Kapitel 6 – Weitergehende Architekturkonzepte

1. **Pipelining und Parallelverarbeitung**

2. Speicherorganisation

Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

Prof. Dr. Christoph Scholl

Institut für Informatik

WS 2015/16

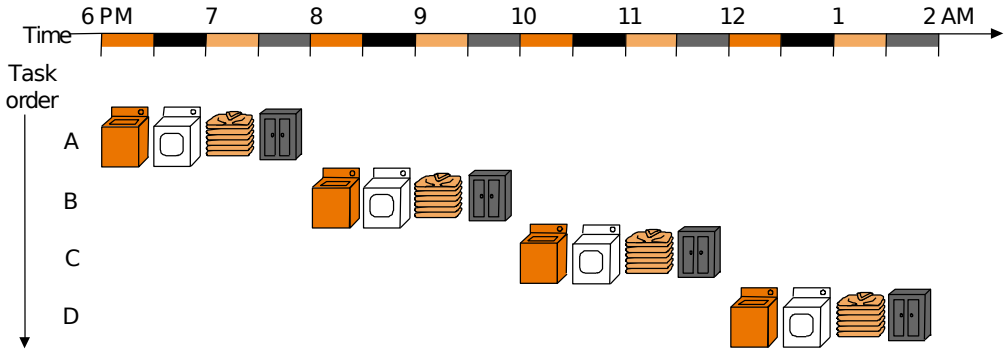
Übersicht Pipelining

- Performanz von Rechnern lässt sich durch Pipelining steigern.
  - Prinzip der Fließbandverarbeitung
  - Probleme der Fließbandverarbeitung

Das Prinzip an einem alltäglichen Beispiel

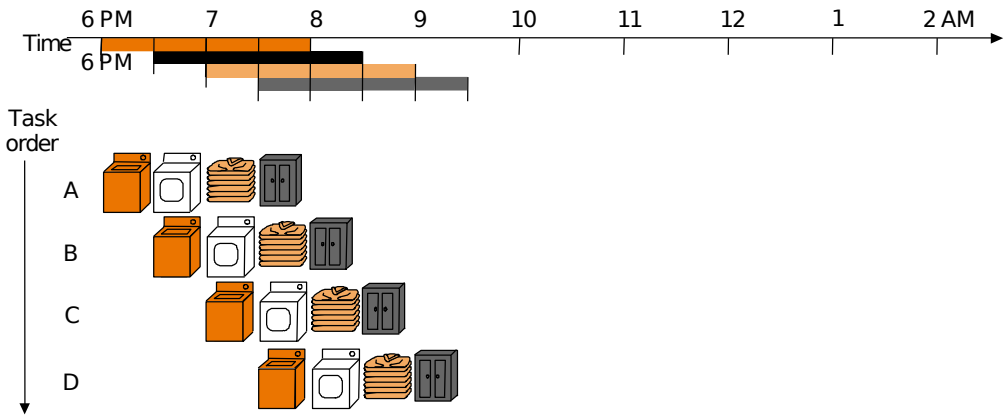
- Personen A, B, C und D kommen aus dem Urlaub; es ist viel schmutzige Wäsche zu waschen!
- Zur Verfügung stehen:
  - eine Waschmaschine (1/2 Stunde Laufzeit)
  - ein Trockner (1/2 Stunde Laufzeit)
  - ein Bügeleisen (1/2 Stunde Arbeit zum Bügeln)
  - ein Wäscheschrank (1/2 Stunde Arbeit zum Einräumen)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

Das Prinzip an einem Beispiel



Dauer der Arbeiten:
8 Stunden

Mit Pipelining:



Dauer der Arbeiten:
$3\frac{1}{2}$ 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 4 Schritten:
  - Befehls-Holphase (instruction fetch)
  - Dekodierphase / Lesen von Operanden aus Registern
  - Ausführung / Adressberechnung
  - Abspeicherphase (result write back phase)

Pipelining: Illustration

- **Annahme**: Aufteilung der Befehlsabarbeitung in 4 gleichlange Phasen.

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

Pipelining: Speedup (1/2)

- **Annahmen**:
  - Abarbeitungszeit eines Befehls ohne Pipelining: t
  - k Pipelinestufen, gleiche Laufzeit der Stufen

  $\Rightarrow$ Laufzeit einer Stufe der Pipeline: $\frac{t}{k}$

- Beschleunigung bei m auszuführenden Instruktionen:

  $m = 1$: Laufzeit mit Pipeline $= k \cdot \frac{t}{k} = t$
  $\Rightarrow$ keine Beschleunigung

Pipelining: Speedup (2/2)

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

  $\Rightarrow$ Beschleunigung um $\frac{mt}{t+(m-1)\cdot\frac{t}{k}} = \frac{mk}{m+k-1} = k - \frac{k(k-1)}{m+k-1}$

- **Ergebnis**:

  Für $m >> k$ nähert sich der Speedup also der Anzahl k der Pipelinestufen.

- Es wurde vorausgesetzt, dass sich die Ausführung der Befehle ohne weiteres „verzahnen" läßt. Dies ist in der Praxis nicht immer der Fall
  $\Rightarrow$ Hazards!

Pipelining: Data Hazards (1/2)

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

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

  - **Problem**:

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

Pipelining: Data Hazards (2/2)

- Nehme 4 Pipelinestufen an:

Takt	Befehlholen	Decodieren / Operand holen	Ausführung / Adressberechnung	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!

Behandlung von Data Hazards (1/2)

- **Lösung 1**: Einfügen von NOPs (NOP = no operation)

Takt	Befehlholen	Decodieren / Operand holen	Ausführung / Adressberechnung	Abspeichern
1	LOAD			
2	NOP	LOAD		
3	NOP	NOP	LOAD	
4	ADD	NOP	NOP	LOAD
5	SUB	ADD	NOP	NOP

- Jetzt korrekt:

  In Takt 4 wurde R1 korrekt belegt und in Takt 5 der richtige Operand aus Register R1 geladen!

Behandlung von Data Hazards (2/2)

- **Lösung 2**: Umordnen von Befehlen

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

- **Neue Befehlsfolge**:

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

- **Wichtig**: Umordnen nicht beliebig möglich (Datenabhängigkeiten beachten!)

Pipelining: Hazards

- **Lösung 2**: Umordnen von Befehlen

Takt	Befehlholen	Decodieren / Operand holen	Ausführung / Adressberechnung	Abspeichern
1	LOAD			
2	SUB	LOAD		
3	MUL	SUB	LOAD	
4	ADD	MUL	SUB	LOAD
5		ADD	MUL	SUB

- Jetzt korrekt:

  In Takt 4 wurde R1 korrekt belegt und in Takt 5 der richtige Operand aus Register R1 geladen!

Pipelining: Control Hazards

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

```
1. ADD R1, R2, R3;        // Addiere Inhalte der Register R1, R2,
                          // speichere Ergebnis in R3 ab.
2. JMP>0 R4 <label>;      // Wenn Inhalt von R4 > 0, springe zu
                          // Programmzeile <label>.
3. MUL R5, R6, R7;        // Multipliziere Inhalt von R5 und R6, Erg. in R7.
```

  - **Problem**:

    Die Ausführung des MUL-Befehls wird angefangen, bevor die Sprungbedingung ausgewertet wurde.

  - **Mögliche Lösung**: Branch Prediction
    - „Raten" der nächsten Instruktion (z.B. durch Analysen der Häufigkeit des einen oder anderen Ausgangs der Abfrage).
    - Spekulativer Sprung
    - Leeren der Pipeline und „Rücksetzen" bei falscher Vorhersage.

Zusammenfassung: Pipelining

- Beschleunigung um bis zu Faktor k durch Einsatz einer Pipeline (k = Anzahl der Pipeline-Stufen).
- Hazards verringern die Beschleunigung.
- Viele Möglichkeiten (z.T. in Hardware) Hazards zu vermeiden:
  - **Softwaremäßig**:
    - NOPs
    - Umstellen der Instruktionen eines Maschinenprogramms durch Code-optimierende Compiler.
  - **Hardwaremäßig**:
    - Branch Prediction (Hardware merkt sich, ob bei den letzten Ausführungen des bedingten Sprungbefehls verzweigt wurde oder nicht.)
    - ...

Übersicht: Formen der Parallelität

- Parallelität auf Bitebene: bis etwa 1985
  - Kombinatorische Addierer und Multiplizierer, etc.
  - wachsende Wortbreite auf 64 Bit
- Parallelität auf Instruktionsebene: 1985 bis heute
  - Pipelining der Instruktionsverarbeitung
  - Mehrere Funktionseinheiten (superskalare Prozessoren), bei mehr als 4 Funktionseinheiten werden Datenabhängigkeiten oft zum Hindernis für eine effiziente Ausnutzung.
  - Vektorprozessoren führen eine Operation parallel auf vielen Daten durch (Bsp: Cray 1 [1974])
- Parallelität auf Prozessor-/Rechnerebene
  - Multiprozessoren mit mehreren Cores führen mehrere Prozesse / Threads parallel aus.