

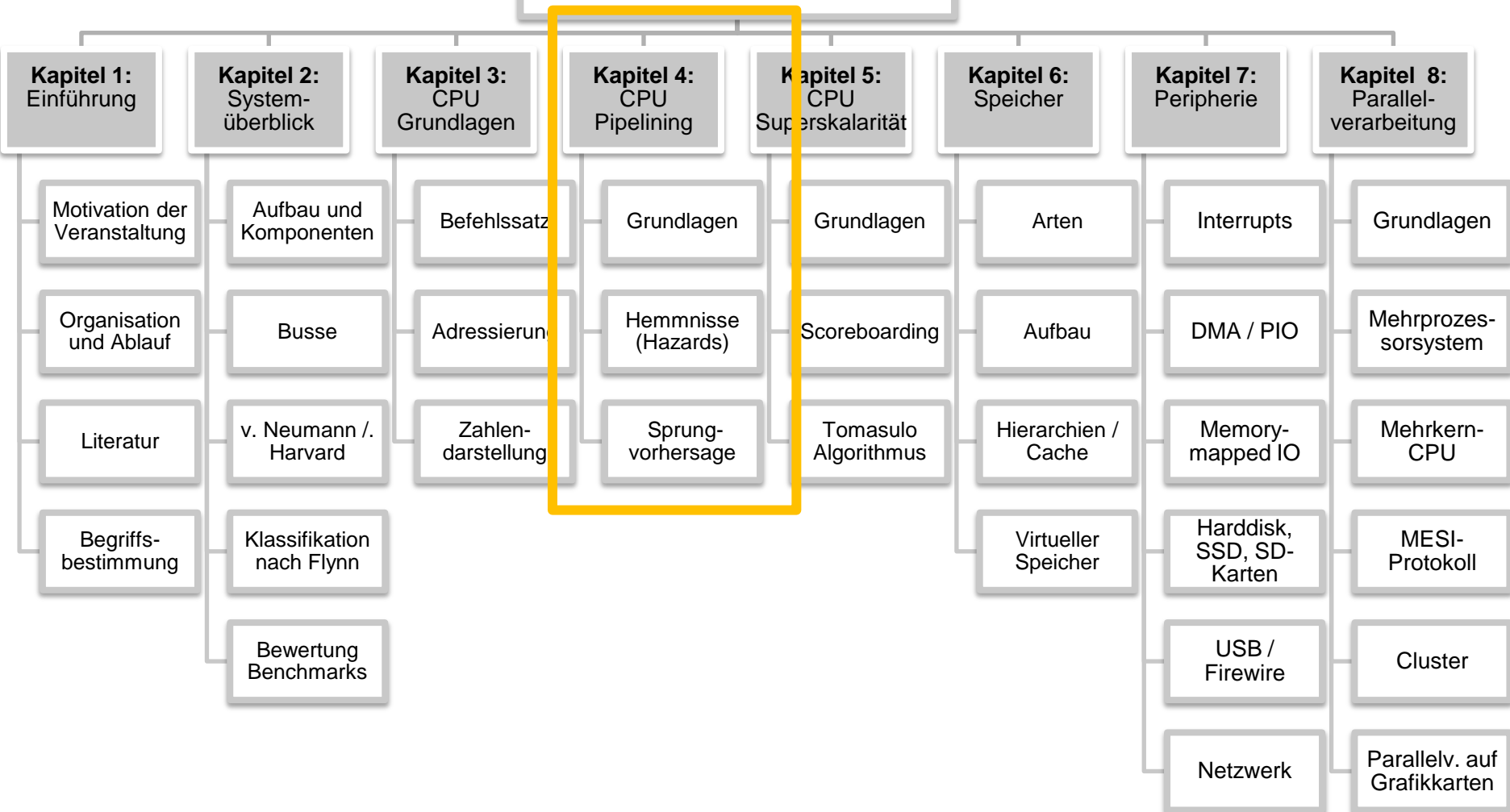
RECHNERARCHITEKTUR

Kapitel 4 – CPU: Pipelining

Prof. Dr. L. Wischhof <wischhof@hm.edu>

Fakultät 07 – Hochschule München

Rechnerarchitektur



CPU: Pipelining

Motivation

Typische Fragestellungen:

- Wie kann die Verarbeitungsgeschwindigkeit von Befehlen durch Pipelining gesteigert werden?
- Auf welche Teilschritte wird sinnvollerweise die Verarbeitung eines Befehls aufgeteilt?
- Welche ungünstigen Umstände (Hemmnisse) beeinflussen die Verarbeitung negativ? Wie können sie vermieden werden?
- Wie kann die CPU die Richtung eines Sprunges vorhersagen, um mit der Verarbeitung der nächsten Befehle zu beginnen?



CPU: Pipelining

Lernziele

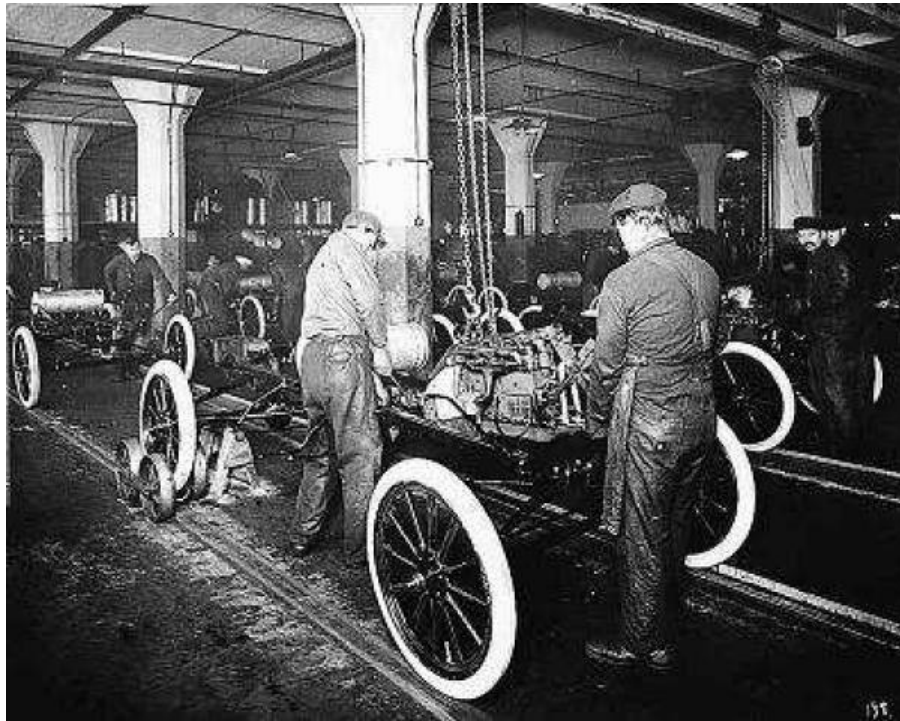
- Vorteile des Pipelinings benennen können
- Wesentliche Begriffe und Fakten zu Pipelining beherrschen
- Hemmnisse erkennen. Benennen können, ob diese durch Struktur, Ablauf oder Datenabhängigkeiten verursacht werden
- Zusammenhang zwischen Taktrate und Arbeitsschritten/Pipelinestufen benennen können
- Programmcode analysieren und die Ausführung auf einer fünfstufigen Pipeline erklären können
- Verfahren zur Sprungvorhersage benennen und anwenden können



Grundlagen

Was ist Pipelining?

„Eine CPU Pipeline ist wie ein Fließband, an dem die Instruktionen verarbeitet werden.“



Ford Werk,
um 1910



Grundlagen

Was ist Pipelining?

- Überlappende Ausführung mehrerer Instruktionen
- Parallelisierung der einzelnen Teilschritte der Befehlsausführung

➔ **Schlüsseltechnik** zur Implementierung einer schnellen CPU!

Begriffe:

- **Pipeline-Stufe:** Abschnitt der Pipeline in welchem ein Teilschritt der Instruktion durchgeführt wird
- **Pipeline-Durchsatz:** Anzahl Befehle pro Zeiteinheit, die die Pipeline verlassen



Grundlagen

Was ist Pipelining?



- Pipeline-Stufen sind verbunden, Übergang zur jeweils nächsten Stufe zum selben Zeitpunkt
- Dauer um eine Stufe zu durchqueren: **Processor Cycle** (entspricht in der Regel dem Takt)
- Langsamste Stufe bestimmt Processor Cycle
→ Ziel ist möglichst ausgeglichene Dauer der Stufen



Grundlagen

Beschleunigte Verarbeitung (ideal)

Unter **idealen Bedingungen** wäre die Zeit zur Ausführung einer Instruktion für eine CPU mit Pipeline

$$\frac{\text{Zeit pro Instruktion bei CPU ohne Pipeline}}{\text{Anzahl Pipeline-Stufen}}$$

→ Beschleunigung um den Faktor „Anzahl Pipeline-Stufen“

Vorteil von Pipelining verglichen mit anderen Beschleunigungstechniken: **für Programmierer „unsichtbar“** (keine Anpassung des Programmes notwendig!)



Grundlagen

Klassische 5-Stufen RISC-Pipeline (1/2)

Abarbeitung des Befehls in fünf Teilschritten:

1. **Holen des Befehls (Fetch, kurz F)**
Befehlszähler liefert Adresse für Speicherzugriff und wird um vier erhöht (bei 32-Bit Befehlslänge).
2. **Decodieren des Befehls (Decode, kurz D)**
 - Bereitstellen von Operanden aus Registern
 - Vorzeichenerweiterung des Sprungoffsets
3. **Ausführen des Befehls (Execute, kurz X)**
 - Im Befehl spezifizierte ALU-Operation
 - Bei Speicherzugriff: Adresse berechnen
 - Bei Verzweigung: ggf. Bedingung prüfen, Adresse des Folgebefehls berechnen



Grundlagen

Klassische 5-Stufen RISC-Pipeline (2/2)

4. **Speicherzugriff (Memory Access, kurz M)**
Lesender oder schreibender Zugriff auf den Speicher
(wenn nötig)
5. **Zurückschreiben des Ergebnisses (Write-Back, kurz W)**
Ergebnis der Operation wird in ein Register geschrieben
(wenn nötig)

Frage dazu:

Welche der Ihnen bekannten MMIX-Befehle

- *besitzen eine leere M-Stufe?*
- *besitzen eine leere W-Stufe?*



Grundlagen

Pipeline-Modell (1/3)

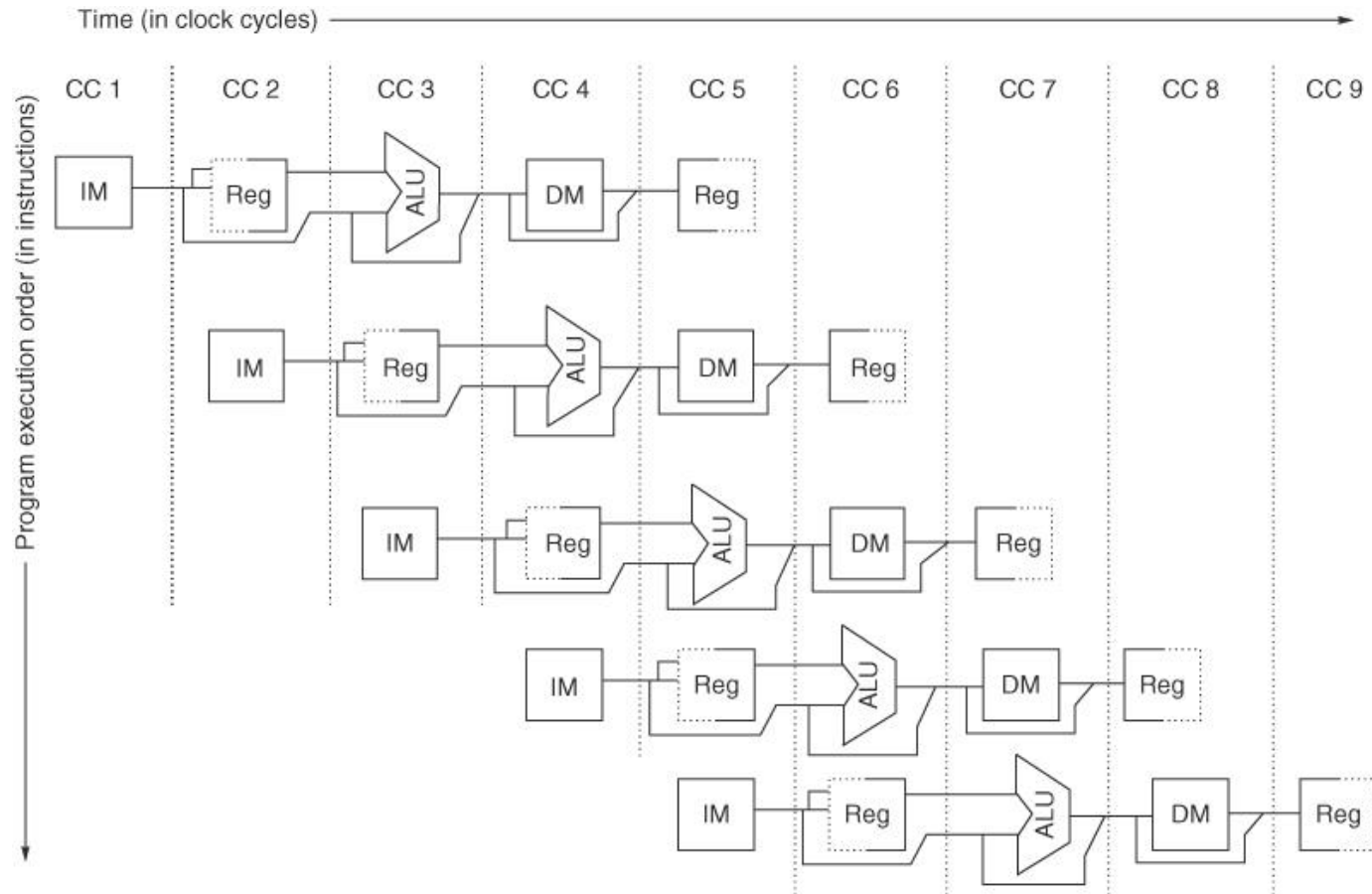
Zeitlich verschobene Datenpfade zur Modellierung des Ablaufs:

- Darstellung der **Verarbeitungszustände der Befehle** über der Zeit
- Komponenten entsprechen einzelnen Stufen:
 1. **Instruction Memory** (IM)
 2. **Register File** (Reg)
(als Quelle, Linie links gestrichelt)
 3. **Arithmetic Logic Unit** (ALU)
 4. **Data Memory** (DM)
 5. **Register File** (Reg)
(als Ziel, Linie rechts gestrichelt)



Grundlagen

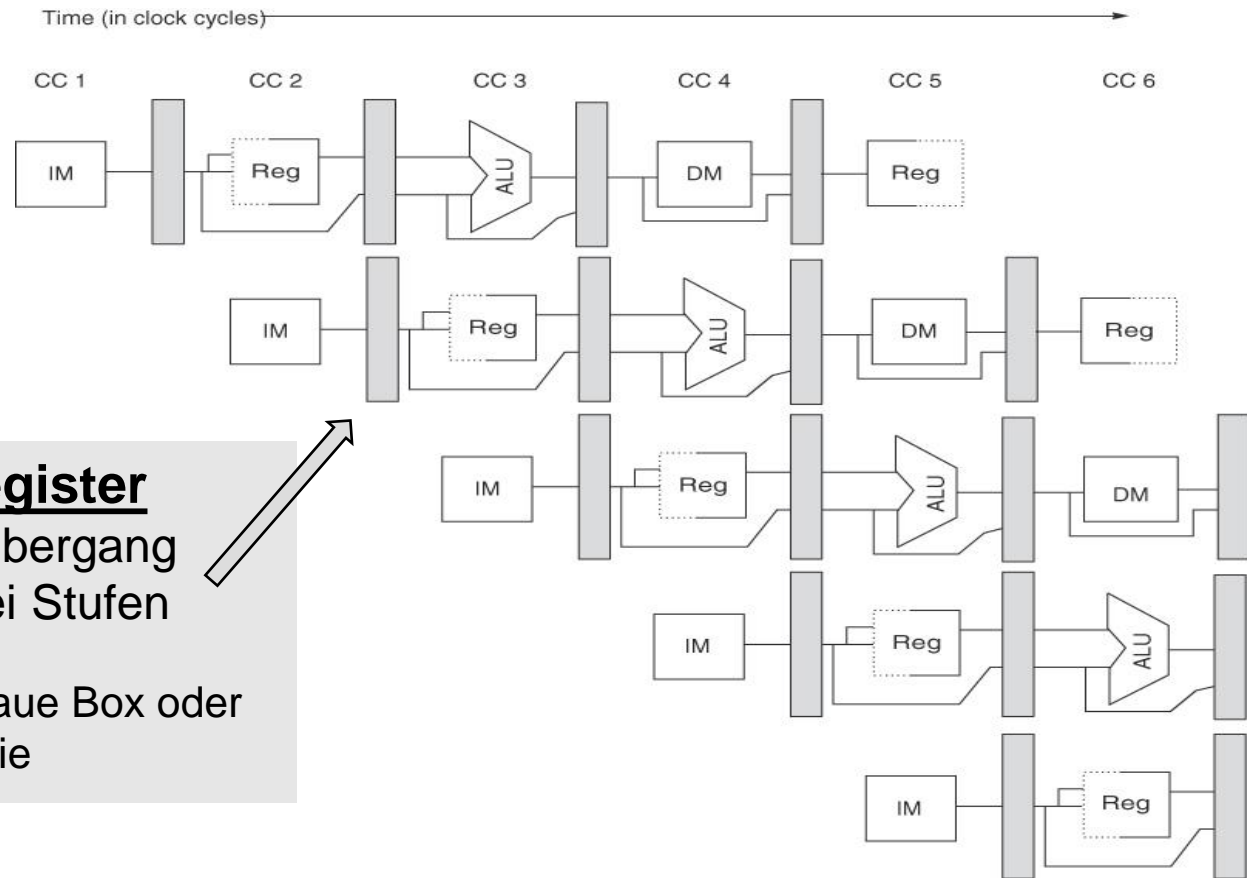
Pipeline-Modell (2/3)



Quelle: [2, S. C-8]

Grundlagen

Pipeline-Modell (3/3)



Pipeline-Register

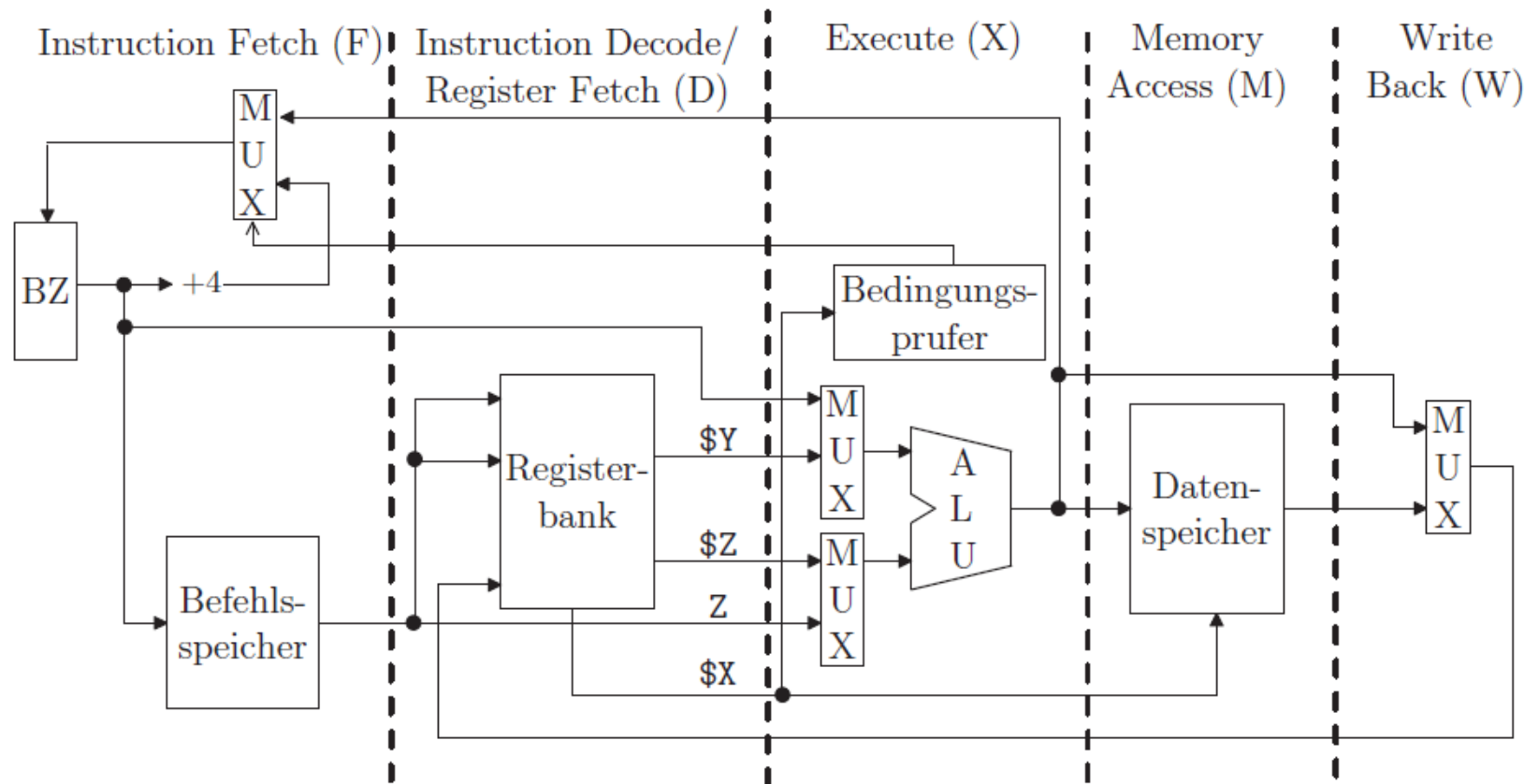
Geordneter Übergang
zwischen zwei Stufen

Darstellung: graue Box oder
gestrichelte Linie

Quelle: [2, S. C-9]

Grundlagen

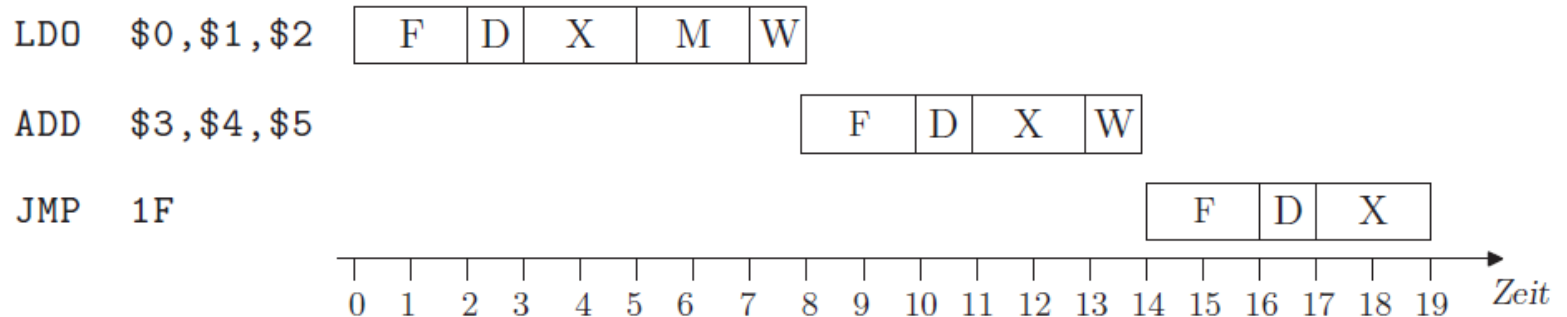
Pipeline-Implementierung am Beispiel MMIX



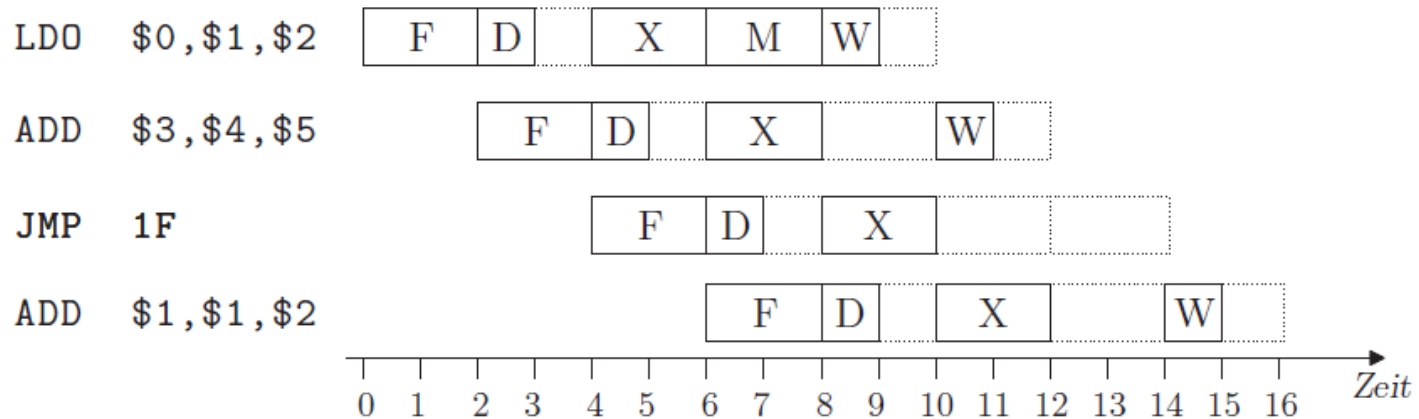
Grundlagen

Beispiel: Verarbeitung mit/ohne Pipelining

Ohne Pipelining:



Mit Pipelining:



Grundlagen

Pipeline-Diagramme (Reservation Tables): Stufen

Es gibt zwei Arten von Pipeline-Diagrammen, die jeweils ineinander überführt werden können.

1. Darstellung mit den Stufen als Zeilen:

	Takt 1	Takt 2	Takt 3	Takt 4	Takt 5	Takt 6
Fetch	Befehl 1	Befehl 2	Befehl 3	Befehl 4	Befehl 5	Befehl 6
Decode		Befehl 1	Befehl 2	Befehl 3	Befehl 4	Befehl 5
Execute			Befehl 1	Befehl 2	Befehl 3	Befehl 4
Memory				Befehl 1	Befehl 2	Befehl 3
Write Back					Befehl 1	Befehl 2



Grundlagen

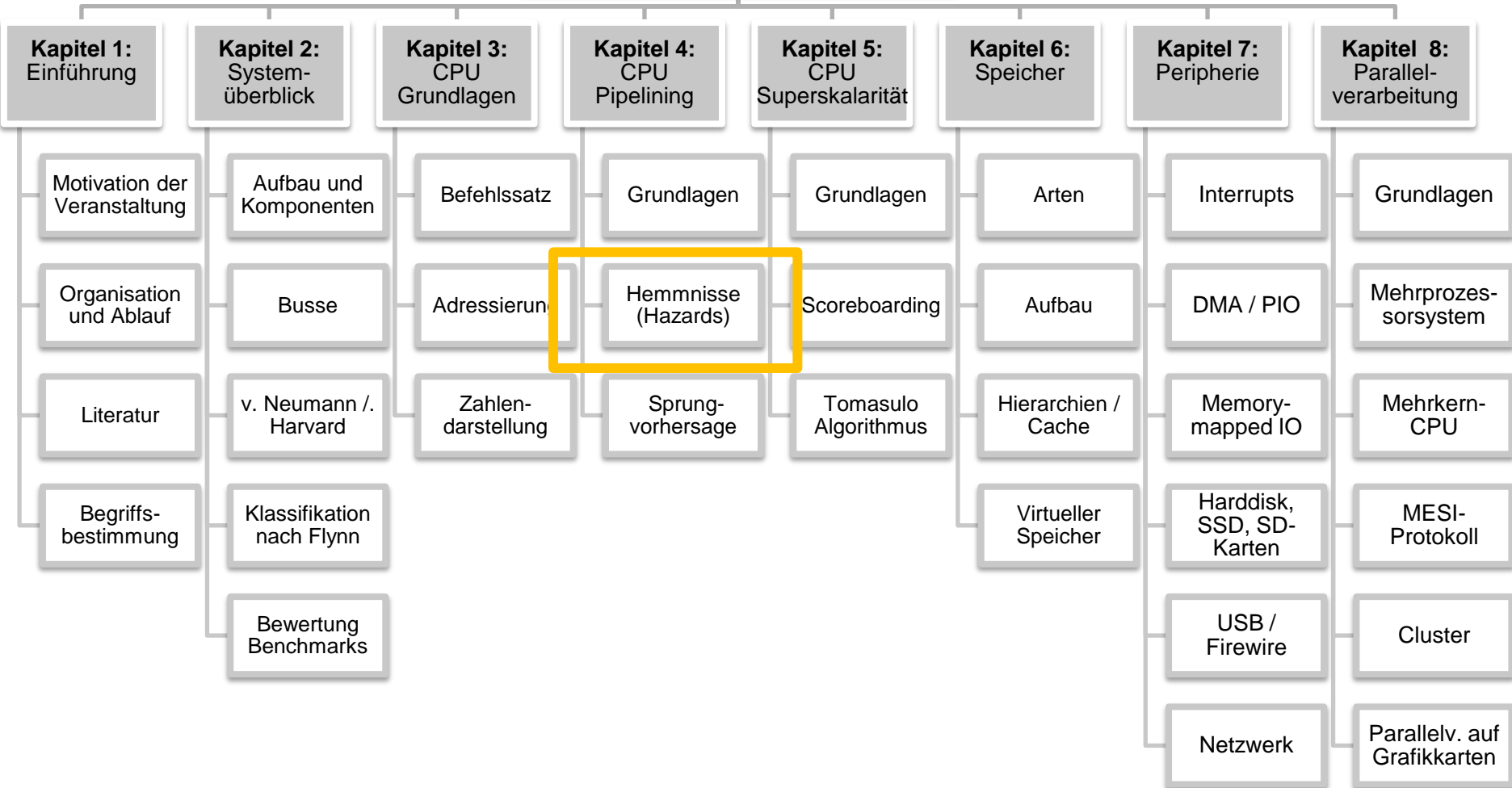
Pipeline-Diagramme (Reservation Tables): Befehle

2. Darstellung mit den Befehlen als Zeilen:
(vergleichbar mit Beispiel auf Folie 15)

	Takt 1	Takt 2	Takt 3	Takt 4	Takt 5	Takt 6
Befehl 1	F	D	X	M	W	
Befehl 2		F	D	X	M	W
Befehl 3			F	D	X	M
Befehl 4				F	D	X
Befehl 5					F	D



Rechnerarchitektur



Pipeline Hemmnisse (Hazards)

Übersicht

Pipeline Hemmnis (Hazard)

Situation, die verhindert, dass ein Befehl seine aktuelle Pipelinestufe ausführt.

- ➔ Pipeline muss teilweise angehalten werden (Stall)
- ➔ Reduktion der Verarbeitungsgeschwindigkeit

Es werden drei Arten unterschieden:

- **Strukturelle Hemmnisse (Structural Hazards)**
verursacht durch Ressourcen-Konflikte wenn Hardware nicht alle Kombinationen der überlappenden Ausführung unterstützt.
- **Hemmnisse durch Datenabhängigkeiten (Data Hazards)**
Abhängigkeit des Ergebnisses der Instruktion von einer vorherigen
- **Ablaufbedingte Hemmnisse (Control Hazards)**
verursacht durch Änderungen des Programmablaufs (z.B. Sprung)



Pipeline Hemmnisse (Hazards)

Structural Hazard: Beispiel Speicher

Wenn nur ein Interface zum Speicher besteht:
Es kann nicht gleichzeitig auf Befehle und Daten zugegriffen werden.

Takt 1	Takt 2	Takt 3	Takt 4	Takt 5	Takt 6	Takt 7
F	D	X	M	W		
	F	D	X	M	W	
		F	D	X	M	W
						F

(rot: Markierung des Hazards)

Abhilfe:

- Befehle werden üblicherweise im Voraus gelesen („Prefetching“)
- Verwendung getrennter Caches für Code und Daten



Pipeline Hemmnisse (Hazards)

Structural Hazard: Beispiel komplexer Befehl

Lange laufende Instruktion (z.B. Gleitkomma-Instruktion) belegt Ausführungseinheit über mehrere Takte:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
SETH $xk, \#4000$	F	D	X	M	W									
FMUL yk, yk, xk		F	D	X	X	X	X	M	W					
FADD yk, yk, q			F	D				X	X	X	X	M	W	
SET $xk, templ$				F				D				X	M	W



Pipeline Hemmnisse (Hazards)

Data Hazards: Beispiel (1/2)

100		XOR	l, l, r	Tauschen von l und r
101		XOR	r, l, r	
102		XOR	l, l, r	
103	1H	CMP	tmp, l, pivot	

Problem:

Ergebnis aus Zeile 100 steht erst am Ende der W-Phase im Register mit dem Alias `l` zur Verfügung, wird aber sofort in Zeile 101 benötigt.

Man spricht von einem **Read-After-Write (RAW)** Konflikt.

Abhilfe:

- Ergebnis von X- und M-Phase als ALU-Input zurückreichen
- Falls notwendig, werden diese zurückgereichten Werte anstelle der Werte aus den Registern verwendet (**Forwarding**, Bypassing)



Pipeline Hemmnisse (Hazards)

Data Hazards: Beispiel (2/2)

	Takt 1	Takt 2	Takt 3	Takt 4	Takt 5	Takt 6
XOR l,l,r	F	D	X	M	W	
XOR r,l,r		F	D			X
XOR l,l,r			F			D
CMP tmp,l,pivot						F

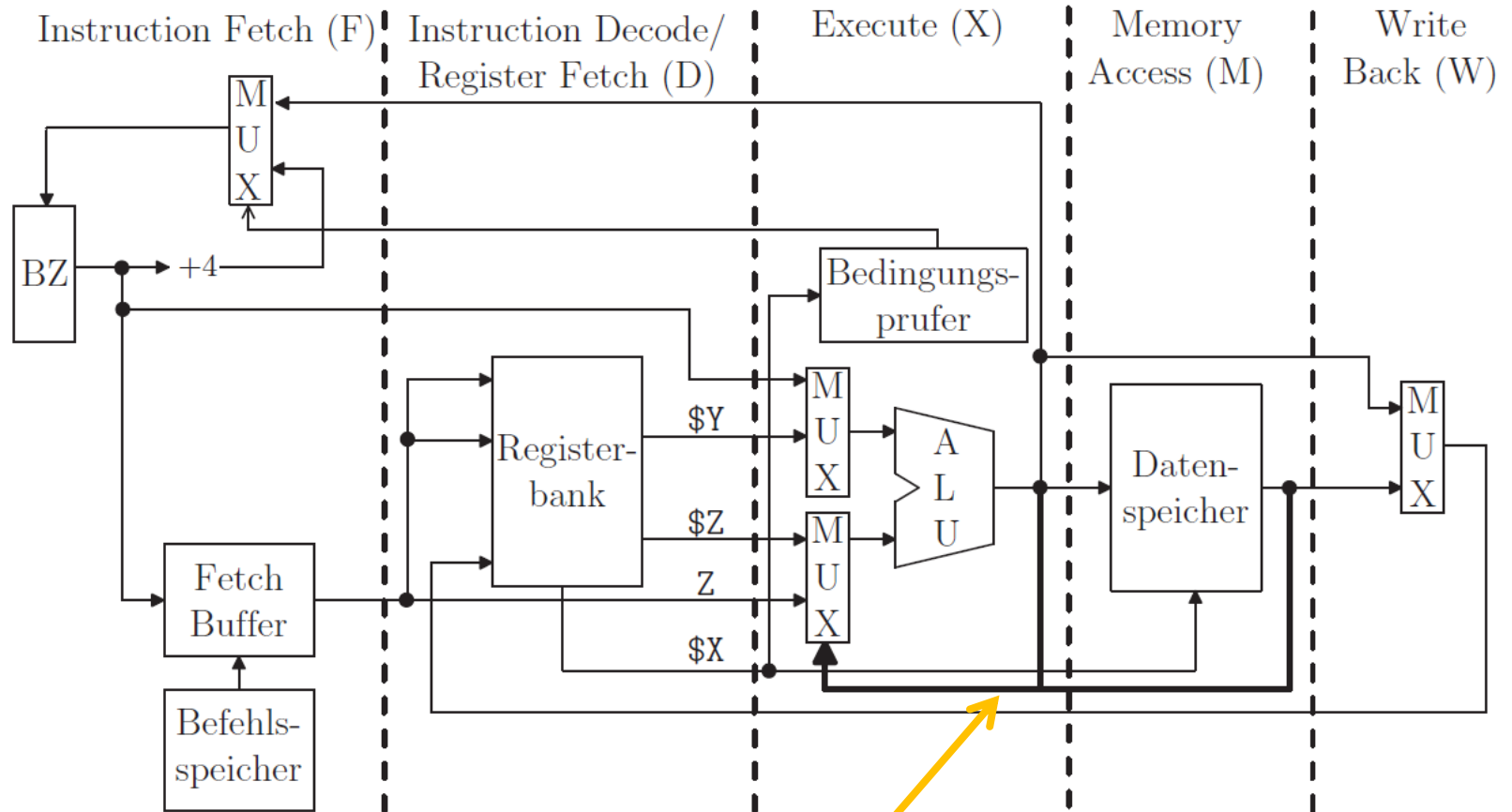
Einbau von Result-Forwarding

	Takt 1	Takt 2	Takt 3	Takt 4	Takt 5	Takt 6
XOR l,l,r	F	D	X	M	W	
XOR r,l,r		F	D	X	M	W
XOR l,l,r			F	D	X	M
CMP tmp,l,pivot				F	D	X



Pipeline Hemmnisse (Hazards)

Pipeline-Struktur mit Forwarding



Einbau von Result-Forwarding

Pipeline Hemmnisse (Hazards)

Data Hazards: Ladeoperationen

ABER:

Forwarding hilft nur teilweise bei Ladeoperationen, da auf die Bereitstellung der Daten aus dem Speicher gewartet werden muss.

	Takt 1	Takt 2	Takt 3	Takt 4	Takt 5	Takt 6
LDO \$1,base,off	F	D	X	M	W	
ADD \$1,\$1,2		F	D		X	M
SUB \$4,\$4,\$5			F		D	X

Wenn Daten nicht im Cache vorhanden sind, können hier **Stalls von bis zu hundert Taktzyklen** auftreten!

(→ vergl. Kapitel „Speicher“)



Pipeline Hemmnisse (Hazards)

Data Hazards: Abhängigkeiten zwischen Befehlen

	Beispiel	Beschreibung
Read After Write (RAW)	ADD \$1, \$2, \$3 SUB \$0, \$4, \$1	Auch „essential dependency“, gelöst durch Result-Forwarding.
Write After Read (WAR)	STO \$1, \$2, \$3 ADD \$1, \$4, \$5	Auch „ordering dependency“ oder „anti-dependency“
Write After Write (WAW)	DIV \$0, \$1, \$2 GET \$0, rR	Auch „output dependency“
Read After Read (RAR)	ADD \$1, \$2, \$3 SUB \$0, \$2, \$3	Stellt normalerweise kein Problem da.

Konflikte/Abhängigkeiten können – müssen aber nicht immer – zu Hazards führen, d.h. nachfolgende Befehle aufhalten.



Pipeline Hemmnisse (Hazards)

Control Hazards: Beispiel (1/2)

100	LDA	Off, Size	
101	STBU	Dat, Off	
102	SRU	Dat, Dat, 8	
103	INCL	Off, 1	
104	BNZ	Dat, @-12	Sprung zu Z. 101
105	SETL	Dat, Wert	
106	GETA	\$255, 6B	

Problem:

Bei Sprungbefehl (Zeile 104) kann Prefetch-Mechanismus Befehle aus der falschen Verzweigungsrichtung holen.



Pipeline Hemmnisse (Hazards)

Control Hazards: Beispiel (2/2)

Sprung in Zeile 104 wird ausgeführt, obwohl „non-probable branch“:

	1	2	3	4	5	6	7	8	9
STBU Dat, Off	F	D	X	M	W				
SRU Dat, Dat, 8		F	D	X	M	W			
INCL Off, 1			F	D	X	M	W		
BNZ Dat, @-12				F	D	X	M	W	
SETL Dat, Wert					F	D			
GETA \$255, 6B						F			
STBU Dat, Off							F	D	X

Abhilfe:

- Sprungvorhersage (siehe folgender Abschnitt der Vorlesung)
- Delayed Branches (eingeschränkt, siehe folgende Folie)



Pipeline Hemmnisse (Hazards)

Control Hazards: Delayed Branches

Technik zur Reduktion der Nachteile von Control Hazards durch (falsch vorhergesagte) Sprünge.

Idee:

Der unmittelbar auf einen bedingten Sprung folgende Befehl wird – **unabhängig vom Verlauf des Sprungs** – noch ausgeführt.

(➔ Anpassung am Programm notwendig, z.B. durch Compiler)

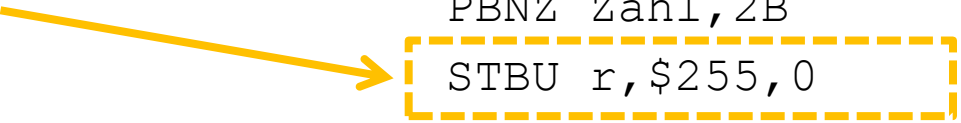


Pipeline Hemmnisse (Hazards)

Control Hazards: Delayed Branches – Beispiel 1/3

```
2H SUB    $255,$255,1
      DIVU  Zahl,Zahl,10
      GET   r,:rR
      INCL  r,'0`
      STBU  r,$255,0
      PBNZ  Zahl,2B
      ADD   a,a,1
```

```
2H SUB    $255,$255,1
      DIVU  Zahl,Zahl,10
      GET   r,:rR
      INCL  r,'0`
      PBNZ  Zahl,2B
      STBU  r,$255,0
      ADD   a,a,1
```



Der bereits in der Pipeline befindliche Befehl STBU wird in jedem Fall ausgeführt (auch wenn „Fetch“ falsche Sprungrichtung annimmt).



Pipeline Hemmnisse (Hazards)

Control Hazards: Delayed Branches – Beispiel 2/3

Pipeline-Diagramm für das Beispiel wenn PBNZ nicht ausgeführt

OHNE Delay Slot:

*Fetch
für
Sprung:
in
diesem
Fall
falsch*

	1	2	3	4	5	6	7	8	9
GET r,:rR	F	D	X	M	W				
INCL r,'0'		F	D	X	M	W			
STBU r,\$255,0			F	D	X	M	W		
PBNZ Zahl,2B				F	D	X	M	W	
SUB \$255,\$255,1					F	D			
DIVU Zahl,Zahl,10						F			
ADD a,a,1							F	D	X



Pipeline Hemmnisse (Hazards)

Control Hazards: Delayed Branches – Beispiel 3/3

Pipeline-Diagramm für das Beispiel wenn PBNZ nicht ausgeführt

MIT Delay Slot:

	1	2	3	4	5	6	7	8	9
GET r, :rR	F	D	X	M	W				
INCL r, '0'		F	D	X	M	W			
PBNZ Zahl, 2B			F	D	X	M	W		
STBU r, \$255, 0				F	D	X	M	W	
SUB \$255, \$255, 1					F				
ADD a, a, 1						F	D	X	M
...							F	D	X

➔ Mit Delay Slot Auswirkung des Hazards um 1 Takt reduziert



Pipeline Hemmnisse (Hazards)

Control Hazards: Delayed Branches – Nachteile

Erfordert besondere Sorgfalt beim Programmieren:

Wenn kein Befehl für Delay-Slot laut Programmablauf geeignet ist, muss ein wirkungsloser Befehl (NoOp-Befehl) eingefügt werden.

Beispiel:

```
1H INCL  step,1
    SET   12,Tsize
    CMP   11,step,12
    BZ    11,voll
    SWYM  0
```



Pipeline Hemmnisse (Hazards)

Control Hazards: Behandlung von Interrupts

- Externe Interrupts fordern schnelle Reaktion
→ Verzweigung in den Interrupt-Handler
- Befehle, die bereits einen Teil der Zustandsänderungen bewirkt haben, dürfen nicht unterbrochen werden!

Beispiel:

Ein Befehl der in den Speicher schreibt darf nicht nach der M- und vor der W-Phase unterbrochen werden, da er möglicherweise das Spezialregister rA ändert. Nach der M-Phase hätte er den Wert in den Speicher übertragen, aber rA noch nicht geändert!

- Befehle die gerade erste dekodiert werden können jedoch wieder verworfen werden.
→ gezieltes Leerlaufen-Lassen der Pipeline (Pipeline Draining)



Pipeline Hemmnisse (Hazards)

Performance mit Stalls

Wie wirken sich Hazards auf die Beschleunigung durch das Pipelining aus?

Annahme: CPI von Prozessor ist 1 (ideale CPI), dann

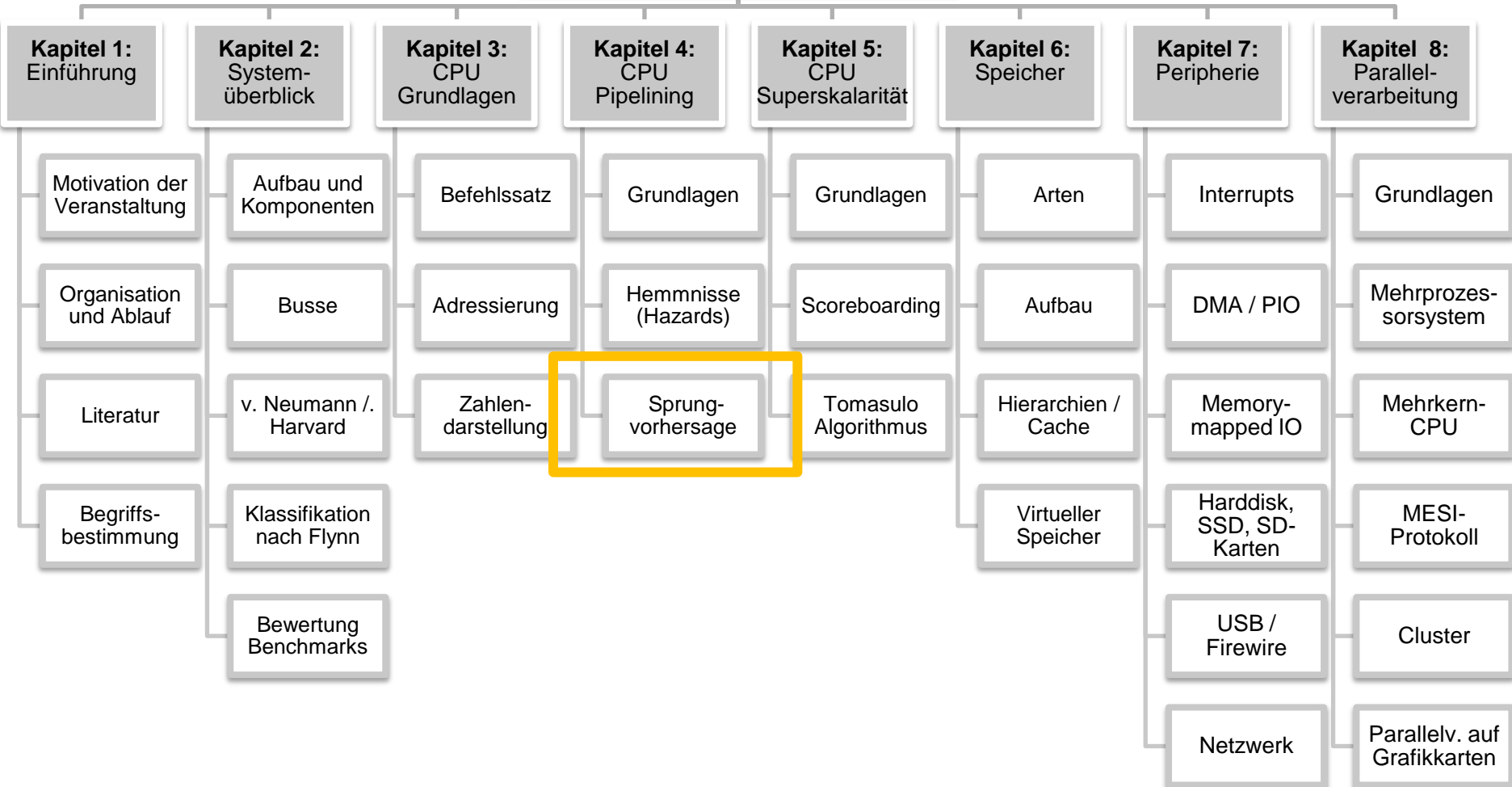
$\text{CPI pipelined} = 1 + \text{pipeline stall clock cycles per instruction}$

Vernachlässigt man den Overhead für das Pipelining und geht von einer ausgeglichenen Pipeline (jede Stufe gleich lang) aus, so ergibt sich damit:

$$\text{speedup} = \frac{\text{CPI unpipelined}}{1 + \text{pipeline stall clock cycles per instruction}}$$



Rechnerarchitektur



Sprungvorhersage

Motivation

Bedingte Sprünge sind für das Pipelining problematisch:

- Pipeline ist befüllt mit Befehlen einer Ablaufvariante
 - Branch Taken (T): Verzweigung/Sprung *wird* ausgeführt
 - Branch Not Taken (N): Verzweigung/Sprung *wird nicht* ausgeführt
- Tritt die andere Ablaufvariante ein (falsche Vorhersage) dann muss die Pipeline neu befüllt werden
 - ➔ Sprungverzögerung (**Misprediction Penalty**)
- Längere Pipeline führt in der Regel zu größerer Misprediction Penalty
- Moderne Prozessoren haben relativ lange Pipeline

➔ Gute **Sprungvorhersage** ist entscheiden für **Performance**!



Grundlagen

Leistungseinbußen durch falsch vorhergesagte Sprünge (1/2)

Wie verlängert sich die Laufzeit des Programms?

Parameter

b : branch rate

m : misprediction rate (=1-prediction rate)

p : penalty (Strafe) für falsch vorhergesagte Sprünge in Takten

Laufzeit T eines Programms mit n Befehlen:

(Einschwingen der Pipeline beim Start vernachlässigt)

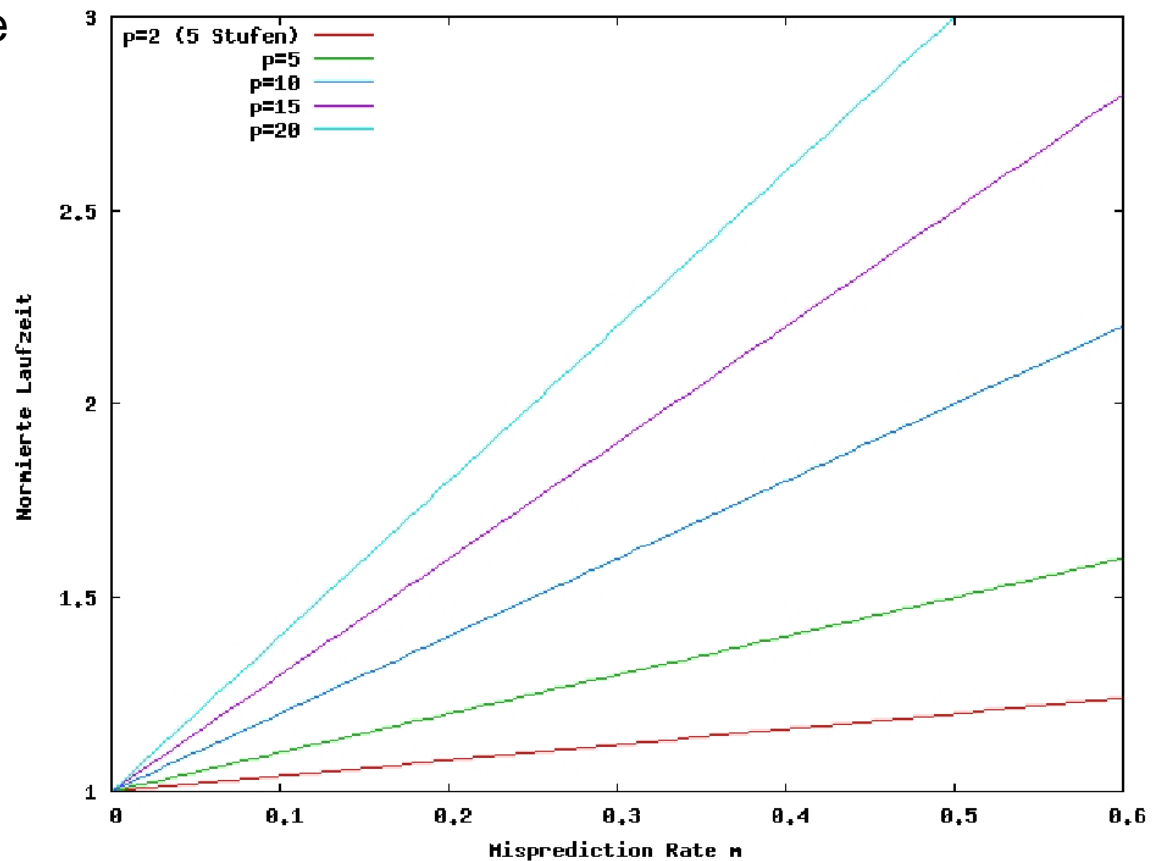
$$T = n(1 + bmp)$$



Grundlagen

Leistungseinbußen durch falsch vorhergesagte Sprünge (2/2)

Für eine Branch-Rate von $b=0.2$ ergibt sich für unterschiedliche Werte der Misprediction Penalty p :



Grundlagen

Reduzierung der Sprungverzögerung – Übersicht

Mögliche Maßnahmen:

- **Sprungvermeidung**

Ersatz von Sprungkonstrukten durch andere Befehle

- **Sprungvorhersage**

Je länger die Pipeline, desto „teurer“ sind falsch vorhergesagte Sprünge (Branch Penalty/Misprediction Penalty)!

→ CPU treibt viel Aufwand zur richtigen Vorhersage der Sprünge

Zwei Arten werden unterschieden:

- **Statische Sprungvorhersage**
- **Dynamische Sprungvorhersage**



Sprungvermeidung

Grundlagen

Vermeidung von Sprüngen (Software-Technik) durch:

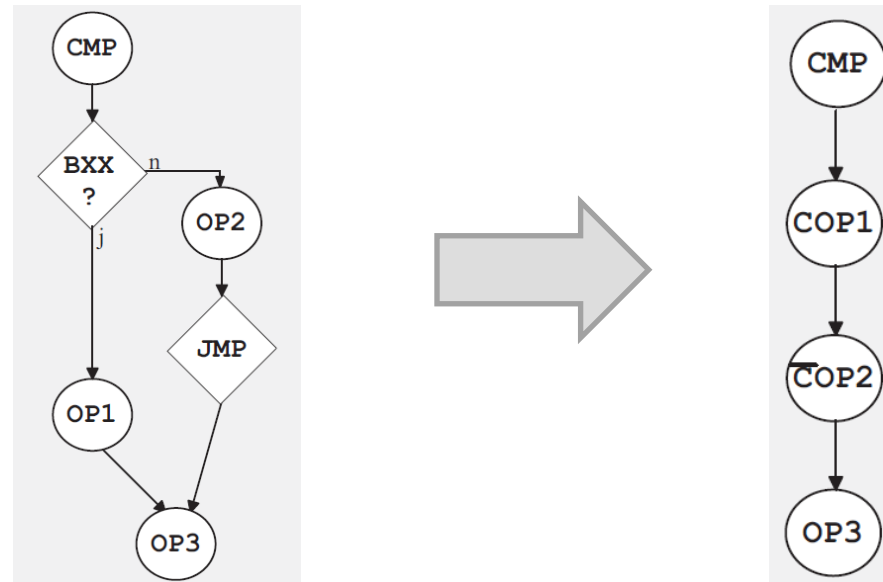
- **Verwendung Bedingte Befehle**, soweit möglich
- **Loop Unrolling/Loop Unwinding**
 - Umschreiben von Schleifen in wiederholte Befehlssequenzen
 - Vermindert Sprünge, erhöht jedoch die Codegröße
 - In der Regel automatisch durch Compiler durchgeführt (für innere Schleifen, Schleifen mit wenigen Iterationen)



Sprungvermeidung durch Bedingte Befehle

Beispiel


Aus Kapitel 3 ist bekannt, dass ein bedingter Sprung durch bedingte Befehle ersetzt werden kann:



Sprungvermeidung durch Loop Unrolling

Beispiel

Dreifach geschachtelte Schleife:



```
for (int i=0; i < 3; i++)  
    for (int j = 0; j < 3; j++)  
        for (int k = 0; k < 3; k++)  
            m3[i][j] += m1[i][k] * m2[k][j];
```

Loop Unrolling für die innere Schleife:

```
for (int i = 0; i < 3; i++)  
    for (int j = 0; j < 3; j++)  
        m3[i][j] =  
            m1[i][0] * m2[0][j]  
            + m1[i][1] * m2[1][j]  
            + m1[i][2] * m2[2][j];
```



Statische Sprungvorhersage

Grundlagen

Idee: Vorhersage wird dem Programmierer/Compiler überlassen

- Ein Bit im Opcode gibt an, ob die Verzweigung wahrscheinlich ausgeführt wird („take/don't take Bit“).
- Bei MMIX: „probable branches“ und „non probable branches“
Opcodes unterscheiden sich am Bit 4 (Wertigkeit $2^4=16=\#10$)
z.B.: BZ (Branch if Zero) hat Wert #42 und PBZ (Probable Branch if Zero) #52
- Besonders geeignet für Zählschleifen
- Eingesetzt bei: PowerPC, Alpha, MMIX



Statische Sprungvorhersage

Beispiel: Prediction Rate bei Quicksort

Messung: Quicksort zum Sortieren von 10.000 zufälligen Werten. Unterprogramm wird 1551 Mal aufgerufen, sortiert im Mittel 5,44 Werte.

Trefferraten (Prediction Rates):

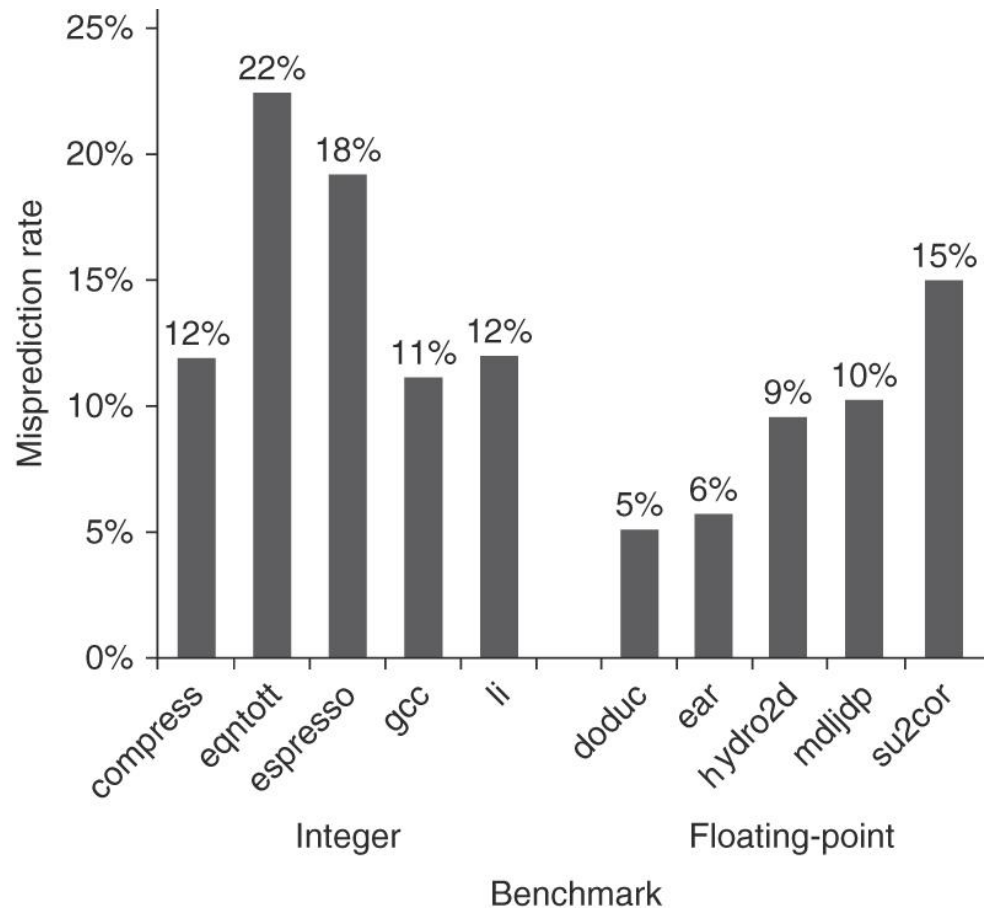
Zeile	Taken	Not Taken	Trefferrate
23	2059	16345	88,8%
27	11505	4840	70,4%
32	6899	1551	81,6%

Quelltext des Programms: [1], Anhang A.2, Seite 191



Statische Sprungvorhersage

Beispiel: Misprediction-Rate bei SPEC 92



Misprediction Rate bei Floating-Point-Programmen ist deutlich geringer als bei Integer-Programmen

Quelle: [2, S. C-27]

Dynamische Sprungvorhersage

Grundlagen

Idee:

- Vorhersage hängt von der Vergangenheit ab.
- Zu jeder Adresse eines Verzweigungsbefehls (Modulo einer bestimmten Speichergröße a) speichert der Prozessor einen Prädiktor.
- Prädiktoren werden in kleinem, über die hinteren Adressbits indizierten Speicher abgelegt (Branch-Prediction Buffer, Branch History Table)



Dynamische Sprungvorhersage

Verfahren

- **Vorhersage mit Zählern**
(1-Bit Predictor, 2-Bit Predictor, n-Bit Predictor)
- **Vorhersage unter Berücksichtigung der Vorgeschichte**
bzw. in Abhängigkeit von anderen Sprüngen
(Correlating Predictors, Two-Level Predictors)
- **Adaptive Kombination** lokaler und globaler Prädiktoren
(Tournament Predictors)
hier nicht weiter behandelt



Dynamische Sprungvorhersage

MMIX Testprogramm

Beispiel-Testprogramm `simple.mms`:

1	YesNo	GREG	#AAAAAAAAAAAAAAAAAAAA
---	-------	------	-----------------------

2

3	P	IS	\$2
---	---	----	-----

4	q	IS	\$3
---	---	----	-----

5

6		LOC	#100
---	--	-----	------

7	Main	PBEV	YesNo,1F
---	------	------	----------

8		ADD	P,P,1
---	--	-----	-------

9	1H	AND	q,YesNo,1
---	----	-----	-----------

10		SLU	q,q,63
----	--	-----	--------

11		SRU	YesNo,YesNo,1
----	--	-----	---------------

12		OR	YesNo,YesNo,q
----	--	----	---------------

13		JMP	Main
----	--	-----	------

*Vorgabe eines Sprungverhaltens
durch Bitvektor*

Misserfolge zählen
rotieren



Dynamische Sprungvorhersage

1-Bit Prädiktor

Vorhersage: nächste Ausführung so, wie die zuletzt beobachtete.

„0“ bedeutet: Verzweigungsrichtung wird übereinstimmend mit der (statisch im Op-Code kodierten) Vorhersage eingeschlagen („in Agreement“, kurz A)

„1“ bedeutet: Verzweigungsrichtung wird entgegen der (statisch im Op-Code kodierten) Vorhersage eingeschlagen („in Opposition“, kurz O)

Nach dem Sprung wird das Bit des Prädiktors entsprechend des beobachteten Verhaltens gesetzt.



Dynamische Sprungvorhersage

1-Bit Prädiktor: Beispiel mit `simple.mms`

Mit Vektor `YesNo=#8888 8888 8888 8888` wird jeder vierte Sprung genommen und es ergibt sich:

Durchlauf	Prädiktor	Verzweigung
1	0 (T)	T
2	0 (T)	T
3	0 (T)	T
4	0 (T)	N *
5	1 (N)	T *
6	0 (T)	T
7	0 (T)	T
8	0 (T)	N *
9	1 (N)	T *
USW.		

*Fehlerhafte
Vorhersage*

*Selten genutzter
Sprung führt zu
zweimalig
falscher
Vorhersage!*

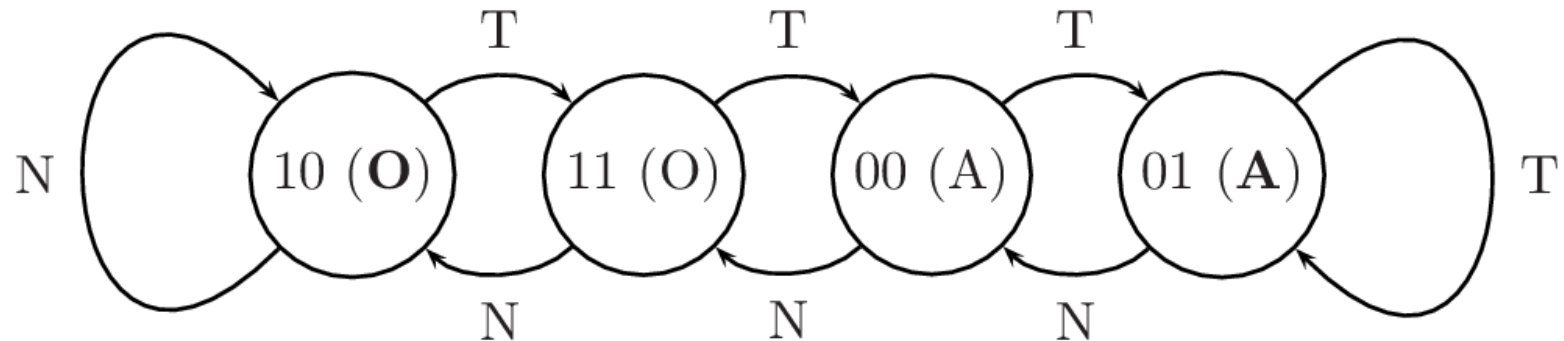


Dynamische Sprungvorhersage

2-Bit Prädiktor

Vorhersage: über saturierenden 2-Bit Zähler

z.B. in Zweierkomplement-Darstellung:



Bitmuster	Wert	Vorhersage
00	0	in Agreement (A)
01	1	in Strong Agreement (A)
11	-1	in Opposition (O)
10	-2	in Strong Opposition (O)



Dynamische Sprungvorhersage

2-Bit Prädiktor: Beispiel mit `simple.mms`

Wiederum mit Vektor `YesNo=#8888 8888 8888 8888` ergibt sich für den 2-Bit Prädiktor

Durchlauf	Prädiktor	Verzweigung
1	00 (T)	T
2	01 (T)	T
3	01 (T)	T
4	01 (T)	N *
5	00 (T)	T
6	01 (T)	T
7	01 (T)	T
8	01 (T)	N *
9	00 (T)	T
usw.		

*Fehlerhafte
Vorhersage*

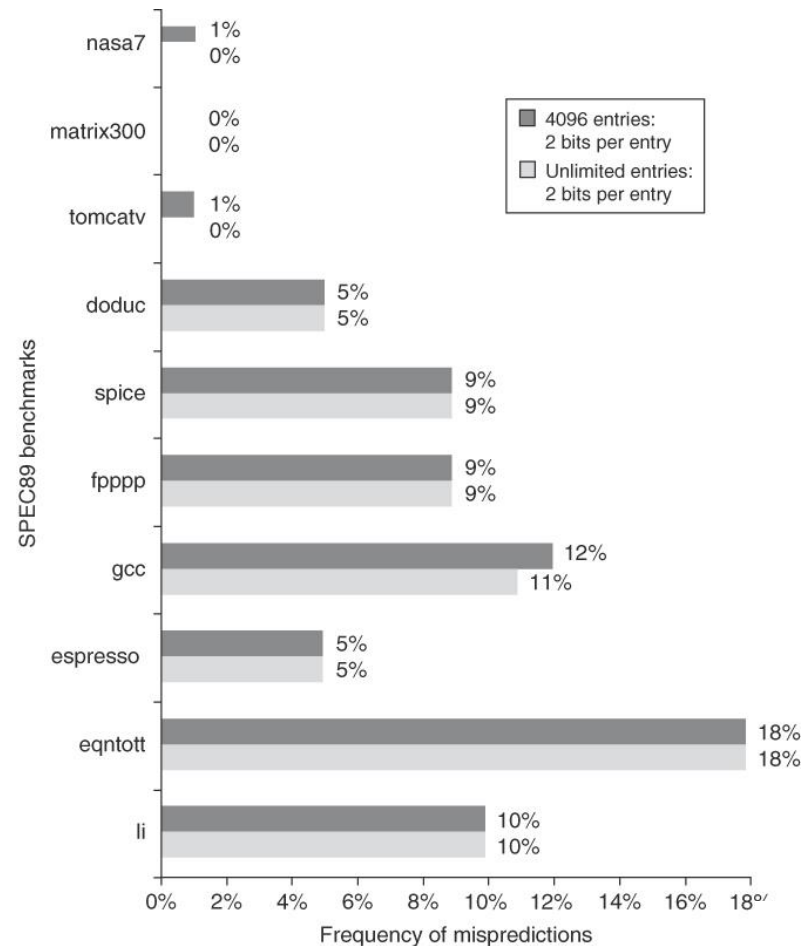
*Misprediction
Rate verglichen
mit 1-Bit Prädiktor
um die Hälfte
gesenkt!*



Dynamische Sprungvorhersage

2-Bit Prädiktor: Beispiel SPEC89 Misprediction Rate

Misprediction Rate
bei 2-Bit Prädiktor
bei Branch History
Table (BHT) mit
4096 bzw.
unbegrenzter Anzahl
an Einträgen:

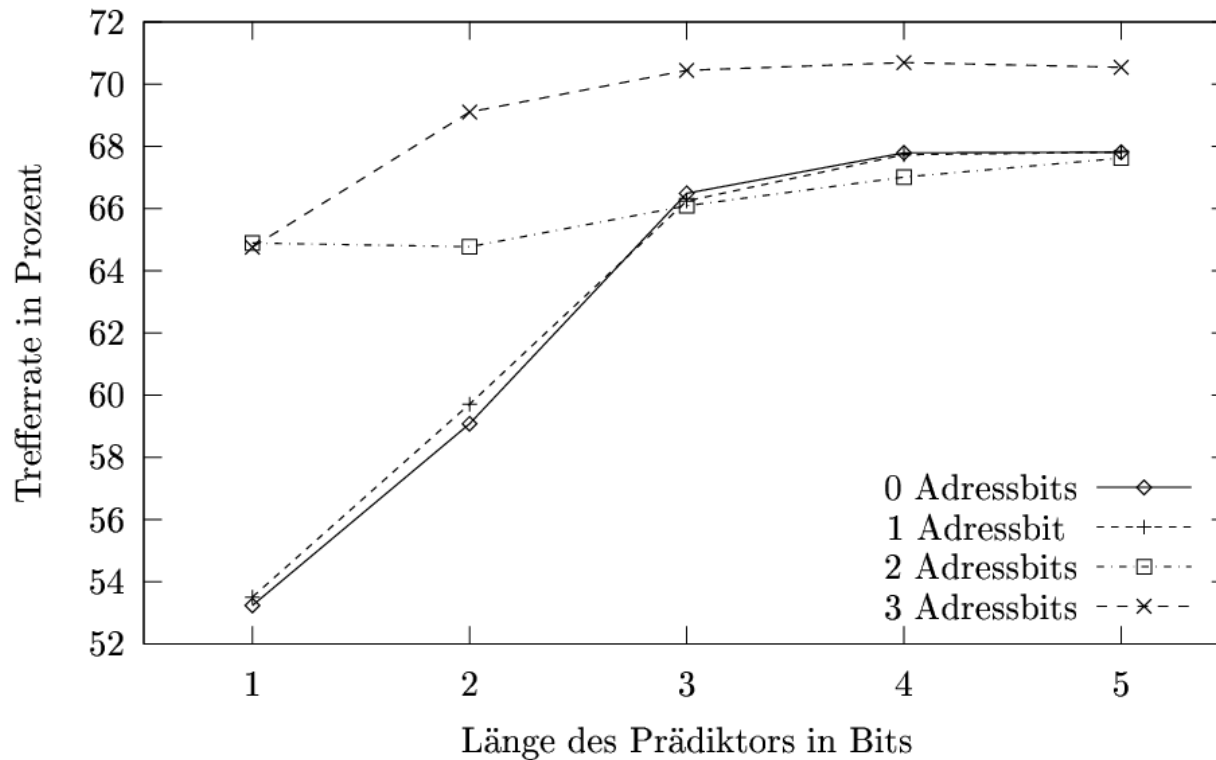


Quelle: [2, S. C-30]

Dynamische Sprungvorhersage

n-Bit Prädiktor: Beispiel Quicksort

Trefferrate (Prediction Rate) in Abhängigkeit von der Länge n des Prädiktors:



Dynamische Sprungvorhersage

Sprungvorhersage mit (globaler) Vorgeschichte

Vorhersage:

Abhängig von den insgesamt in der nahen Vorgeschichte beobachteten Sprungbefehlen (global history) in zwei Stufen:

1. Bitvektor mit m -Bit speichert Vorgeschichte (T/NT) als Bitfolge
2. Dieser dient als Index in Tabelle von Prädiktoren

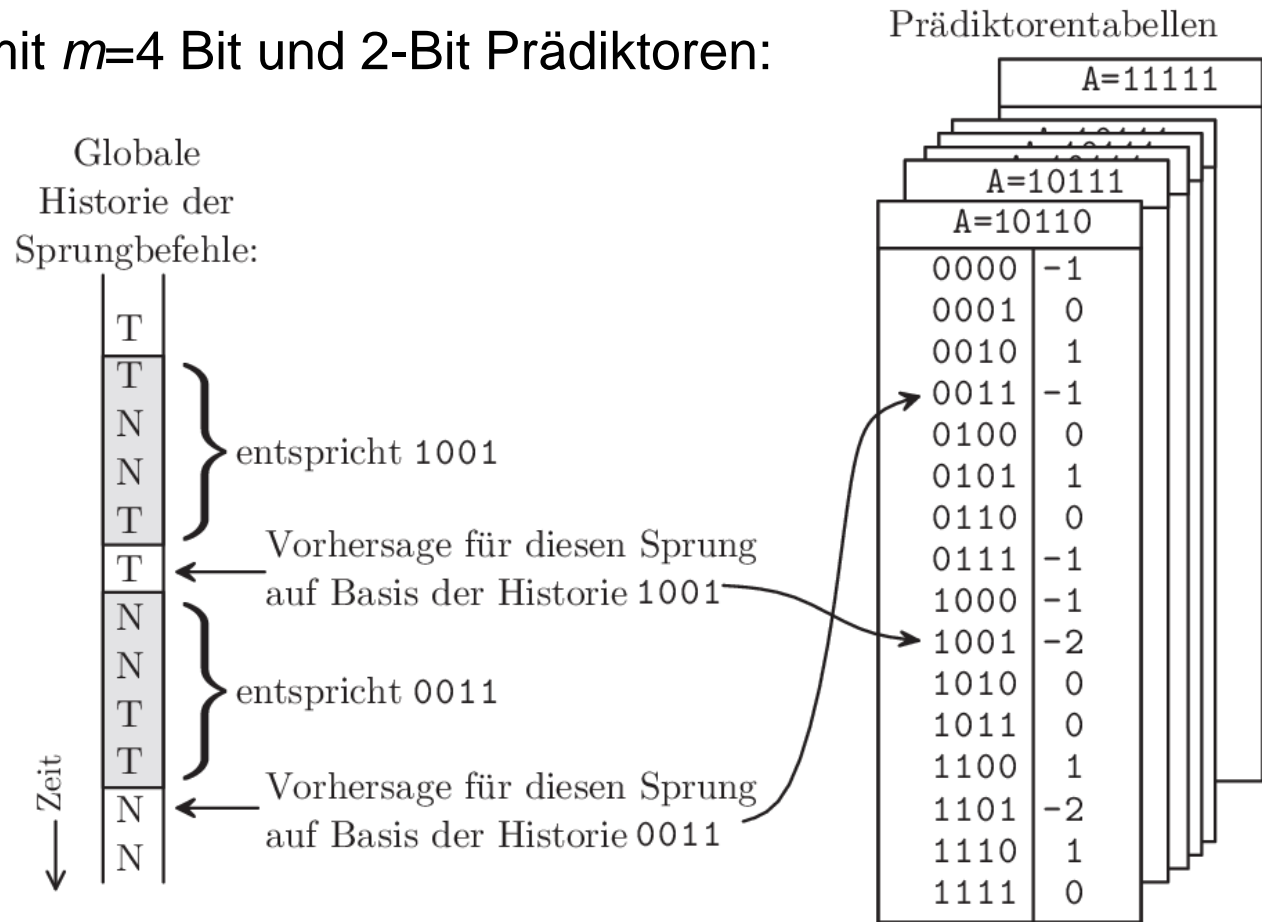
Damit gibt es pro Sprungbefehl nicht nur einen Prädiktor sondern jeweils eine Tabelle mit 2^m Prädiktoren!



Dynamische Sprungvorhersage

Sprungvorhersage mit Vorgeschichte: Beispiel

History mit $m=4$ Bit und 2-Bit Prädiktoren:



Dynamische Sprungvorhersage

Sprungvorhersage mit Vorgeschichte: Beispiel

Beob. Folge	Historie	Prädiktoren (eigentlich 16 Spalten)						
		NNNN	NNNT	NNTT	NTTN	TTNN	TNNT
T	NNNN	00	00	00	00	00	00	...
T	NNNT	01	00	00	00	00	00	...
N	NNTT	01	01	00	00	00	00	...
N	NTTN	01	01	10	00	00	00	...
T	TTNN	01	01	10	10	00	00	...
T	TNNT	01	01	10	10	01	00	...
N	NNTT	01	01	10	10	01	01	...
N	NTTN	01	01	11	10	01	01	...
T	TTNN	01	01	11	11	01	01	...
T	TNNT	01	01	11	11	01	01	...

00, 01: Prediction „Taken“

10, 11: Prediction „Not Taken“



Prädiktor sagt falsch vorher

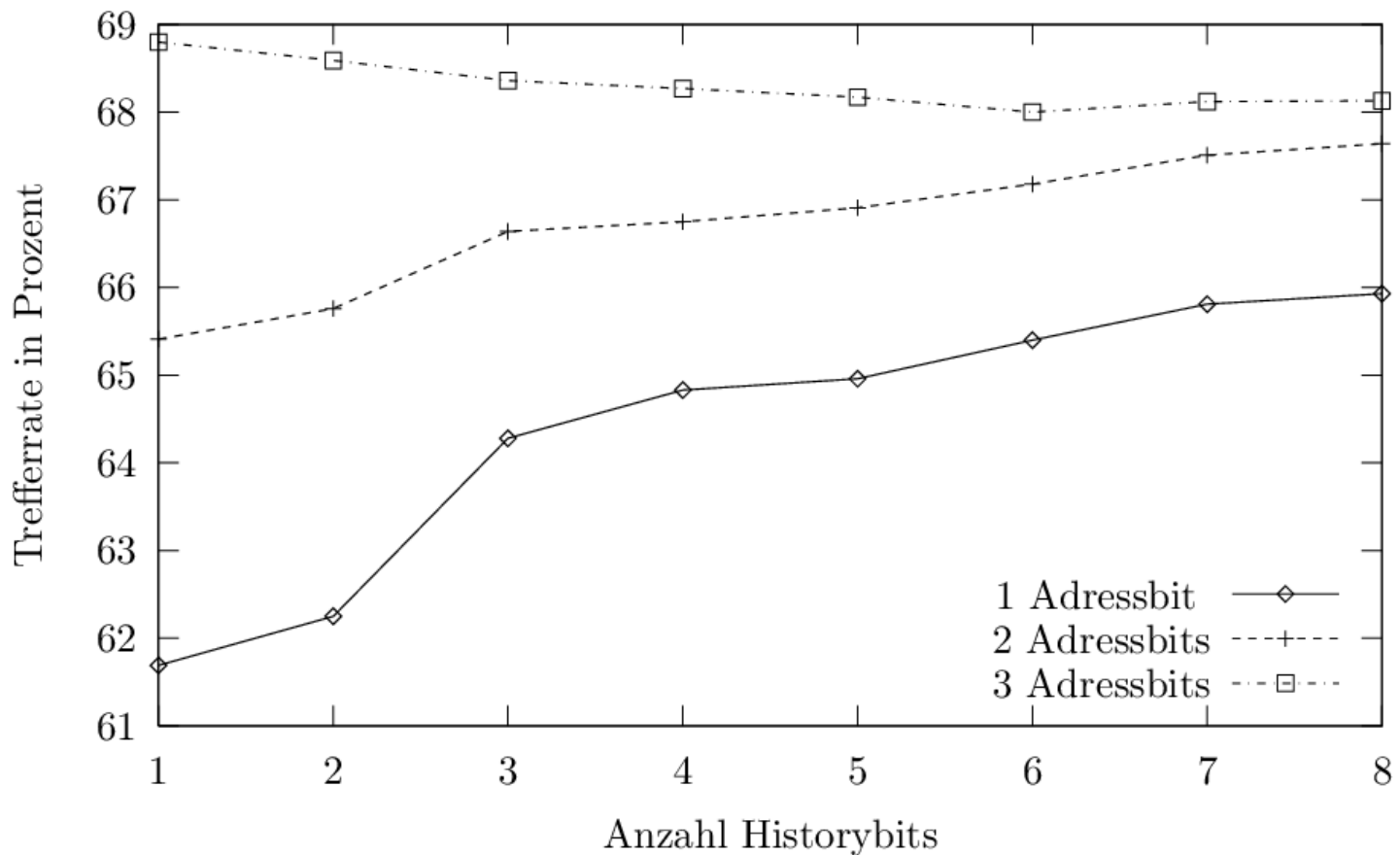


Prädiktor sagt richtig vorher



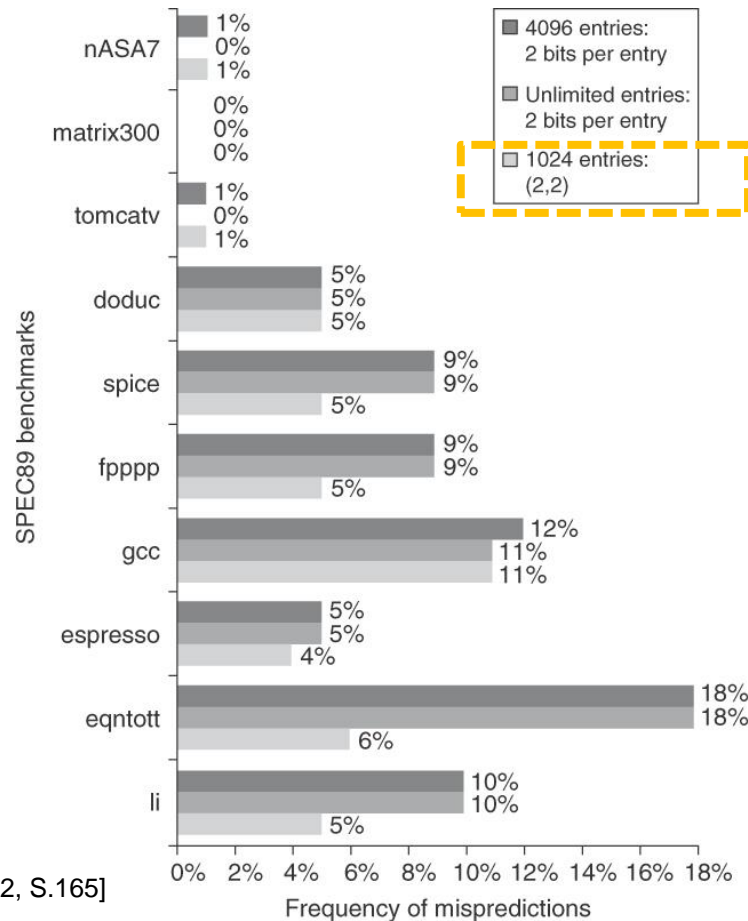
Dynamische Sprungvorhersage

Sprungvorhersage mit Vorgeschichte: Quicksort



Dynamische Sprungvorhersage

Sprungvorhersage mit Vorgeschichte: SPEC89



Prädiktor mit $m=2$ Bit Vorgeschichte und 1024 Einträgen ist deutlich besser als Prädiktor ohne Vorgeschichte mit 4096 oder sogar unendlich vielen Einträgen!

Quelle: [2, S.165]

Dynamische Sprungvorhersage

Branch Target Buffer (BTB)

Problem:

Vorhersage, dass Sprung stattfindet, reicht nicht!

Um Prefetching durchzuführen und die Sprungverzögerung zu vermeiden muss die Zieladresse des Sprungs vorhergesagt werden!

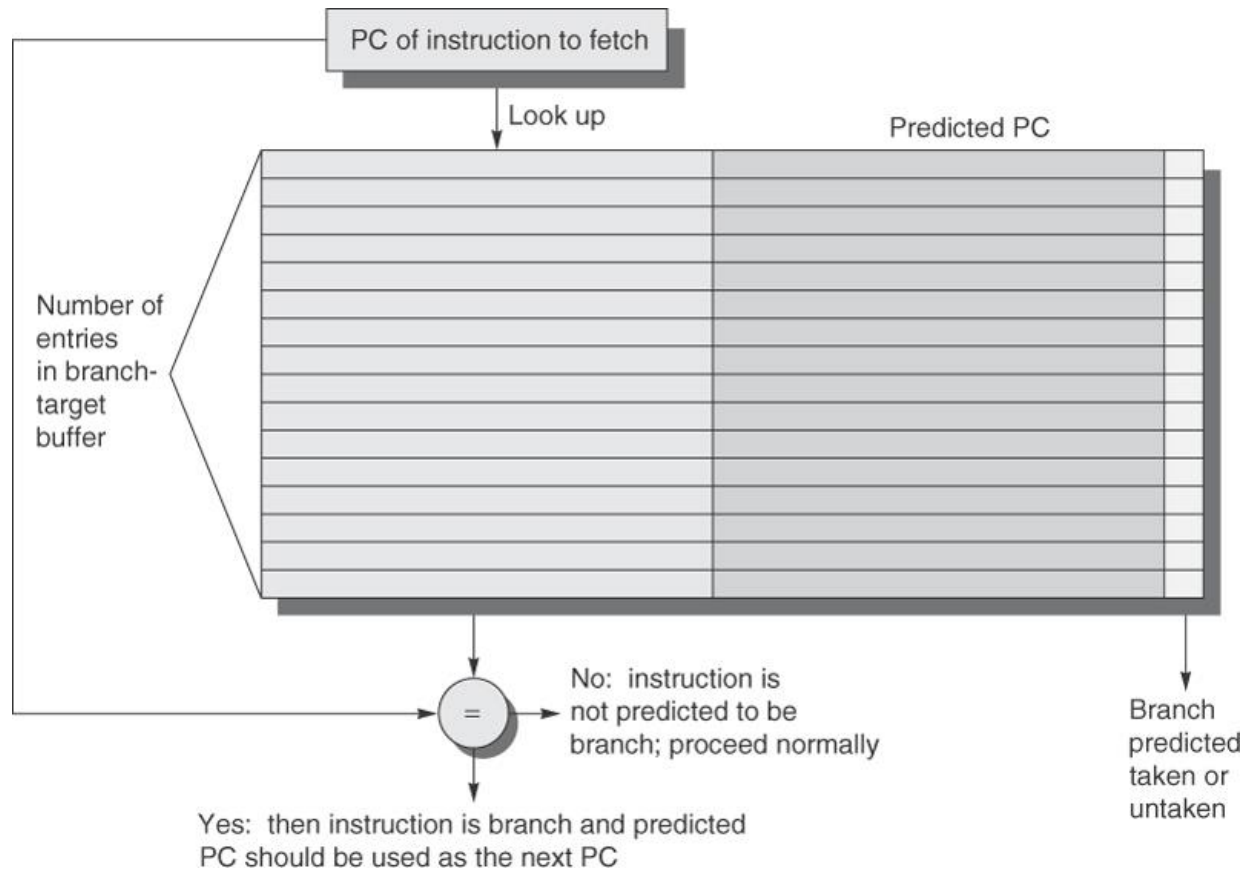
Lösungsansatz: Branch Target Buffer (BTB)

- Zwischenspeicher (Cache), in dem vorhergesagte Sprungadressen für durchgeführte Sprünge gespeichert werden
- Im Fall eines als „Taken“ vorhergesagten Sprungs liefert der BTB die vorhergesagte zugehörige Adresse. Von dieser wird gefetcht.
- Im Fall eines als „Not Taken“ vorhergesagten Sprungs wird Fetch-Adresse regulär erhöht (z.B. um 4 bei MMIX).



Dynamische Sprungvorhersage

Branch Target Buffer (BTB)



Quelle: [2, S. 203]

Dynamische Sprungvorhersage

Praxisbeispiel: Intel Core i7 Branch Predictor Quelle: [2]

- **Zweistufiger Prädiktor**
 - Einfacher First-Level Prädiktor
(schnelle Vorhersage: ein Mal pro Takt)
 - Komplexerer Second-Level Prädiktor (als Backup)
- **Kombination dreier Prädiktoren**
 - Einfacher 2-Bit Prädiktor
 - Prädiktor unter Berücksichtigung der Vorgeschichte (global history predictor)
 - Prädiktor zur Vorhersage des Schleifenendes (loop exit predictor)
- Pro Branch wird Güte der Vorhersage der drei Prädiktoren verfolgt
- Jeweils bester Prädiktor wird für Vorhersage herangezogen
- Branch Target Buffer zur Vorhersage der Zieladresse bei indirekten Sprüngen, Stack zur Vorhersage von Rücksprungadressen



Kontrollfragen zu diesem Kapitel

- Wie trägt Pipelining zur Steigerung der Verarbeitungsgeschwindigkeit bei?
- Welche Stufen hat die klassische 5-Stufen RISC Pipeline?
 - Was passiert in der jeweiligen Stufe?
 - Welche Komponente ist jeweils betroffen?
- Welche Hemmnisse (Hazards) können auftreten und was kann jeweils dagegen getan werden?
- Was unterscheidet statische und dynamische Sprungvorhersage? Welche Techniken gibt es jeweils?
- Erläutern Sie die Funktionsweise der dynamischen Sprungvorhersage mit Vorgeschichte und geben Sie ein Beispiel an, in dem diese deutlich besser als eine Vorhersage ohne Vorgeschichte ist.



Danksagung und Quellen

- Dieser Foliensatz basiert inhaltlich in großen Teilen auf einem älteren von Prof. Axel Böttcher, Hochschule München, entwickelten Foliensatz zur Rechnerarchitektur sowie dem entsprechenden Buch [1].
- Sämtliche Fehler im Foliensatz hingegen entstammen meiner Feder – falls Sie Fehler finden, bin ich Ihnen für einen kurzen Hinweis dankbar.
- Eine Liste weiterer Quellen finden Sie im Abschnitt „Empfohlene Literatur“ des Foliensatzes zu Kapitel 1.

