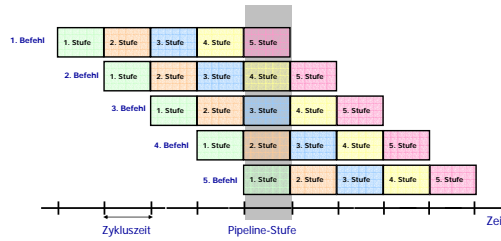
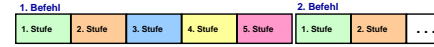


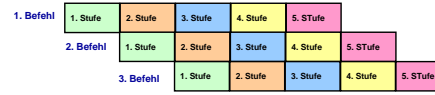
Pipeline-Verarbeitung



Sequentielle Ausführung:

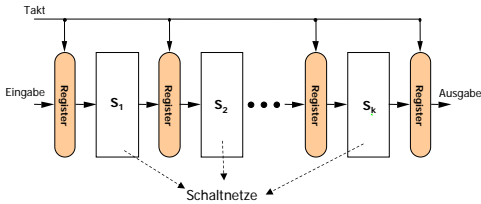


Pipelining:



- Pipelining:** Zerlegung einer Maschinenoperation in mehrere Phasen oder Suboperationen, die dann von hintereinander geschalteten Verarbeitungseinheiten **taktsynchron** bearbeitet werden, wobei jede Verarbeitungseinheit genau eine spezielle Teiloperation ausführt
- Die Gesamtheit dieser Verarbeitungseinheiten nennt man eine **Pipeline**.
- Bei einer **Befehlspipeline** (Instruction Pipeline) wird die Ausführung eines Maschinenbefehls in verschiedene Phasen unterteilt, aufeinanderfolgende Maschinenbefehle werden jeweils um einen Taktzyklus versetzt ausgeführt

5.2 Pipeline-Stufen und Pipeline-Register



Verzögerungszeiten:
 > der Schaltnetze: $\tau_i (i = 1, \dots, k)$
 > der Pipeline-Register: τ_{reg}

Länge eines Taktzyklus:
 $\tau = \max\{\tau_1, \tau_2, \dots, \tau_k\} + \tau_{reg}$

Definitionen

- Jede Stufe der Pipeline heißt **Pipeline-Stufe** oder **Pipeline-Segment**.
- Pipeline-Stufen werden durch getaktete **Pipeline-Register** (auch *latches* genannt) getrennt.
- Ein Pipeline-Maschinentakt ist die Zeit, die benötigt wird, um einen Befehl eine Stufe weiter durch die Pipeline zu schieben.
- Idealerweise wird ein Befehl in einer **k-stufigen Pipeline** in **k** Takten von **k** Stufen ausgeführt.
- Wird in jedem Takt ein neuer Befehl geladen, dann werden zu jedem Zeitpunkt unter idealen Bedingungen **k** Befehle gleichzeitig behandelt und jeder Befehl benötigt **k** Takte, bis zum Verlassen der Pipeline.

Definitionen

- Latenz:** die Zeit, die ein Befehl benötigt, um alle **k** Pipeline-Stufen zu durchlaufen.
- Ideale Verhältnisse:**
 - Ausführung eines Befehls in **k** Takten.
 - Es werden gleichzeitig **k** Befehle bearbeitet.
- Durchsatz einer Pipeline:** Anzahl der Befehle, die eine Pipeline pro Takt verlassen können. Dieser Wert spiegelt die Rechenleistung einer Pipeline wider.

Leistungssteigerung durch Pipelining

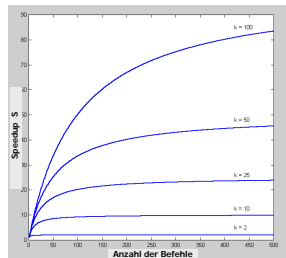
- n Befehle in einer Pipeline mit k Stufen**
- Hypothetischen Prozessor ohne Pipeline: **n * k** Taktzyklen
- Pipeline-Prozessor mit einer k-stufigen Pipeline: **k + (n-1)** Taktzyklen (unter der Annahme idealer Bedingungen mit einer Latenz von k Takten und einem Durchsatz von 1)
 - k Taktzyklen, um die Pipeline zu füllen
 - (n-1) Taktzyklen, um die restlichen (n-1) Befehle auszuführen.

Leistungssteigerung **S** (speedup): $S = \frac{n * k}{k + (n-1)}$

Leistungssteigerung durch Pipelining

$$S = \frac{n * k}{k + (n-1)}$$

$$\lim_{n \rightarrow \infty} S = k$$



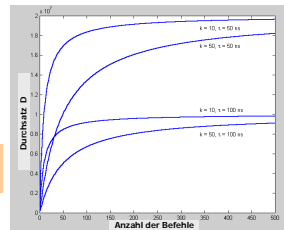
Durchsatz

- Der **Durchsatz** gibt an, wie viele Befehle in einem Zeitraum $T_k * \tau$ ausgeführt werden.

$$D = \frac{n}{T_k * \tau}$$

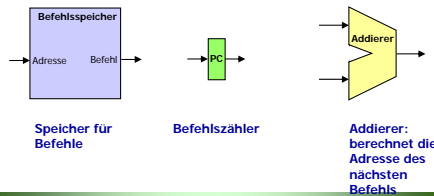
$$= \frac{n}{(k + (n-1)) * \tau}$$

$$\lim_{n \rightarrow \infty} D = \frac{1}{\tau} = D_{max}$$



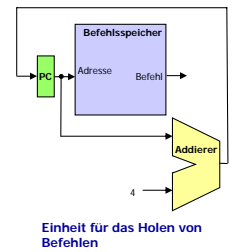
5.3 Befehlsabarbeitung und Datenpfade der MIPS-Befehle

- Welche **Hardware-Komponenten** sind zur Ausführung der **MIPS-Befehle** notwendig?
 > Für alle Befehlsklassen werden folgende Komponenten benötigt:



5.3 Befehlsabarbeitung und Datenpfade

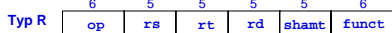
- Befehl aus dem Befehlspeicher holen
 - Befehl im Befehlspeicher adressieren
 - Befehlszähler um 4 inkrementieren



5.3 Befehlsabarbeitung und Datenpfade

- Befehle vom R-Typ:** opcode r_s, r_m, r_n

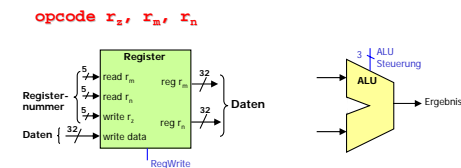
- Arithmetisch logische Befehle: **add, sub, and or**
- Vergleichsbefehle: **slt**



- Befehle haben 3 Operanden
- Operanden stehen in Registern
- Lesezugriff auf beiden Operanden-Register und
- ein Schreibzugriff auf das Zielregister

5.3 Befehlsabarbeitung und Datenpfade

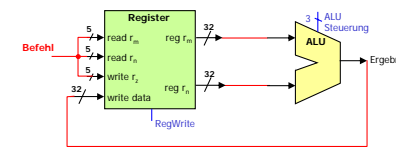
- Befehle vom R-Typ:** opcode r_s, r_m, r_n



- Lesezugriff auf beiden Operanden-Register und
- ein Schreibzugriff auf das Zielregister

5.3 Befehlsabarbeitung und Datenpfade

- Befehle vom R-Typ:**

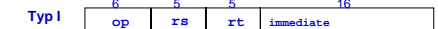


- Lesezugriff auf beiden Operanden-Register und
- ein Schreibzugriff auf das Zielregister

5.3 Befehlsabarbeitung und Datenpfade

- Lade- und Speicherbefehle (load and store)**

```
lw r_s, offset(r_n)    sw r_n, offset(r_n)
```



- Speicheradresse wird durch die Addition einer Basisadresse im Register r_n zu einem vorzeichenbehafteten 16-bit offset berechnet

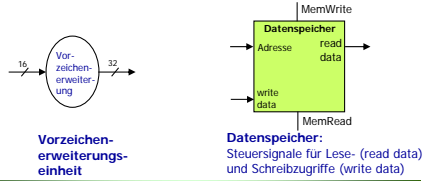
- Bei Speicherbefehle wird das zu speichernde Wort aus dem Register r_n gelesen. Bei Ladebefehlen wird das geladene Wort ins Register r_s geladen

5. 3 Befehlsabarbeitung und Datenpfade

□ Lade- und Speicherbefehle (load and store)

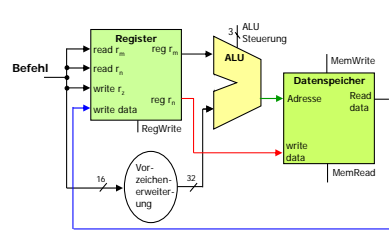
lw r_r , offset(r_n) **sw** r_n , offset(r_n)

➤ Weitere benötigte Komponenten:



5. 3 Befehlsabarbeitung und Datenpfade

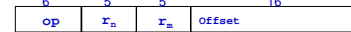
□ Lade- und Speicherbefehle (load and store)



5. 3 Befehlsabarbeitung und Datenpfade

□ Verzweigungsbefehle

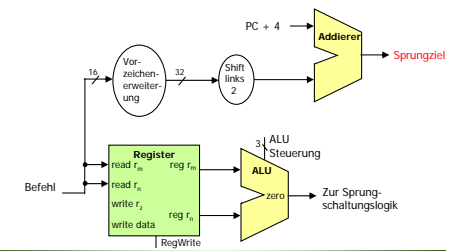
beq r_n , r_m , offset



- 16-bit vorzeichenbehaftetes Offset
- $2^{15}-1$ Befehle vorwärts und 2^{15} rückwärts
- Basisadresse zur Berechnung der Sprungadresse ist die Adresse des Befehls hinter dem Verzweigungsbefehl, d. h. (PC+4)
- Offset: um 2 Bits nach links verschieben, um Wörter zu adressieren
- Ergebnis des Vergleichs von r_n und r_m entscheidet, ob der Sprung „genommen“ bzw. „nicht genommen“ wird

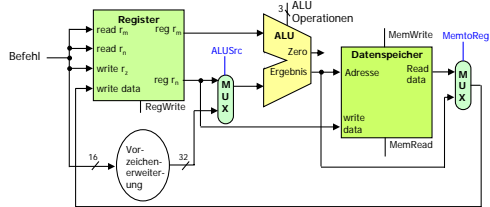
5. 3 Befehlsabarbeitung und Datenpfade

□ Verzweigungsbefehle

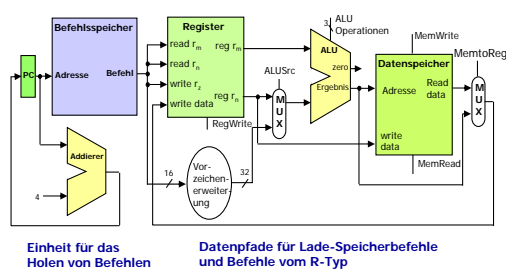


5. 3 Befehlsabarbeitung und Datenpfade

□ Datenpfad für Lade-Speicherbefehle und Befehle vom R-Typ

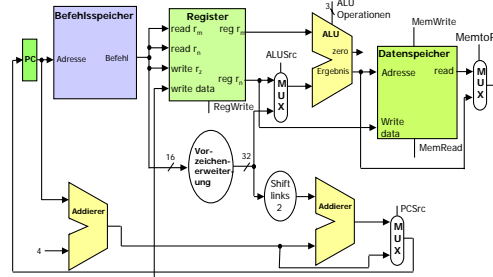


5. 3 Befehlsabarbeitung und Datenpfade



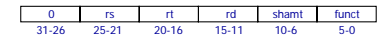
Einheit für das Holen von Befehlen Datenpfad für Lade-Speicherbefehle und Befehle vom R-Typ

Datenpfad für die MIPS-Architektur

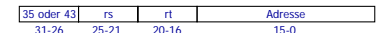


Erinnerung: MIPS Befehlsformate

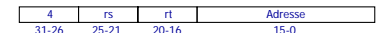
➤ R-Typ Befehl



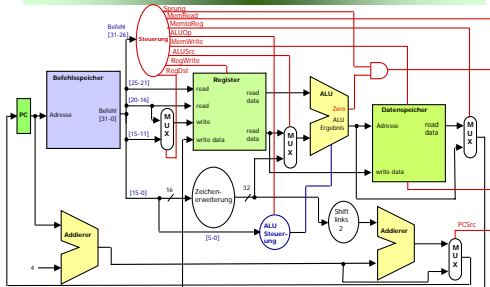
➤ Lade/Speicher Befehl



➤ Sprung Befehl



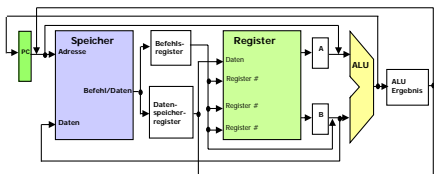
Datenpfad für die MIPS Architektur



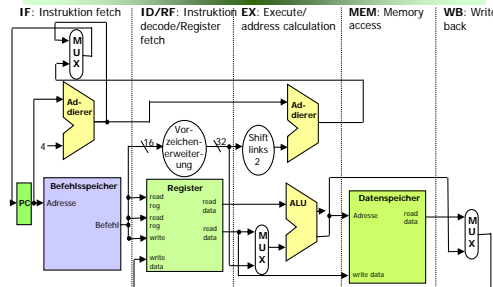
5.4. Pipelining in MIPS Architektur

□ Schlüsseleinheiten des Datenpfades

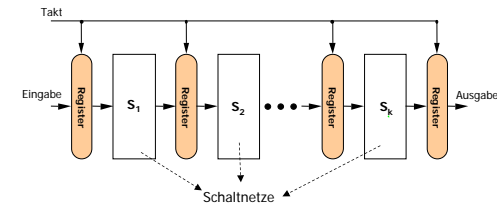
- Eine ALU
- Eine Speichereinheit für Daten und Befehle
- Register: Befehlsregister, Datenspeicherregister, A, B und ALU-Ergebnisregister



5.4. Pipelining in MIPS Architektur



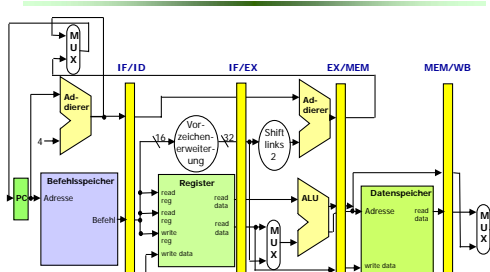
Pipeline-Stufen und Pipeline-Register



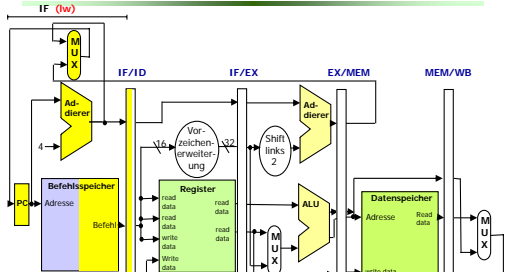
Verzögerungszeiten:
 ➤ der Schaltzette: τ_i ($i = 1, \dots, k$)
 ➤ der Pipeline-Register: τ_{reg}

Länge eines Taktzyklus:
 $\tau = \max\{\tau_1, \tau_2, \dots, \tau_k\} + \tau_{reg}$

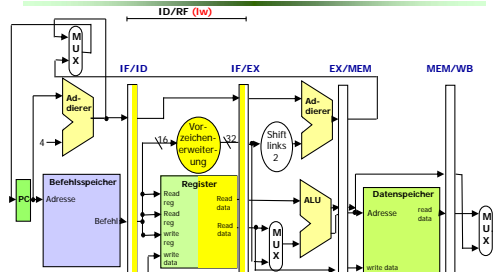
5.4. Pipelining in MIPS Architektur



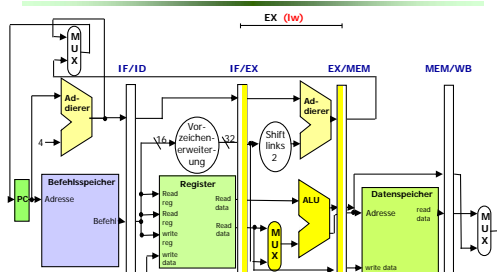
5.4. DLX Pipelinestufen



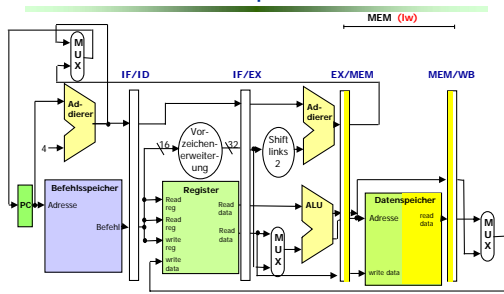
5.4. DLX Pipelinestufen



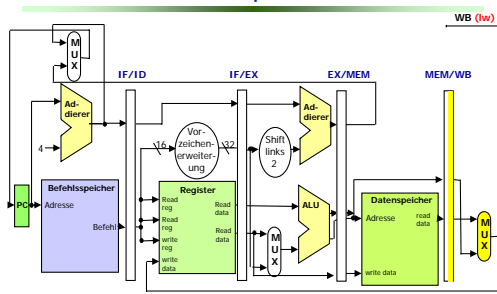
5.4. DLX Pipelinestufen



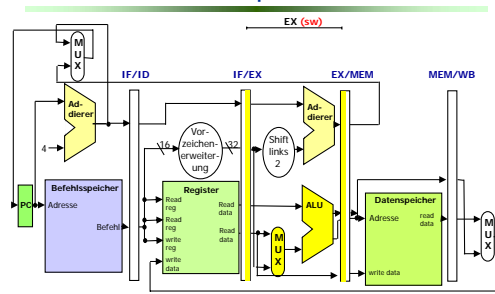
5.4. DLX Pipelinestufen



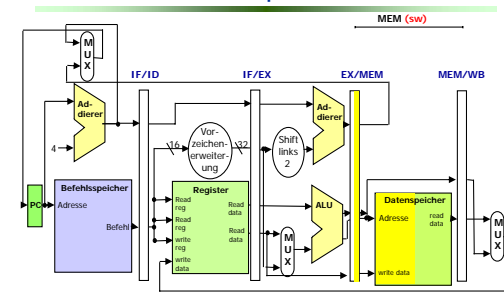
5.4. DLX Pipelinestufen



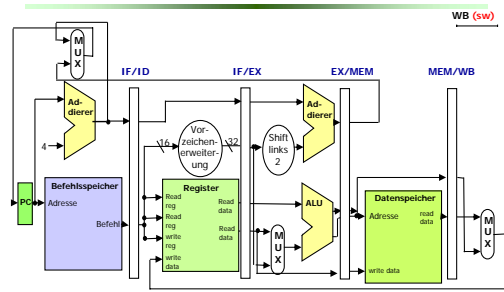
5.4. DLX Pipelinestufen



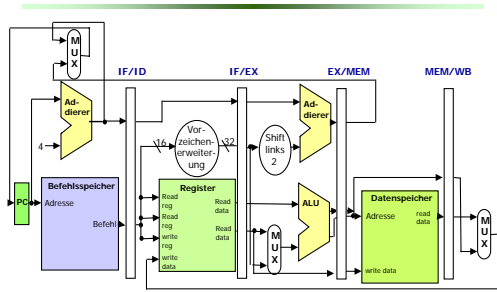
5.4. DLX Pipelinestufen



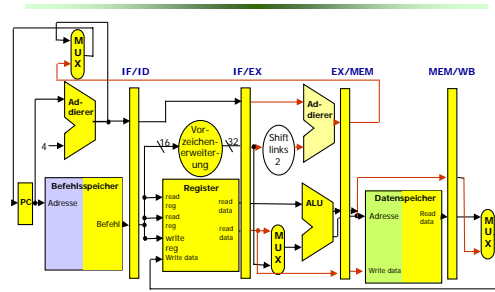
5.4. DLX Pipelinestufen



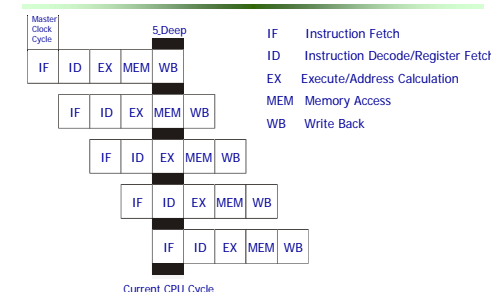
5.4. DLX Pipelinestufen



5.4. DLX Pipelinestufen



5.4. DLX Pipelinestufen



Phasen der Befehlsausführung in einer fünfstufigen Pipeline (DLX-Pipeline)

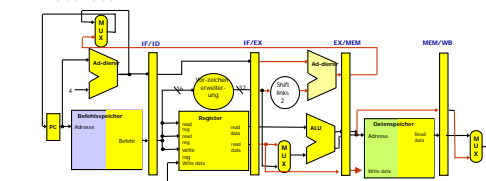
- Befehlsbereitstellungs-Phase (IF-Phase: Instruction Fetch)**
 Der durch den Befehlszähler adressierte Befehl wird aus dem Arbeitsspeicher (bzw. dem Befehlsache) in einen Befehlspuffer geladen. Der Befehlszähler wird weitergeschaltet.
- Dekodier- und Operandenbereitstellungsphase (ID-Phase: Instruction Decode & Register Fetch)**
 Aus dem Operationscode des Maschinenbefehls werden prozessorinterne Steuersignale erzeugt. Die Operanden werden aus (Universal-)Register bereit gestellt (2. Takthälfte).
- Ausführungsphase / Berechnung der effektiven Adresse (EX-Phase: Execution/Effective Address Calculation)**
 Die Operation wird auf den Operanden ausgeführt. Bei Lade- und Speicherbefehlen oder Verzweigungen berechnet die ALU die effektive Adresse

Phasen der Befehlsausführung in einer fünfstufigen Pipeline (DLX-Pipeline)

- Speicherzugriffsphase (MEM-Phase: memory access)**
 Der Speicherzugriff (bei Lade- und Speicherbefehlen) wird durchgeführt
- Resultatspeicherphase (WB-Phase: write back):**
 Das Ergebnis wird in ein (Universal-)Register geschrieben (1. Takthälfte). Befehle ohne Ergebnis durchlaufen diese Phase passiv.

5.5 Pipeline Konflikte

- Bei einer gefüllten Pipeline wird pro Taktzyklus ein Befehl beendet.



5.5 Pipeline Konflikte

- Es gibt leider mehrere potentielle Probleme, die den Durchfluss durch die Pipeline hemmen bzw. verzögern. Man spricht von **Pipeline-Hemmnissen**
- Diese Verzögerungen entstehen durch **Daten-, Struktur- und Steuerflussabhängigkeiten**

Drei Arten von Pipeline Konflikten

- Datenkonflikte:** Treten auf, wenn ein Operand ist in der Pipeline (noch) nicht verfügbar.
 - Datenkonflikte werden durch Datenabhängigkeiten im Befehlsstrom erzeugt
- Struktur- oder Ressourcenkonflikte:** Treten auf, wenn zwei Pipeline-Stufen dieselbe Ressource benötigen, auf diese aber nur einmal zugegriffen werden kann.
- Steuerflusskonflikte** treten bei Programmsteuerbefehlen auf:
 - wenn in der Befehlsbereitstellungsphase die Zieladresse des nächsten auszuführenden Befehls noch nicht berechnet ist bzw.
 - wenn im Falle eines bedingten Sprunges noch nicht klar ist, ob überhaupt gesprungen wird.

5.5.1 Datenabhängigkeiten

- Zwischen zwei aufeinander folgenden Befehle $Inst_1$ und $Inst_2$ besteht eine **echte Datenabhängigkeit (true dependence) δ^t** von $Inst_1$ zu $Inst_2$, wenn $Inst_1$ seine Ausgabe in ein Register Reg (oder in den Speicher) schreibt, das von $Inst_2$ als Eingabe gelesen wird.

5.5.1 Datenabhängigkeiten

- Zwischen zwei aufeinander folgenden Befehle $Inst_1$ und $Inst_2$ besteht eine **Gegenabhängigkeit (antidependence) δ^a** von $Inst_1$ zu $Inst_2$, falls $Inst_1$ Daten von einem Register Reg (oder einer Speicherstelle) liest, das anschließend von $Inst_2$ überschrieben wird.

5.5.1 Datenabhängigkeiten

- Zwischen zwei aufeinander folgenden Befehle $Inst_1$ und $Inst_2$ besteht eine **Ausgabeabhängigkeit (output dependence) δ^o** von $Inst_2$ zu $Inst_1$, wenn beide in das gleiche Register Reg (oder eine Speicherstelle) schreiben und $Inst_2$ sein Ergebnis nach $Inst_1$ schreibt.