# Pipelining

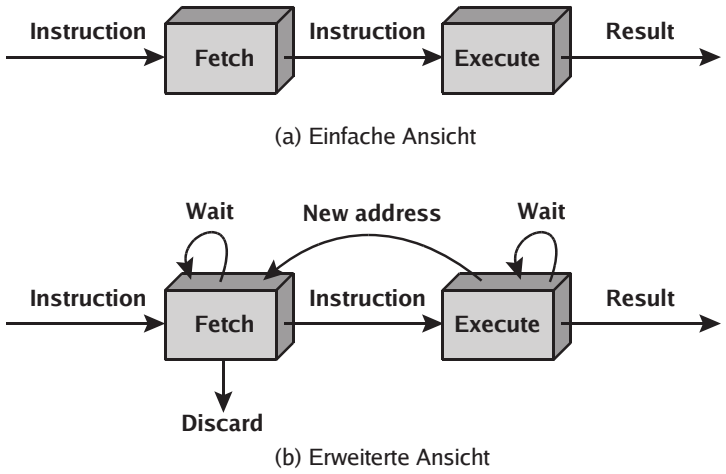


## Pipelining – Ziele

- Wesentliche Begriffe und Fakten über Pipelining kennen
- Vorteile des Pipelining kennen
- Bearbeitung von Sprungbefehlen – Arbeitsschritte für alle Befehle
  - Pipelining Hemmnisse verstehen: Sprünge, Speicherzugriff, Langläufer, Datenabhängigkeiten
- Zusammenhang Arbeitsschritte – Taktrate:
  - Implementierung analysieren können
  - MMIX-Programmstück analysieren und die Ausführung auf einer 5-stufigen Pipeline nachvollziehen können



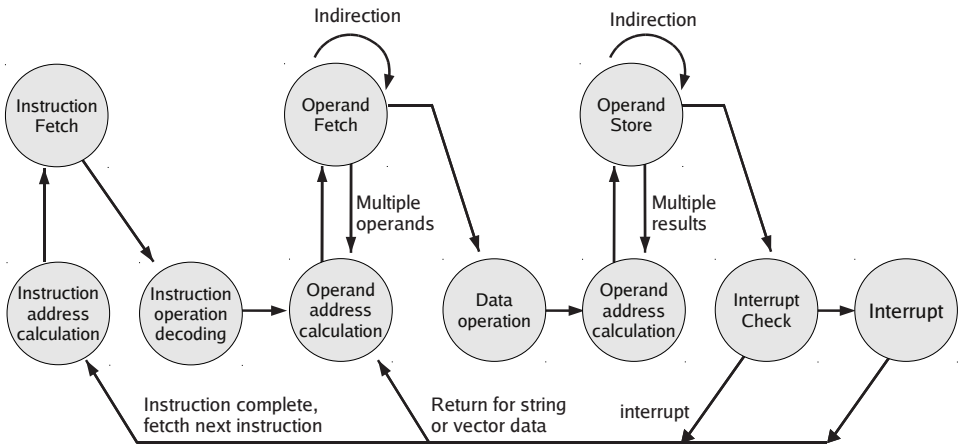
## Einleitung – einfache Pipeline




Quelle: Stallings, Computer Organization and Architecture, 8th ed., 2010, S. 463



## Instruction Cycle State Diagram




Quelle: Stallings, Computer Organization and Architecture, 8th ed., 2010, S. 460

# Pipeline-Designs

- 5-stufige RISC-Pipeline
  - Fetch – Decode – Execute – Memory Access – Write Back
  - siehe auch http://en.wikipedia.org/wiki/Classic_RISC_pipeline
- 6-stufige CISC-Pipeline
  - Fetch Instruction – Decode – Calculate Operands – Fetch Operands – Execute – Write Back
  - siehe Stallings, S. 464



# Klassische 5-Stufen-RISC-Pipeline (1/3)

- Holen des Befehls – Fetch (**F**)
  Befehlszähler liefert Adresse für Speicherzugriff und wird (um vier) erhöht.
- Dekodieren des Befehls – Decode (**D**)
  Bei RISC üblicherweise fixed field decoding
  - 2 Register-Operanden (OP $X,**$Y,$Z**):
    Bereitstellen der beiden Operanden aus Registern
  - 1 Register-Operand, 1 Direktoperand (OP $X,**$Y,Z**):
    Register $Y auslesen; Z aus Befehlswort
  - Speicherzugriff: Adresse mit Register- oder Direktoperand berechnen. Bei schreibendem Zugriff: Wert aus $X lesen
  - Sprung: Für Bedingung $X auslesen; Direktoperand aus Befehlswort (für Sprungzielberechnung)



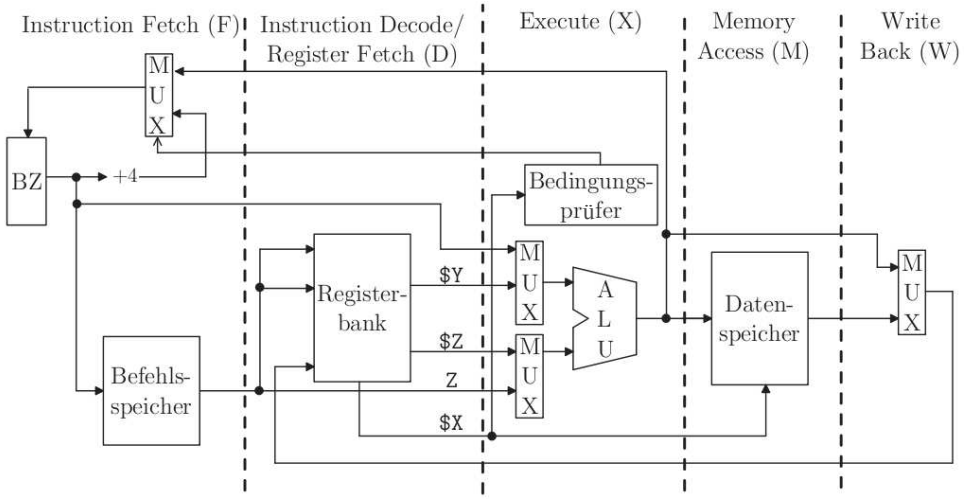
# Klassische 5-Stufen-RISC-Pipeline (2/3)

- Ausführen des Befehls – Execute (**X**)
  - Im Befehl spezifizierte ALU-Operation ausführen
  - Bei Speicherzugriff: Adresse berechnen
  - Bei Verzweigungen: Bedingungen prüfen, Adresse des Folgebefehls berechnen
- Speicherzugriff – Memory Access (**M**)
  Lesender oder schreibender Speicherzugriff, falls erforderlich
- Ergebnis zurückschreiben – Result Write-back (**W**)
  Ergebnis einer Operation in ein Register schreiben, falls erforderlich (ALU-Output oder Speicherwort)

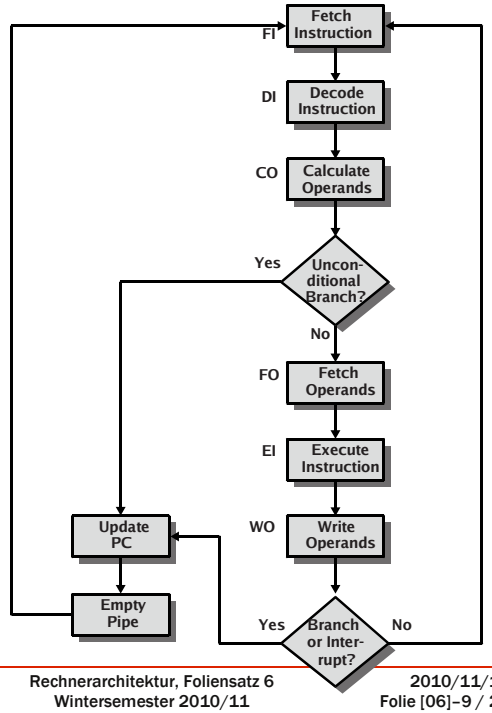


# Klassische 5-Stufen-RISC-Pipeline (3/3)

**MMIX:**



# 6-Stufen-CISC-Pipeline




Quelle: Stallings, Computer Organization and Architecture



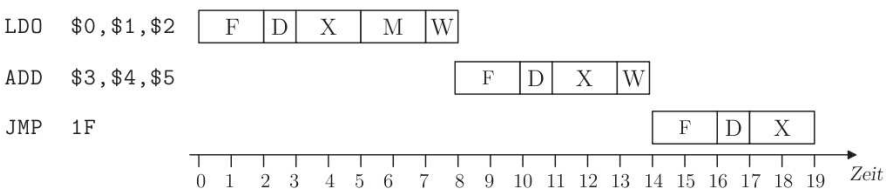
# Takt

- Übergang zwischen Pipeline-Stufen muss getaktet sein
- Bei Übergang Werte-Weitergabe (CPU-interne Pipeline-Register)
- Annahmen:
  - Fetch, Execute, Memory Access: je 2 *t*
  - Decode, Write Back: je *t*
  - Dauer einer Takteinheit hängt von längster Stufe ab, hier also 2 *t*

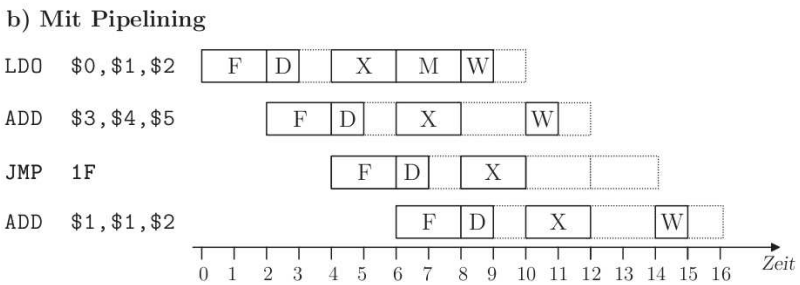


# Ausführung mit und ohne Pipelining

a) Ohne Pipelining




b) Mit Pipelining






# Pipeline-Diagramme („Reservation Tables“)

**a) aus Sicht der Stufen**

| | Takt 1 | Takt 2 | Takt 3 | Takt 4 | Takt 5 | Takt 6 | Takt 7 |
|---|---|---|---|---|---|---|---|
| Fetch | Befehl 1 | Befehl 2 | Befehl 3 | Befehl 4 | Befehl 5 | Befehl 6 | Befehl 7 |
| Decode | | Befehl 1 | Befehl 2 | Befehl 3 | Befehl 4 | Befehl 5 | Befehl 6 |
| Execute | | | Befehl 1 | Befehl 2 | Befehl 3 | Befehl 4 | Befehl 5 |
| Memory | | | | Befehl 1 | Befehl 2 | Befehl 3 | Befehl 4 |
| Write Back | | | | | Befehl 1 | Befehl 2 | Befehl 3 |

**b) aus Sicht der Befehle**

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

# Pipeline-Hemmnisse

- Theoretisch: In jedem Takt ein Befehl fertig
- Praktisch: „Hemmnisse“ („Hazards“) verhindern das:
  - Strukturelle Hemmnisse
  - Datenabhängigkeiten (data hazard)
  - Ablaufbedingte Hemmnisse



# Pipeline-Hemmnisse: strukturell (1)

- Speicherzugriff (von Neumann)

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

- komplexe Befehle (z. B. Gleitkomma)

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SETH $3,#4000 | F | D | X | M | W | | | | | | | | | |
| FMUL $4,$4,$3 | | F | D | X | X | X | X | M | W | | | | | |
| FADD $4,$4,$2 | | | F | D | -- | -- | -- | X | X | X | X | M | W | |
| SET $3, $10 | | | | F | -- | -- | -- | D | -- | -- | -- | X | M | W |



# Pipeline-Hemmnisse: strukturell (2)

- Beispiele für Länge der Execute-Phase komplexer Befehle (bei MMIX):
  - Integer-Multiplikation: 10 Takte
  - Integer-Division: 60 Takte
  - Gleitkommabefehle: 4 Takte (i.d.R.)
- Taktrate auf Dauer des längsten Befehls anheben?



# Pipeline-Hemmnisse: strukturell (3)

- Lösung für Speicherzugriff: Prefetching
  - In jeder Fetch-Phase mehrere Befehle aus dem RAM lesen und puffern („Fetch Buffer“)
  - z. B. MMIX: Befehlslänge 4 Bytes, eine Leseoperation liefert aber 8 Bytes
  - Ungenutzte Memory-Phasen (Befehle ohne Speicherzugriff) für weitere Fetch-Operationen verwenden
- getrennte Prozessor-Caches für Befehle und Daten (→ Pseudo-Harvard-Architektur)

# Pipeline-Hemmnisse: Datenabhängigkeiten (1)

- Data hazards: Befehl benötigt ein Ergebnis, das noch nicht fertig berechnet wurde
- Beispiel: Tausch von zwei Werten (aus Quicksort)

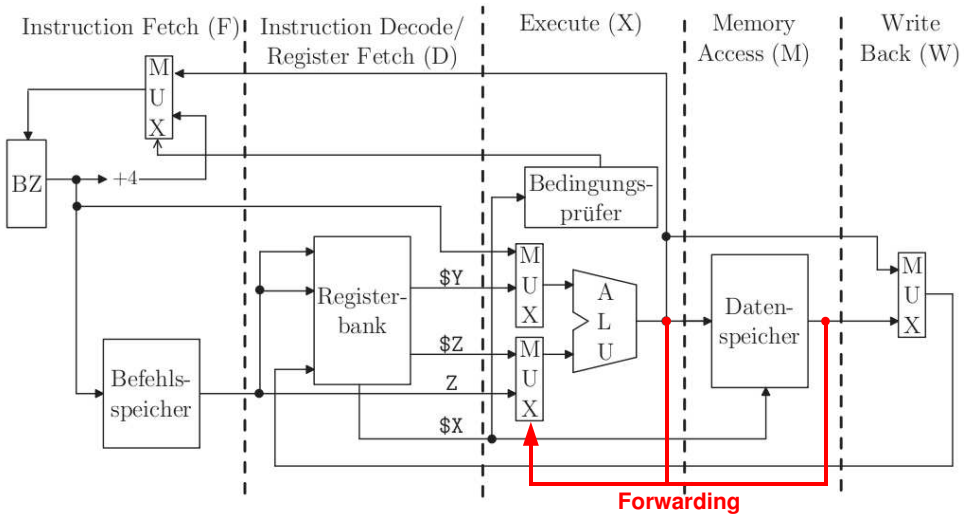
```
                1  2  3  4  5  6  7
XOR l,l,r       F  D  X  M  W
XOR r,l,r          F  D  -- -- X  M     (l nicht bekannt)
XOR l,l,r             F  -- -- D  --    (r nicht bekannt)
CMP tmp,l,pivot          -- -- F  --
```

**Read-after-Write**- (**RAW**-) Konflikt (zweiter Befehl braucht das im ersten berechnete l)

- Lösung: Result Forwarding / Bypassing



# Pipeline-Hemmnisse: Datenabhängigkeiten (2)






# Pipeline-Hemmnisse: Datenabhängigkeiten (3)

- ohne Forwarding

```
                1  2  3  4  5  6  7
XOR l,l,r       F  D  X  M  W
XOR r,l,r          F  D  -- -- X  M     (l nicht bekannt)
XOR l,l,r             F  -- -- D  --    (r nicht bekannt)
CMP tmp,l,pivot          -- -- F  --
```

- mit Forwarding

```
                1  2  3  4  5  6  7
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  W
CMP tmp,l,pivot          F  D  X  M
```



# Pipeline-Hemmnisse: Datenabhängigkeiten (4)

- Result Forwarding hilft nicht immer
  - Lade-Befehle: Erst nach M-Phase steht Ergebnis zur Verfügung

```
                1  2  3  4  5  6  7
LDO $1,base,off F  D  X  M  W
ADD $1,$1,$2       F  D  -- X  M  W
SUB $3,$4,$5          F  -- D  X  M
```

# Pipeline-Hemmnisse: Ablaufbedingt

- bei bedingtem Sprungbefehl:
  unklar, an welcher Stelle es weiter geht
- Sprungziel steht erst nach Execution-Phase des Sprungbefehls fest
- Frage: Was in die Pipeline schreiben?
  - einfach: Immer davon ausgehen, dass *nicht* gesprungen wird
  - komplizierter: Sprungvorhersage (eigenes Thema)



# Zwischen-Evaluation

- Bitte füllen Sie die Fragebögen aus... Danke :)



# Vorschau 18.11.2010

- Mehr Pipelining