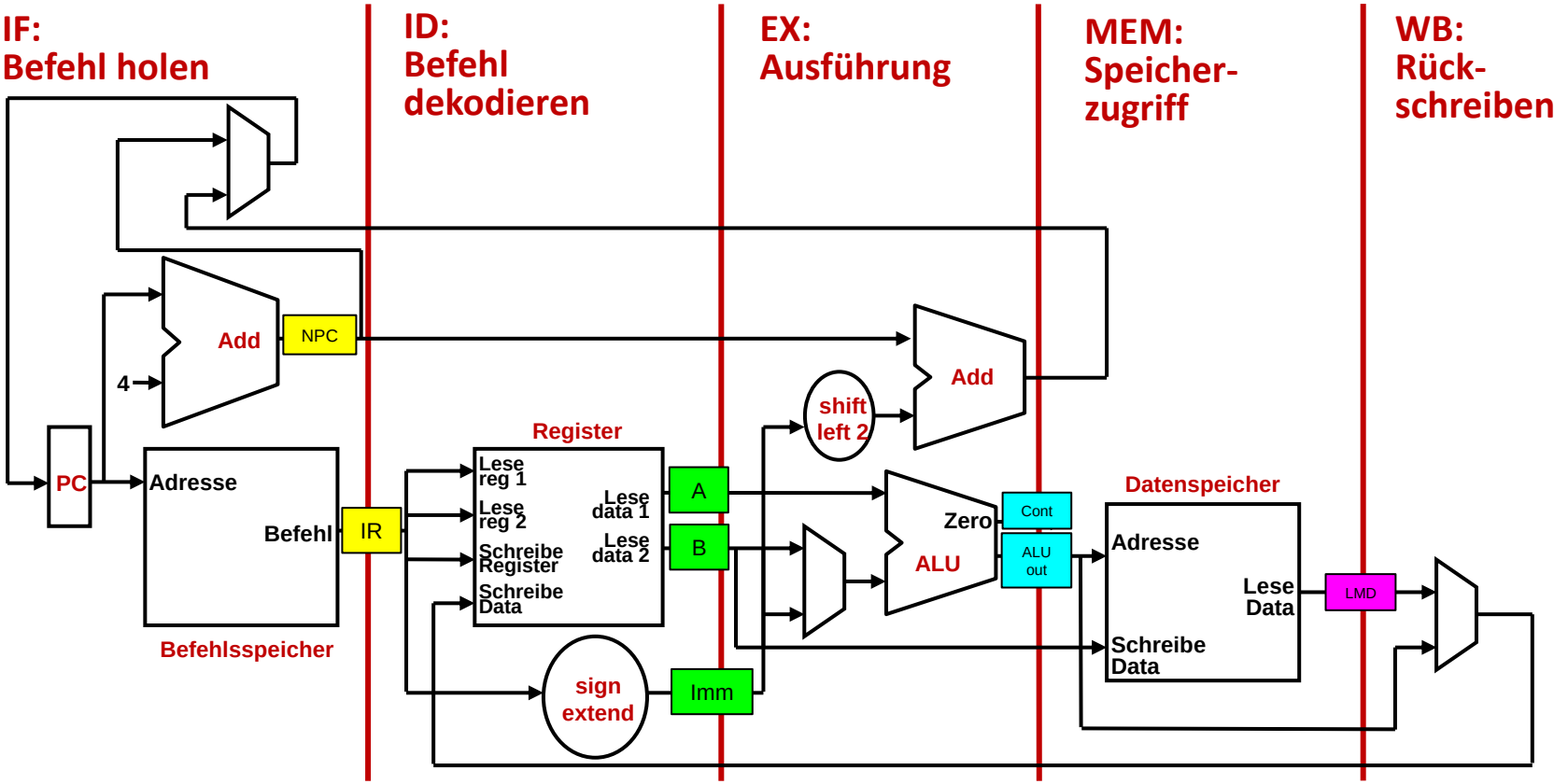
Pipelining und ISA-Design

Technische Universität Berlin

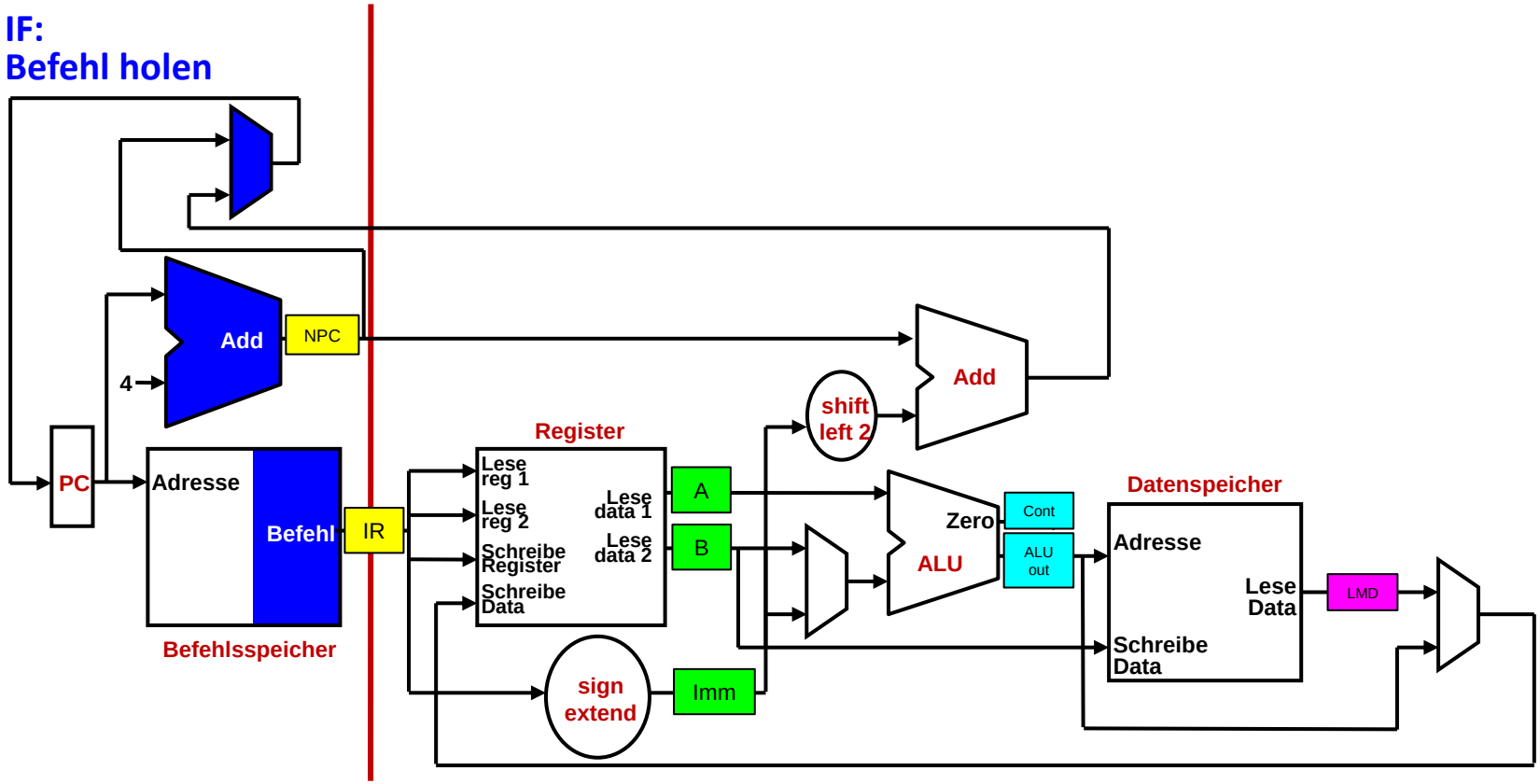
Instruction Set Architecture (ISA)

- MIPS-ISA wurde für Pipelining entworfen
  - Alle Befehle 32 Bit lang.
    - Einfach in einem Takt Befehl holen und PC zu inkrementieren
    - Vgl. x86: 1- bis 17-Byte Instruktionen
- Wenige und regelmäßige Befehlsformate
  - Dadurch können in der selben Stufe Befehle dekodiert und Register gelesen werden.
- Load-/Store-Adressierung
  - Kann in der 3. Stufe Adresse berechnen und in der 4. Stufe auf den Speicher zugreifen.

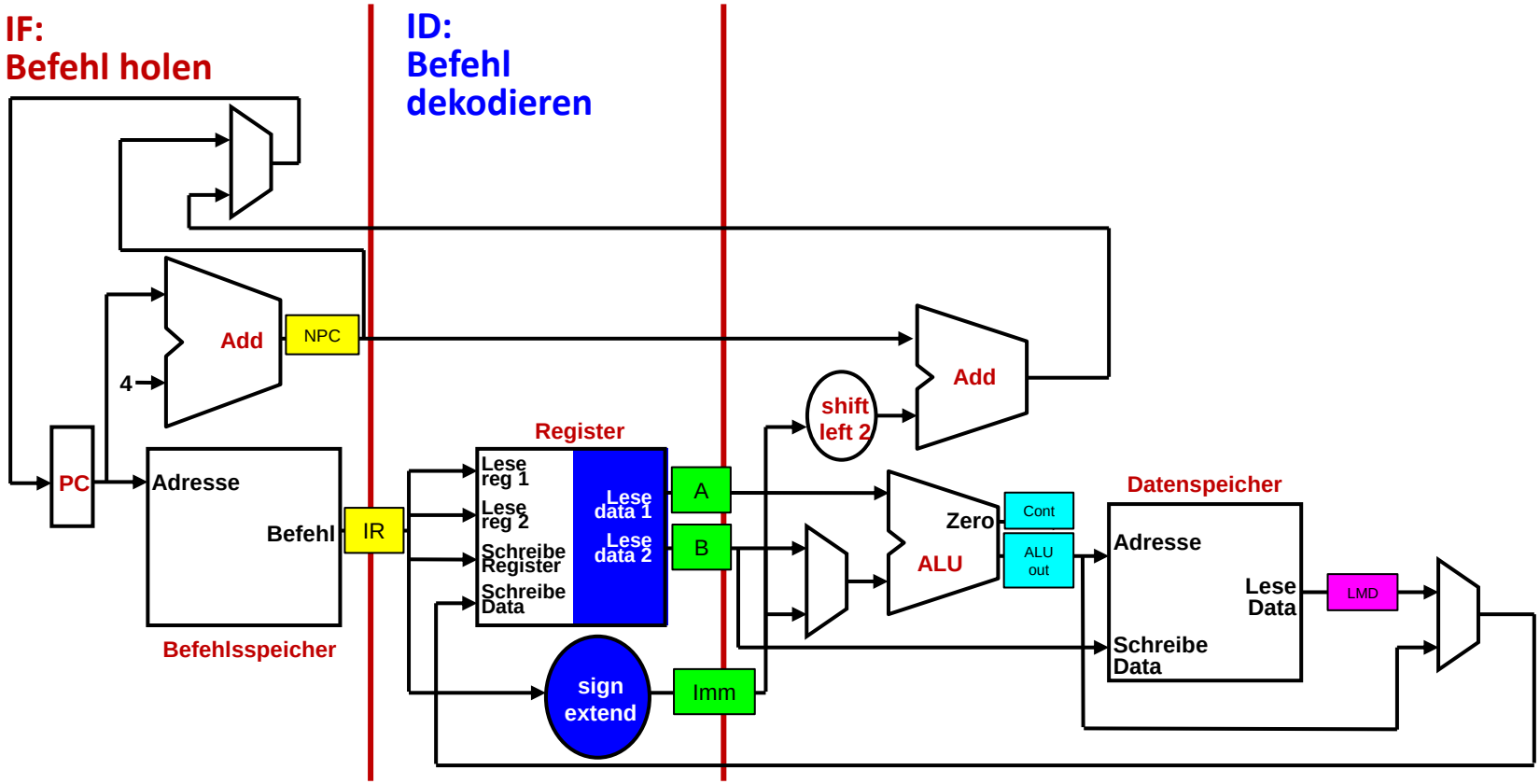
Grundidee: Unterteile Eintaktdatenpfad



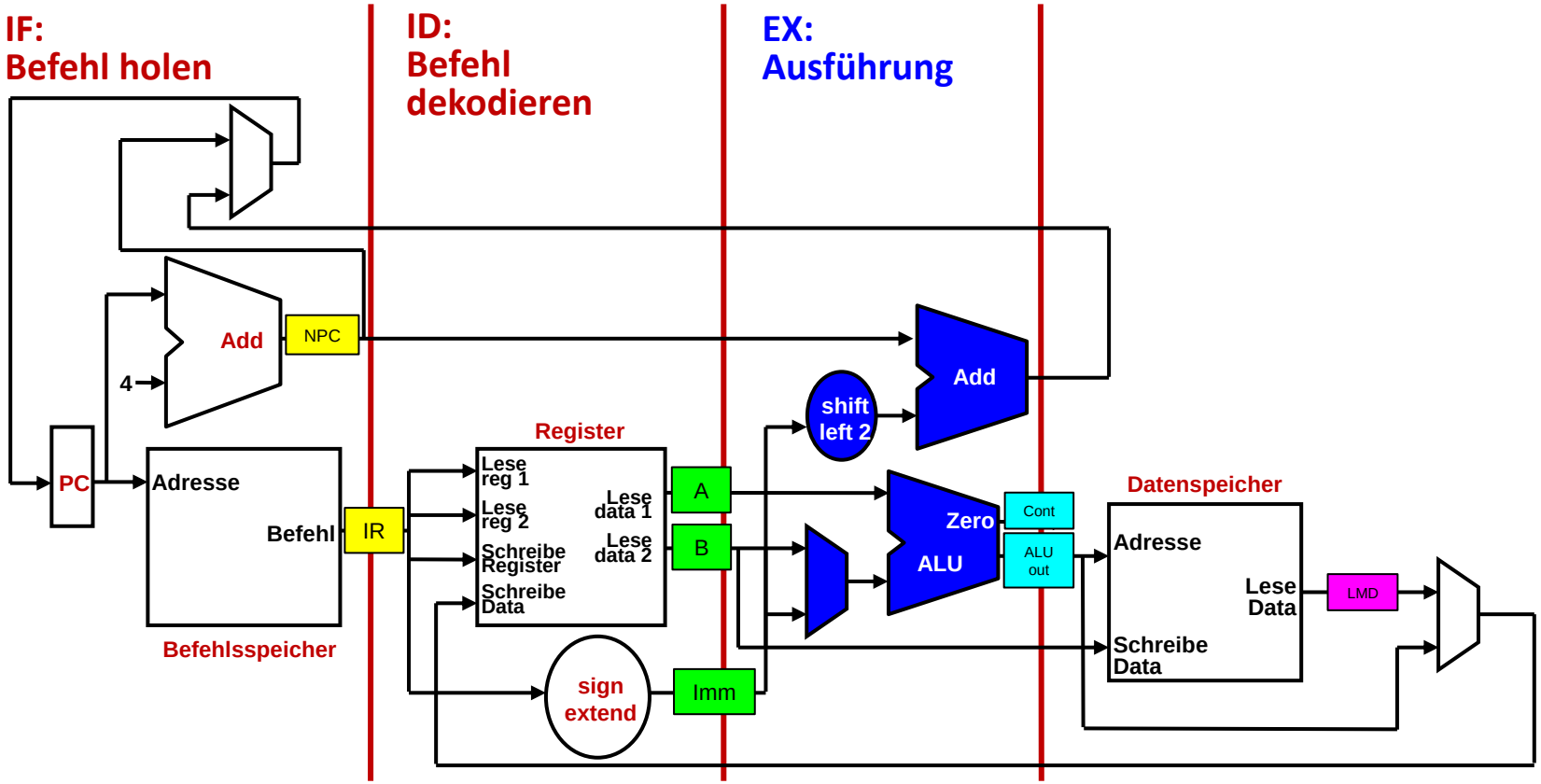
Grundidee: Unterteile Eintaktdatenpfad



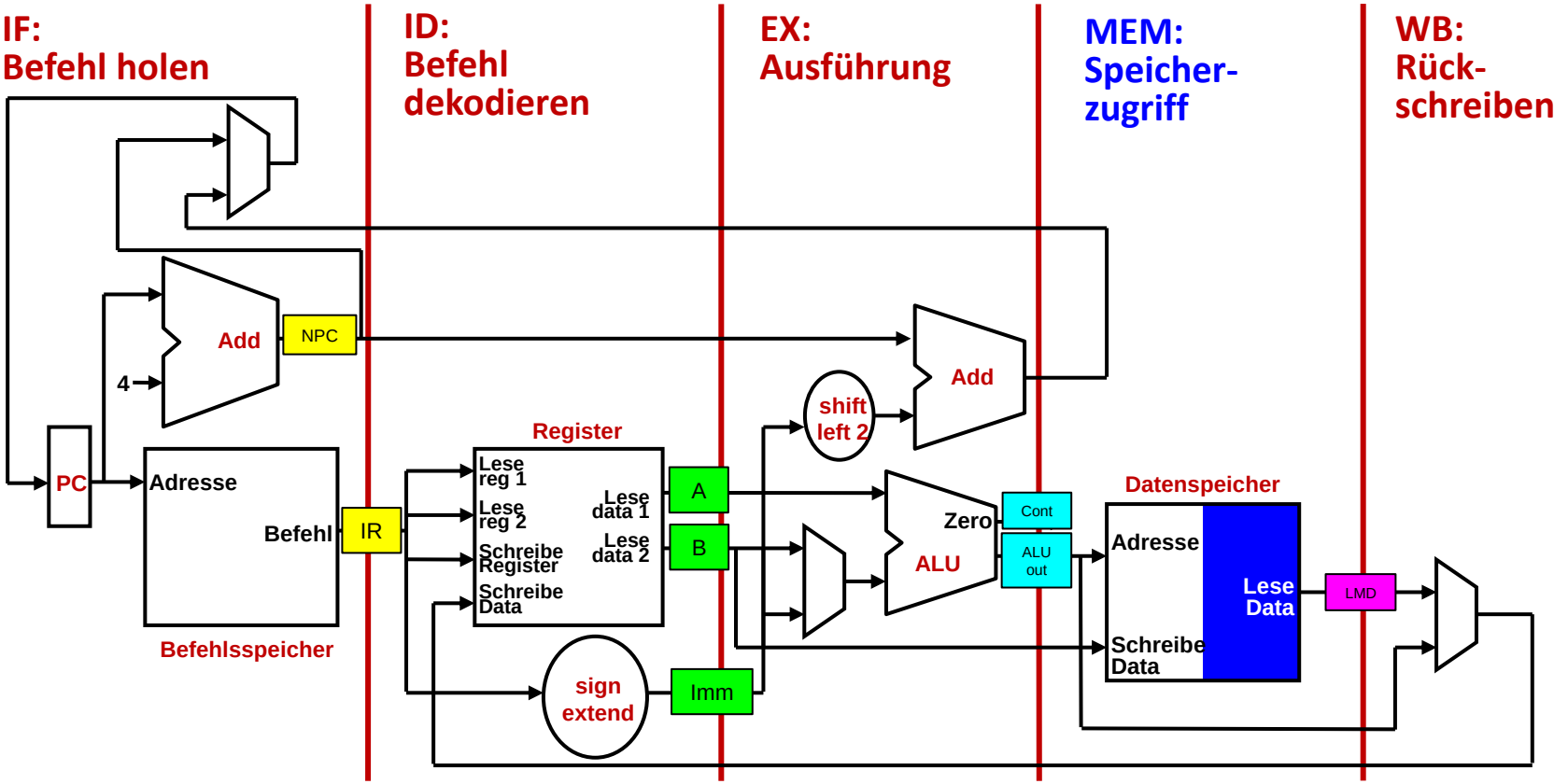
Grundidee: Unterteile Eintaktdatenpfad



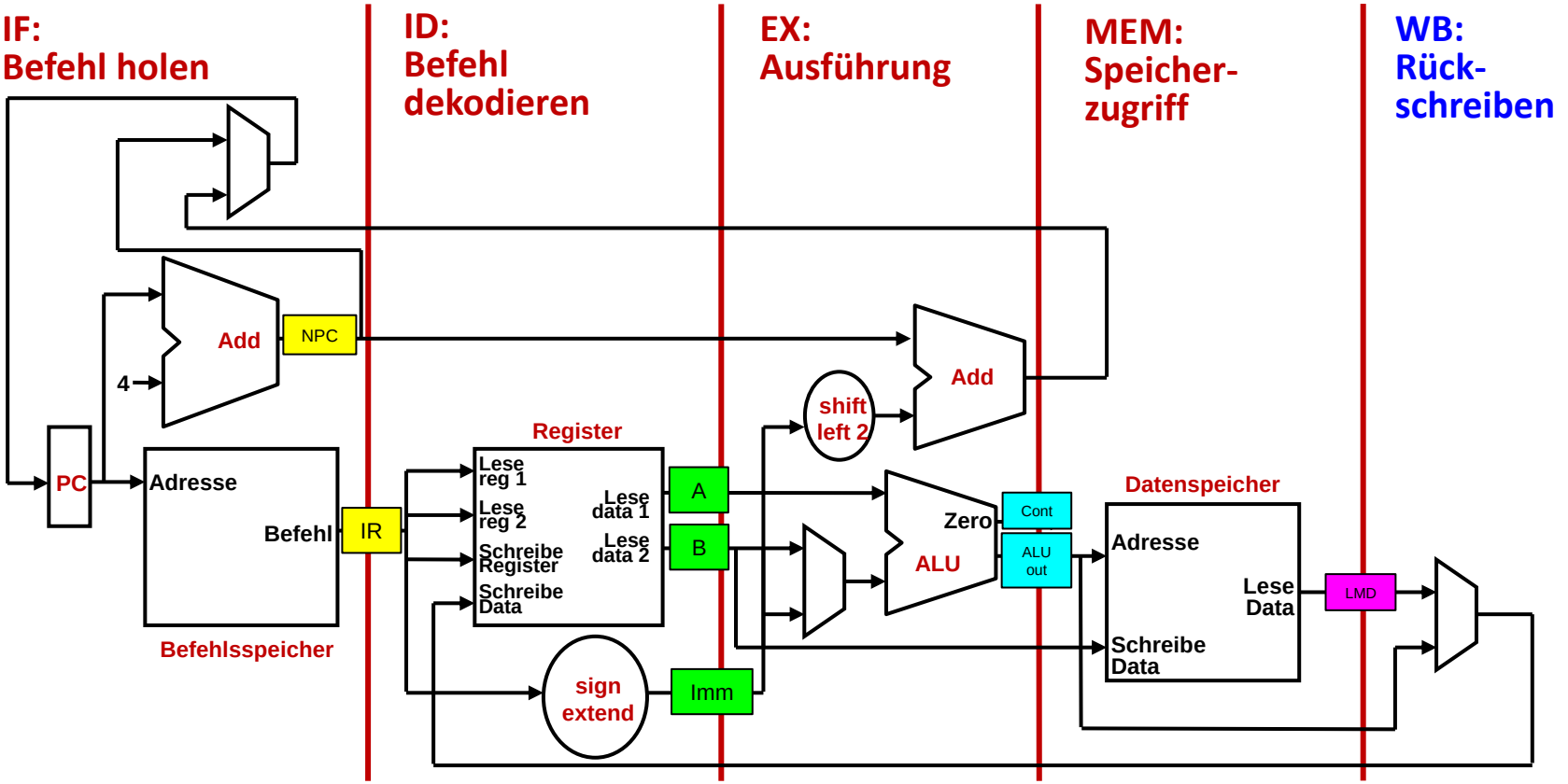
Grundidee: Unterteile Eintaktdatenpfad



Grundidee: Unterteile Eintaktdatenpfad



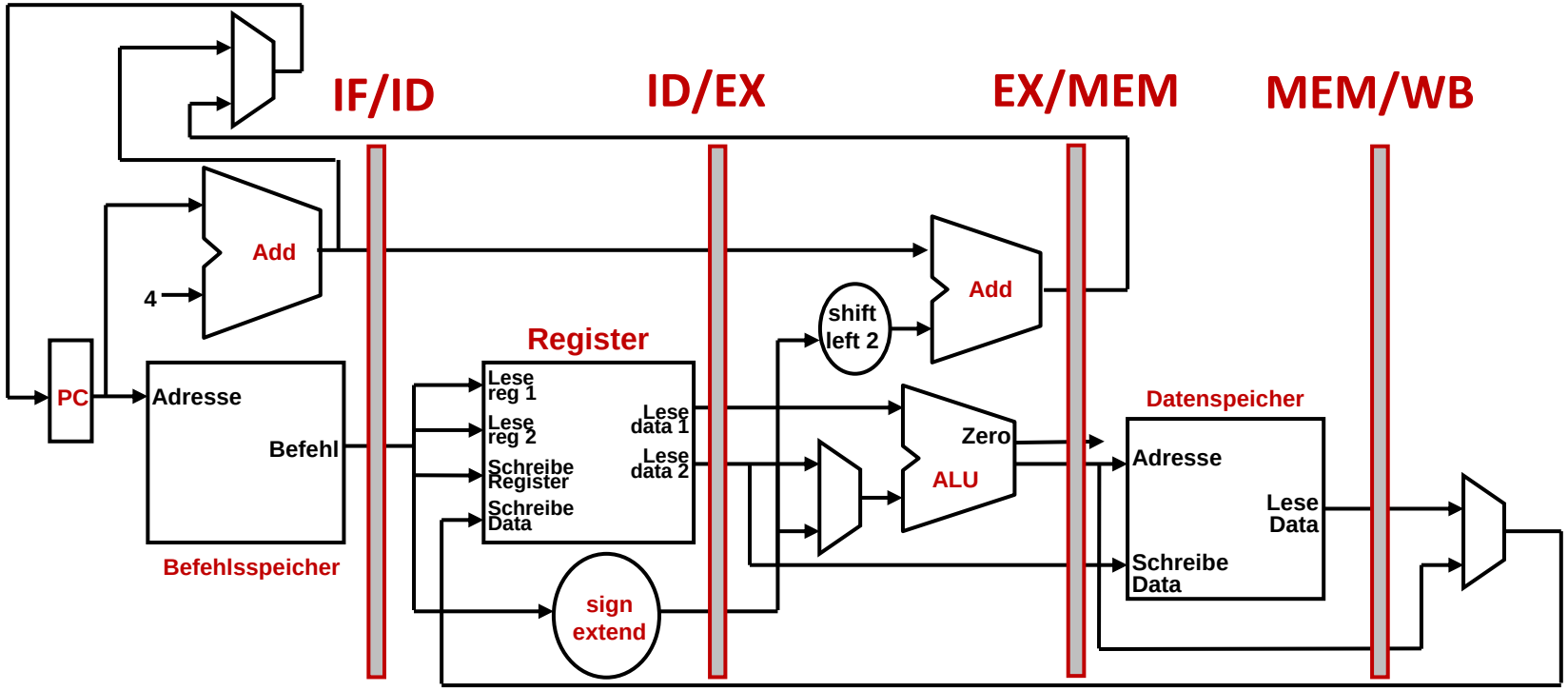
Grundidee: Unterteile Eintaktdatenpfad



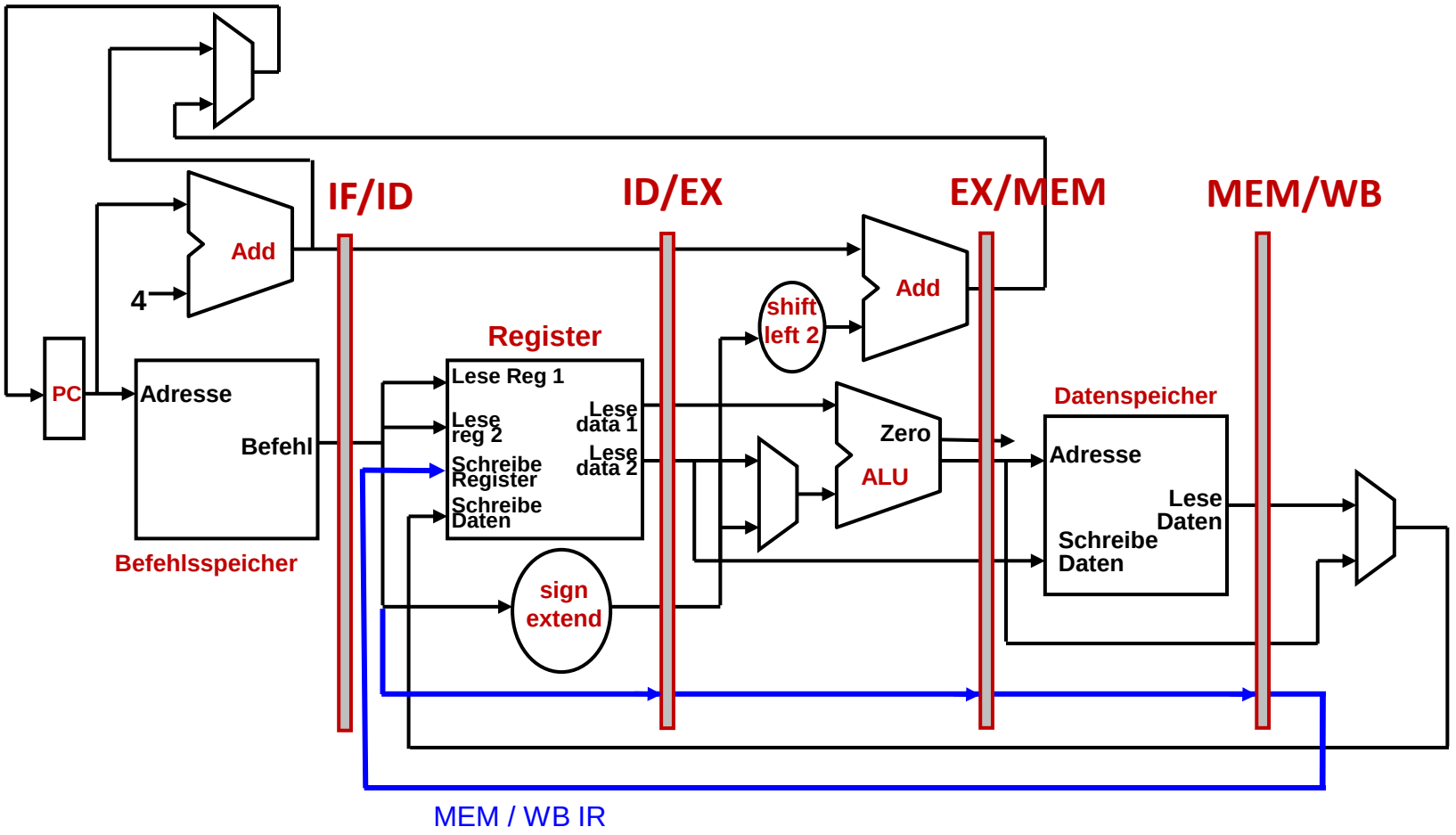
Datenpfad mit Pipelining

- **Es gibt einen Fehler im Design. Können Sie ihn finden? Welche Befehle werden fehlerhaft ausgeführt?**

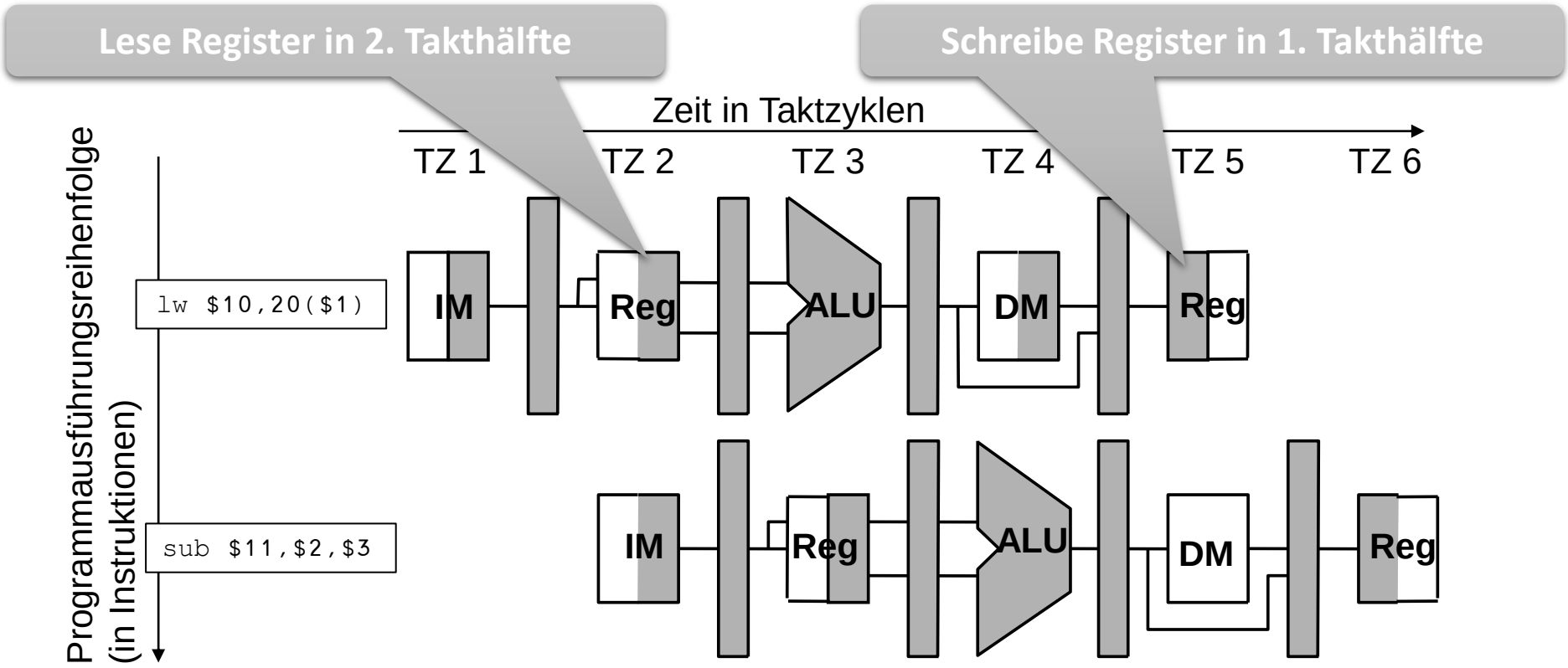
  Tipp: Welche Stufe schreibt in den Registersatz?



Datenpfad mit Pipelining



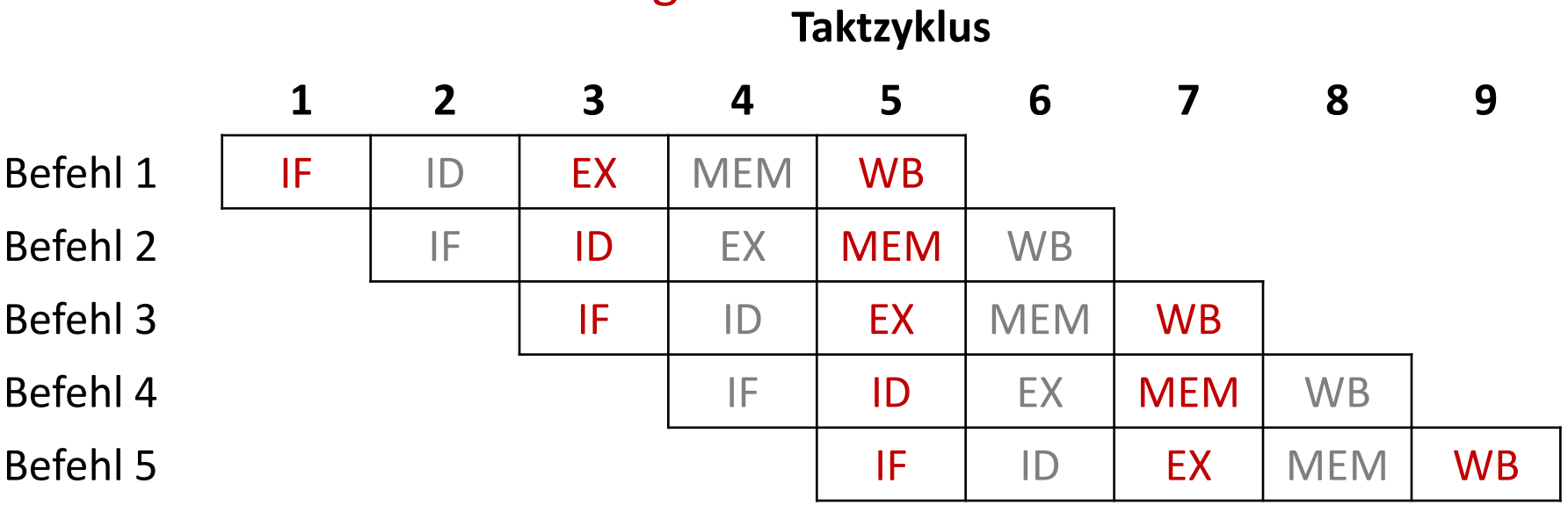
Graphische Darstellung von Pipelining / 1



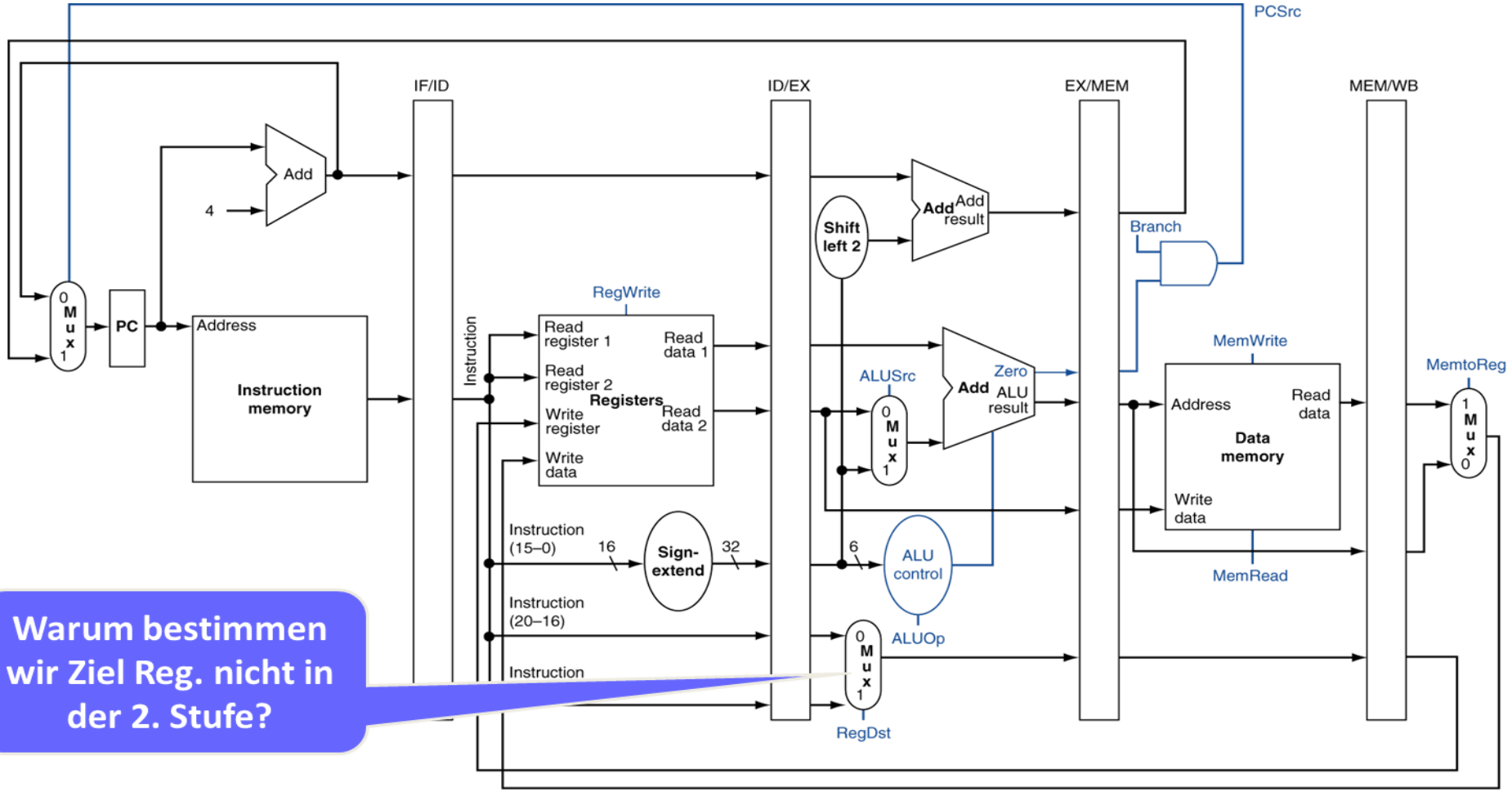
- Wie viele Takte werden zum Ausführen dieses Codes benötigt?
- Was macht die ALU im 4. Taktzyklus?

Graphische Darstellung von Pipelining / 2

- Traditionelle Darstellung:

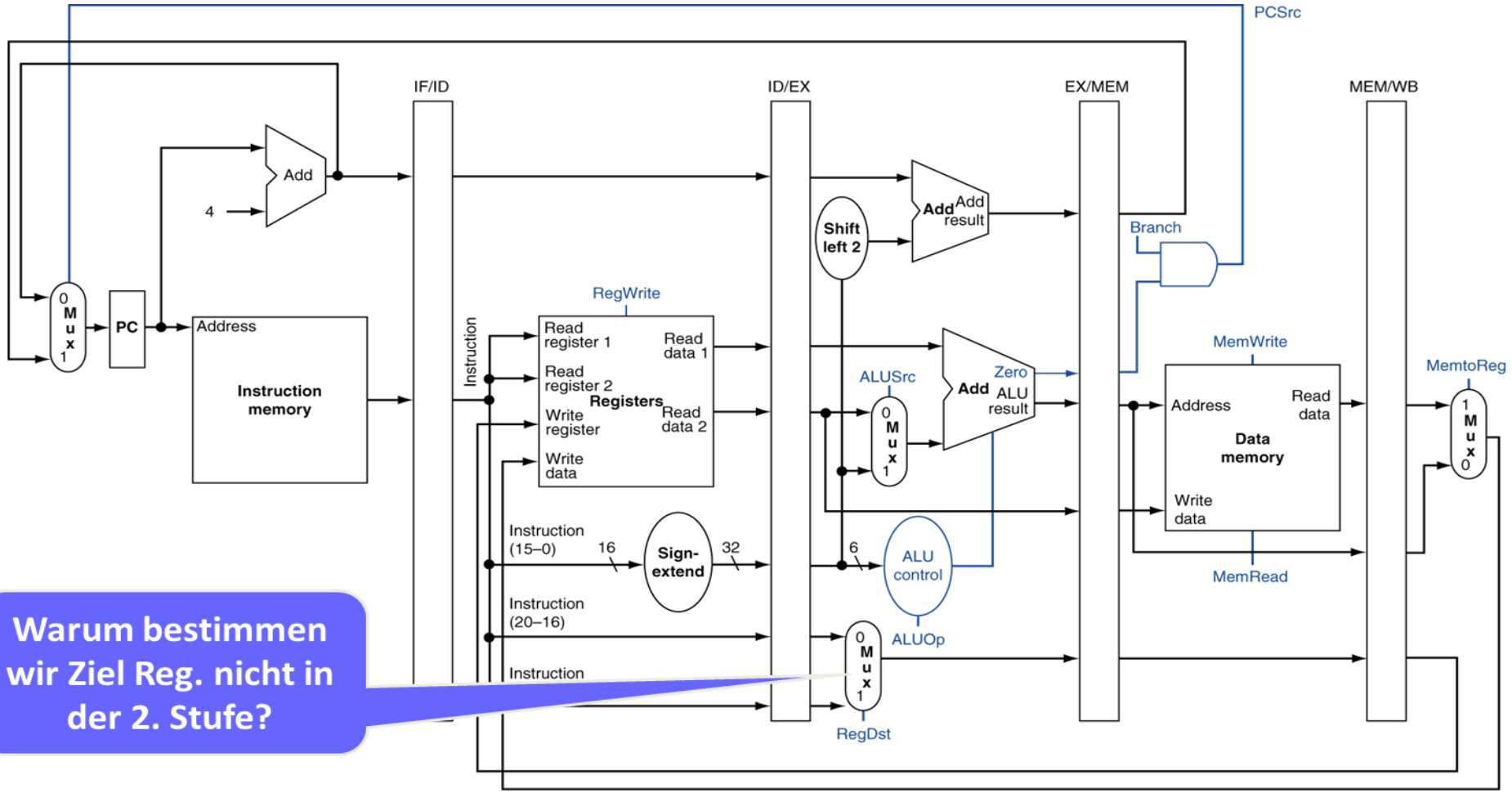


Steuerung der Pipeline



Warum bestimmen wir Ziel Reg. nicht in der 2. Stufe?

Steuerung der Pipeline 1



Warum bestimmen wir Ziel Reg. nicht in der 2. Stufe?

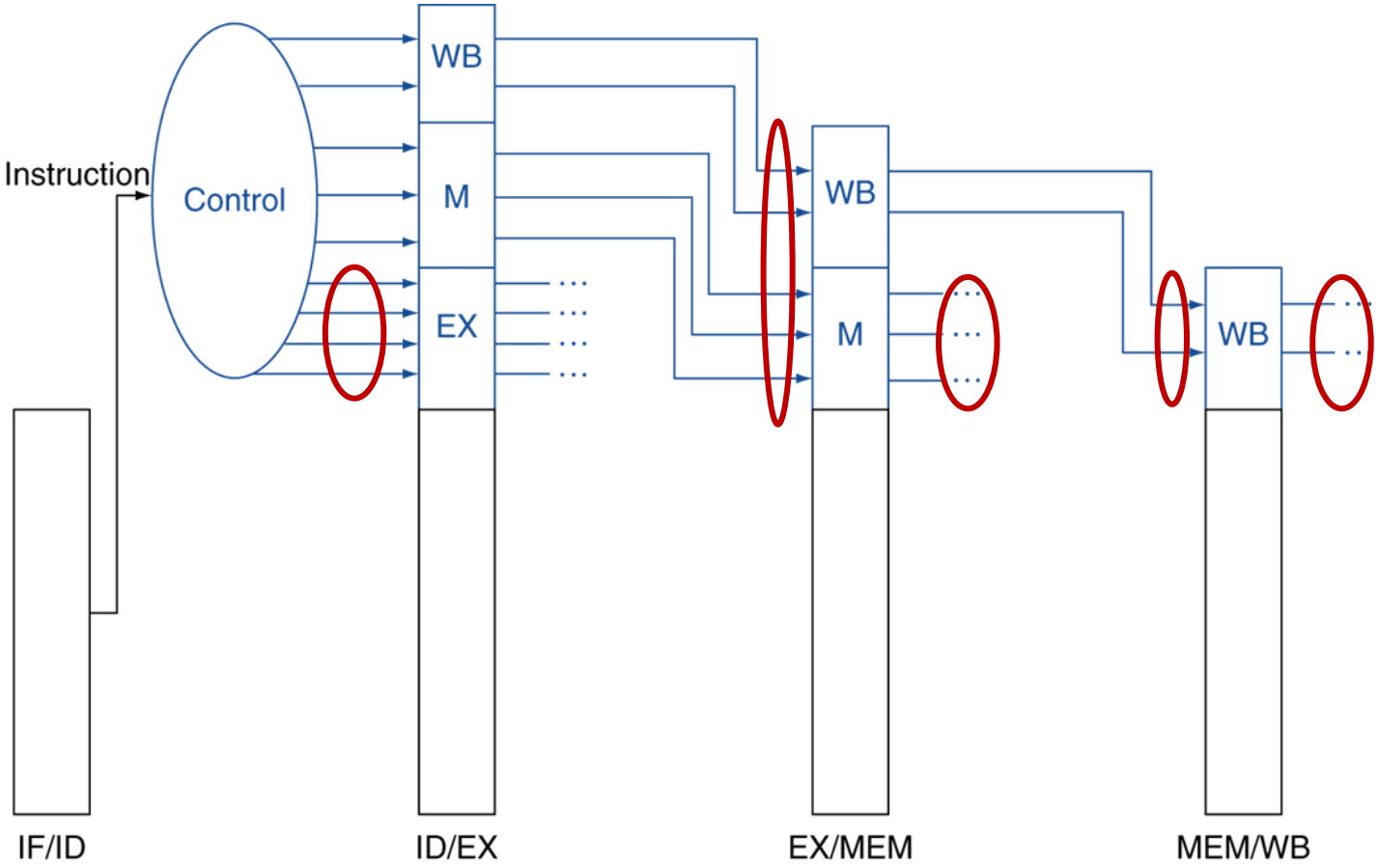
Steuerung der Pipeline 2

- Steuersignale sind identisch zu denen in der Eintakt-Implementierung
- Was muss in jeder Stufe gesteuert werden?
  - Befehl holen und PC Inkrementieren
  - Befehl dekodieren / Register lesen
  - Befehl ausführen oder Adresse berechnen
  - Speicherzugriff
  - Ergebnis Rückschreiben

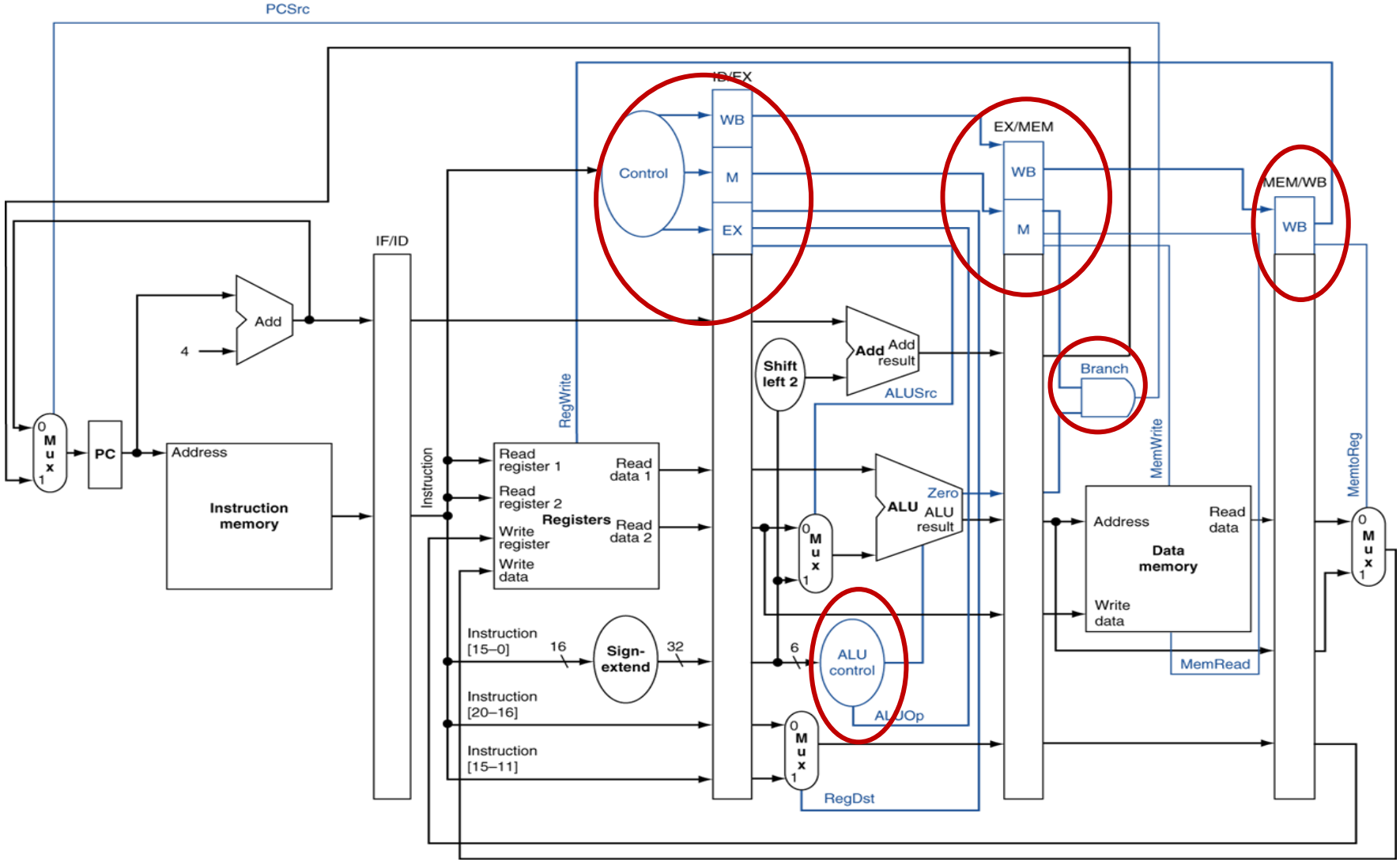
Befehl	Ausführung / Adresse berechnen				Speicherzugriff			Rückschreiben	
	Reg Dst	ALU Op1	ALU OP0	ALU Src	Branch	Mem Read	Mem Write	Reg Write	Mem to Reg
R-Format	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

Steuerung der Pipeline 3

- Durchschalten der Steuersignale durch die Stufen wie Daten



Steuerung der Pipeline 4



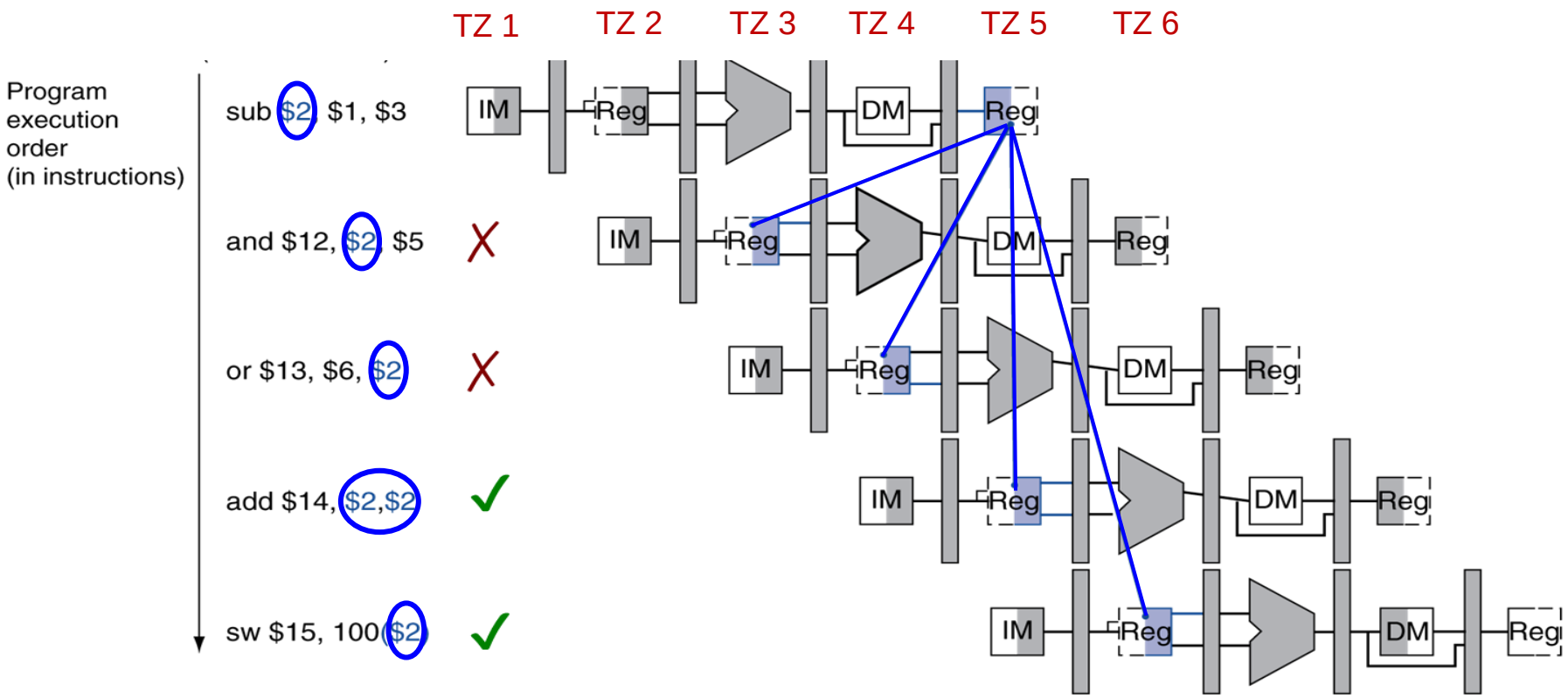
Pipeline Konflikte / Hazards

- Konflikte (Hemmnisse) (*Hazards*):
  - Situationen, die das Ausführen des nächsten Befehls im nächsten Takt verhindert.

- Strukturkonflikte (*structural hazards*)
  - benötigte Ressource ist belegt
- Datenkonflikte (*data hazards*)
  - auf das Ergebnis eines vorherigen Befehls muss gewartet werden
- Steuerkonflikte (*control hazards*)
  - Falls verzweigt (*branch*) wird, befinden sich bereits andere Befehle in der Pipeline

Datenkonflikte

- Ergebnis wird im Takt **i** geschrieben, soll aber bereits im Taktzyklus **i-2** und **i-1** gelesen werden



Software-Lösung

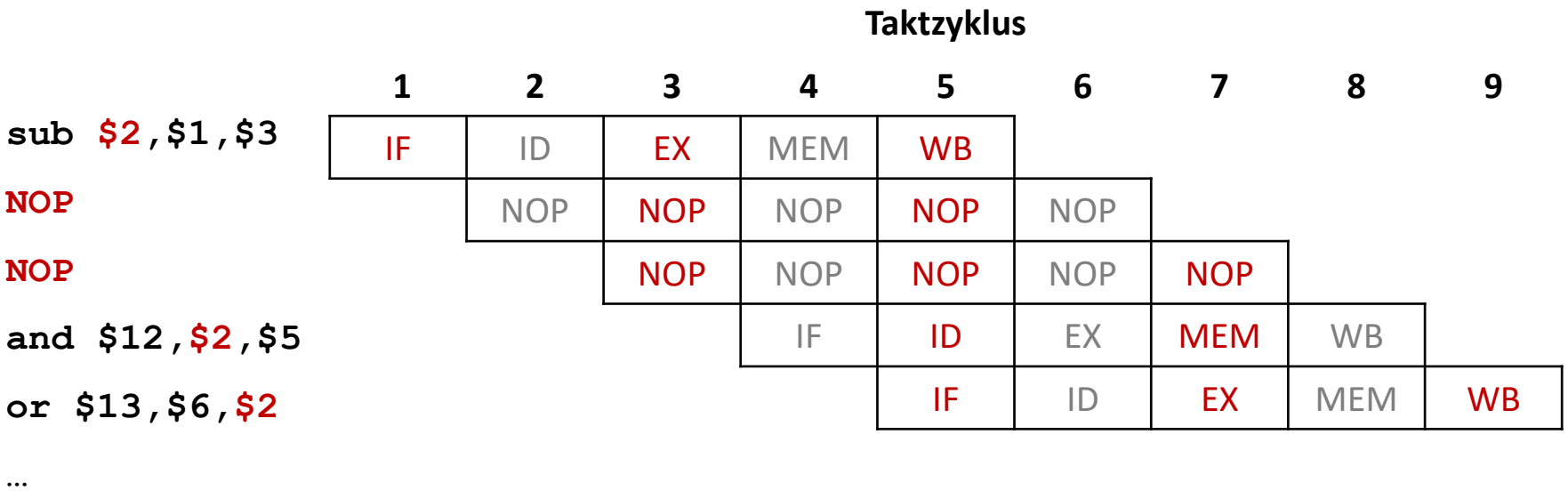
- Compiler muss garantieren, dass keine Konflikte auftreten
- Wo müssen wir „No Operations“ (NOPs) einfügen?

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

NOP
NOP

- Aber das verlangsamt die Ausführung extrem!
  - Kann manchmal durch Veränderung der Befehlsfolge vermieden werden.

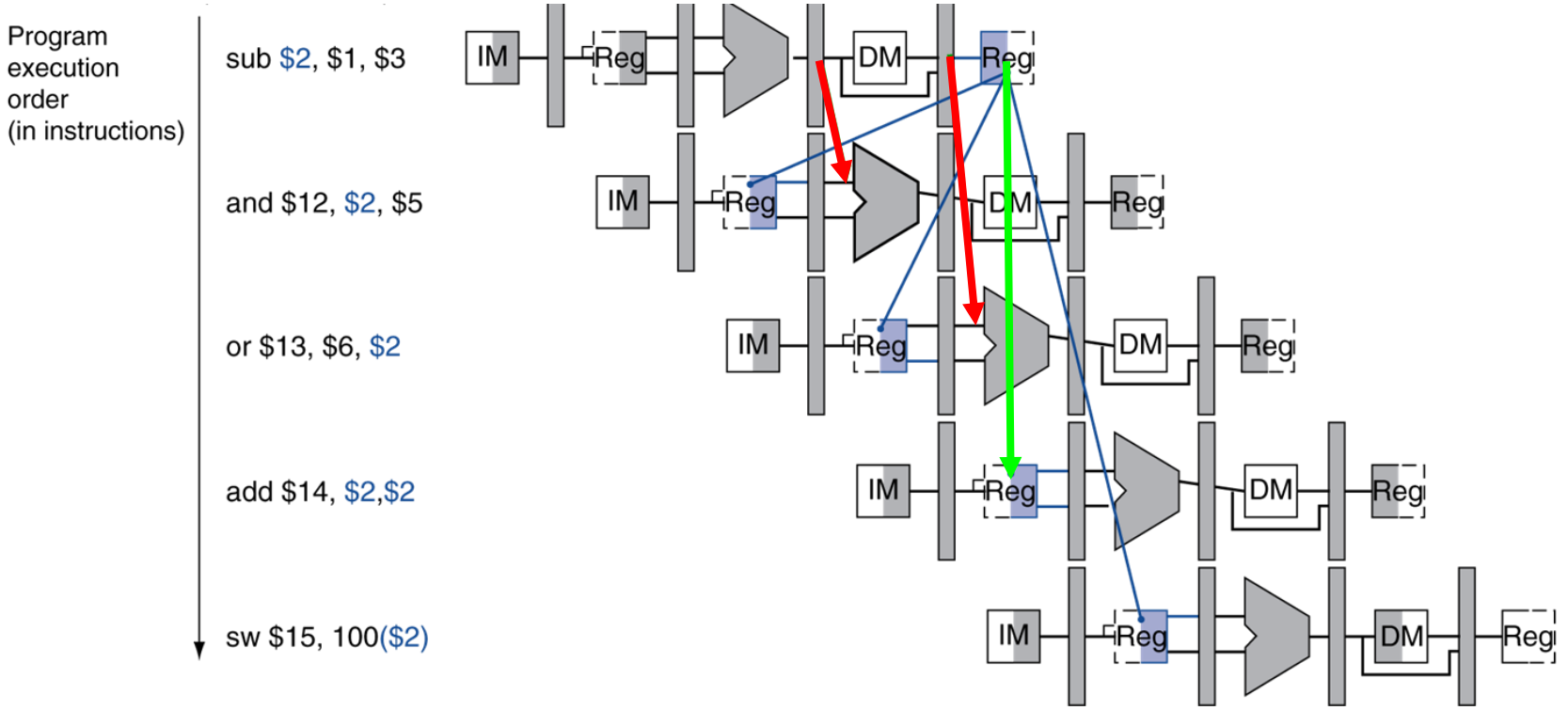
Konfliktlösung durch NOPs



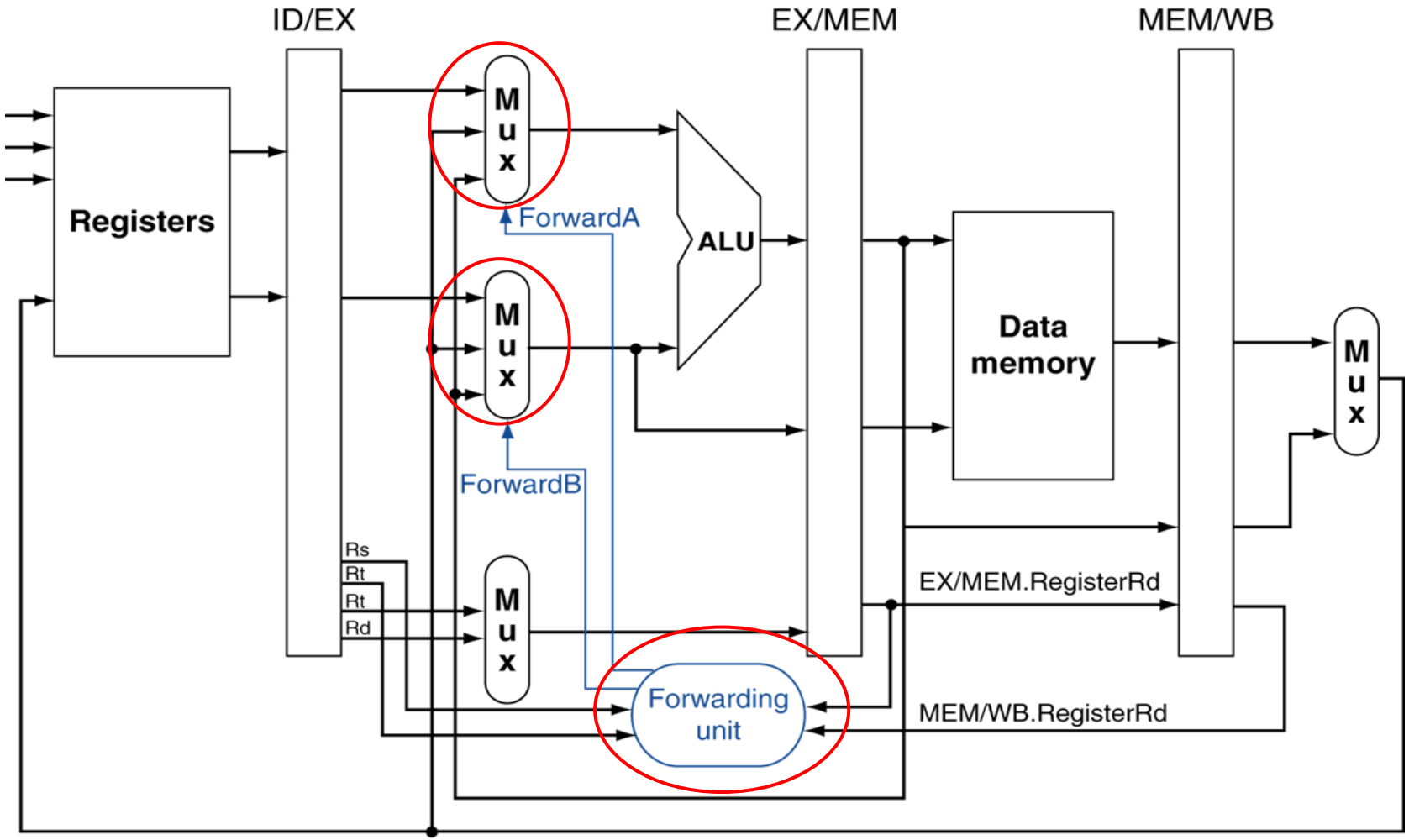
...

Hardware-Lösung: Forwarding / Bypassing

- Ergebnisse werden sofort weitergeleitet, ohne darauf zu warten bis sie in den Registersatz zurückgeschrieben wurden



Forwarding Einheit



b. With forwarding

Bedingungen zum Erkennen von Konflikten

- Eine Notation, die die Felder der Pipeline-Register benennt, ermöglicht eine genauere Notation von Abhängigkeiten: `ID/EX.Rs`

- EX/MEM-Konflikt Detektion und Control:

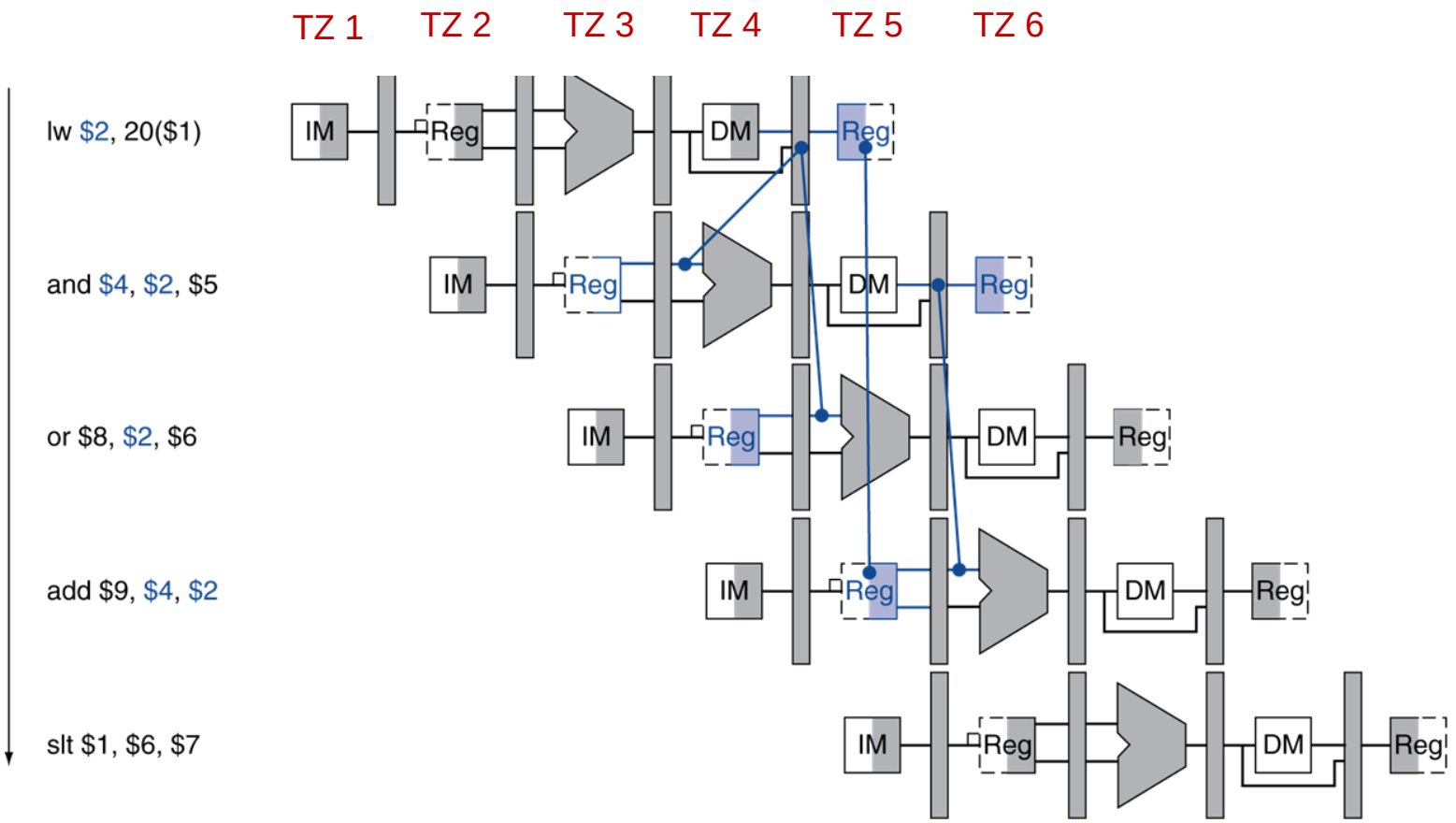
```
if (EX/MEM.RegWrite && EX/MEM.Rd≠0){
    if (EX/MEM.Rd == ID/EX.Rs) ForwardA = 10;
    if (EX/MEM.Rd == ID/EX.Rt) ForwardB = 10;
}
```

- MEM/WB-Konflikt Detektion und Control :

```
if (MEM/WB.RegWrite && MEM/WB.Rd≠0){
    if (EX/MEM.Rd≠ID/EX.Rs && MEM/WB.Rd==ID/EX.Rs)
        ForwardA = 01;
    if (EX/MEM.Rd≠ID/EX.Rt && MEM/WB.Rd==ID/EX.Rt)
        ForwardB = 01;
}
```

Load-Use Konflikt

- Kann nicht immer “forwarden”: `lw` kann Konflikt auslösen.
  - Befehl nutzt Register, dass durch vorherigen Ladebefehl geschrieben wird



Abhandeln von Load-Use Konflikten

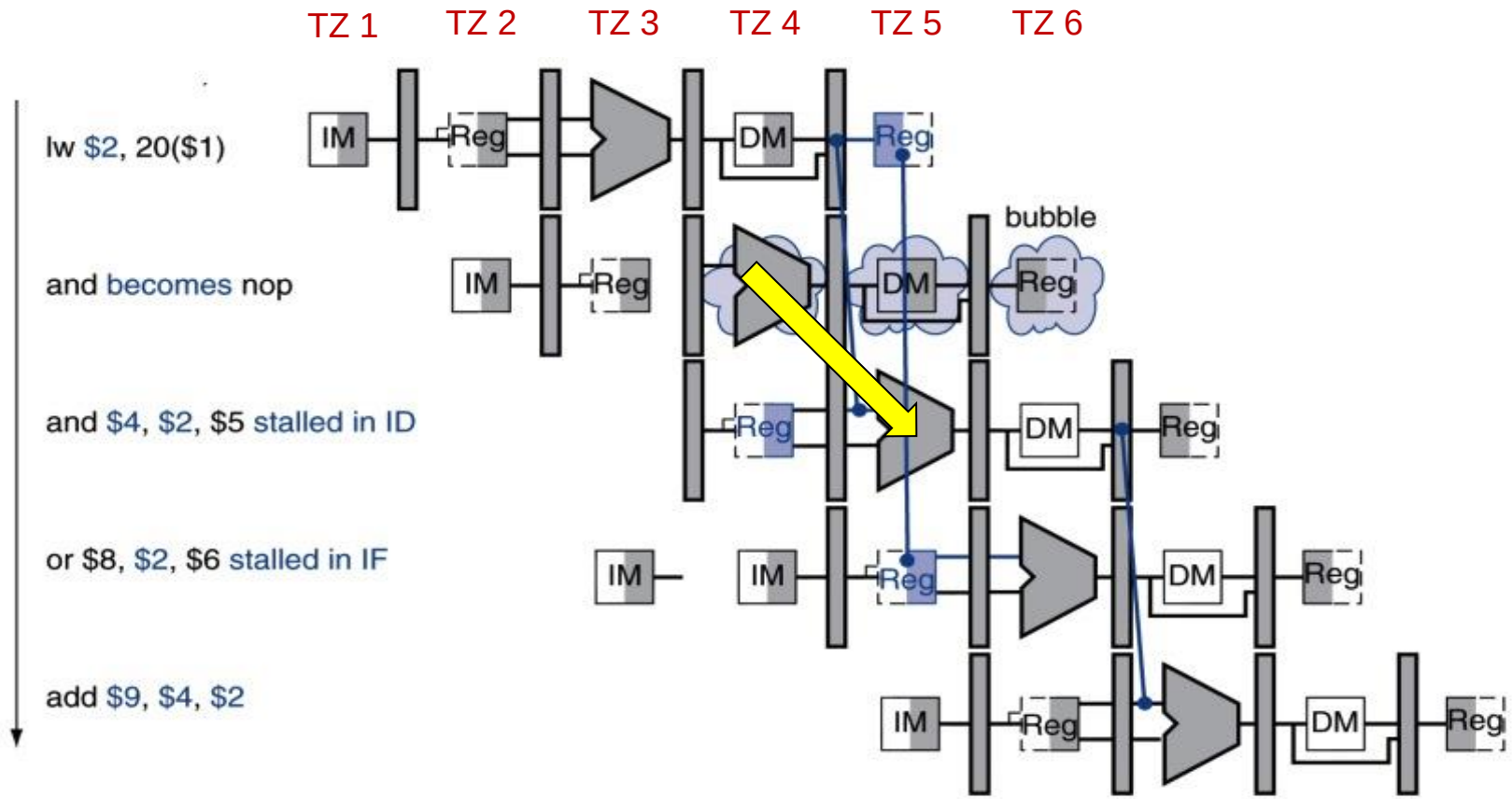
- Man benötigt
  - Einheit zum Erkennen von solchen Konflikten (Hazard Detection Unit)
  - Möglichkeit die Pipeline anzuhalten

- Erkennen solcher Konflikte:

```
ID/EX.MemRead &
     (ID/EX.Rt=IF/ID.Rs |
      ID/EX.Rt=IF/ID.Rt)
```

- Anhalten der Pipeline:
  - Befehle nach dem Ladebefehl in der selben Stufe halten
    - Write-Signalen zu PC- und IF/ID-Pipeline-Register hinzufügen
  - NOP-Befehl einfügen → Einfügen einer „Blase“ (bubble)

Verzögerung der Pipeline



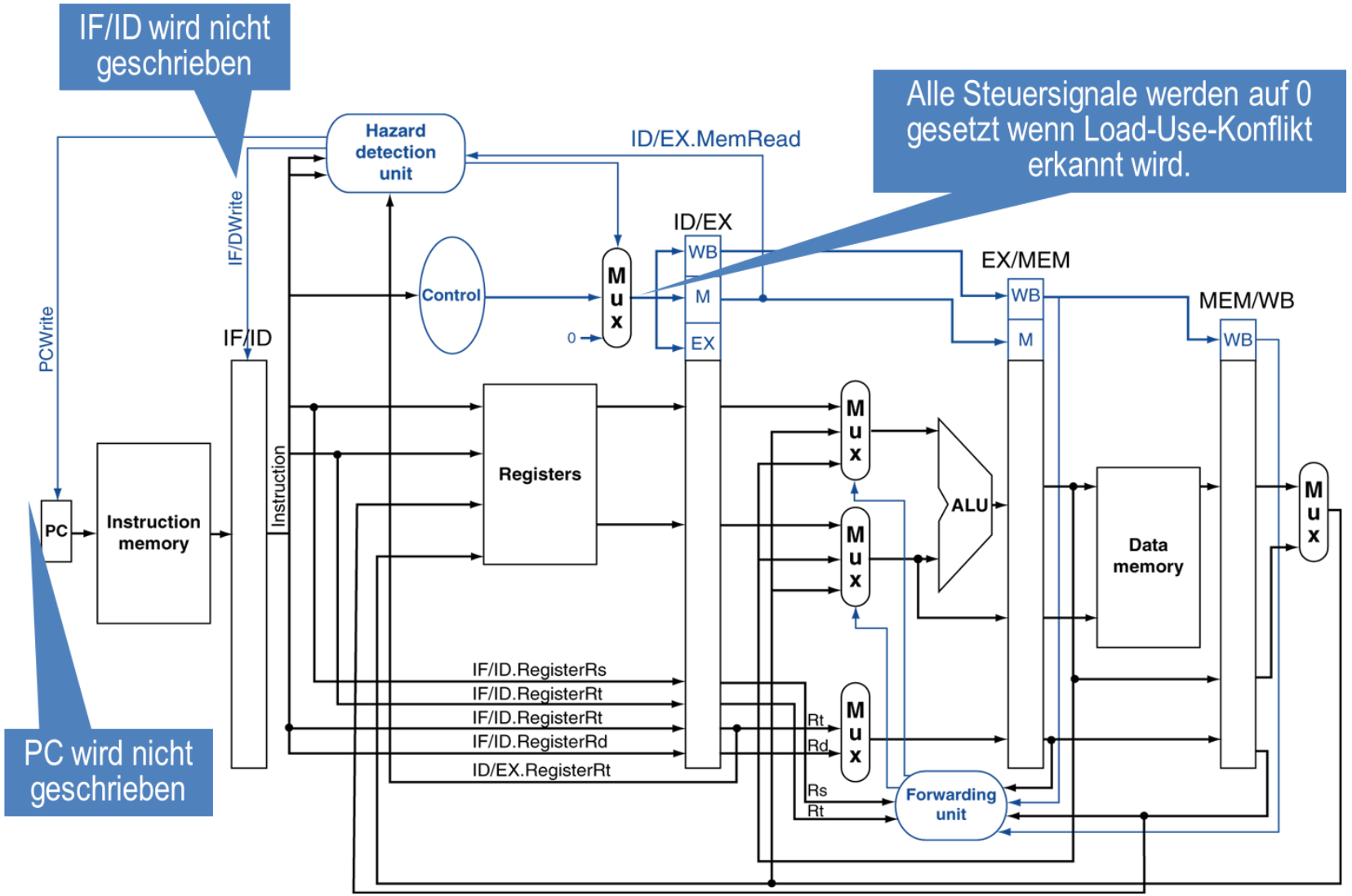
Alternative Darstellung

Zeit (Taktzyklen) →

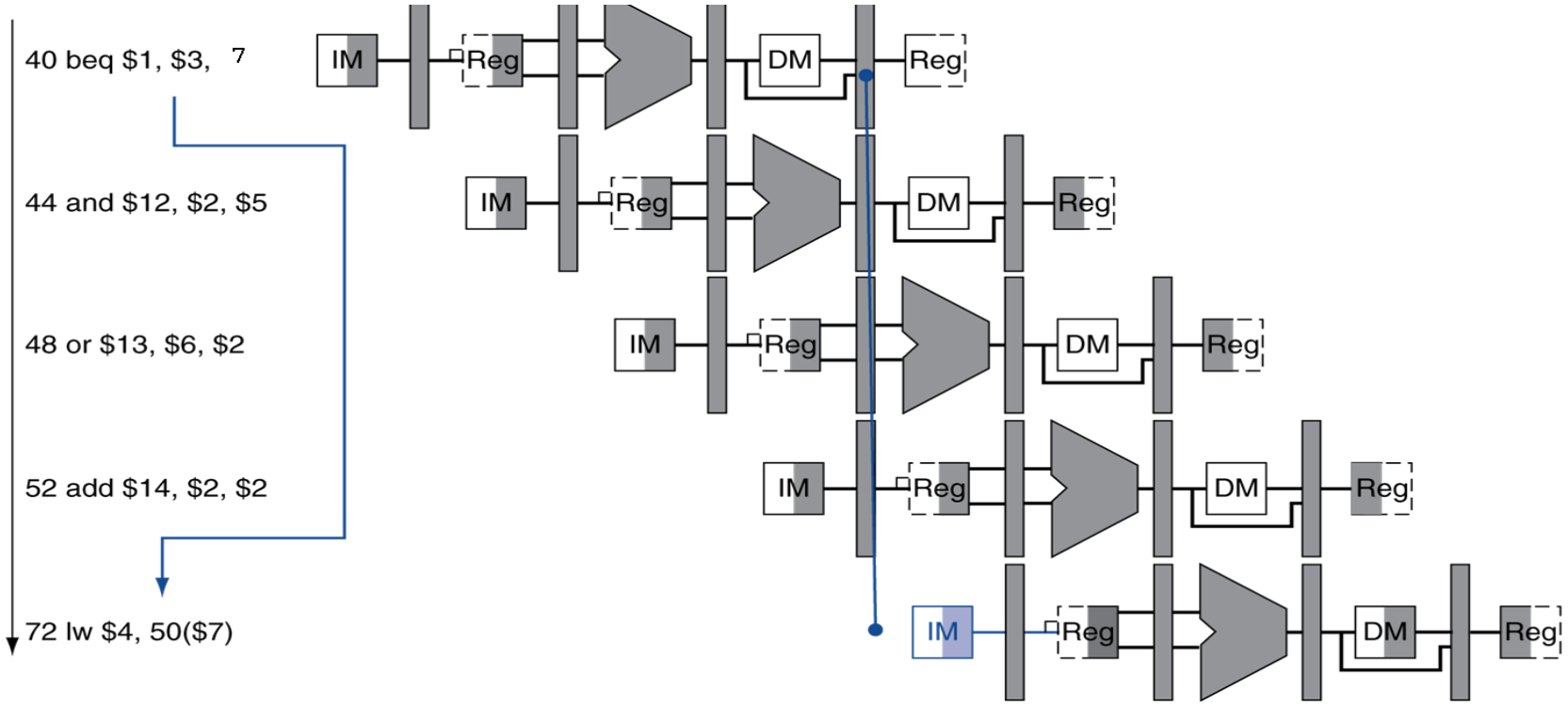
	TZ 1	TZ 2	TZ 3	TZ 4	TZ 5	TZ 6	TZ 7	TZ 8	TZ 9	TZ 10
`lw $2,20($1)`	IF	ID	EX	MEM	WB					
`and $4,$2,$5`		IF	ID		EX	MEM	WB			
`or $8,$2,$6`			IF		ID	EX	MEM	WB		
`add $9,$4,$2`					IF	ID	EX	MEM	WB	

Bubble

Datenpfad mit Hazard-Detection-Einheit

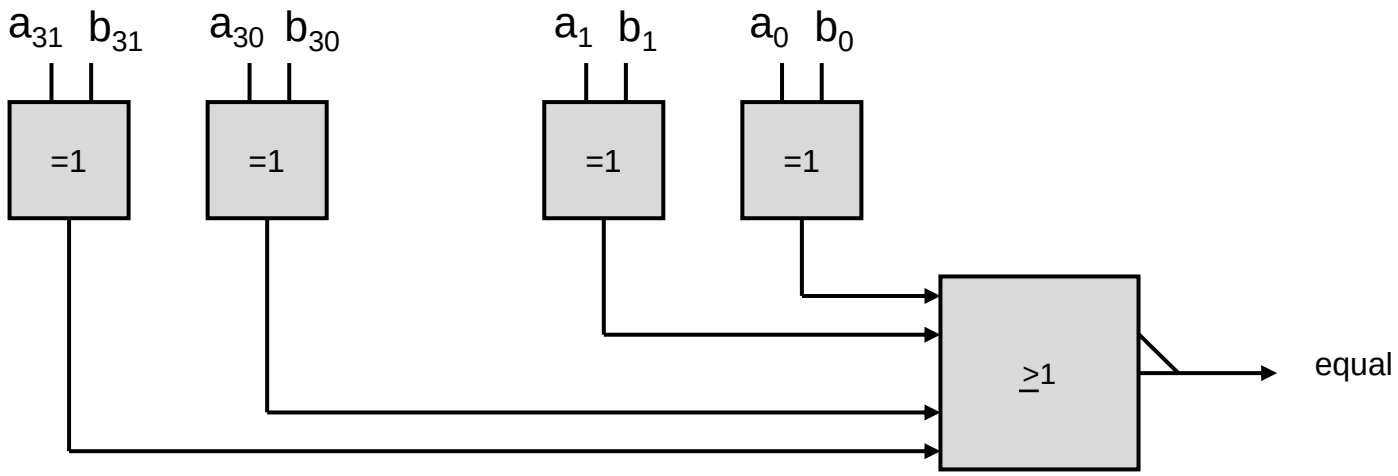


Steuerkonflikte / Verzweigungskonflikte

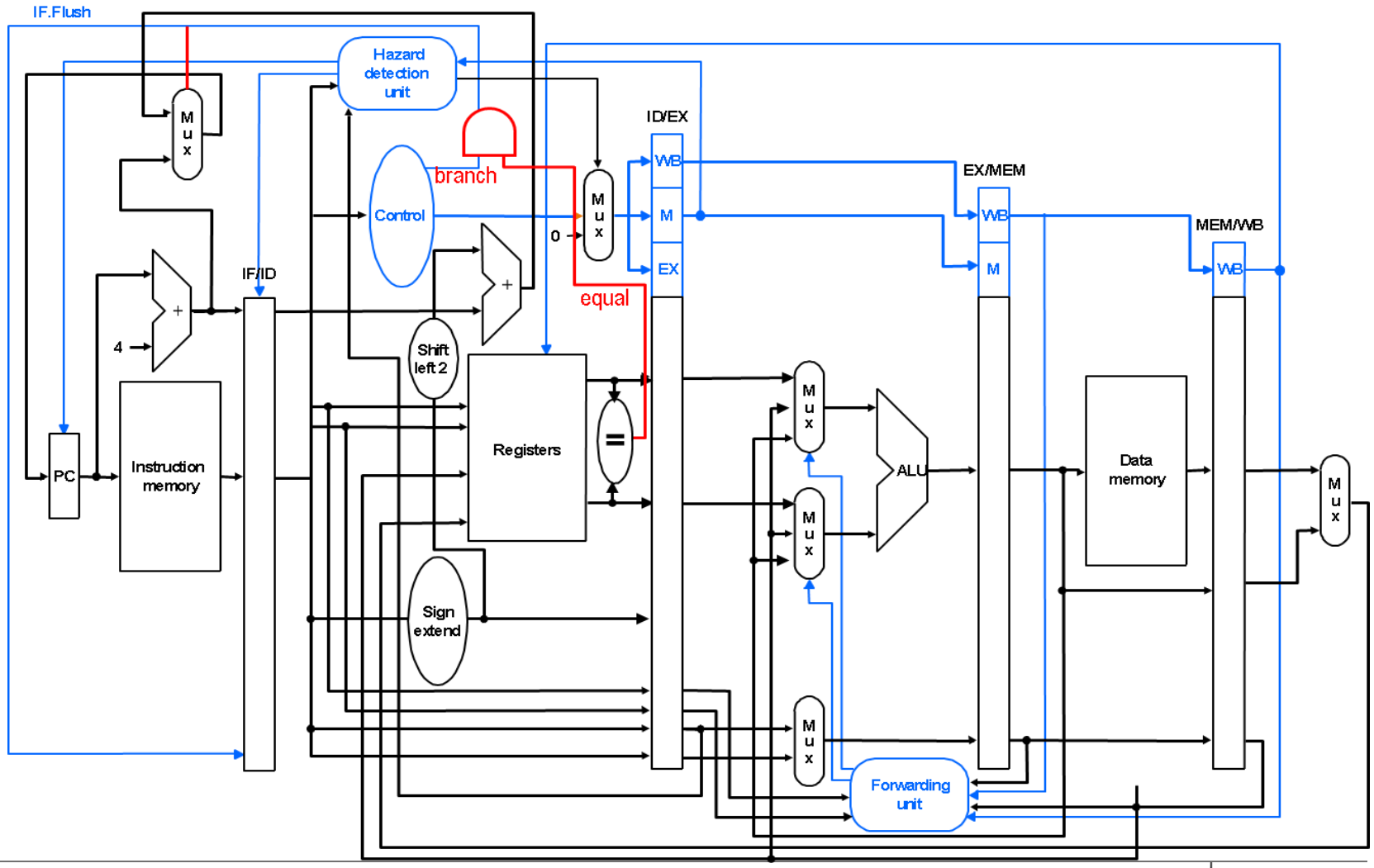


Lösungen für Steuerkonflikte 1

- Pipeline anhalten, wenn eine Verzweigung auftritt
  - 3 Warte-Zyklen bei jeder Verzweigung
- Sprungausführung in frühere Stufe verlegen
  - zur EX-Stufe ist einfach (s. Folie 19)
  - beim MIPS auch zur ID-Stufe möglich
    - MIPS unterstützt nur einfache Verzweigungen (beq,bne,bltz,…)



Verzweigen in ID-Stufe und Annahme, dass Sprung nicht ausgeführt wird



Lösungen für Steuerkonflikte 2

- Verzweigung in ID-Stufe verlagern und
  - Annehmen, dass nicht verzweigt wird
    - Lösche (flush) nächsten Befehl, falls Annahme falsch
    - Ist Sprungbedingung falsch, so entsteht kein Verlust
    - Wird jedoch gesprungen, so geht 1 Zyklus verloren

- Branch Delay Slot (Verzögerter Sprung) hinzufügen
  - Befehl nach der Verzweigung wird immer ausgeführt
  - Compiler muss Delay Slot mit sinnvollem Befehl oder NOP füllen
  - Lösungsansatz wurde im ersten MIPS-Prozessor verwendet
  - Delay Slot existiert immer noch beim MIPS. Warum?

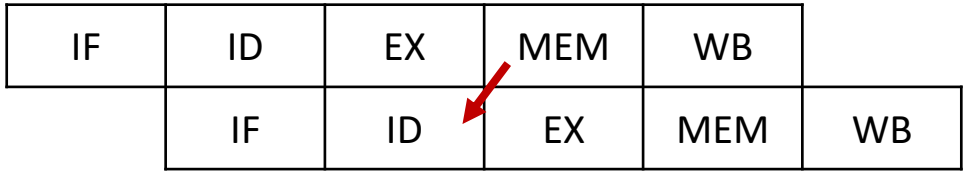
- Dynamische Sprungvorhersage (dynamic branch prediction)

Der Teufel steckt im Detail

- **Forwarding** bei Befehlen mit nur einem (`lw`,`addi`,...) oder keine (`j`,...) Quellregistern

- Um Verzweigungen in ID-Stufe zu verlagern
  - Zusätzliche Forwardlogik in ID-Stufe nötig
  - Verzweigungen können keine Register verwenden, welches durch einen vorherigen ALU- oder Ladebefehl geschrieben wurde.
    - Solche Datenkonflikte erkennen und Pipeline anhalten

```
addi $t0,$t0,1
beq $t0,$t1,-4
```



Dynamische Sprungvorhersage

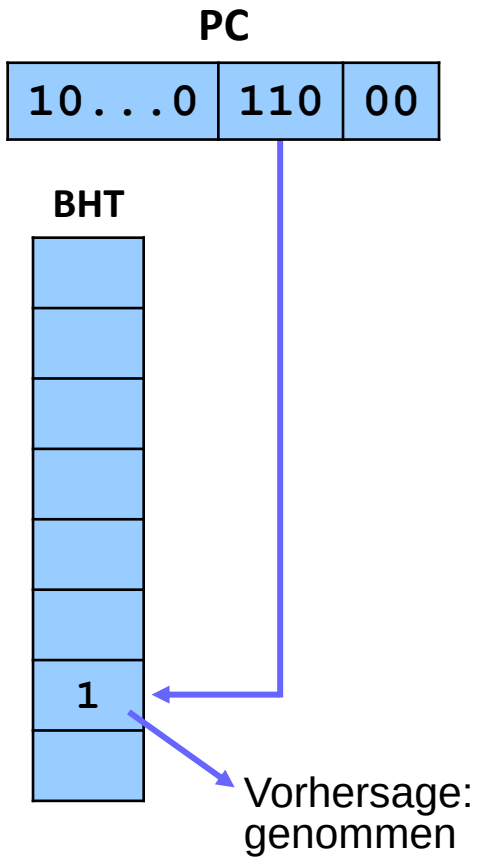
- Moderne Prozessoren haben tiefere Pipelines
  - Z. B. Intel Core i7: 16 Stufen
  - Sprungverzögerung (branch penalty) nimmt zu
  - Unmöglich alle delay slots mit nützlichen Befehlen zu füllen

- Dynamische Sprungvorhersage (dynamic branch prediction)
  - Vorhersage von Sprung an Hand seines Verlaufs
    - Falls voriges Mal genommen (taken), nehme an wieder genommen
    - Falls nicht genommen (not taken), nehme an wieder nicht

Dynamische Sprungvorhersage

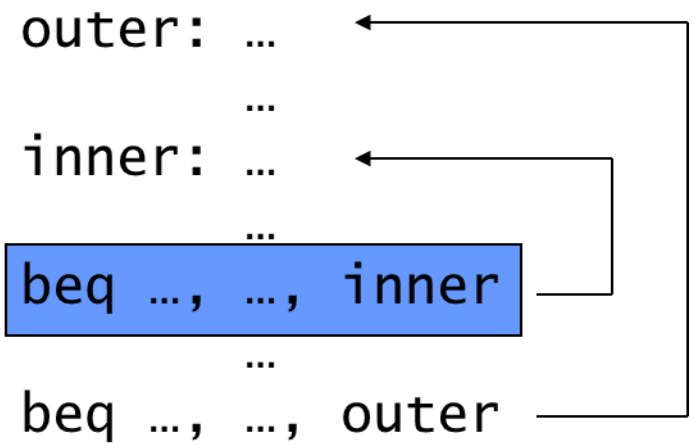
- Sprungvorhersagepuffer (branch prediction buffer) / Sprungverlaufstabelle (branch history table [BHT])
  - Indiziert durch unteren Teil der Befehlsadresse (PC)
  - Speichert die vorherigen Sprungergebnisse (genommen/nicht genommen [taken/not taken])

- Vorhersage eines Sprungs:
  - Tabelle nachschlagen
    - Annahme: Sprungergebnis bleibt gleich
  - Nächster Befehl holen aus vorhergesagter Richtung
  - Wenn Vorhersage falsch, lösche (flush) Pipeline und invertiere Vorhersagebit



Nachteil 1-Bit-Sprungvorhersage

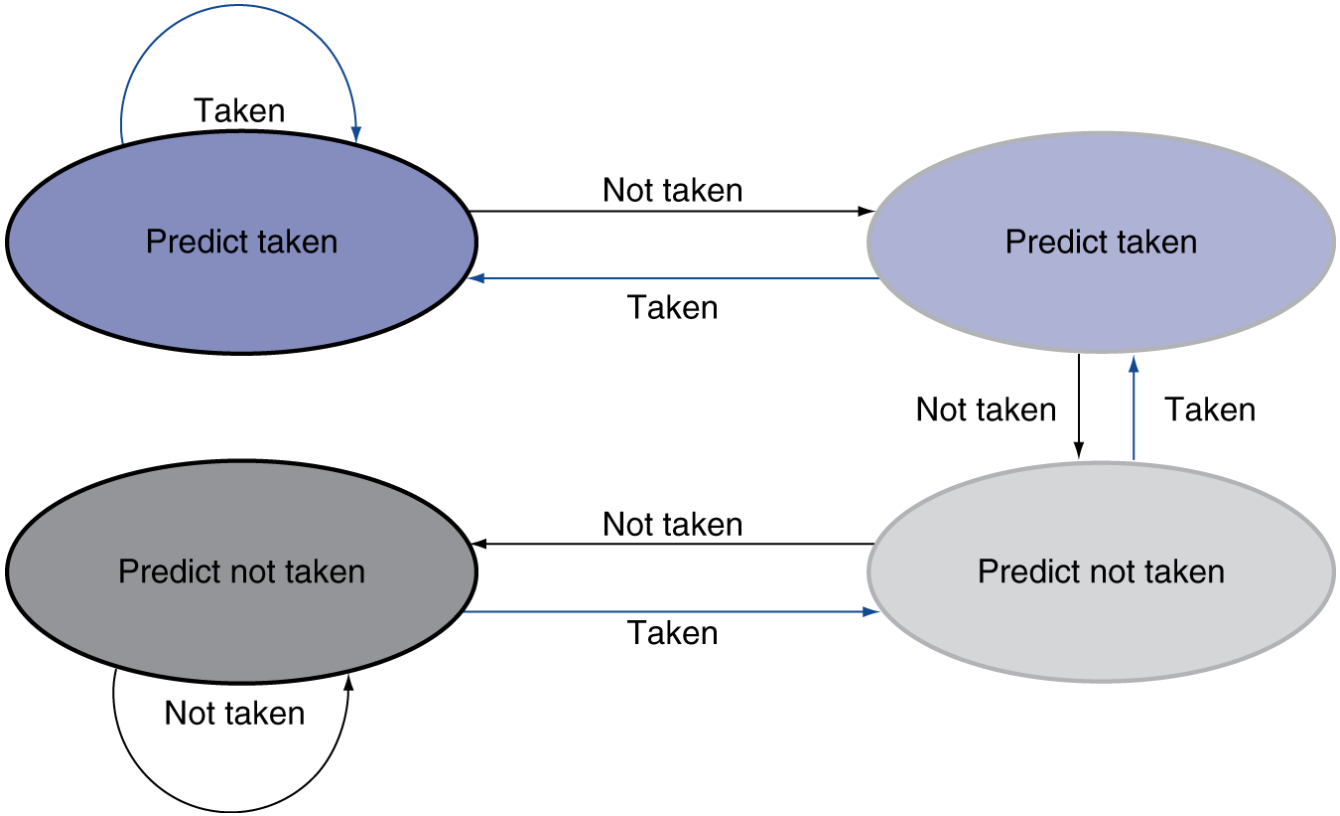
- Zwei falsche Vorhersagen pro Durchgang einer inneren Schliefe



- Fehlvorhersage (taken) bei letzte Iteration der innere Schleife
- Beim nächsten Eintreten in innere Schleife wiederum Fehlvorhersage (not taken)

2-Bit (bimodeal) Prädikator

- Ändere Vorhersage erst, wenn 2 Fehlvorhersagen auftreten



Sprungzielpuffer (*Branch Target Buffer*)

- Bisherige Sprung-Prädiktoren sagen nur Sprungrichtung voraus
  - Leistungssteigerung nur wenn Sprungzieladresse schon bekannt

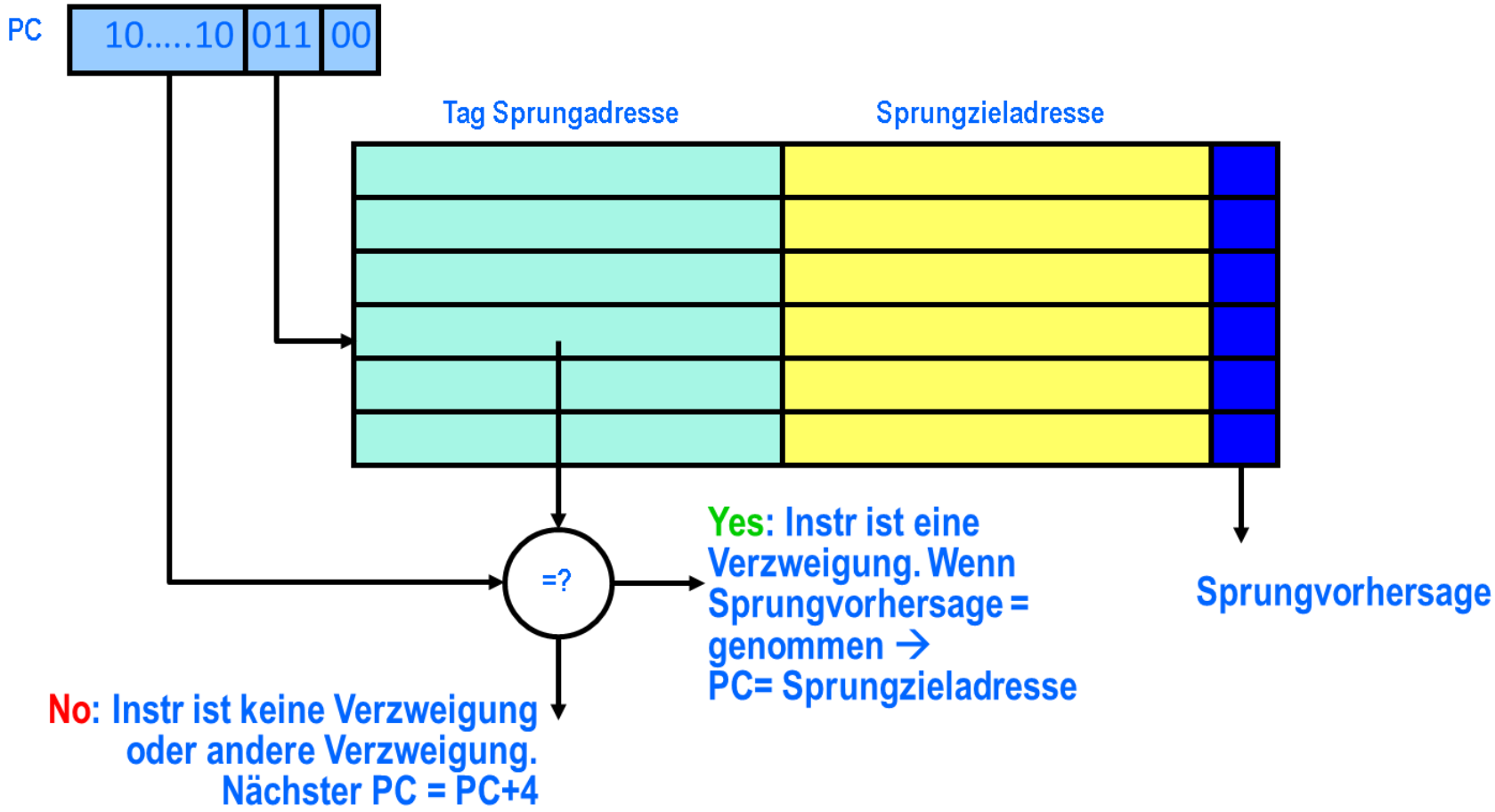
- In 5-stufigen MIPS Pipeline Sprungrichtung und Zieladresse in ID-Stufe berechnet (+ branch delay slot)

- Um Sprungvorhersage zur IF-Stufe zu verlagern, brauchen wir sowohl die Zieladresse als auch die Vorhersage bevor der potentielle Sprungbefehl geholt wurde

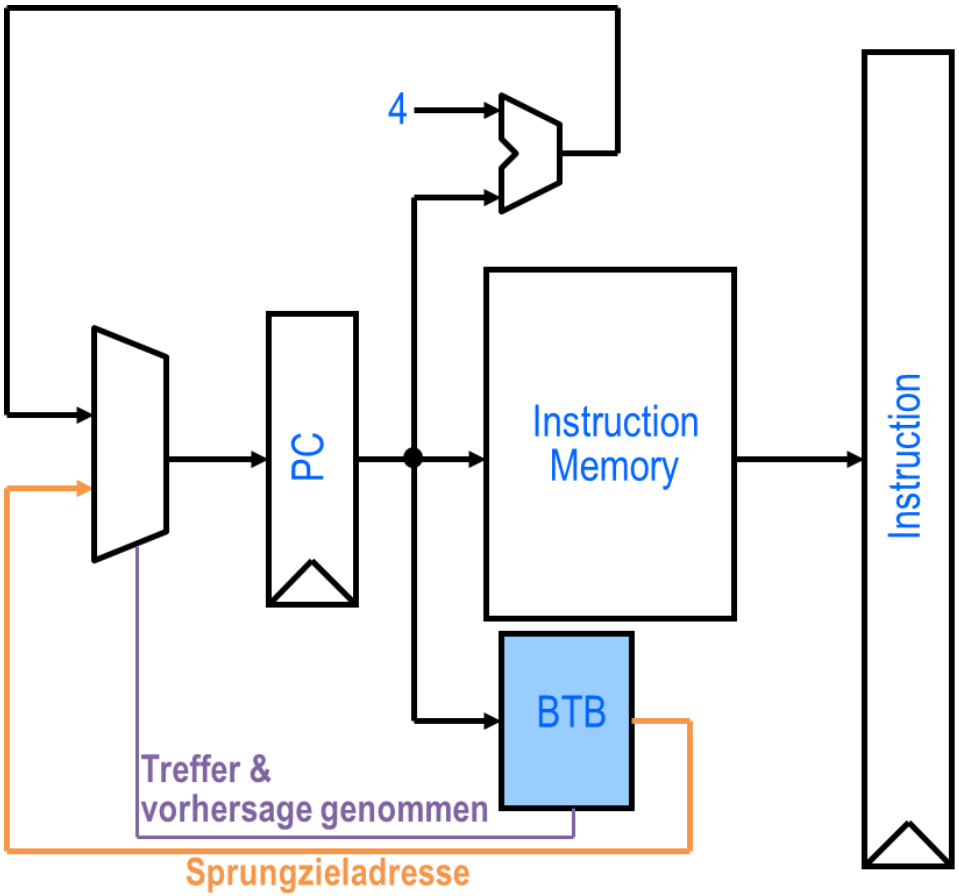
Sprungzielpuffer (*Branch Target Buffer*)

- Sprungzielpuffer (*Branch Target Buffer* (BTB)): Sprungvorhersage UND Zieladresse
  - Ist die nächste Instruktion ein Sprung?
    - Könnte jeder Befehl sein (Befehl noch nicht entschlüsselt)
  - Ist der obere Teil des Befehlszählers = obere Teil der Befehlsadresse des Sprungs im BTB (Tag)?
    - BTB eine Art Cache (nächste Vorlesung)

Sprungzielpuffer (*Branch Target Buffer*)



BTB integriert in MIPS-Pipeline



Moderne Hochleistungsprozessoren

- Moderne Prozessoren sind sehr kompliziert
- Intel Core i7:
  - 16 Pipeline Stufen
  - verarbeitet bis zu 4/6 Befehle gleichzeitig
  - gleichzeitig holen, dekodieren, ausführen, etc. (*Superskalar*)
  - CPU bestimmt in welcher Reihenfolge Befehle ausgeführt werden (*out-of-order execution, dynamische Scheduling*)
  - Register werden umbenannt (*register renaming*) um Parallelität auf Befehlsebene (*instruction-level parallelism*) zu erhöhen
  - 1 Kern kann 2 Programme/Threads gleichzeitig bearbeiten (*Hardwareseitiges Multithreading*)
  - Sehr fortgeschrittene Sprung-Prädiktoren
- Diese und andere Merkmale moderner Prozessoren werden im Modul Advanced Computer Architecture (Fachgebiet AES) vertieft diskutiert

Fazit

- Pipelining erhöht den Durchsatz, indem die Ausführung von verschiedenen Befehlen überlappt wird.

- MIPS für Pipelining entworfen
  - Alle Befehle 32 Bit lang
  - Quellregister immer an gleicher Stelle
  - Sehr einfache Sprungbefehle

- Strukturkonflikte
  - repliziere Funktionelle Einheiten

Fazit

- Datenabhängigkeiten/-konflikte
  - Forwarding zur frühzeitigen Bereitstellung der Operanden
  - Erkennen von Konflikten (Hazard Detection Unit)
  - Umordnen der Reihenfolge der Befehle (Compiler, out-of-order execution)
- Steuerkonflikte
  - Vorziehen der Sprungentscheidung +
  - nächster Befehl verwerfen (Pipeline Flush) oder Branch Delay Slot

- Nächste Vorlesung: Speicherhierarchie (Buchkapitel 7)

Fazit Eintakt- vs. Mehrzyklen- vs. Pipelined Prozessor

- Eintaktprozessor

lw					add				beq		
IF	ID	EX	MA	WB	IF	ID	EX	WB	IF	ID	EX

- Mehrzyklenprozessor

lw					add				beq		
IF	ID	EX	MA	WB	IF	ID	EX	WB	IF	ID	EX

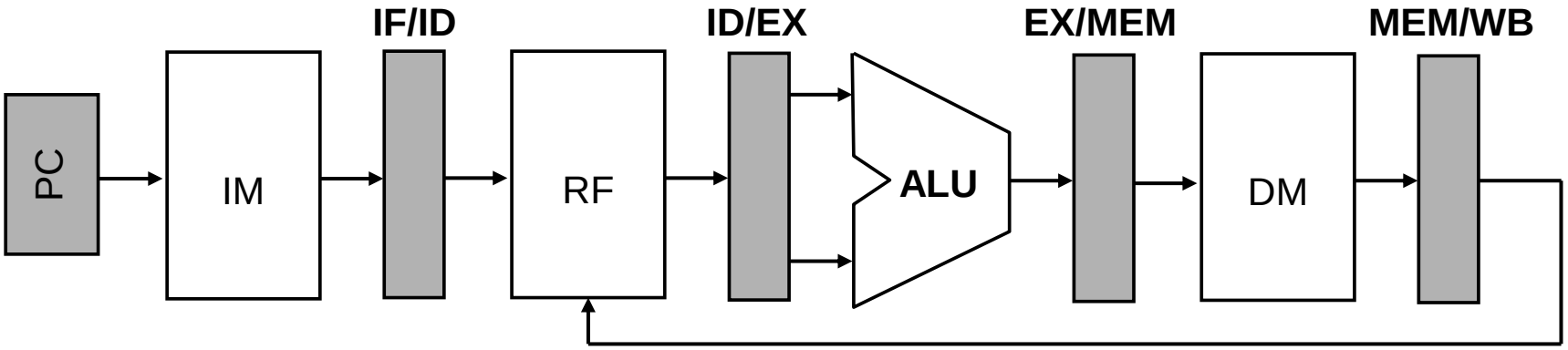
- Pipelined Prozessor

	cc 1	cc 2	cc3	cc 4	cc 5	cc 6	cc 7
lw	IF	ID	EX	MA	WB		
add		IF	ID	EX	MA	WB	
beq			IF	ID	EX	MA	WB

Back-up folien

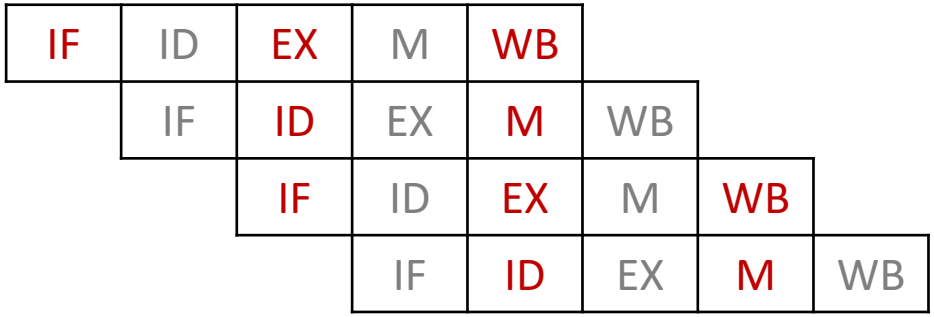
Wann forwarding?

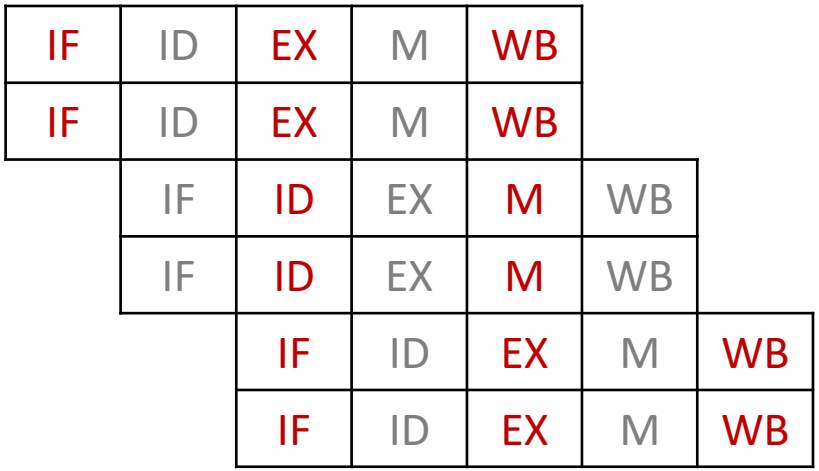
- Wenn (einer) der Quellregister des Befehls in der EX-Stufe mit dem Zielregister des Befehls in der MEM- oder WB-Stufe übereinstimmt
  - EX/MEM.Rd==ID/EX.Rs
  - EX/MEM.Rd==ID/EX.Rt
  - MEM/WB.Rd==ID/EX.Rs
  - MEM/WB.Rd==ID/EX.Rt
- + Befehl soll zum Registersatz schreiben (RegWrite) und Zielreg. nicht $0



Moderne Prozessoren

- Moderne Hochleistungsprozessoren nutzen tiefere Pipelines.
  - 10 bis 20 Pipeline-Stufen sind nicht unüblich.
- Zusätzlich sind moderne Prozessoren meist superskalar.
  - In jedem Taktzyklus wird versucht mehrere Befehle zu laden und auszuführen.





→ Falsche Sprungvorhersagen sind noch problematischer

→ Verwenden der dynamischen Sprungvorhersage (dynamic branch prediction)