Kapitel 3: Pipelining

Befehlssätze

Pipelining

Lernziele

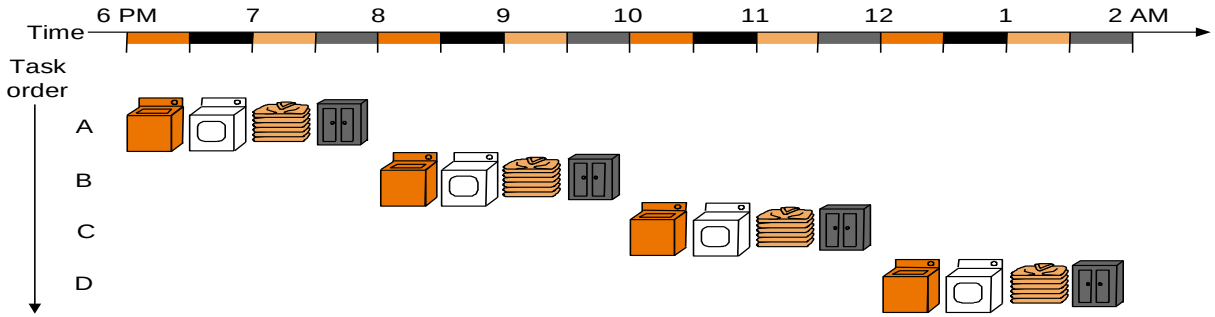
- Pipelining als Technik der Performanzsteigerung kennenlernen und verstehen
- Prinzip der Fließbandverarbeitung (Pipelining) wiedergeben und anwenden können
- Probleme der Fließbandverarbeitung (Pipelining) kennen und mögliche Problemlösungen hierfür angeben können

Das Prinzip an einem alltäglichen Beispiel

- Sie kommen aus dem Urlaub und es ist viel schmutzige Wäsche zu waschen
- Zur Verfügung stehen:
  - eine Waschmaschine (1/2 Stunde Laufzeit)
  - ein Trockner (1/2 Stunde Laufzeit)
  - eine Bügelmaschine (1/2 Stunde Arbeit zum Bügeln)
  - ein Wäscheschrank (1/2 Stunde Arbeit zum Einräumen)
- Jede der Personen A, B, C, und D aus dem Haushalt wäscht seine Wäsche selbst
- Es gibt zwei Möglichkeiten, die vier Waschvorgänge auszuführen

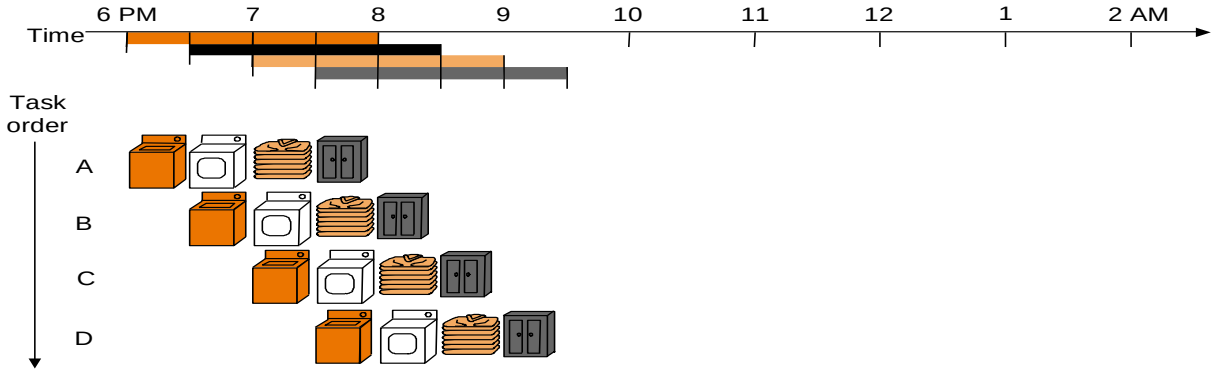
Das Prinzip an einem Beispiel

Ohne Pipelining:



Dauer der Arbeiten:
8 Stunden

Mit Pipelining:

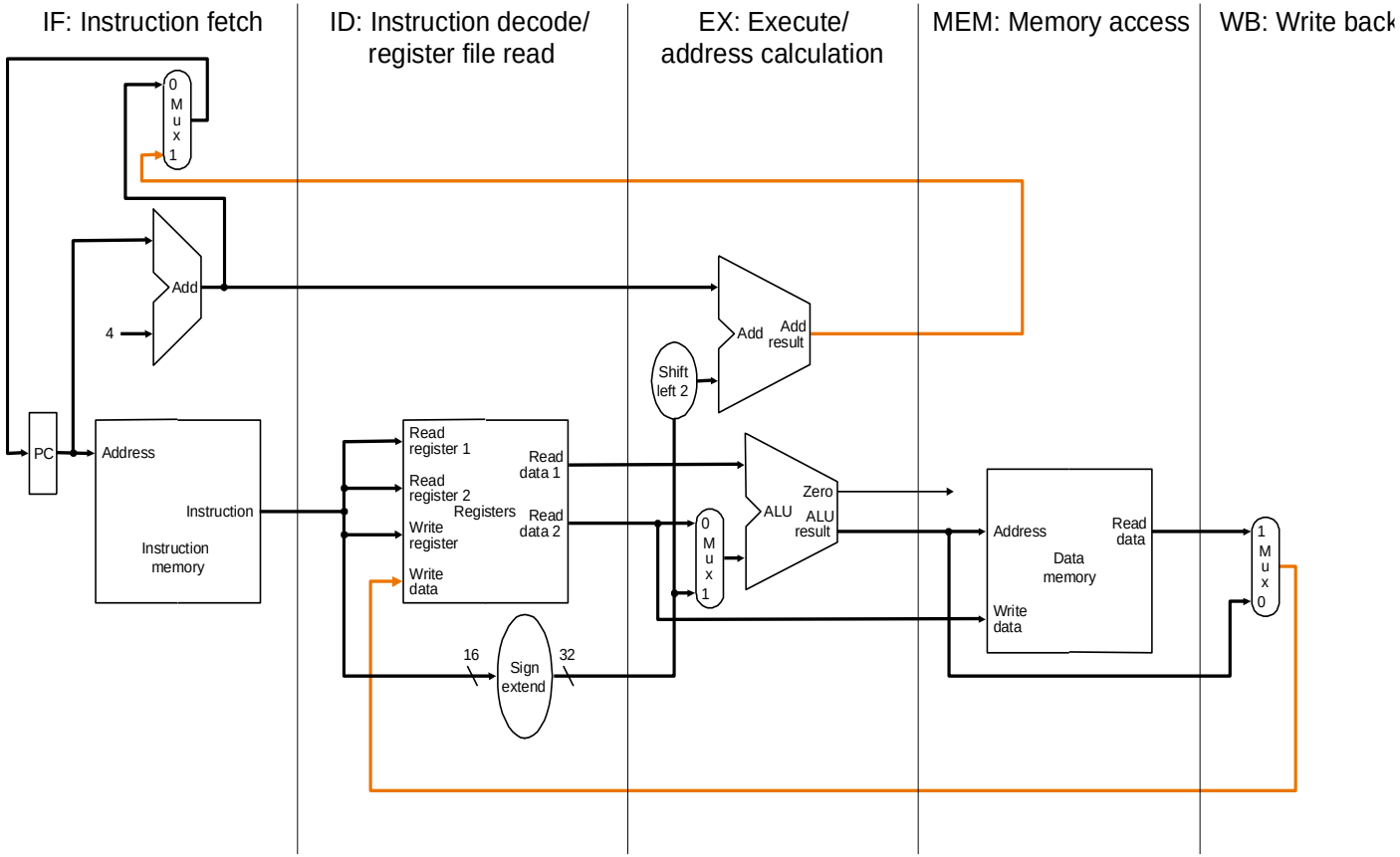


Dauer der Arbeiten:
3,5 Stunden

Aufteilung der Befehlsabarbeitung in Phasen

- Um Pipelining im Datenpfad ausnutzen zu können, muss die Abarbeitung eines Maschinenbefehls in mehrere Phasen mit möglichst gleicher Dauer aufgeteilt werden
- Eine sinnvolle Aufteilung ist abhängig vom Befehlssatz und der verwendeten Hardware
- Beispiel: Abarbeitung in fünf Schritten:
  - Befehl-Holen-Phase (*instruction fetch*)
  - Dekodierphase / Lesen von Operanden aus Registern (*decode*)
  - Ausführung / Adressberechnung (*execute*)
  - Speicherzugriff (*memory access*)
  - Abspeicherphase (*write back*)
- Bei CISC ist Pipelining schwierig, da die Dauer der Dekodier- und Ausführungsphase (evtl. mehrere Mikroprogrammbefehle) bei den verschiedenen Maschinenbefehlen sehr unterschiedlich ist

Aufteilung des Datenpfades in fünf Phasen



Pipelining: Illustration

- Annahme: Aufteilung der Befehlsabarbeitung in fünf gleichlange Phasen (P1-P5)

Zeitschritt:	1	2	3	4	5	6	7	8	9	10	11
Befehl 1:	P1	P2	P3	P4	P5						
Befehl 2:		P1	P2	P3	P4	P5					
Befehl 3:			P1	P2	P3	P4	P5				
Befehl 4:				P1	P2	P3	P4	P5			
Befehl 5:					P1	P2	P3	P4	P5		
Befehl 6:						P1	P2	P3	P4	P5	
Befehl 7:							P1	P2	P3	P4	P5

Pipelining: Beschleunigung (1)

- Annahmen:
  - Abarbeitungszeit eines Befehls ohne Pipelining: t
  - k Pipelinestufen, gleiche Laufzeit der Stufen
  - Laufzeit einer Stufe der Pipeline: t/k

- Beschleunigung bei m auszuführenden Instruktionen:

$m = 1$: Laufzeit mit Pipeline : $k \cdot \ t/k \ = \ t$

➡ keine Beschleunigung

$m = 2$: Laufzeit mit Pipeline: $t \ + \frac{t}{k} = \ (k+1) \cdot \frac{t}{k}$

➡ Beschleunigung um Faktor $2 \cdot k \ / \ (k+1)$

Pipelining: Beschleunigung (2)

$m \geq 1$: Laufzeit mit Pipeline: $t + (m-1) \cdot \frac{t}{k}$

Laufzeit ohne Pipeline: $m \cdot t$

➡ Beschleunigung um den Faktor: $$\frac{m \cdot t}{t + (m-1) \cdot \frac{t}{k}} = \frac{m \cdot t}{t \cdot (1 + (m-1) \cdot \frac{1}{k})} = \frac{m \cdot k}{m + k - 1} = \mathrm{k} - \frac{k \cdot (k-1)}{m + k - 1}$$

Ergebnis: Für $m \gg k$ nähert sich der Beschleunigung der Anzahl k der Pipelinestufen an.

Aber: Es wurde vorausgesetzt, dass sich die Ausführung der Befehle ohne weiteres „verzahnen“ lässt. Dies ist in der Praxis nicht immer der Fall ➡ Hazards!

Pipelining: Hazards

- **Datenabhängigkeit (data hazards):**
  - Befehlsfolge:

```
1. LOAD R1, 5 (R0);      // Adresse = Inhalt von Register R0 + 5,  lade Speicherinhalt an Adresse in Register R1
2. ADD R3, R2, R1;       // Addiere Inhalte der Register R2 und R1, Ergebnis in R3
3. SUB R6, R5, R4;       // Subtrahiere Inhalt von R4 von R5, Ergebnis in R6
4. MUL R9, R8, R7;       // Multipliziere Inhalt von R8 und R7, Ergebnis in R9
```

  - Problem:

    Der Wert in R1 steht dem `ADD`-Befehl nicht rechtzeitig zur Verfügung, falls man die Pipeline nicht stoppt.

Pipelining: Data Hazards

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Pipelining: Data Hazards

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Takt	Befehl holen	Dekodieren/Operand holen	Ausführen	Abspeichern

Pipelining: Data Hazards

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Takt	Befehl holen	Dekodieren/Operand holen	Ausführen	Abspeichern
1	LOAD			

Pipelining: Data Hazards

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Takt	Befehl holen	Dekodieren/Operand holen	Ausführen	Abspeichern
1	LOAD			
2	ADD	LOAD		

Pipelining: Data Hazards

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Takt	Befehl holen	Dekodieren/Operand holen	Ausführen	Abspeichern
1	LOAD			
2	ADD	LOAD		
3	SUB	ADD	LOAD	

Pipelining: Data Hazards

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Takt	Befehl holen	Dekodieren/Operand holen	Ausführen	Abspeichern
1	LOAD			
2	ADD	LOAD		
3	SUB	ADD	LOAD	
4	MUL	SUB	ADD	LOAD

Pipelining: Data Hazards

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Takt	Befehl holen	Dekodieren/Operand holen	Ausführen	Abspeichern
1	LOAD			
2	ADD	LOAD		
3	SUB	ADD	LOAD	
4	MUL	SUB	ADD	LOAD
5		MUL	SUB	ADD

Pipelining: Data Hazards

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern

Takt	Befehl holen	Dekodieren/Operand holen	Ausführen	Abspeichern
1	LOAD			
2	ADD	LOAD		
3	SUB	ADD	LOAD	
4	MUL	SUB	ADD	LOAD
5		MUL	SUB	ADD

Problem:
In Takt 3 ist Register `R1` noch nicht mit dem richtigen Wert belegt!

Pipelining: Data Hazards - NOPs

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Lösungsidee: Einfügen von NOPs (NOP = no operation)

Takt	Befehl holen	Dekodieren/Operand holen	Ausführen	Abspeichern
1	LOAD			

Pipelining: Data Hazards - NOPs

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Lösungsidee: Einfügen von NOPs (NOP = no operation)

Takt	Befehl holen	Dekodieren/Operand holen	Ausführen	Abspeichern
1	LOAD			
2	NOP	LOAD		

Pipelining: Data Hazards - NOPs

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Lösungsidee: Einfügen von NOPs (NOP = no operation)

Takt	Befehl holen	Dekodieren/Operand holen	Ausführen	Abspeichern
1	LOAD			
2	NOP	LOAD		
3	NOP	NOP	LOAD	

Pipelining: Data Hazards - NOPs

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Lösungsidee: Einfügen von NOPs (NOP = no operation)

Takt	Befehl holen	Dekodieren/Operand holen	Ausführen	Abspeichern
1	LOAD			
2	NOP	LOAD		
3	NOP	NOP	LOAD	
4	ADD	NOP	NOP	LOAD

Pipelining: Data Hazards - NOPs

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Lösungsidee: Einfügen von NOPs (NOP = no operation)

Takt	Befehl holen	Dekodieren/Operand holen	Ausführen	Abspeichern
1	LOAD			
2	NOP	LOAD		
3	NOP	NOP	LOAD	
4	ADD	NOP	NOP	LOAD
5	SUB	ADD	NOP	NOP

Pipelining: Data Hazards - NOPs

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Lösungsidee: Einfügen von NOPs (NOP = no operation)

Takt	Befehl holen	Dekodieren/Operand holen	Ausführen	Abspeichern
1	LOAD			
2	NOP	LOAD		
3	NOP	NOP	LOAD	
4	ADD	NOP	NOP	LOAD
5	SUB	ADD	NOP	NOP

Korrekt:
In Takt 5 greift die `ADD` Operation auf den richtigen Wert in Register 1 zu, der in Takt 4 gespeichert wurde.

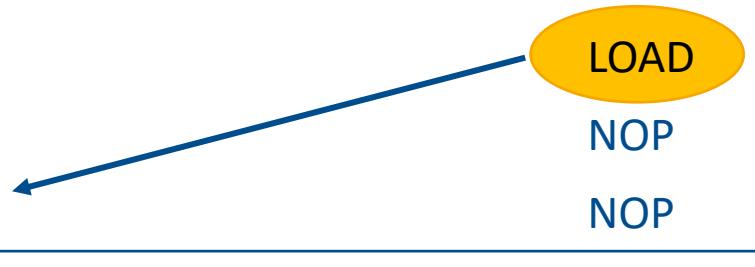
Pipelining: Data Hazards - Forwarding

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Lösungsidee: Einfügen von NOPs (NOP = no operation)

Beobachtung: Ergebnis der Execute (Ausführungs-)Phase direkt der ALU zur Verfügung zu stellen, spart einen NOP

Takt	Befehl holen	Dekodieren/Operand holen	Ausführen	Abspeichern
1	LOAD			
2	NOP	LOAD		
3	NOP	NOP	LOAD	
4	ADD	NOP	NOP	LOAD
5	SUB	ADD	NOP	NOP



Pipelining: Data Hazards - Forwarding

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Lösungsidee: Einfügen von NOPs (NOP = no operation)

Beobachtung: Ergebnis der Execute (Ausführungs-)Phase direkt der ALU zur Verfügung zu stellen, spart einen NOP

Takt	Befehl holen	Dekodieren/Operand holen	Ausführen	Abspeichern
1	LOAD			
2	NOP	LOAD		
3	NOP	NOP	LOAD	
4	ADD	NOP	NOP	LOAD
5	SUB	ADD	NOP	NOP

Pipelining: Data Hazards - Forwarding

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Lösungsidee: Einfügen von NOPs (NOP = no operation)

Beobachtung: Ergebnis der Execute (Ausführungs-)Phase direkt der ALU zur Verfügung zu stellen, spart einen NOP

Takt	Befehl holen	Dekodieren/Operand holen	Ausführen	Abspeichern
1	LOAD			
2	NOP	LOAD		
3	ADD	NOP	LOAD	
4	SUB	ADD	NOP	LOAD
5	MUL	SUB	ADD	NOP

Beachte:

Dazu ist Zusatzhardware notwendig, die diese Abhängigkeiten erkennt und das **Forwarding** durchführt.

Pipelining: Data Hazards – Umordnen von Befehlen

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Lösungsidee 2: Umordnen von Befehlen

Takt	Befehl holen	Dekodieren /Operand holen	Ausführen	Abspeichern
1	LOAD			
2	ADD	LOAD		
3	SUB	ADD	LOAD	
4	MUL	SUB	ADD	LOAD
5		MUL	SUB	ADD

Pipelining: Data Hazards – Umordnen von Befehlen

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Lösungsidee 2: Umordnen von Befehlen

Takt	Befehl holen	Dekodieren /Operand holen	Ausführen	Abspeichern
1	LOAD			
2	ADD	LOAD		
3	SUB	ADD	LOAD	
4	MUL	SUB	ADD	LOAD
5		MUL	SUB	ADD

Ursprüngliches Programm:

```
1. LOAD R1, 5 (R0);
2. ADD R3, R2, R1;
3. SUB R6, R5, R4;
4. MUL R9, R8, R7;
```

Pipelining: Data Hazards – Umordnen von Befehlen

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Lösungsidee 2: Umordnen von Befehlen

Takt	Befehl holen	Dekodieren /Operand holen	Ausführen	Abspeichern
1	LOAD			
2	ADD	LOAD		
3	SUB	ADD	LOAD	
4	MUL	SUB	ADD	LOAD
5		MUL	SUB	ADD

Ursprüngliches Programm:

```
1. LOAD R1, 5 (R0);
2. ADD R3, R2, R1;
3. SUB R6, R5, R4;
4. MUL R9, R8, R7;
```

Pipelining: Data Hazards – Umordnen von Befehlen

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Lösungsidee 2: Umordnen von Befehlen

Takt	Befehl holen	Dekodieren /Operand holen	Ausführen	Abspeichern

Ursprüngliches Programm:

```
1. LOAD R1, 5 (R0);

2. SUB R6, R5, R4;
3. MUL R9, R8, R7;
4. ADD R3, R2, R1;
```

Pipelining: Data Hazards – Umordnen von Befehlen

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Lösungsidee 2: Umordnen von Befehlen

Takt	Befehl holen	Dekodieren /Operand holen	Ausführen	Abspeichern

Ursprüngliches Programm:

```
1. LOAD R1, 5 (R0);
2. SUB R6, R5, R4;
3. MUL R9, R8, R7;
4. ADD R3, R2, R1;
```

Pipelining: Data Hazards – Umordnen von Befehlen

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Lösungsidee 2: Umordnen von Befehlen

Takt	Befehl holen	Dekodieren /Operand holen	Ausführen	Abspeichern
1	LOAD			

Ursprüngliches Programm:

```
1. LOAD R1, 5 (R0);
2. SUB R6, R5, R4;
3. MUL R9, R8, R7;
4. ADD R3, R2, R1;
```

Pipelining: Data Hazards – Umordnen von Befehlen

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Lösungsidee 2: Umordnen von Befehlen

Takt	Befehl holen	Dekodieren /Operand holen	Ausführen	Abspeichern
1	LOAD			
2	SUB	LOAD		

Ursprüngliches Programm:

```
1. LOAD R1, 5 (R0);
2. SUB R6, R5, R4;
3. MUL R9, R8, R7;
4. ADD R3, R2, R1;
```

Pipelining: Data Hazards – Umordnen von Befehlen

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Lösungsidee 2: Umordnen von Befehlen

Takt	Befehl holen	Dekodieren /Operand holen	Ausführen	Abspeichern
1	LOAD			
2	SUB	LOAD		
3	MUL	SUB	LOAD	

Ursprüngliches Programm:

```
1. LOAD R1, 5 (R0);
2. SUB R6, R5, R4;
3. MUL R9, R8, R7;
4. ADD R3, R2, R1;
```

Pipelining: Data Hazards – Umordnen von Befehlen

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Lösungsidee 2: Umordnen von Befehlen

Takt	Befehl holen	Dekodieren /Operand holen	Ausführen	Abspeichern
1	LOAD			
2	SUB	LOAD		
3	MUL	SUB	LOAD	
4	ADD	MUL	SUB	LOAD

Ursprüngliches Programm:

```
1. LOAD R1, 5 (R0);
2. SUB R6, R5, R4;
3. MUL R9, R8, R7;
4. ADD R3, R2, R1;
```

Pipelining: Data Hazards – Umordnen von Befehlen

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Lösungsidee 2: Umordnen von Befehlen

Takt	Befehl holen	Dekodieren /Operand holen	Ausführen	Abspeichern
1	LOAD			
2	SUB	LOAD		
3	MUL	SUB	LOAD	
4	ADD	MUL	SUB	LOAD
5		ADD	MUL	SUB

Ursprüngliches Programm:

```
1. LOAD R1, 5 (R0);
2. SUB R6, R5, R4;
3. MUL R9, R8, R7;
4. ADD R3, R2, R1;
```

Pipelining: Data Hazards – Umordnen von Befehlen

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Lösungsidee 2: Umordnen von Befehlen

Takt	Befehl holen	Dekodieren /Operand holen	Ausführen	Abspeichern
1	LOAD			
2	SUB	LOAD		
3	MUL	SUB	LOAD	
4	ADD	MUL	SUB	LOAD
5		ADD	MUL	SUB

Ursprüngliches Programm:

```
1. LOAD R1, 5 (R0);
2. SUB R6, R5, R4;
3. MUL R9, R8, R7;
4. ADD R3, R2, R1;
```

Pipelining: Data Hazards – Umordnen von Befehlen

Annahme: Vier Pipelinestufen (Befehl holen, Dekodieren, Ausführen und Abspeichern)

Lösungsidee 2: Umordnen von Befehlen

Takt	Befehl holen	Dekodieren /Operand holen	Ausführen	Abspeichern
1	LOAD			
2	SUB	LOAD		
3	MUL	SUB	LOAD	
4	ADD	MUL	SUB	LOAD
5		ADD	MUL	SUB

Ursprüngliches Programm:

```
1. LOAD R1, 5 (R0);
2. SUB R6, R5, R4;
3. MUL R9, R8, R7;
4. ADD R3, R2, R1;
```

Beachte:
Umordnen ist nicht beliebig möglich, es müssen Datenabhängigkeiten beachtet werden.

Pipelining: Control Hazards

- **Bedingte Verzweigungen (control hazards):**
  - Befehlsfolge:

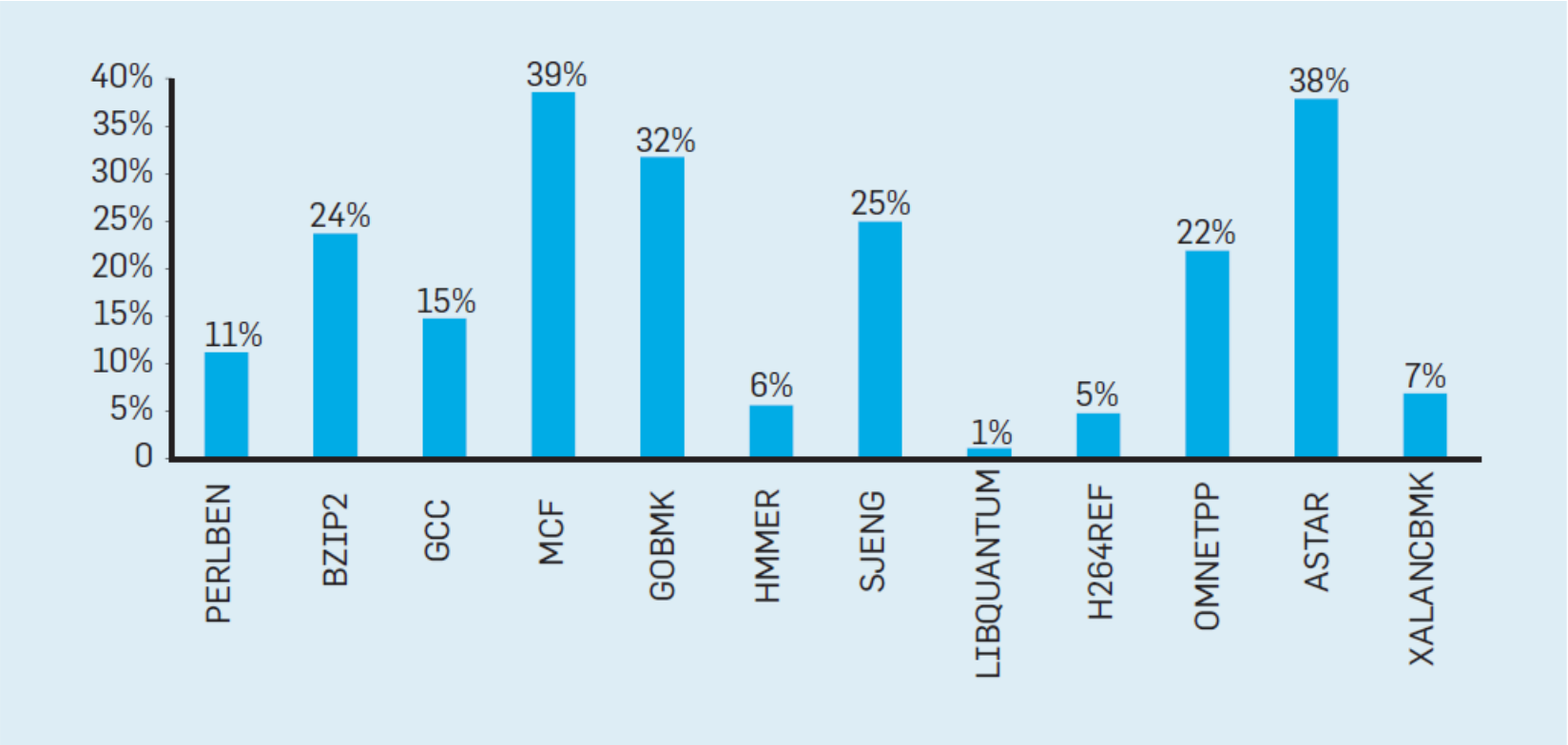
```
1. ADD R3, R2, R1;          // Addiere den Inhalt von R1 und R2, Erg. In R3
2. JMP>0 R4, <label>;       // Wenn Inhalt von R4 > 0, springe zu <label>
3. MUL R7, R6, R5;          // Multipliziere Inhalt von R6 und R5, Erg. in R7
```

  - Problem:
    Ausführung von `MUL` wird angefangen, bevor die Sprungbedingung ausgewertet wurde

  - **Lösungsidee**: Branch Prediction
    - „Raten“ der nächsten Instruktion (z.B. Analyse der Häufigkeit der möglichen Ausgänge der Abfrage)
    - Spekulativer Sprung
    - Leeren der Pipeline und „Rücksetzen“ bei falscher Vorhersage

Spekulative Ausführung

- z.T. hoher Anteil von Instruktionen, die „irrtümlich“ berechnet werden



Zusammenfassung

- Beschleunigung um bis zu Faktor k durch Einsatz einer Pipeline (k entspricht der Anzahl der Pipeline-Stufen)
- Hazards verringern die Beschleunigung
- Viele Möglichkeiten (z.T. in Hardware), Hazards zu vermeiden
  - **NOPs**
  - **Forwarding**
  - **Umstellen der Instruktionen** eines Maschinenprogramms durch Code-optimierende Compiler
  - **Branch Prediction** (Hardware merkt sich die letzte Ausführung des aktuellen Sprungbefehls)