

# Pipelining

Technische Grundlagen der Informatik

# Recap

---

- Maschinenbefehle
  - Erweiterung der Prozessorfunktionalität
  - Können „mehr“ als Mikroinstruktionen
  - Bsp.: Multiplizieren, Push, Stackmaschine (siehe TUWEL)
- Funktionen
  - Rücksprungadresse (am Stack)
- Interrupts
  - Asynchron
  - Können den Programmablauf unterbrechen
- Stack

# Adressierungsarten (für den Hauptspeicher)

---

- Direkte Adressierung

- Adresse wird direkt (immediate) angegeben

- z.B.:  $R1 \leftarrow \text{memory}[0x\text{CAFE}]$

- Register-indirekte Adressierung

- Inhalt eines Registers stellt die Adresse dar

- z.B.:

- $R2 \leftarrow 0x\text{CAFE}$

- $R1 \leftarrow \text{memory}[R2]$

- Indirekte Adressierung

- Inhalt einer Speicherzelle stellt die Adresse dar

- z.B.:  $R1 \leftarrow \text{memory}[\text{memory}[0x\text{CAFE}]]$

- Pre- und Post inkrement / dekrement

- Registerinhalt wird vor bzw. nach dem Zugriff verändert

- z.B.:

- $R2 \leftarrow 0x\text{CAFE}$

- $R1 \leftarrow \text{memory}[(R2) +]$

➔ Zugriff auf 0xCAFE, R2 danach 0xCAFF

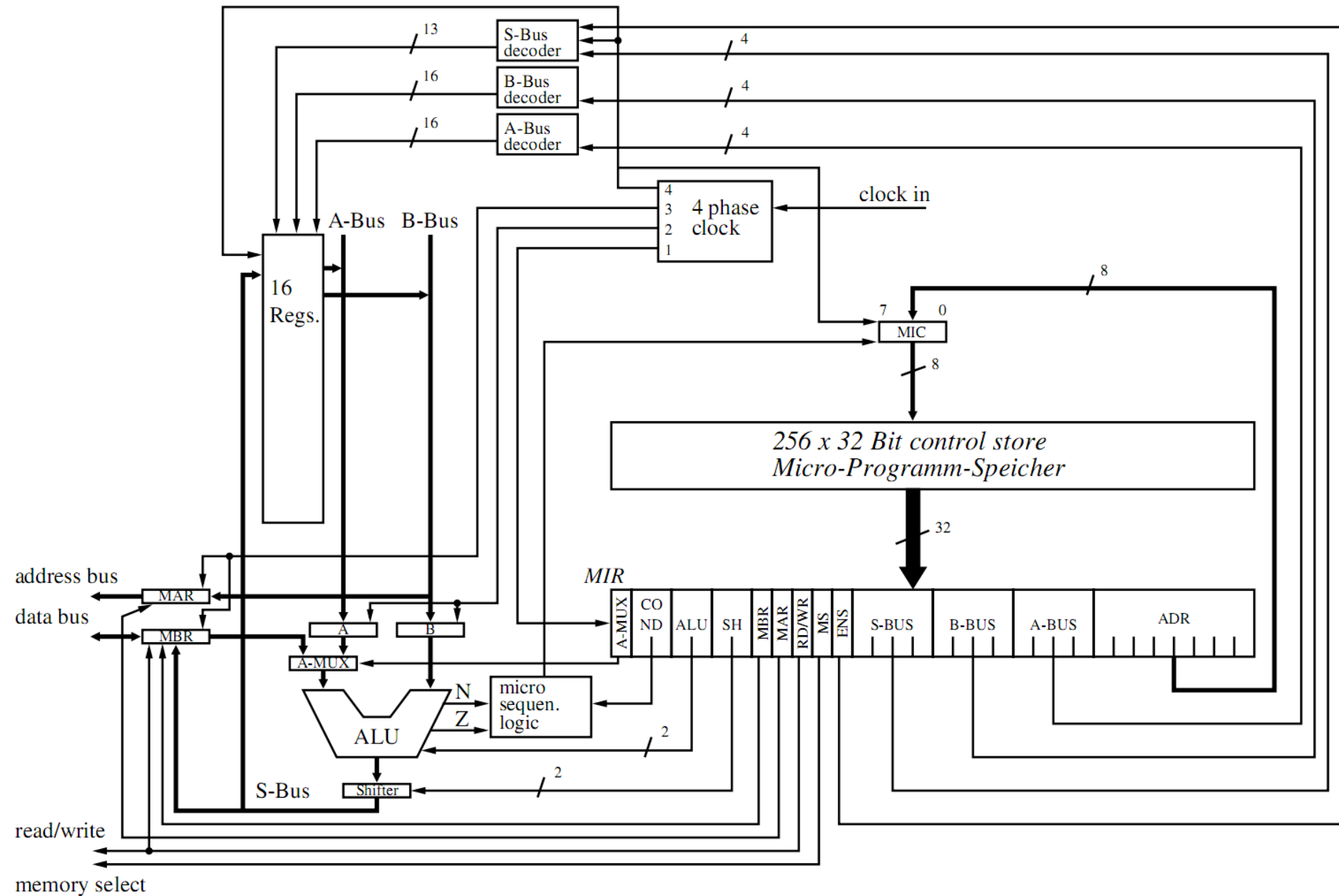
- $R1 \leftarrow \text{memory}[+ (R2) ]$

➔ Zugriff auf 0xCAFF, R2 danach 0xCAFF

# Beispielprogramm Micro16

---

1.  $R6 \leftarrow R6 + R5$                       # ADD
2.  $MAR \leftarrow R4; rd$                       # RD1
3.  $rd$     # RD2
4.  $R2 \leftarrow lsh(R1)$                       # SHIFT



# Befehlsdurchsatz (Micro16)

Takt	Phase 1 (MIR)	Phase 2 (REG)	Phase 3 (Adr)	Phase 4 (WB)
0	ADD			
1		ADD		
2			ADD	
3				ADD
4	RD1			
5		RD1		
6			RD1	
7				RD1
8	RD2			
9		RD2		
10			RD2	
11				RD2
12	SHIFT			
13		SHIFT		
14			SHIFT	
15				SHIFT

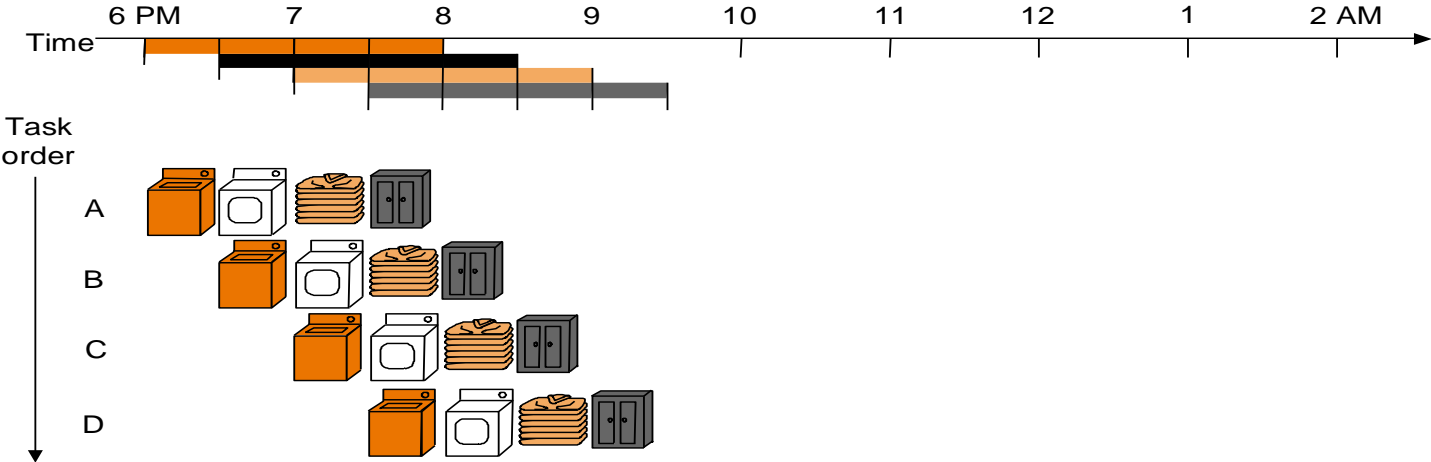
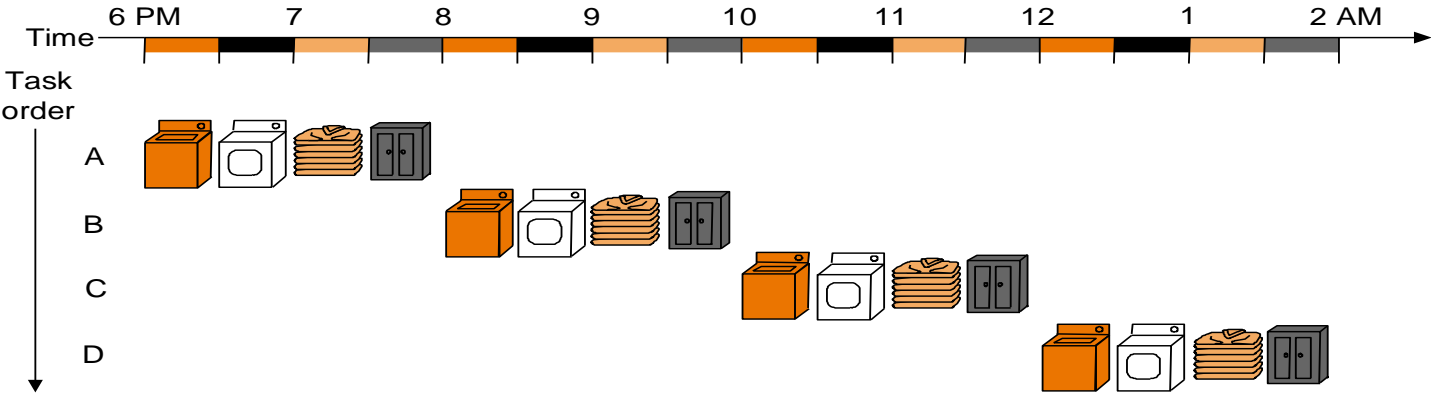
# Kennwerte Micro16

---

Vorgaben:  
4-Phasen Rechenwerk  
z.B. 2 ns Taktung

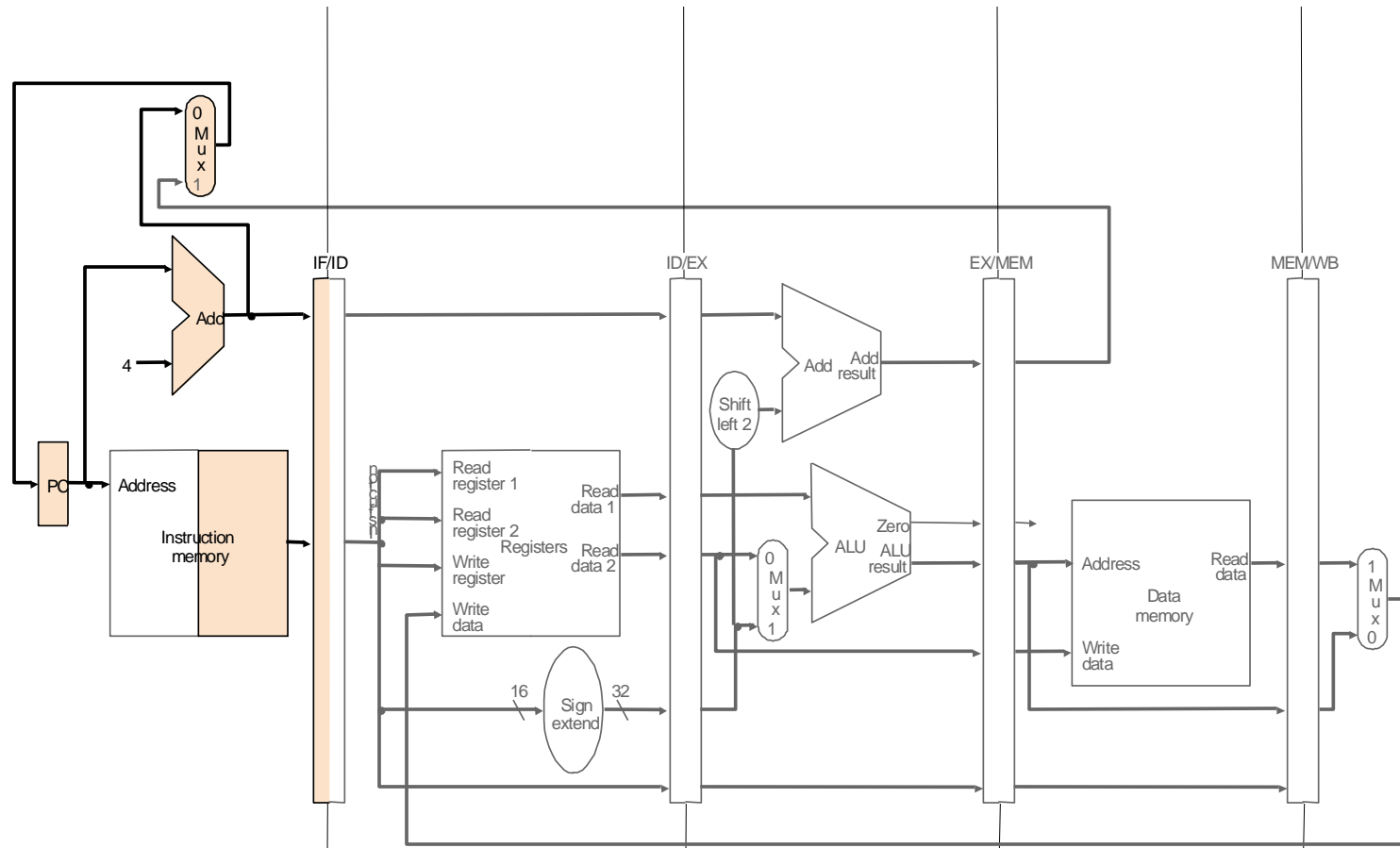
- Ausführungszeit:
  - 4 Takte pro Befehl
  - 8 ns pro Befehl
  - 125 Millionen Instruktionen pro Sekunde (MIPS)
    - FLOPS = Floating Point Operations per Second
- Unnötige Phasen können nicht übersprungen werden

# Idee





# MIPS (Microprocessor without interlocked pipeline stages)



## Verarbeitungsschritte beim MIPS

Befehl	IF	ID	EXE	MEM	WB	$\Sigma$
<i>lw</i>	2 ns	1 ns	2 ns	2 ns	1 ns	8 ns
<i>sw</i>	2 ns	1 ns	2 ns	2 ns		7 ns
<i>add, sub, and, or, slt</i>	2 ns	1 ns	2 ns		1 ns	6 ns
<i>beq</i>	2 ns	1 ns	2 ns			5 ns

IF ... *instruction fetch*

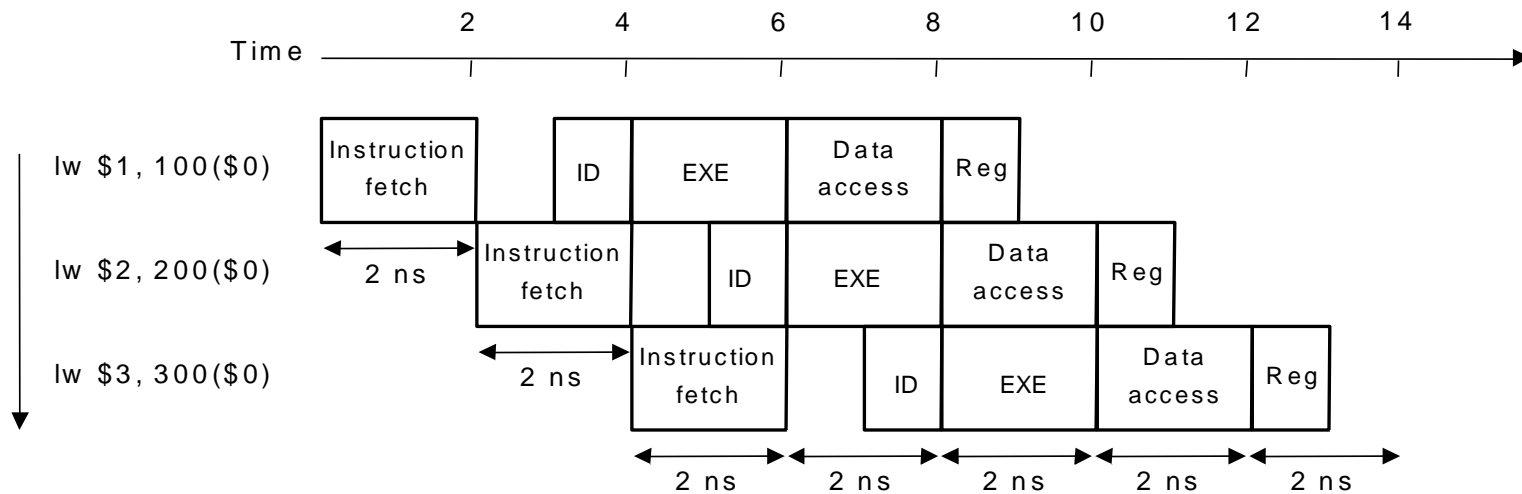
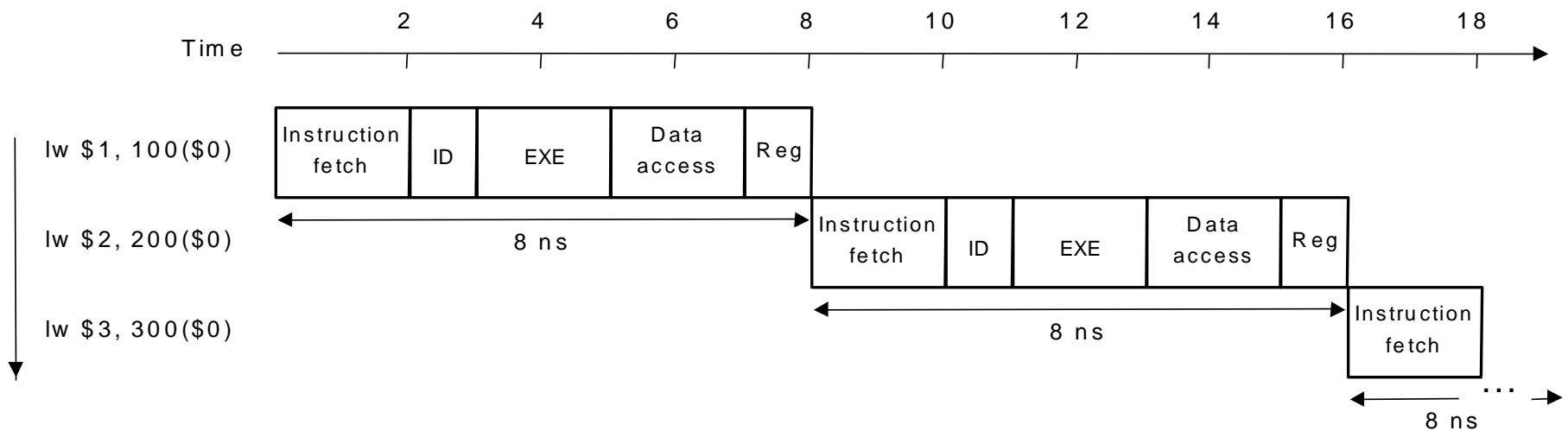
ID ... *instruction decode and source register read*

EXE ... *instr. execution / addr. computation / compare*

MEM ... *memory data access*

WB ... *target register write*

# Pipelining



## Bsp.: Perfekte Pipeline – 6 Befehle

Takt	IF	ID	EXE	MEM	WB
0	I1				
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					

## Bsp.: Perfekte Pipeline – 6 Befehle

Takt	IF	ID	EXE	MEM	WB
0	I1				
1	I2	I1			
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					

## Bsp.: Perfekte Pipeline – 6 Befehle

Takt	IF	ID	EXE	MEM	WB
0	I1				
1	I2	I1			
2	I3	I2	I1		
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					

## Bsp.: Perfekte Pipeline – 6 Befehle

Takt	IF	ID	EXE	MEM	WB
0	I1				
1	I2	I1			
2	I3	I2	I1		
3	I4	I3	I2	I1	
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					

## Bsp.: Perfekte Pipeline – 6 Befehle

Takt	IF	ID	EXE	MEM	WB
0	I1				
1	I2	I1			
2	I3	I2	I1		
3	I4	I3	I2	I1	
4	I5	I4	I3	I2	I1
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					



## Bsp.: Perfekte Pipeline – 6 Befehle

Takt	IF	ID	EXE	MEM	WB
0	I1				
1	I2	I1			
2	I3	I2	I1		
3	I4	I3	I2	I1	
4	I5	I4	I3	I2	I1
5	I6	I5	I4	I3	I2
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					

## Bsp.: Perfekte Pipeline – 6 Befehle

Takt	IF	ID	EXE	MEM	WB
0	I1				
1	I2	I1			
2	I3	I2	I1		
3	I4	I3	I2	I1	
4	I5	I4	I3	I2	I1
5	I6	I5	I4	I3	I2
6		I6	I5	I4	I3
7					
8					
9					
10					
11					
12					
13					
14					
15					

## Bsp.: Perfekte Pipeline – 6 Befehle

Takt	IF	ID	EXE	MEM	WB
0	I1				
1	I2	I1			
2	I3	I2	I1		
3	I4	I3	I2	I1	
4	I5	I4	I3	I2	I1
5	I6	I5	I4	I3	I2
6		I6	I5	I4	I3
7			I6	I5	I4
8					
9					
10					
11					
12					
13					
14					
15					

## Bsp.: Perfekte Pipeline – 6 Befehle

Takt	IF	ID	EXE	MEM	WB
0	I1				
1	I2	I1			
2	I3	I2	I1		
3	I4	I3	I2	I1	
4	I5	I4	I3	I2	I1
5	I6	I5	I4	I3	I2
6		I6	I5	I4	I3
7			I6	I5	I4
8				I6	I5
9					
10					
11					
12					
13					
14					
15					

## Bsp.: Perfekte Pipeline – 6 Befehle


Takt	IF	ID	EXE	MEM	WB
0	I1				
1	I2	I1			
2	I3	I2	I1		
3	I4	I3	I2	I1	
4	I5	I4	I3	I2	I1
5	I6	I5	I4	I3	I2
6		I6	I5	I4	I3
7			I6	I5	I4
8				I6	I5
9					I6
10					
11					
12					
13					
14					
15					

# Kennwerte

---

Vorgaben:

5-stufige Pipeline, 2 ns Taktung

- Absolute Ausführungszeit:
  - 10 Takte = 20 ns
  - Ein Befehl: nach wie vor 10 ns
- Realer Durchsatz: 
  - 6 Befehle in 20ns **3,33 ns** pro Befehl
  - 300 Millionen Instruktionen pro Sekunde (MIPS)
- Theoretischer Durchsatz (kein Ein-/Auslaufen):
  - 1 Befehl in 2 ns
  - 500 MIPS

# Performance-Gewinn

---

- $k$ -stufige Pipeline verbessert Durchsatz um den Faktor  $k$  (im Idealfall)
- verbessert NICHT die Ausführungszeit des einzelnen Befehls
- Begrenzungen:
  - “Balance” der Pipeline-Stufen (siehe zuvor)
  - “Einlaufen der Pipeline”
  - Abhängigkeiten (“Hazards”)

# Hazards

---

- **Strukturelle Hazards**

mehrere Pipeline-Stufen benötigen dieselbe Ressource

**Maßnahmen:** geeignete Architektur, *stall*

- **Control Hazards (Branch Hazards)**

Nachfolgebefehl hängt vom Ausgang des Sprunges ab

**Maßnahmen:** prediction, delayed branching, *stall*

- **Data Hazards**

Berechnung erfordert Ergebnis des Vorgängerbefehls

**Maßnahmen:** forwarding, Code Optimierung, *stall*



## Bsp.: Stall (z.B. bei strukturellem Hazard)

Takt	IF	ID	EXE	MEM	WB
0	ADD				
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					

## Bsp.: Stall (z.B. bei strukturellem Hazard)

Takt	IF	ID	EXE	MEM	WB
0	ADD				
1	LOAD	ADD			
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					

## Bsp.: Stall (z.B. bei strukturellem Hazard)

Takt	IF	ID	EXE	MEM	WB
0	ADD				
1	LOAD	ADD			
2		LOAD	ADD		
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					

## Bsp.: Stall (z.B. bei strukturellem Hazard)

Takt	IF	ID	EXE	MEM	WB
0	ADD				
1	LOAD	ADD			
2		LOAD	ADD		
3			LOAD	ADD	
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					

## Bsp.: Stall (z.B. bei strukturellem Hazard)

Takt	IF	ID	EXE	MEM	WB
0	ADD				
1	LOAD	ADD			
2		LOAD	ADD		
3			LOAD	ADD	
4				LOAD	ADD
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					

Diagram illustrating a pipeline stall (Bsp.: Stall) due to a structural hazard. The pipeline stages are IF (Instruction Fetch), ID (Instruction Decode), EXE (Execute), MEM (Memory Access), and WB (Write Back). The instructions are ADD, LOAD, and ADD. The stall occurs at the MEM stage of the second instruction (LOAD) because it needs the result of the first instruction (ADD), which is still in the MEM stage of the first instruction. The stall is indicated by an orange arrow pointing to the MEM stage of the second instruction, labeled "STALL".

## Bsp.: Stall (z.B. bei strukturellem Hazard)

Takt	IF	ID	EXE	MEM	WB
0	ADD				
1	LOAD	ADD			
2		LOAD	ADD		
3			LOAD	ADD	
4				LOAD	ADD
5				LOAD	
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					

## Bsp.: Stall (z.B. bei strukturellem Hazard)

Takt	IF	ID	EXE	MEM	WB
0	ADD				
1	LOAD	ADD			
2		LOAD	ADD		
3			LOAD	ADD	
4				LOAD	ADD
5				LOAD	
6					LOAD
7					
8					
9					
10					
11					
12					
13					
14					
15					

# Typen von Data-Hazards

---

- read-after-write (RAW)

Register wird gelesen,  
bevor vorheriger Befehl es beschrieben hat

$$R2 \leftarrow R1 + R3$$

$$R4 \leftarrow R2 + R1$$

- write-after-read (WAR)

Register wird beschrieben,  
bevor vorheriger Befehl es gelesen hat

$$R2 \leftarrow R1 + R3$$

$$R3 \leftarrow 1$$

- write-after-write (WAW)

Register wird beschrieben,  
jedoch danach vom vorherigen Befehl überschrieben

$$R2 \leftarrow R1 + R3$$

$$R2 \leftarrow 1$$



# Beispiel: Data Hazards durch Code Optimierung auflösen

---

1. lw \$1, 100(\$0)                   # lade Mem(100) ins Register 1
2. add \$3, \$1, \$1                   # addiere Register R1 + R1, Ergebnis --> Register 3
3. sll \$1, \$3, 1                    # Shift left von R3, Ergebnis --> Register 1
4. sw \$3, 300(\$0)                   # Speichere Register 3 nach Mem(300)
5. lw \$2, 200(\$0)                   # lade Mem(200) ins Register 2
6. sll \$4, \$2, 1                    # Shift left von R2, Ergebnis --> Register 4
7. add \$2, \$4, \$3                   # addiere Register R4 + R3, Ergebnis --> Register 2
8. sw \$2, 400(\$0)                   # Speichere Register 2 nach Mem(400)
9. jr \$8                            # Springe an die Speicheradresse \$8

- IF - Instruction Fetch: Holt die Instruktion aus dem Speicher
- ID - Instruction Decode: Dekodiert die Instruktion und greift lesend auf Register zu
- EXE - Execute: Führt ALU-Operationen aus
- MEM - Memory: Führt lesende und schreibende Zugriffe auf den Speicher aus
- WB - Writeback: Schreibt Ergebnisse in die Register zurück

# Data Hazards

Takt	IF	ID	EXE	MEM	WB
0	1				
1	NOP	1			
2	NOP	NOP	1		
3	NOP	NOP	NOP	1	
4	2	NOP	NOP	NOP	1
5	NOP	2	NOP	NOP	NOP
6	NOP	NOP	2	NOP	NOP
7	NOP	NOP	NOP	2	NOP
8	3	NOP	NOP	NOP	2
9	4	3	NOP	NOP	NOP
10	5	4	3	NOP	NOP
11	NOP	5	4	3	NOP
12	NOP	NOP	5	4	3
13	NOP	NOP	NOP	5	4
14	6	NOP	NOP	NOP	5
15	NOP	6	NOP	NOP	NOP

The diagram illustrates data hazard resolution in a 6-stage pipeline. Arrows indicate the forwarding of data from the MEM stage of one instruction to the ID stage of a subsequent instruction. Specifically, the value 1 from the MEM stage of instruction 4 is forwarded to the ID stage of instruction 5. The value 2 from the MEM stage of instruction 8 is forwarded to the ID stage of instruction 9. The value 5 from the MEM stage of instruction 14 is forwarded to the ID stage of instruction 15. The values in the ID stage (2, 3, and 6) are circled in red to indicate the resolved values.

# Beispiel: Data Hazards durch Code Optimierung auflösen

---

1) lw \$1, 100(\$0)	# lade Mem(100) ins Register 1
5) lw \$2, 200(\$0)	# lade Mem(200) ins Register 2
2) add \$3, \$1, \$1	# addiere Register R1 + R1, Ergebnis --> Register 3
6) sll \$4, \$2, 1	# Shift left von R2, Ergebnis --> Register 4
3) sll \$1, \$3, 1	# Shift left von R3, Ergebnis --> Register 1
7) add \$2, \$4, \$3	# addiere Register R4 + R3, Ergebnis --> Register 2
4) sw \$3, 300(\$0)	# Speichere Register 3 nach Mem(300)
8) sw \$2, 400(\$0)	# Speichere Register 2 nach Mem(400)
9) jr \$8	# Springe an die Speicheradresse \$8

- IF - Instruction Fetch: Holt die Instruktion aus dem Speicher
- ID - Instruction Decode: Dekodiert die Instruktion und greift lesend auf Register zu
- EXE - Execute: Führt ALU-Operationen aus
- MEM - Memory: Führt lesende und schreibende Zugriffe auf den Speicher aus
- WB - Writeback: Schreibt Ergebnisse in die Register zurück

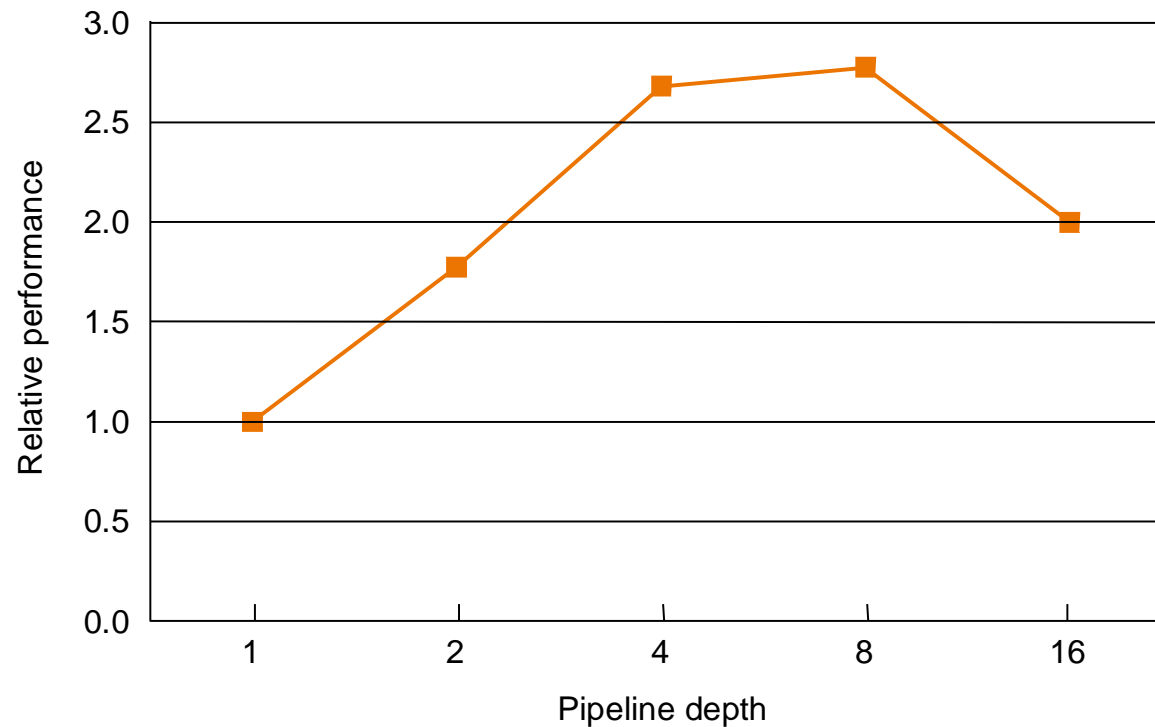
# Data Hazards durch Code Optimierung aufgelöst

Takt	IF	ID	EXE	MEM	WB
0	1				
1	5	1			
2	NOP	5	1		
3	NOP	NOP	5	1	
4	2	NOP	NOP	5	1
5	6	2	NOP	NOP	5
6	NOP	6	2	NOP	NOP
7	NOP	NOP	6	2	NOP
8	3	NOP	NOP	6	2
9	7	3	NOP	NOP	6
10	4	7	3	NOP	NOP
11	NOP	4	7	3	NOP
12	NOP	NOP	4	7	3
13	8	NOP	NOP	4	7
14	9	8	NOP	NOP	4
15		9	8	NOP	NOP

The diagram illustrates data hazard resolution through code optimization. Red circles highlight the values in the ID and WB stages that are being resolved. Arrows point from the WB stage to the ID stage, indicating the flow of data.

# “Super-Pipelining”

= höhere Anzahl von Pipeline-Stages



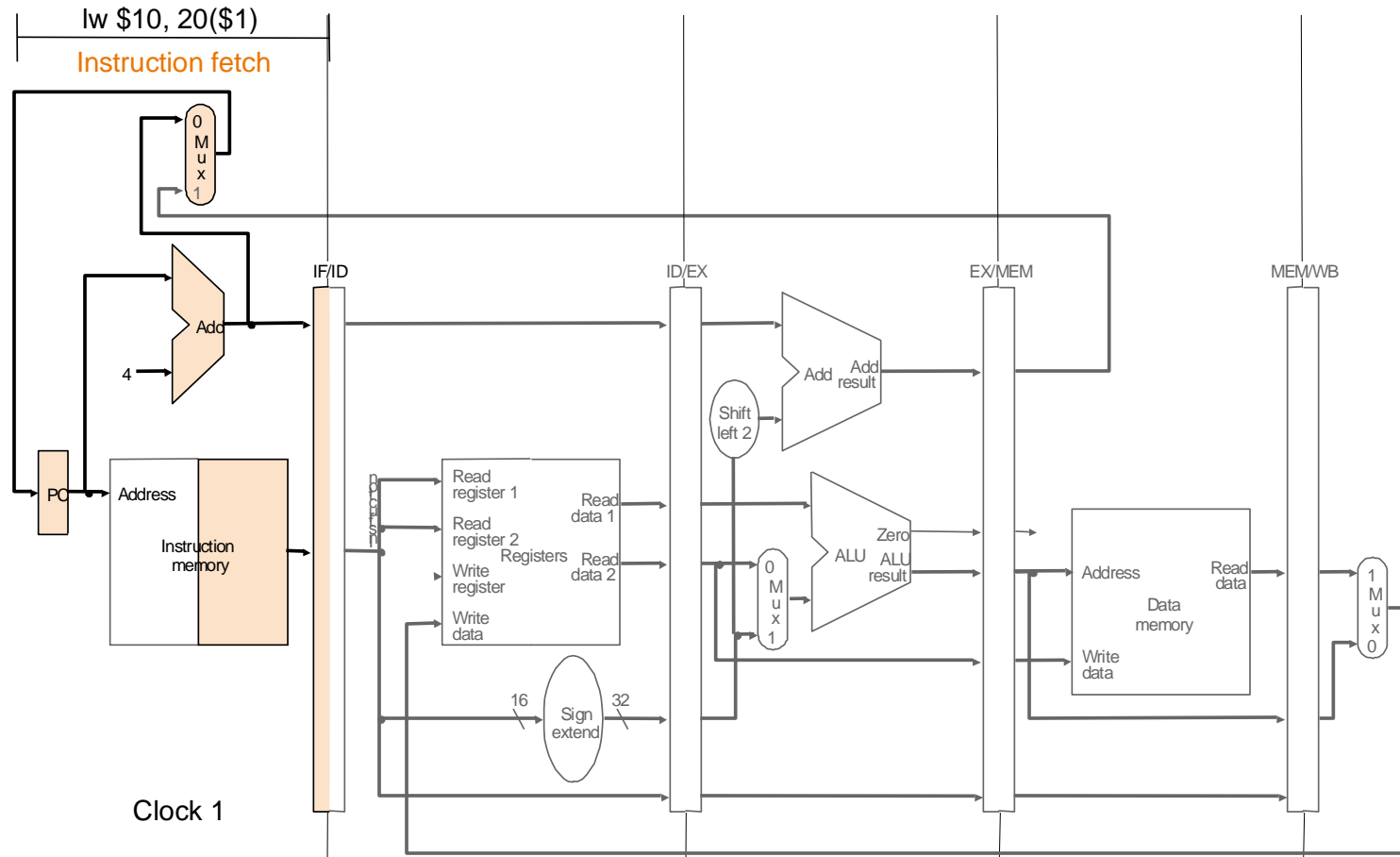
- Data- und Control-Hazards und
- Register-Verzögerungen werden dominanter

# “Superscalar Pipelining” vs. „Hyperthreading“

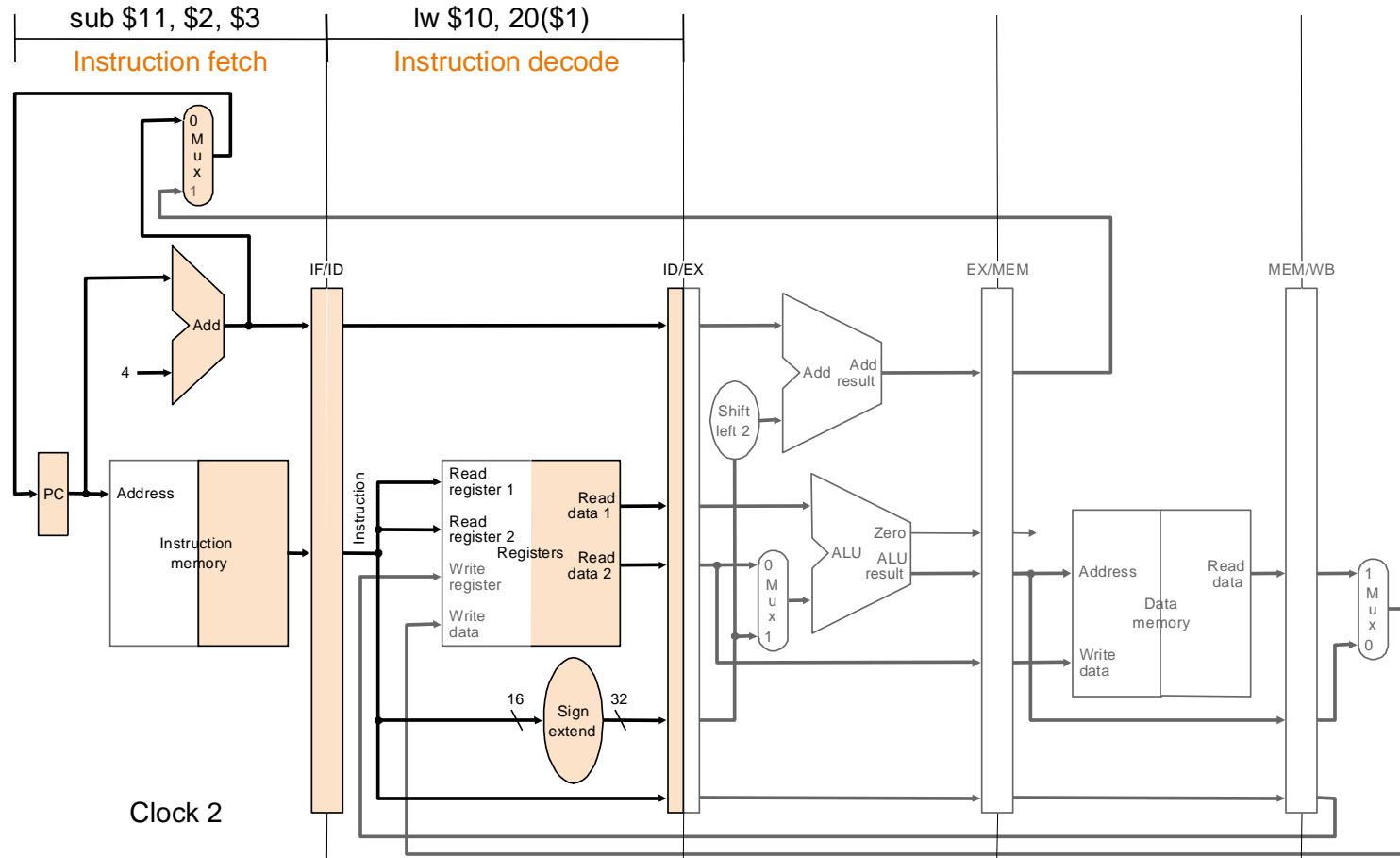
---

- Es gibt mehrere (typ. 2...6) parallele Datenpfade:
  - z.B. Integer, FP, load/store
- Pro Taktzyklus werden mehrere Befehle bearbeitet
  - z.B. CPI=0,25
- Pro Taktzyklus werden mehrere Befehle geladen (!)
  - z.B. 2 Befehle über 64 bit
- Hyperthreading
  - Alle oben angeführten Eigenschaften
  - Plus: **Doppelter Registersatz!**

# Zeitlicher Schnappschuß (1)

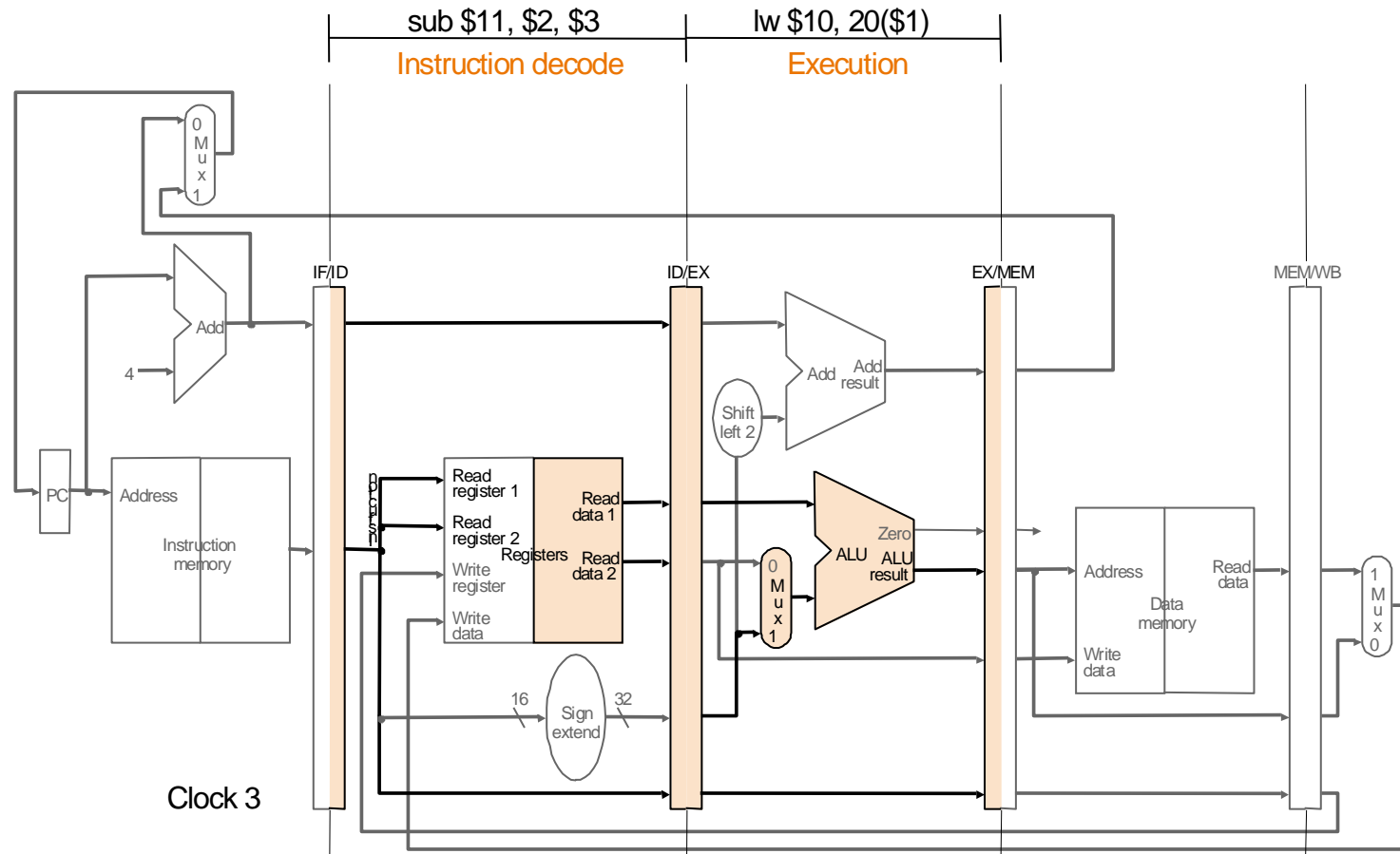


# Zeitlicher Schnappschuß (2)

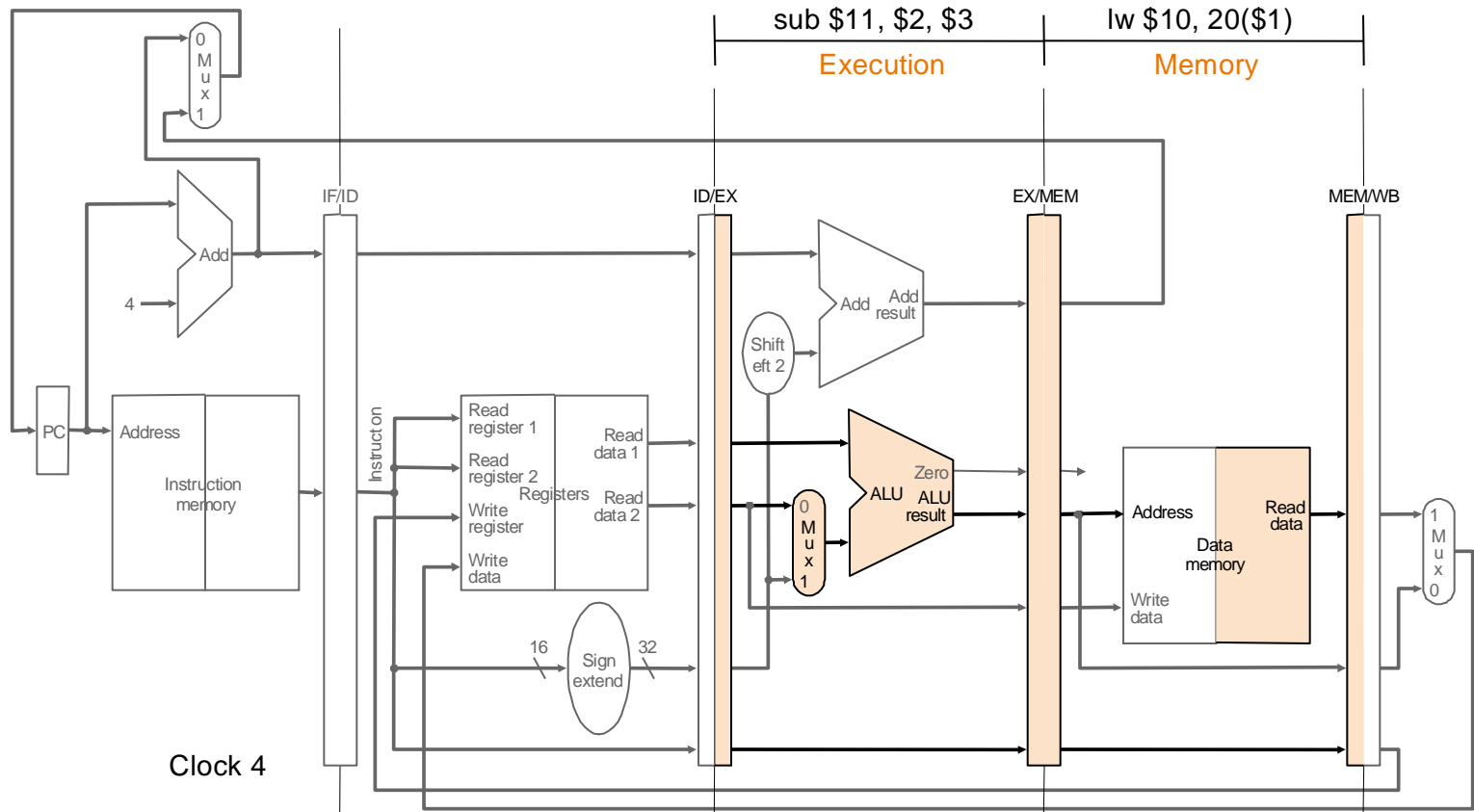




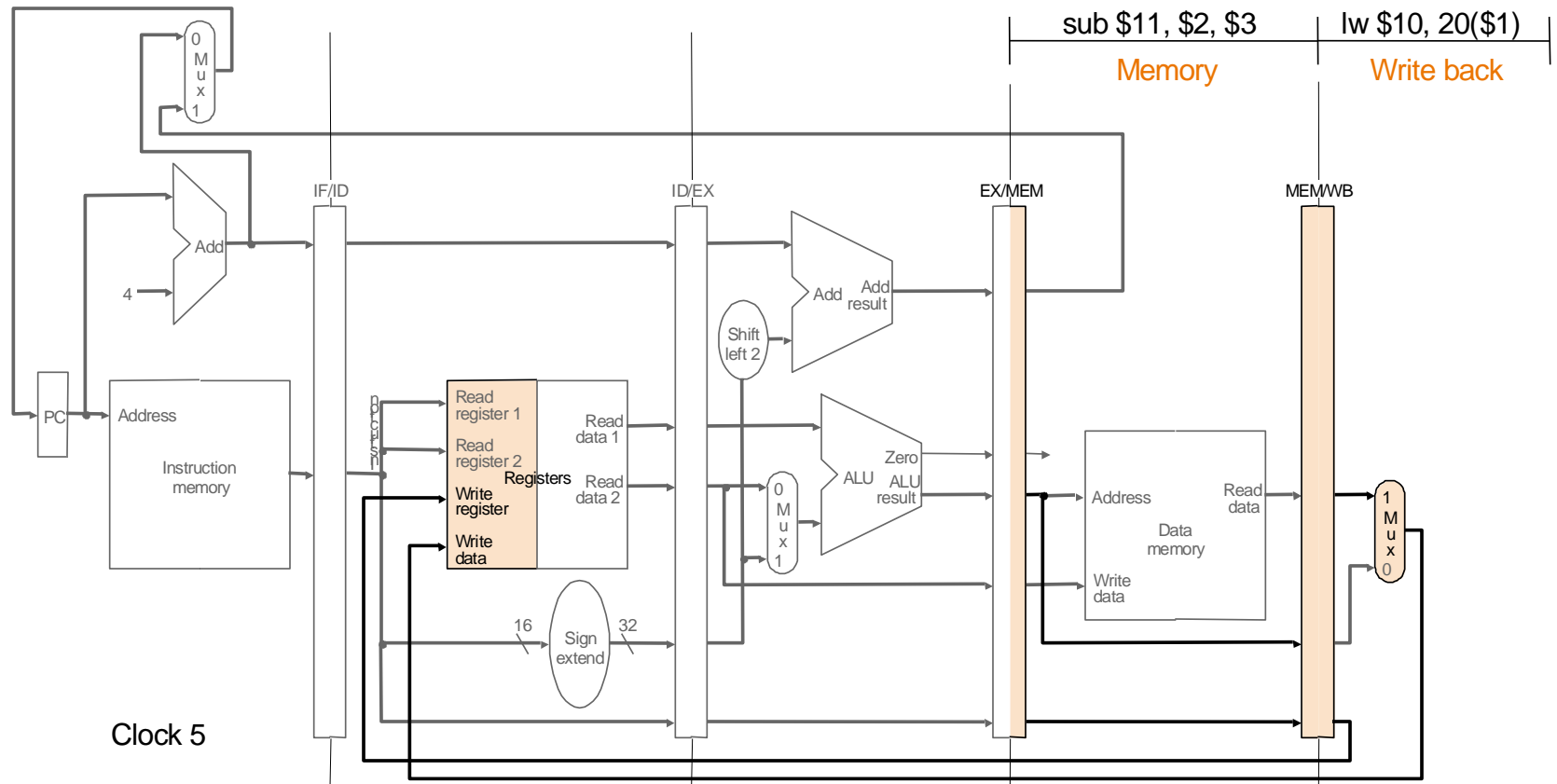
# Zeitlicher Schnappschuß (3)



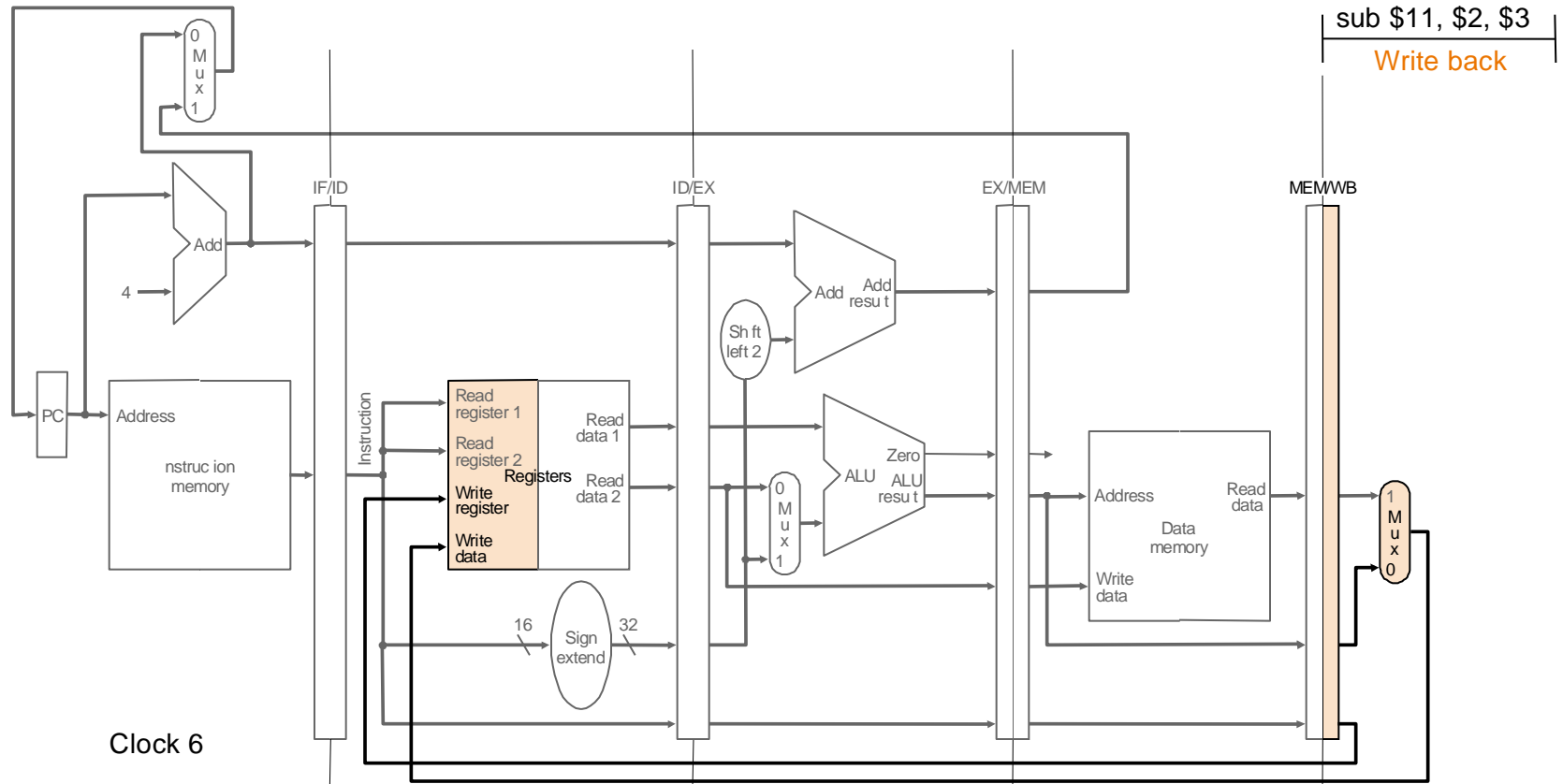
# Zeitlicher Schnappschuß (4)



# Zeitlicher Schnappschuß (5)



# Zeitlicher Schnappschuß (6)



# Zusammenfassung

---

- Pipelining
- Stufen
  - k Stufen – k-facher Durchsatz
- Stalling
  - Kann durch die Architektur oder den Compiler passieren
  - Senkt den Durchsatz einer Pipeline
- Hazards
  - 3 Arten, davon sind Data Hazards die wichtigsten
    - Wiederum 3 Arten von Data-Hazards