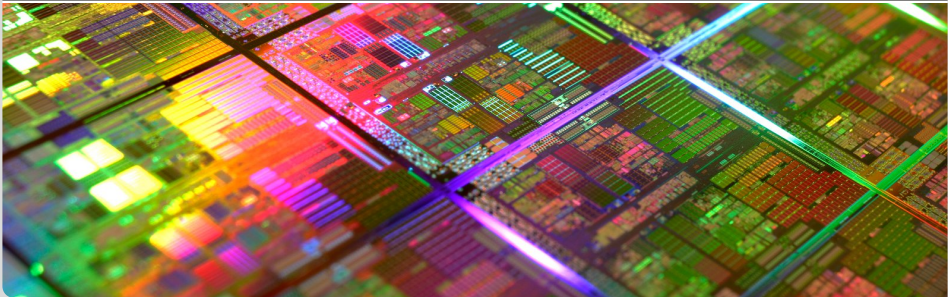


# RO-Tutorien 15 und 16

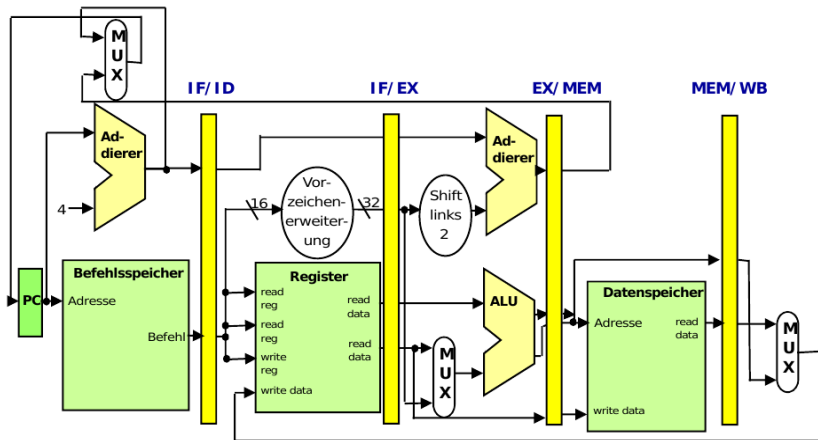
Tutorien zur Vorlesung "Rechnerorganisation"

Tutorienwoche 7 am 08.06.2011



- Datenpfad bei DLX/MIPS
- Datenabhängigkeiten und -konflikte
- Kontrollabhängigkeiten und -konflikte
- Übungsaufgabe

# Datenpfad bei DLX/MIPS



# Was für Abhängigkeiten gibt es?

- **Datenabhängigkeiten:**

Zwei Instruktionen lesen/schreiben die selbe Datenressource (z.B. Register)

- **Kontrollabhängigkeiten:**

Eine Instruktion bestimmt, ob die andere ausgeführt wird (bedingte Sprünge)

- **Strukturabhängigkeiten:**

Zwei Instruktionen benötigen die selbe Ressource (Betriebsmittelabhängigkeit)

# Was für Abhängigkeiten gibt es?

- **Datenabhängigkeiten:**  
Zwei Instruktionen lesen/schreiben die selbe Datenressource (z.B. Register)
- **Kontrollabhängigkeiten:**  
Eine Instruktion bestimmt, ob die andere ausgeführt wird (bedingte Sprünge)
- **Strukturabhängigkeiten:**  
Zwei Instruktionen benötigen die selbe Ressource (Betriebsmittelabhängigkeit)

# Was für Abhängigkeiten gibt es?

- **Datenabhängigkeiten:**  
Zwei Instruktionen lesen/schreiben die selbe Datenressource (z.B. Register)
- **Kontrollabhängigkeiten:**  
Eine Instruktion bestimmt, ob die andere ausgeführt wird (bedingte Sprünge)
- **Strukturabhängigkeiten:**  
Zwei Instruktionen benötigen die selbe Ressource (Betriebsmittelabhängigkeit)

- **Read After Write (RAW):** Echte Abhängigkeit
- **Write After Read (WAR):** Gegenabhängigkeit
- **Write After Write (WAW):** Ausgabeabhängigkeit
- Welche der Abhängigkeiten können auf einer MIPS-/DLX-Pipeline überhaupt zu Konflikten führen und warum?

- **Read After Write (RAW):** Echte Abhängigkeit
- **Write After Read (WAR):** Gegenabhängigkeit
- **Write After Write (WAW):** Ausgabeabhängigkeit
- Welche der Abhängigkeiten können auf einer MIPS-/DLX-Pipeline überhaupt zu Konflikten führen und warum?



- **Read After Write (RAW):** Echte Abhängigkeit
- **Write After Read (WAR):** Gegenabhängigkeit
- **Write After Write (WAW):** Ausgabeabhängigkeit
- Welche der Abhängigkeiten können auf einer MIPS-/DLX-Pipeline überhaupt zu Konflikten führen und warum?

- **Read After Write (RAW):** Echte Abhängigkeit
- **Write After Read (WAR):** Gegenabhängigkeit
- **Write After Write (WAW):** Ausgabeabhängigkeit
- Welche der Abhängigkeiten können auf einer MIPS-/DLX-Pipeline überhaupt zu Konflikten führen und warum?

- Eine Instruktion schreibt Daten, die von einer Instruktion danach gelesen werden:  
 $d := a + b$   
 $e := c + d$
- Konflikt tritt auf, wenn die erste Instruktion noch nicht geschrieben hat, sobald die zweite Instruktion liest

- Eine Instruktion liest Daten, die von einer Instruktion danach geschrieben werden:  
 $c := a + b$   
 $a := d + e$
- Konflikt tritt auf, wenn die erste Instruktion noch nicht gelesen hat, sobald die zweite Instruktion schreibt

- Zwei Instruktionen schreiben Daten an die selbe Stelle.

a := b + c

a := d + e

- Konflikt tritt auf, wenn die erste Instruktion nach der zweiten Instruktion schreibt

- Entweder **in Software**:  
Compiler erzeugt konfliktfreien Code (für eine bestimmte Pipeline).
- Oder **in Hardware**:  
Hardware erkennt Abhängigkeiten und verhindert Konflikte.
- Was sind Vor- und Nachteile der beiden Möglichkeiten?

- Entweder **in Software**:  
Compiler erzeugt konfliktfreien Code (für eine bestimmte Pipeline).
- Oder **in Hardware**:  
Hardware erkennt Abhängigkeiten und verhindert Konflikte.
- Was sind Vor- und Nachteile der beiden Möglichkeiten?

- Entweder **in Software**:  
Compiler erzeugt konfliktfreien Code (für eine bestimmte Pipeline).
- Oder **in Hardware**:  
Hardware erkennt Abhängigkeiten und verhindert Konflikte.
- Was sind Vor- und Nachteile der beiden Möglichkeiten?



# Behandlung von Datenabhängigkeiten in Software

- Compiler erkennt Datenabhängigkeiten
- Einfügen von NOPs verhindert das Auftreten von Konflikten
- Änderung der Befehlsreihenfolge (Umordnung) statt NOPs, falls möglich

# Behandlung von Datenabhängigkeiten in Software

- Compiler erkennt Datenabhängigkeiten
- **Einfügen von NOPs** verhindert das Auftreten von Konflikten
- **Änderung der Befehlsreihenfolge** (Umordnung) statt NOPs, falls möglich

# Behandlung von Datenabhängigkeiten in Software

- Compiler erkennt Datenabhängigkeiten
- **Einfügen von NOPs** verhindert das Auftreten von Konflikten
- **Änderung der Befehlsreihenfolge** (Umordnung) statt NOPs, falls möglich

# Behandlung von Datenabhängigkeiten in Hardware

- **Guter Aufbau der Pipeline** verringert Anzahl von möglichen Konfliktfällen
  - Art und Anordnung der Pipeline-Stufen
  - Forwarding
- Hardware verhindert Konflikte durch **Sperren der Pipeline** (pipeline interlock):  
Statt neuen Befehlen werden solange “Blasen” in die Pipeline eingefügt, bis es sicher ist, die Folgeinstruktion zu laden

# Behandlung von Datenabhängigkeiten in Hardware

- **Guter Aufbau der Pipeline** verringert Anzahl von möglichen Konfliktfällen
  - Art und Anordnung der Pipeline-Stufen
  - Forwarding
- Hardware verhindert Konflikte durch **Sperren der Pipeline** (pipeline interlock):  
Statt neuen Befehlen werden solange “Blasen” in die Pipeline eingefügt, bis es sicher ist, die Folgeinstruktion zu laden

- Treten immer bei bedingten Sprüngen auf
- Erst wenn der Sprungbefehl ausgewertet wurde, ist die Adresse des Folgebefehls bekannt

- Treten immer bei bedingten Sprüngen auf
- Erst wenn der Sprungbefehl ausgewertet wurde, ist die Adresse des Folgebefehls bekannt

# Behandlung von Kontrollabhängigkeiten in Software

- Compiler erkennt Kontrollabhängigkeiten
- Umordnung oder Einfügen von NOPs (wie bei Datenkonflikten)
- Alternative: Effekte sind durch den Assembler-Programmierer zu bedenken (z.B. branch delay slots)



# Behandlung von Kontrollabhängigkeiten in Software

- Compiler erkennt Kontrollabhängigkeiten
- Umordnung oder Einfügen von NOPs (wie bei Datenkonflikten)
- Alternative: Effekte sind durch den Assembler-Programmierer zu bedenken (z.B. branch delay slots)

# Behandlung von Kontrollabhängigkeiten in Software

- Compiler erkennt Kontrollabhängigkeiten
- Umordnung oder Einfügen von NOPs (wie bei Datenkonflikten)
- Alternative: Effekte sind durch den Assembler-Programmierer zu bedenken (z.B. branch delay slots)

# Behandlung von Kontrollabhängigkeiten in Hardware

- Hardware erkennt bedingte Sprungbefehle
- **Sperren der Pipeline** (pipeline interlock) sorgt dafür, dass bis zur Sprungentscheidung keine Befehle nachgeladen werden
- Effizienter: **Sprungvorhersage**
  - **Statische Techniken**: "predict taken", "predict not taken"
  - **Dynamische Techniken**: Zustandsautomat, noch ausgefeiltere Verfahren
- Was passiert, wenn die Sprungvorhersage falsch liegt?

# Behandlung von Kontrollabhängigkeiten in Hardware

- Hardware erkennt bedingte Sprungbefehle
- **Sperren der Pipeline** (pipeline interlock) sorgt dafür, dass bis zur Sprungentscheidung keine Befehle nachgeladen werden
- Effizienter: **Sprungvorhersage**
  - **Statische Techniken:** "predict taken", "predict not taken"
  - **Dynamische Techniken:** Zustandsautomat, noch ausgefeiltere Verfahren
- Was passiert, wenn die Sprungvorhersage falsch liegt?

# Behandlung von Kontrollabhängigkeiten in Hardware

- Hardware erkennt bedingte Sprungbefehle
- **Sperren der Pipeline** (pipeline interlock) sorgt dafür, dass bis zur Sprungentscheidung keine Befehle nachgeladen werden
- Effizienter: **Sprungvorhersage**
  - **Statische Techniken:** “predict taken”, “predict not taken”
  - **Dynamische Techniken:** Zustandsautomat, noch ausgefeiltere Verfahren
- Was passiert, wenn die Sprungvorhersage falsch liegt?

# Behandlung von Kontrollabhängigkeiten in Hardware

- Hardware erkennt bedingte Sprungbefehle
- **Sperren der Pipeline** (pipeline interlock) sorgt dafür, dass bis zur Sprungentscheidung keine Befehle nachgeladen werden
- Effizienter: **Sprungvorhersage**
  - **Statische Techniken:** “predict taken”, “predict not taken”
  - **Dynamische Techniken:** Zustandsautomat, noch ausgefeiltere Verfahren
- Was passiert, wenn die Sprungvorhersage falsch liegt?

Betrachten Sie das folgende MIPS-Programmstück:

```
# In Register $v0 steht die Adresse 0x10008000
lw $t7, 0($v0)
lw $s0, 4($v0)
add $s0, $s0, $t7
mul $t7, $t7, $s0
```

- Bestimmen Sie alle Daten- und Kontrollabhängigkeiten in dem gegebenen Programmstück.

Betrachten Sie das folgende MIPS-Programmstück:

```
# In Register $v0 steht die Adresse 0x10008000
lw $t7, 0($v0)
lw $s0, 4($v0)
add $s0, $s0, $t7
mul $t7, $t7, $s0
```

- Wie viele Pipelinekonflikte treten auf? Begründen Sie Ihre Antwort.



Betrachten Sie das folgende MIPS-Programmstück:

```
# In Register $v0 steht die Adresse 0x10008000
lw $t7, 0($v0)
lw $s0, 4($v0)
add $s0, $s0, $t7
mul $t7, $t7, $s0
```

- Unter der Voraussetzung, dass die auftretenden Pipelinekonflikte von der Hardware nicht erkannt werden, müssen die Pipelinekonflikte vom Compiler behandelt werden. Ergänzen Sie obiges Programmstück, sodass die auftretenden Pipelinekonflikte berücksichtigt werden.

Betrachten Sie das folgende MIPS-Programmstück:

```
# In Register $v0 steht die Adresse 0x10008000
lw $t7, 0($v0)
lw $s0, 4($v0)
add $s0, $s0, $t7
mul $t7, $t7, $s0
```

- Welche der von Ihnen getroffenen Maßnahmen im letzten Aufgabenteil sind noch erforderlich, falls die auftretenden Pipelinekonflikte von der Hardware erkannt und durch Load-Forwarding und Result-Forwarding behandelt werden.



Quelle: <http://xkcd.com/749/>