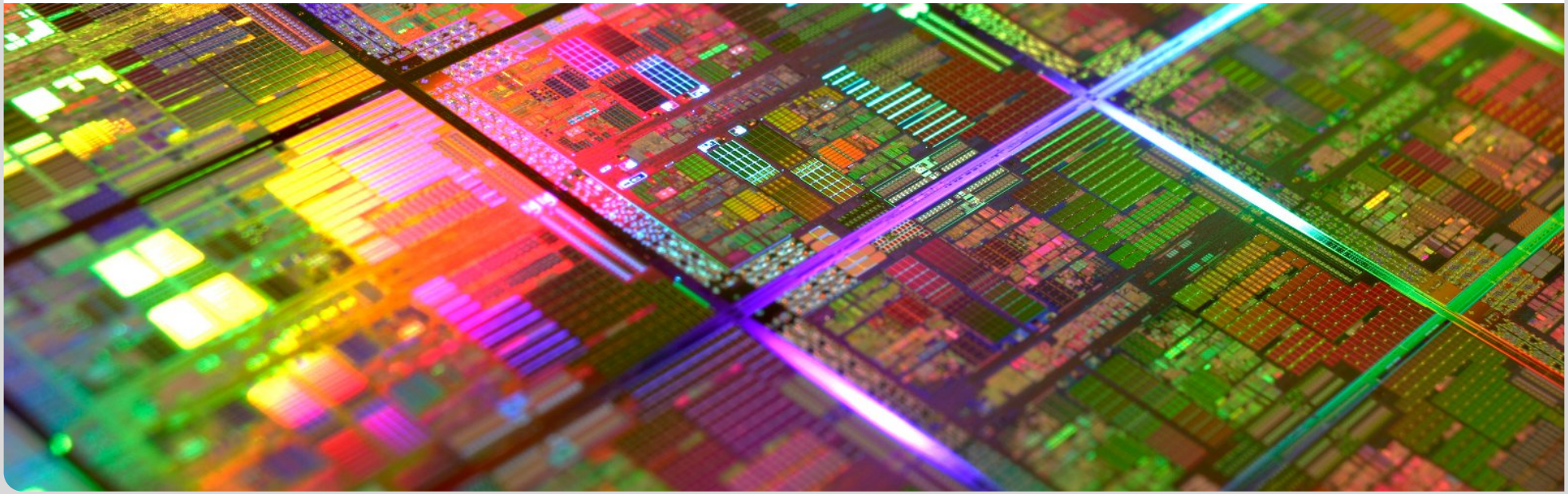
Karlsruher Institut für Technologie

RO-Tutorien 15 und 16

Tutorien zur Vorlesung “Rechnerorganisation”

Tutorienwoche 6 am 01.06.2011



Heute

- Pipelining
- Übungsaufgaben

Parallelität innerhalb der CPU - Wozu?

- Viele Komponenten der CPU sind die meiste Zeit inaktiv
- Eine schlechte Datenbusarchitektur erschwert Parallelität, indem eine Komponenten einen geteilten Bus für andere Komponenten blockiert (Flaschenhals)
- Ziel: Geschwindigkeitsvorteil durch optimale Parallelisierung

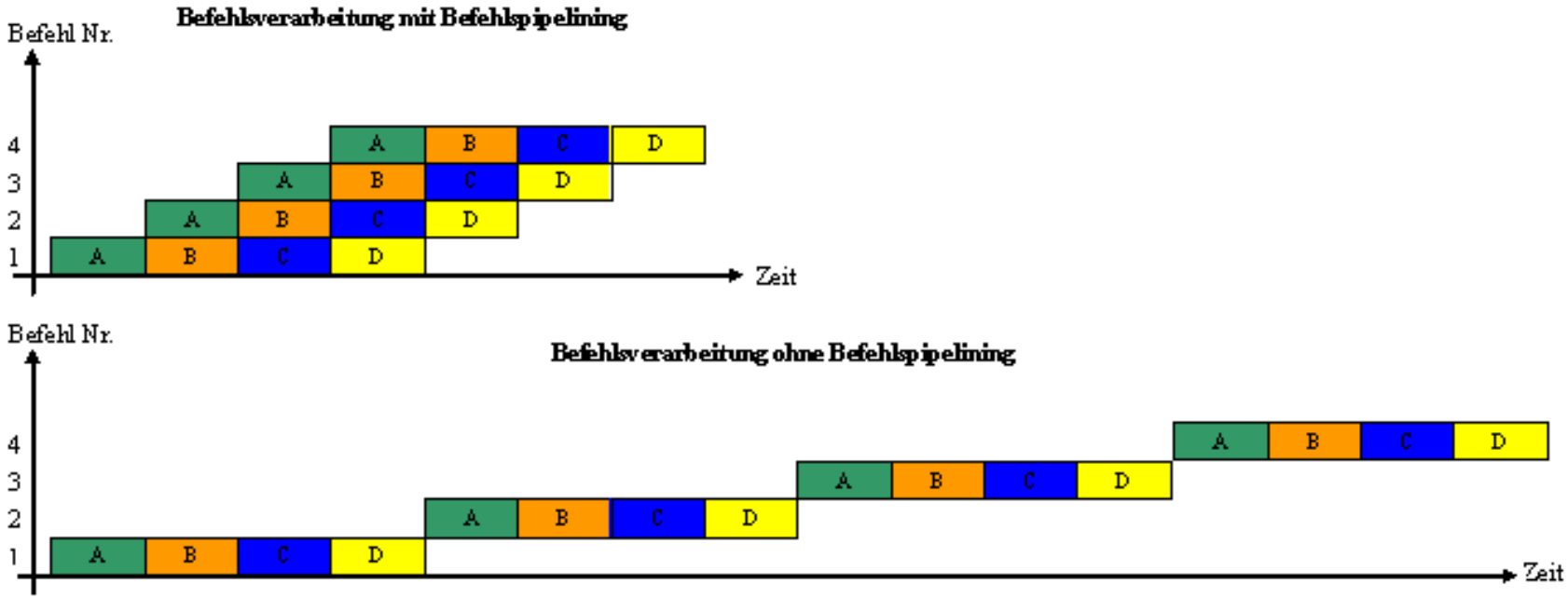
Parallelität innerhalb der CPU - Wozu?

- Viele Komponenten der CPU sind die meiste Zeit inaktiv
- Eine schlechte Datenbusarchitektur erschwert Parallelität, indem eine Komponenten einen geteilten Bus für andere Komponenten blockiert (Flaschenhals)
- Ziel: Geschwindigkeitsvorteil durch optimale Parallelisierung

Parallelität innerhalb der CPU - Wozu?

- Viele Komponenten der CPU sind die meiste Zeit inaktiv
- Eine schlechte Datenbusarchitektur erschwert Parallelität, indem eine Komponenten einen geteilten Bus für andere Komponenten blockiert (Flaschenhals)
- Ziel: Geschwindigkeitsvorteil durch optimale Parallelisierung

Die Idee der Pipeline



- Mögliche Phasen der Befehlsabarbeitung bekannt von MIMA/MIPS

Die DLX-Pipeline

1. IF: Instruction Fetch
2. ID: Instruction Decode
3. EX: Execute
4. MEM: Memory Access
5. WB: Write Back

Die DLX-Pipeline

1. IF: Instruction Fetch
2. ID: Instruction Decode
3. EX: Execute
4. MEM: Memory Access
5. WB: Write Back

Die DLX-Pipeline

1. IF: Instruction Fetch
2. ID: Instruction Decode
3. EX: Execute
4. MEM: Memory Access
5. WB: Write Back

Die DLX-Pipeline

1. IF: Instruction Fetch
2. ID: Instruction Decode
3. EX: Execute
4. MEM: Memory Access
5. WB: Write Back

Die DLX-Pipeline

1. IF: Instruction Fetch
2. ID: Instruction Decode
3. EX: Execute
4. MEM: Memory Access
5. WB: Write Back

Vorteile von Pipelining

- Zeitdauer für Abarbeitung **eines** Befehls wird nicht reduziert
- Aber: Pipelining erhöht den Durchsatz bei der Ausführung einer Befehlsfolge
- Im Idealfall: Nach Befüllen der Pipeline wird jeden Takt die Ausführung eines Befehls abgeschlossen
- Geschwindigkeit der Pipeline richtet sich nach der langsamsten Pipeline-Stufe

Vorteile von Pipelining

- Zeitdauer für Abarbeitung **eines** Befehls wird nicht reduziert
- Aber: Pipelining erhöht den Durchsatz bei der Ausführung einer Befehlsfolge
- Im Idealfall: Nach Befüllen der Pipeline wird jeden Takt die Ausführung eines Befehls abgeschlossen
- Geschwindigkeit der Pipeline richtet sich nach der langsamsten Pipeline-Stufe

Vorteile von Pipelining

- Zeitdauer für Abarbeitung **eines** Befehls wird nicht reduziert
- Aber: Pipelining erhöht den Durchsatz bei der Ausführung einer Befehlsfolge
- Im Idealfall: Nach Befüllen der Pipeline wird jeden Takt die Ausführung eines Befehls abgeschlossen
- Geschwindigkeit der Pipeline richtet sich nach der langsamsten Pipeline-Stufe

Vorteile von Pipelining

- Zeitdauer für Abarbeitung **eines** Befehls wird nicht reduziert
- Aber: Pipelining erhöht den Durchsatz bei der Ausführung einer Befehlsfolge
- Im Idealfall: Nach Befüllen der Pipeline wird jeden Takt die Ausführung eines Befehls abgeschlossen
- Geschwindigkeit der Pipeline richtet sich nach der langsamsten Pipeline-Stufe

Probleme beim Pipelining

- Erhöhter Planungs- und Schaltungsaufwand
- Daten werden benötigt, die noch nicht zur Verfügung stehen (Datenkonflikte)
- Es ist noch nicht klar, welcher Befehl in die Pipeline geladen werden muss (Kontrollflusskonflikte)
- Was macht man, wenn man Komponenten (z.B. Addierer oder Speicherbus) in mehr als einer Pipeline-Stufe benötigt (Ressourcenkonflikte)?

Probleme beim Pipelining

- Erhöhter Planungs- und Schaltungsaufwand
- Daten werden benötigt, die noch nicht zur Verfügung stehen (Datenkonflikte)
- Es ist noch nicht klar, welcher Befehl in die Pipeline geladen werden muss (Kontrollflusskonflikte)
- Was macht man, wenn man Komponenten (z.B. Addierer oder Speicherbus) in mehr als einer Pipeline-Stufe benötigt (Ressourcenkonflikte)?

Probleme beim Pipelining

- Erhöhter Planungs- und Schaltungsaufwand
- Daten werden benötigt, die noch nicht zur Verfügung stehen (Datenkonflikte)
- Es ist noch nicht klar, welcher Befehl in die Pipeline geladen werden muss (Kontrollflusskonflikte)
- Was macht man, wenn man Komponenten (z.B. Addierer oder Speicherbus) in mehr als einer Pipeline-Stufe benötigt (Ressourcenkonflikte)?

Probleme beim Pipelining

- Erhöhter Planungs- und Schaltungsaufwand
- Daten werden benötigt, die noch nicht zur Verfügung stehen (Datenkonflikte)
- Es ist noch nicht klar, welcher Befehl in die Pipeline geladen werden muss (Kontrollflusskonflikte)
- Was macht man, wenn man Komponenten (z.B. Addierer oder Speicherbus) in mehr als einer Pipeline-Stufe benötigt (Ressourcenkonflikte)?

Durchsatz

- Durchsatz als Metrik, z.B. gemessen in IPC (Instructions per cycle)
- $\textit{Durchsatz} := \frac{\textit{Ausgeführte Befehle}}{\textit{Benötigte Zeit}}$
- Durchsatz erhöht sich durch Pipelining
- Im Idealfall:
  - N-stufige Pipeline mit k Befehlen: $k + (n - 1)$ benötigte Taktzyklen
  - Für $n \longrightarrow \infty$: Durchsatz $\longrightarrow 1$

Durchsatz

- Durchsatz als Metrik, z.B. gemessen in IPC (Instructions per cycle)
- $\textit{Durchsatz} := \frac{\textit{Ausgeführte Befehle}}{\textit{Benötigte Zeit}}$
- Durchsatz erhöht sich durch Pipelining
- Im Idealfall:
  - N-stufige Pipeline mit k Befehlen: $k + (n - 1)$ benötigte Taktzyklen
  - Für $n \longrightarrow \infty$: Durchsatz $\longrightarrow 1$

Durchsatz

- Durchsatz als Metrik, z.B. gemessen in IPC (Instructions per cycle)
- $\textit{Durchsatz} := \frac{\textit{Ausgeführte Befehle}}{\textit{Benötigte Zeit}}$
- Durchsatz erhöht sich durch Pipelining
- Im Idealfall:
  - N-stufige Pipeline mit k Befehlen: $k + (n - 1)$ benötigte Taktzyklen
  - Für $n \longrightarrow \infty$: Durchsatz $\longrightarrow 1$

Durchsatz

- Durchsatz als Metrik, z.B. gemessen in IPC (Instructions per cycle)
- $\textit{Durchsatz} := \frac{\textit{Ausgeführte Befehle}}{\textit{Benötigte Zeit}}$
- Durchsatz erhöht sich durch Pipelining
- Im Idealfall:
  - N-stufige Pipeline mit k Befehlen: $k + (n - 1)$ benötigte Taktzyklen
  - Für $n \longrightarrow \infty$: Durchsatz $\longrightarrow 1$

Durchsatz

- Durchsatz als Metrik, z.B. gemessen in IPC (Instructions per cycle)
- $\textit{Durchsatz} := \frac{\textit{Ausgeführte Befehle}}{\textit{Benötigte Zeit}}$
- Durchsatz erhöht sich durch Pipelining
- Im Idealfall:
  - N-stufige Pipeline mit k Befehlen: $k + (n - 1)$ benötigte Taktzyklen
  - Für $n \rightarrow \infty$: Durchsatz $\rightarrow 1$

Speedup

- Der Speedup gibt den Geschwindigkeitsgewinn durch das Pipelining an
- $\textit{Speedup} := \frac{\textit{Ohne Pipelining nötige Taktzyklen}}{\textit{Mit Pipeline nötige Taktzyklen}}$
- Im Idealfall: Für $n \longrightarrow \infty$, Speedup $\longrightarrow$ Anzahl Pipeline-Stufen

Speedup

- Der Speedup gibt den Geschwindigkeitsgewinn durch das Pipelining an
- $\textit{Speedup} := \frac{\textit{Ohne Pipelining nötige Taktzyklen}}{\textit{Mit Pipeline nötige Taktzyklen}}$
- Im Idealfall: Für $n \longrightarrow \infty$, Speedup $\longrightarrow$ Anzahl Pipeline-Stufen

Speedup

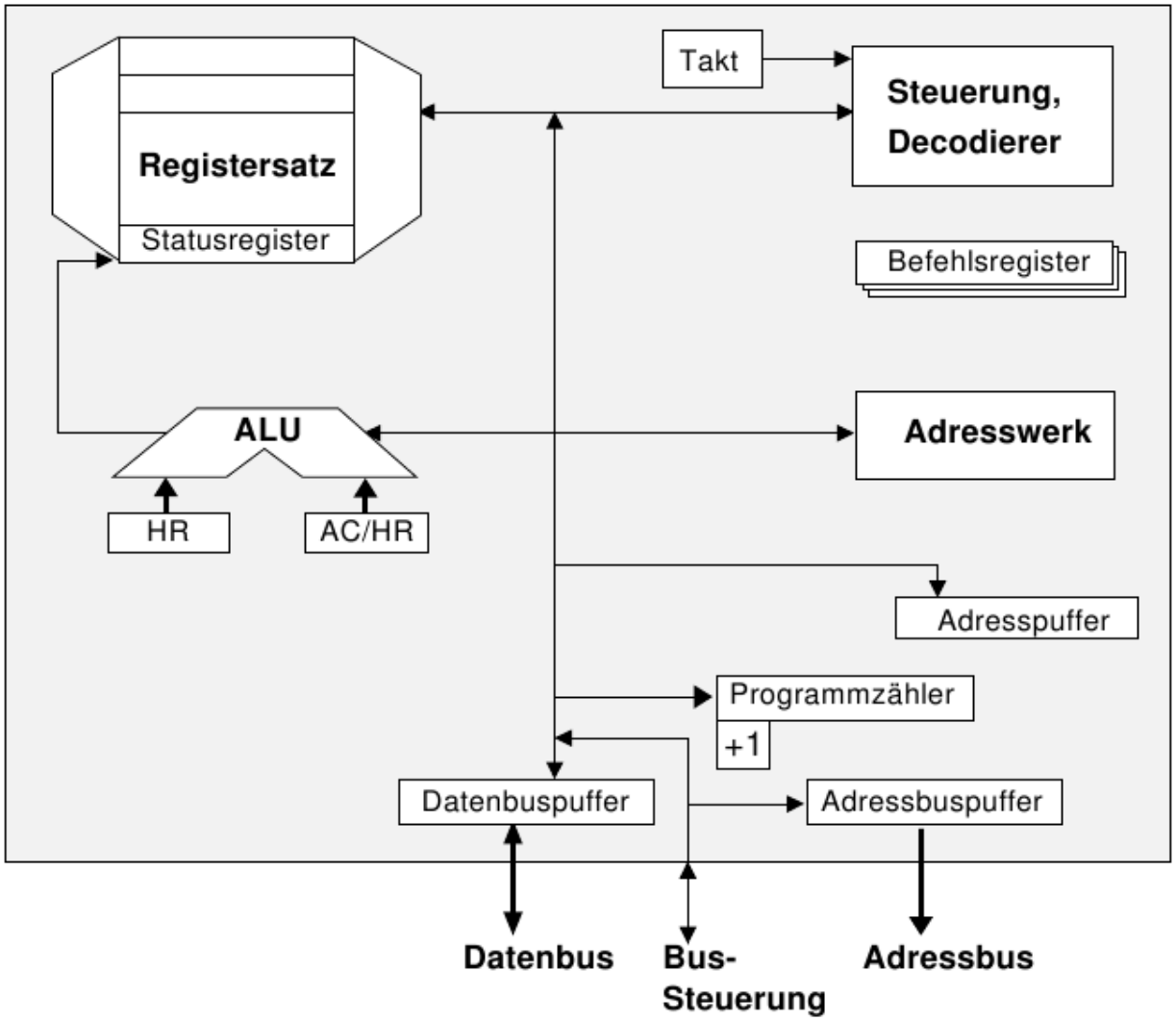
- Der Speedup gibt den Geschwindigkeitsgewinn durch das Pipelining an
- $\textit{Speedup} := \frac{\textit{Ohne Pipelining nötige Taktzyklen}}{\textit{Mit Pipeline nötige Taktzyklen}}$
- Im Idealfall: Für $n \longrightarrow \infty$, Speedup $\longrightarrow$ Anzahl Pipeline-Stufen

Übungsaufgabe 1

Ihre Aufgabe besteht darin, die Architektur des internen Bussystems zu entwerfen, so dass eine hohe prozessorinterne Parallelität bei der Befehlsverarbeitung möglich ist, d.h.

- Opcode Prefetching
- Gleichzeitiges Schreiben der ALU-Ergebnisse in den Registersatz und paralleles Laden der ALU-Eingänge mit Operanden aus dem Registersatz oder dem Datenbuspuffer (sofern nicht das gleiche Register hierfür benötigt wird)
- Direkter Zugriff des Adresswerks auf die Adressregister im Registersatz

Übungsaufgabe 1



Übungsaufgabe 2

Die Befehlsabarbeitung in einem Prozessor erfolgt in den folgenden Stufen:

- B: Befehl holen
- D: Befehl dekodieren
- O: Operand(en) holen
- A: Befehl ausführen
- S: Ergebnis speichern

Diskutieren Sie die folgende Situation: Der Befehl n legt sein Ergebnis in der Speicherstelle a ab. Der nächste Befehl (n + 1) benutzt den soeben erzeugten Wert als Operanden. Zeigen Sie die Probleme auf, die dadurch entstehen, dass die beiden Befehle in einer Pipeline ausgeführt werden, die aus den obigen Bearbeitungseinheiten besteht.

Fertig!

Quelle: http://xkcd.com/838/