3.2 Pipelining

Ziel: Performanzsteigerung

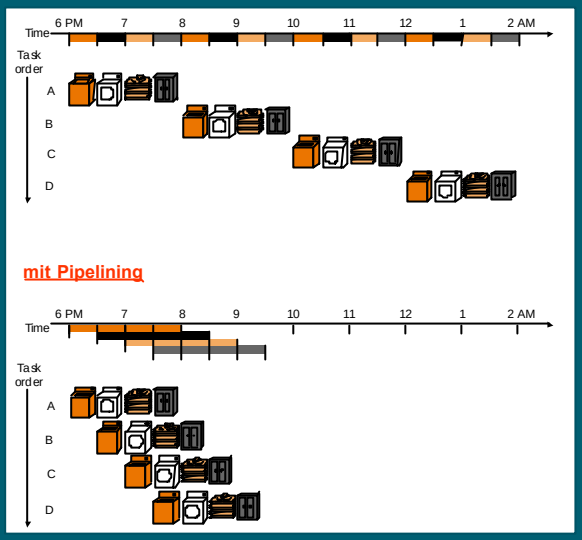
- Prinzip der Fließbandverarbeitung
- Probleme bei Fließbandverarbeitung

Das Prinzip an einem alltäglichen Beispiel

- Sie kommen aus dem Urlaub und es ist viel schmutzige Wäsche zu waschen!
- Zur Verfügung stehen:
  - eine Waschmaschine (1/2 Stunde Laufzeit)
  - ein Trockner (1/2 Stunde Laufzeit)
  - eine Bügelmaschine (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 seine Wäsche selbst

Es gibt zwei Möglichkeiten die vier Waschvorgänge auszuführen!

Das Prinzip an einem Beispiel



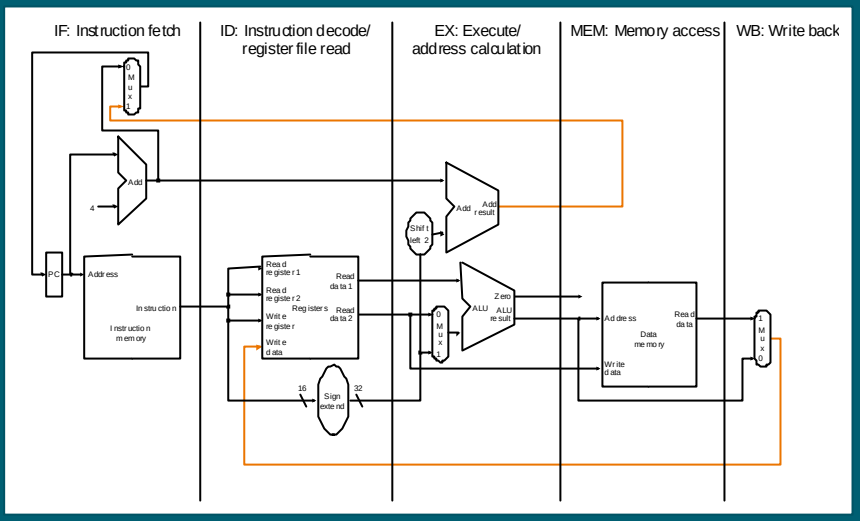
Dauer der Arbeiten:
8 Stunden

Dauer der Arbeiten:
3 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 5 Schritten:
  - Befehls-Holphase (instruction fetch)
  - Dekodierphase / Lesen von Operanden aus Registern
  - Ausführung / Adressberechnung
  - Speicherzugriff (memory access)
  - Abspeicherphase (result write back phase)
- Bei CISC ist Pipelining schwierig, da die Dauer der Dekodier- und Ausführungsphase (evtl. mehrere Mikroprogrammbefehle) bei den verschiedenen Maschinenbefehlen sehr unterschiedlich ist.

Aufteilung des Datenpfades in 5 Phasen



Pipelining: Illustration

- Annahme: Aufteilung der Befehlsabarbeitung in 5 gleichlange Phasen

Befehl 1:	P1	P2	P3	P4	P5						
Befehl 2		P1	P2	P3	P4	P5					
Befehl 3:			P1	P2	P3	P4	P5				
Befehl 4:				P1	P2	P3	P4	P5			
Befehl 5:					P1	P2	P3	P4	P5		
Befehl 6:						P1	P2	P3	P4	P5	
Befehl 7:							P1	P2	P3	P4	P5
Zeitschritt:	1	2	3	4	5	6	7	8	9	10	11

Pipelining: Speedup (1)

- Annahmen:
  - Abarbeitungszeit eines Befehls ohne Pipelining: *t*
  - *k* Pipelinestufen, gleiche Laufzeit der Stufen
  - ⇒ Laufzeit einer Stufe der Pipeline: *t/k*

- Beschleunigung bei *m* auszuführenden Instruktionen:

  m = 1: Laufzeit mit Pipeline = *k * t/k = t*
  ⇒ keine Beschleunigung

  m = 2: Laufzeit mit Pipeline = *t + t/k = (k+1)t / k*
  ⇒ Beschleunigung um Faktor *2 k / (k+1)*

Pipelining: Speedup (2)

m ³ 1: Laufzeit mit Pipeline = *t + (m-1)t / k*,

Laufzeit ohne Pipeline = *m t*

⇒ 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 ⇒ Hazards!

Pipelining: Hazards

- 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: Hazards

Nehme 4 Pipelinestufen an:

Takt	Befehlholen	Decodieren/ Operand holen	Ausführen	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!

Pipelining: Hazards

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

Takt	Befehlholen	Decodieren/ Operand holen	Ausführen	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!

Pipelining: Hazards

Beobachtung: 1 NOP kann eingespart werden, wenn das Ergebnis des LOAD-Befehls nach der Execute-Phase direkt der ALU zur Verfügung gestellt wird, um den ADD-Befehl auszuführen.

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

- **Aber:** Dazu ist Zusatzhardware nötig, die diese Datenabhängigkeiten erkennt und das **„Forwarding“** durchführt!

Pipelining: Hazards

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	Ausführen	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: Hazards

- Bedingte Verzweigungen (control hazards):
  - Befehlsfolge:

    1. ADD R1, R2, R3; // Addiere Inhalte der Register R1, R2, Erg. in R3
    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

- 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
  - **Forwarding**
  - **Branch Prediction** (Hardware merkt sich, ob bei der letzten Ausführung des bedingten Sprungbefehls verzweigt wurde)
  - **Umstellen der Instruktionen** eines Maschinenprogramms durch Code-optimierende Compiler