

Vorlesung Rechnerstrukturen

□ **Kapitel 2: Architektur und Mikroarchitektur von Mikroprozessoren**

RISC

□ RISC - Superskalar

- ◆ Skalare RISC-Prozessoren:
 - ◆ Entwurfsziel: durchschnittlich eine Befehlsausführung pro Takt
 - ◆ CPI – *cycles per instruction*
- ◆ Die Superskalar-Technik ermöglicht es heute, pro Takt bis zu vier Befehle den Ausführungseinheiten zuzuordnen und eine gleiche Anzahl von Befehlsausführungen pro Takt zu beenden.
- ◆ Solche Prozessoren werden als **superskalare (RISC)-Prozessoren** bezeichnet, da die oben definierten RISC-Charakteristika auch heute noch weitgehend beibehalten werden.
- ◆ Heutige Mikroprozessoren nutzen Befehlsebenenparallelität durch die Pipelining- und Superskalartechnik

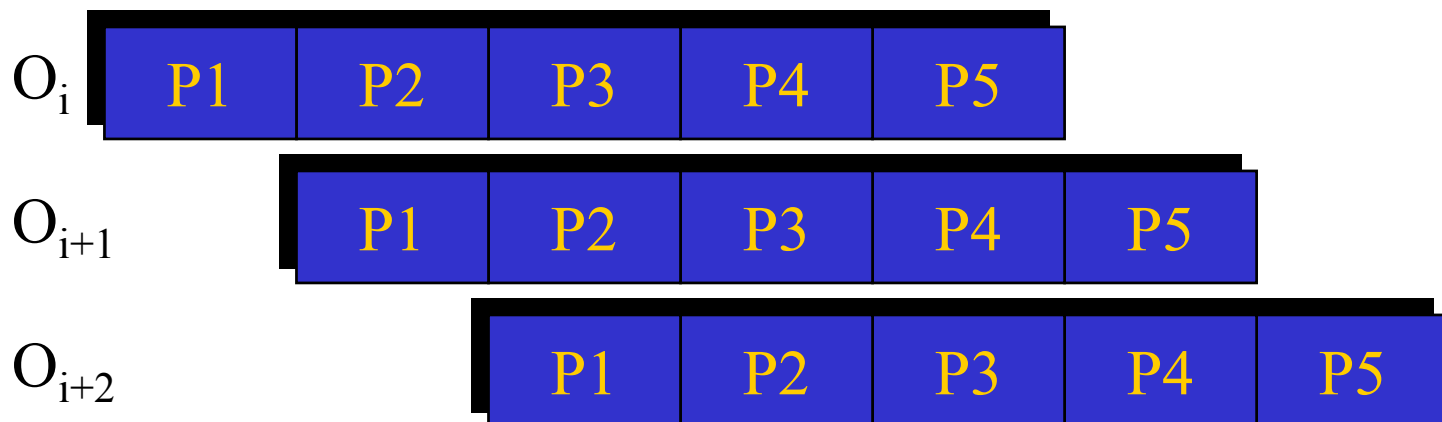
Vorlesung Rechnerstrukturen

- **Kapitel 2: Architektur und Mikroarchitektur von Mikroprozessoren**
 - ◆ **2.6 Grundlegende Pipeline-Stufen**

Pipelining

□ Definition (siehe Vorlesung 2)

- ◆ *Pipelining auf einer Maschine liegt dann vor, wenn die Bearbeitung eines Objektes in Teilschritte zerlegt und diese in einer sequentiellen Folge (Phasen der Pipeline) ausgeführt werden. Die Phasen der Pipeline können für verschiedene Objekte überlappt abgearbeitet werden. (Bode 95)*



Pipelining

□ Befehlspipelining (Instruction Pipelining)

- ◆ *Zerlegung der Ausführung einer Maschinenoperation in Teilphasen, die dann von hintereinander geschalteten Verarbeitungseinheiten taktsynchron bearbeitet werden, wobei jede Einheit genau eine spezielle Teiloperation ausführt.*

□ Pipeline

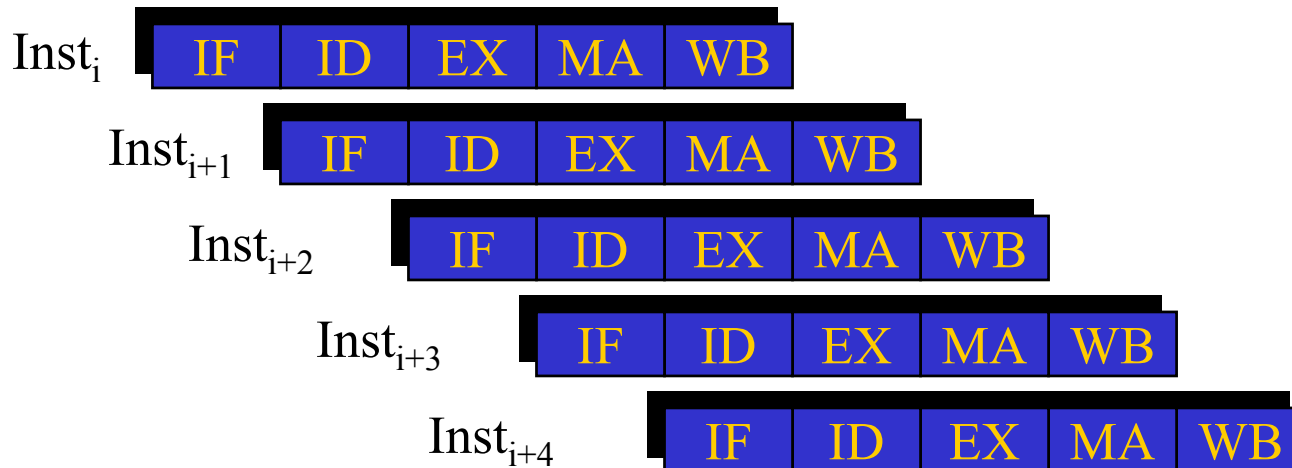
- ◆ Gesamtheit der Verarbeitungseinheiten

□ Pipeline-Stufe

- ◆ Stufen der Pipeline, die jeweils durch Pipeline-Register getrennt sind

Pipelining

□ K-stufige Befehlspipeline (k=5)



IF: Befehl holen
ID: Befehl dekodieren
EX: Befehl ausführen
MA: Speicherzugriff
WB: Zurückschreiben

Pipeline-
Stufe | 1 Takt-
zyklus

Pipelining

❑ Phasen der Befehlspipeline

❑ Befehlsbereitstellungsphase (*Instruction Fetch*):

- ♦ Der Befehl, der durch den Befehlszähler adressiert ist, wird aus dem Hauptspeicher oder dem Cache-Speicher in einen Befehlspeicher geladen. Der Befehlszähler wird weitergeschaltet.

❑ Decodier- und Operandenbereitstellungsphase (*Decode/Operand fetch*):

- ♦ Aus dem Operationscode des Maschinenbefehls werden prozessorinterne Steuersignale erzeugt. Die Operanden werden aus Registern bereitgestellt.

❑ Ausführungsphase (*ALU Operation*):

- ♦ Die Operation wird auf den Operanden ausgeführt. Bei Lade-/Speicherbefehlen wird die effektive Adresse berechnet.

Pipelining

- ❑ **Phasen der Befehlspipeline**
- ❑ Speicherzugriffsphase (*memory access*):
 - ◆ Der Speicherzugriff wird durchgeführt.
- ❑ Resultatspeicherphase (*Write Back*):
 - ◆ Das Ergebnis wird in ein Register geschrieben.

Pipelining

□ Latenz L

- ♦ Zeit, die ein Befehl zum Durchlaufen der Pipeline benötigt;
 - ♦ Ausführung eines Befehls in k Takten (ideale Verhältnisse)
 - ♦ Gleichzeitige Behandlung von k Befehlen gleichzeitig (ideale Verhältnisse), jeder Befehl benötigt k Takte bis zum Verlassen der Pipeline

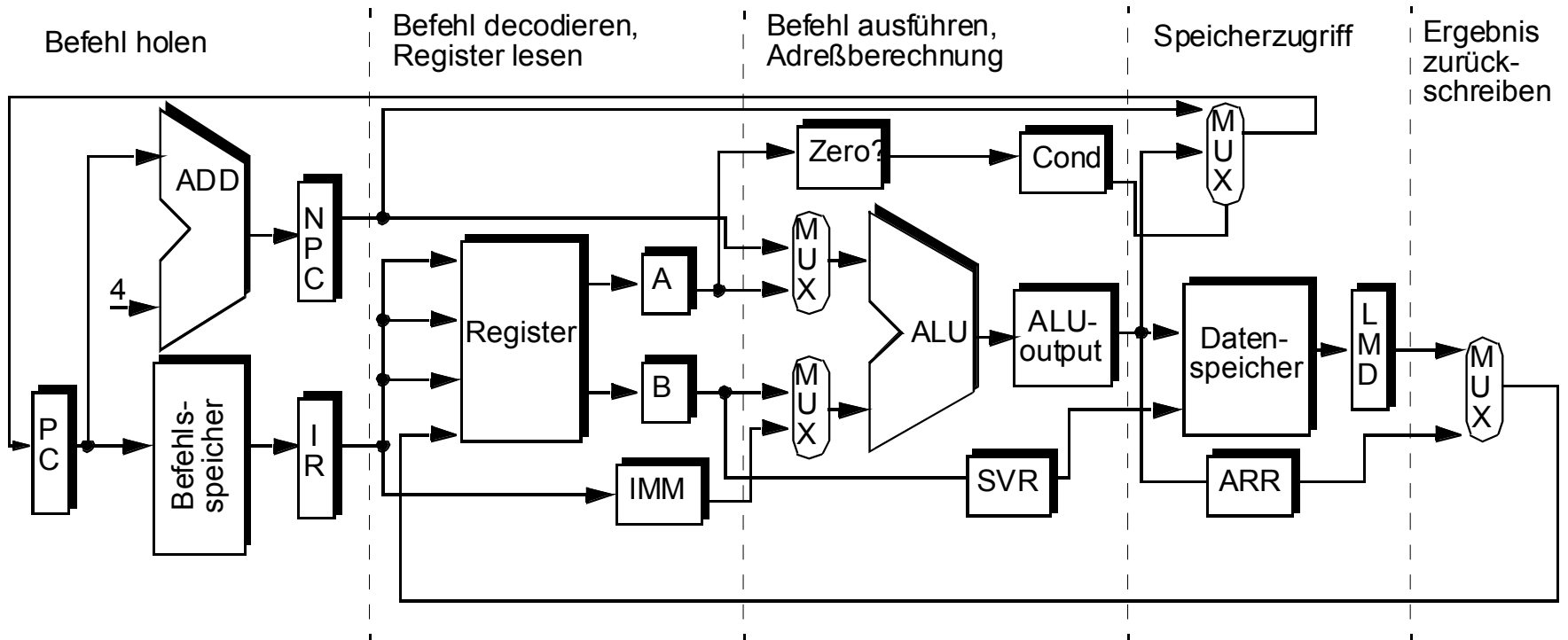
□ Durchsatz T

- ♦ Anzahl der Befehle, die eine Pipeline pro Takt verlassen können.
- ♦ $T = n + k - 1$; n : Anzahl der Befehle in einem Programm
(Annahme: ideale Verhältnisse, Latenz $L = k$ Takte, $T = 1$)

□ Beschleunigung S

- ♦ $S = n * k / (k + n - 1) = k / (k/n + 1 - 1/n)$

Beispiel: DLX-Pipeline



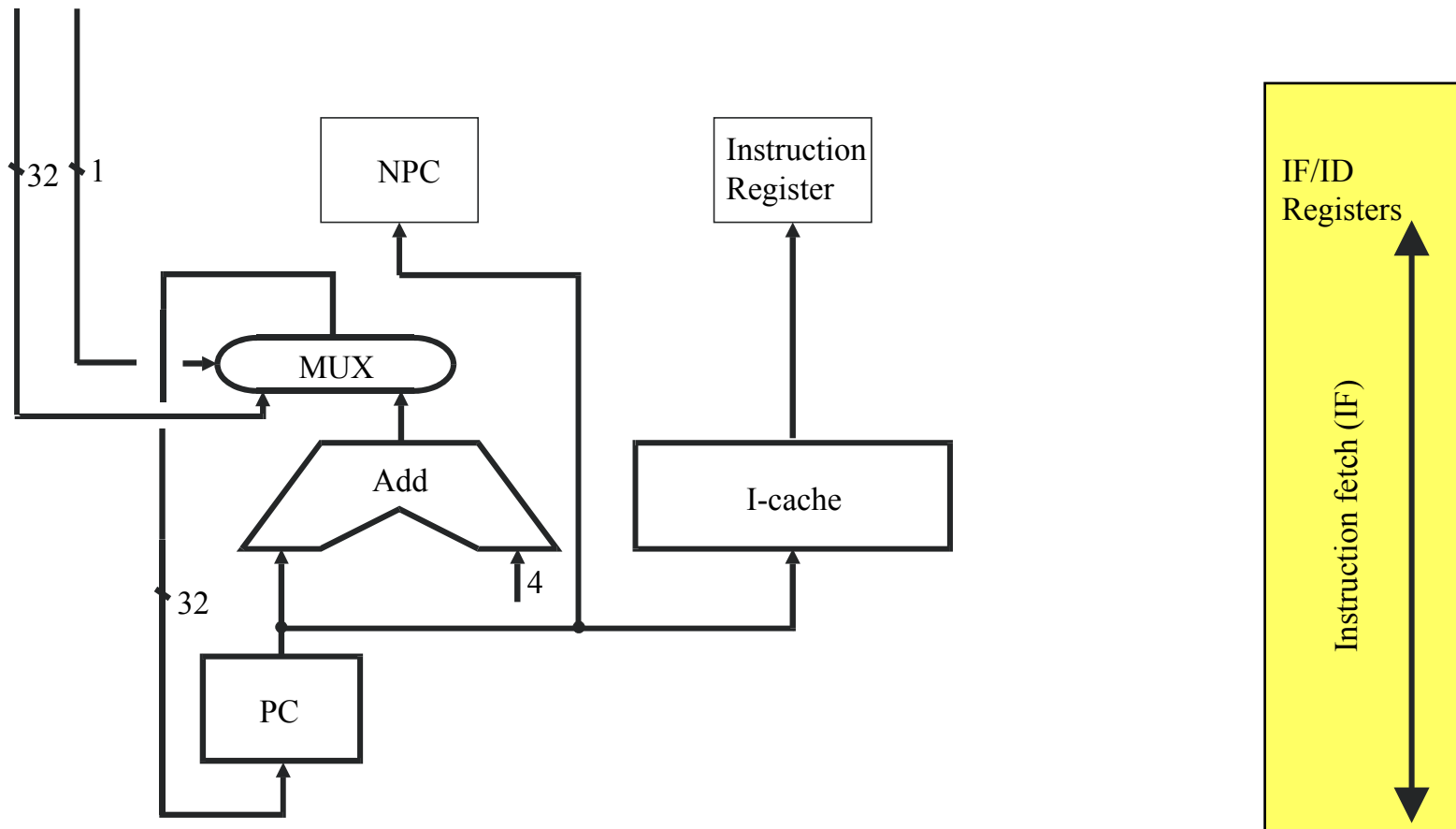
Phasen der Befehlspipeline

□ Befehlsbereitstellungsphase (Instruction Fetch)

- ◆ Holen des Befehls, auf den der PC zeigt, aus dem Befehlsspeicher (Befehls-cache) in das Befehlsregister (Instruction Register, IR)
- ◆ Erhöhung des PCs um 4
(Annahme: 32 Bit Befehlsformat, 4 Byte-adressierbarer Speicher)
- ◆ Bei vorangegangenen Sprungbefehl kann die Zieladresse aus der MA-Stufe benutzt werden, um den PC auf die im nächsten Takt zu holende Anweisung zu setzen

Phasen der Befehlspipeline

□ Befehlsbereitstellungsphase (Instruction Fetch)



Phasen der Befehlspipeline

- ❑ Dekodier- und Operandenbereitstellungsphase (Instruction Decode / Register Fetch, ID)
 - ◆ Dekodieren des Befehls
 - ◆ Weitere Aktionen in Abhängigkeit des Befehls:
 - ◆ **Register-Register** (arithmetische oder logische Befehle)
Laden der ALU-Eingaberegister (Latches)

Phasen der Befehlspipeline

□ Dekodier- und Operandenbereitstellungsphase (Instruction Decode / Register Fetch, ID)

♦ Aktionen in Abhängigkeit des Befehls:

♦ Speichereferenz (Laden, Speichern)

Weitergeben eines Teils der Speicheradresse in ALU-Eingaberegister;

Weiterleiten einer konstanten Abstandsgröße des Befehls
(vorzeichenerweitert) in das Immediate Register

Bei einem Sprungbefehl: Laden des ALU-Eingaberegisters

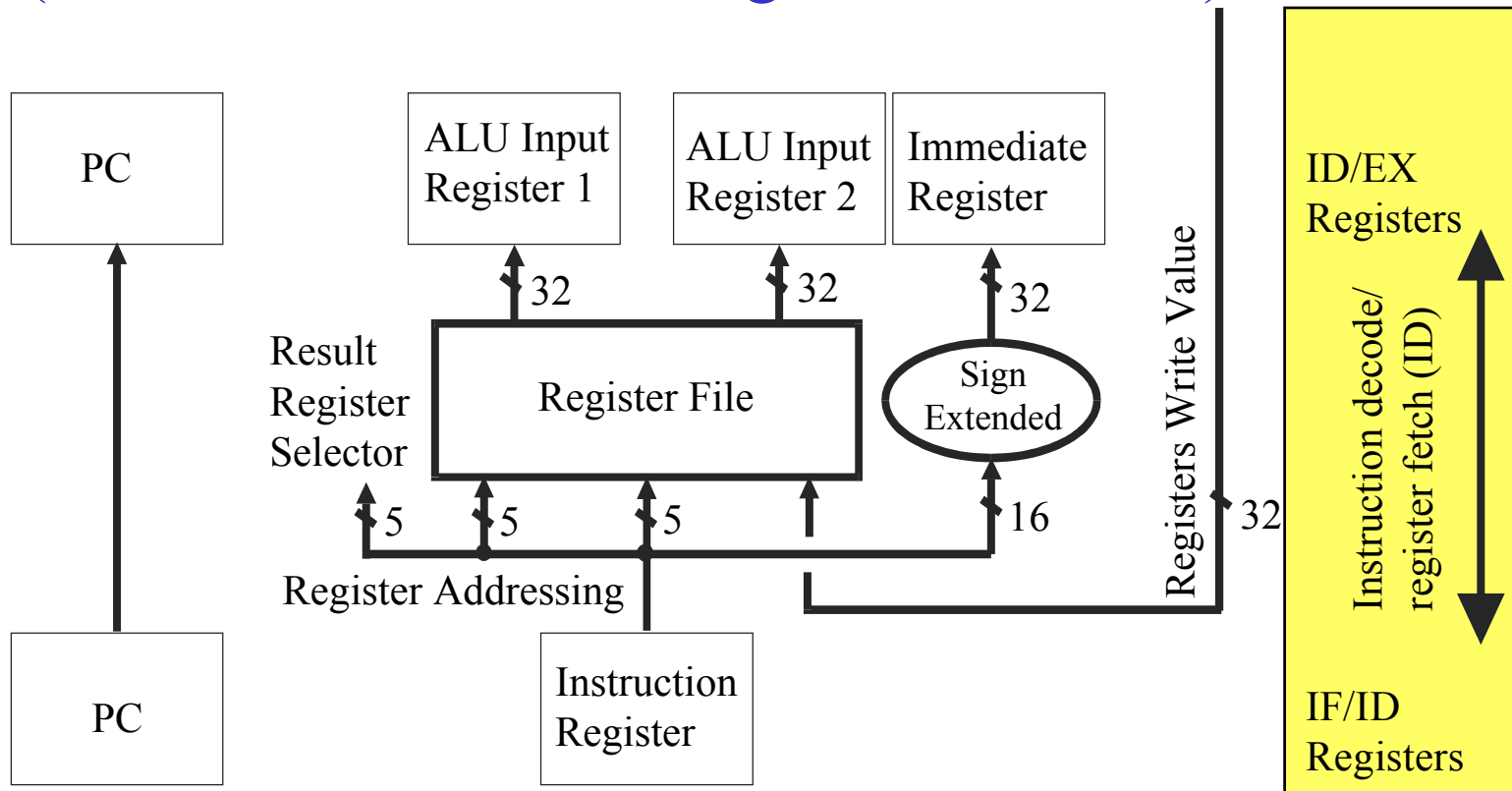
♦ Steuerungstransfer

Laden des Immediate Registers mit der im Befehl angegebenen
konstanten Abstandsgröße (PC-relative Adressierung);

Bedingter Sprung: Registerwert, der Sprungrichtung angibt in ALU-
Eingangsregister

Phasen der Befehlspipeline

- ❑ Dekodier- und Operandenbereitstellungsphase (Instruction Decode / Register Fetch, ID)

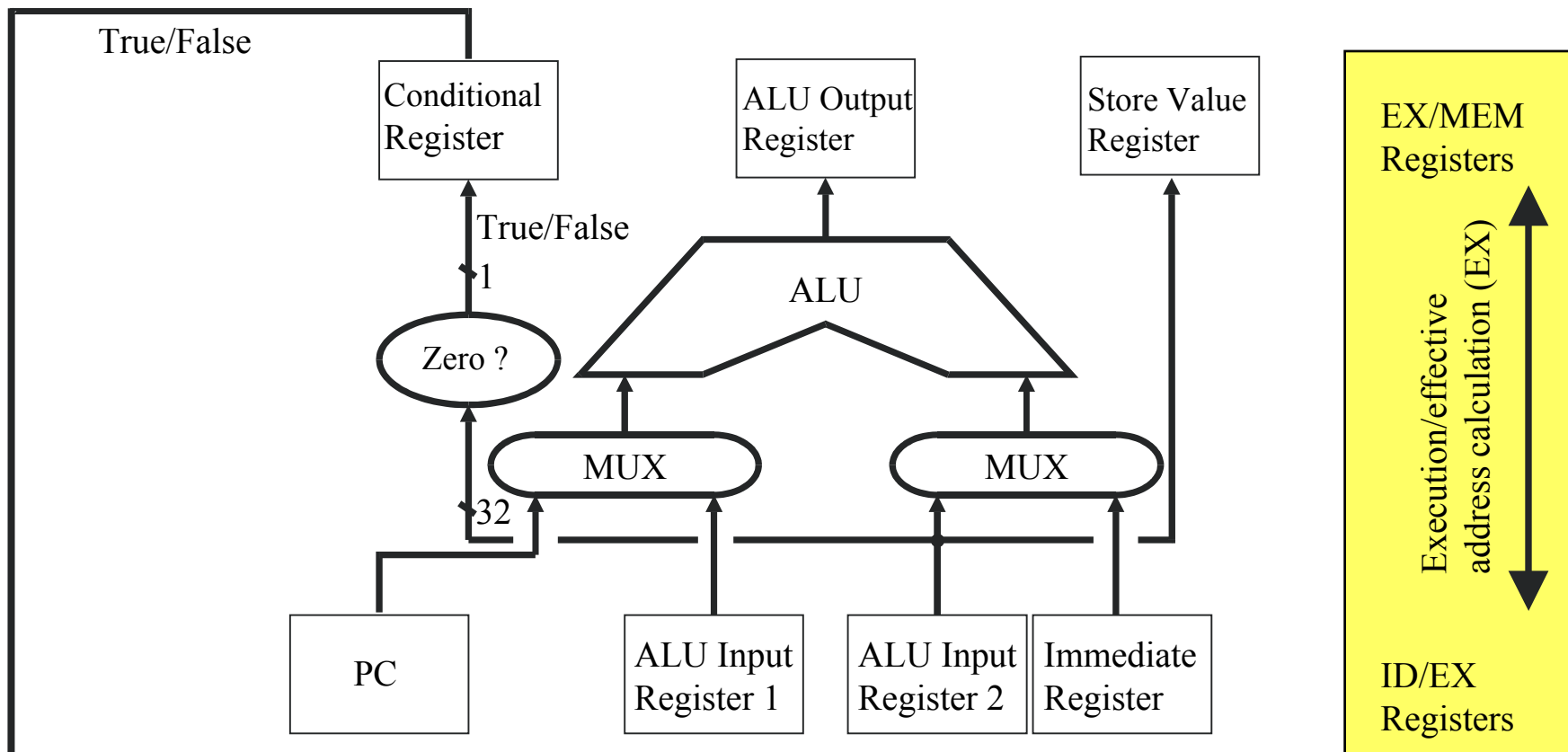


Phasen der Befehlspipeline

- ❑ Ausführungsphase / Effektive Adressberechnung (Execution / effective Address Calculation, EX)
 - ◆ Befehlsausführung
 - ◆ Verarbeiten der Operanden in den ALU-Eingaberegistern, im PC oder im IMM und Ablegen des Ergebnisses in ALU-Ausgaberegister;
 - ◆ *Register-Register* (arithmetische, logische Befehle)
Ausgeben des Ergebnisses in ALU-Ausgaberegister
 - ◆ *Speicherreferenz* (Lade, Speicher)
ALU-Ausgaberegister enthält die effektive Speicheradresse;
Bei einem Speicherbefehl wird der im ALU-Eingaberegister 2 stehende zu speichernde Wert in das Speicherwertregister geschrieben.
 - ◆ *Steuerungstransfer*
Berechnung der Zieladresse;
Test der Sprungbedingung und Speichern des Ergebnisses in Bedingungsregister

Phasen der Befehlspipeline

- Ausführungsphase / Effektive Adressberechnung (Execution / effective Address Calculation, EX)



Phasen der Befehlspipeline

- Speicherzugriff / Sprungvollendung (Memory Access / Branch Completion)
 - ◆ Ausführung für Lade-, Speicher- und Sprungbefehle
 - ◆ *Register-Register*
Weitergeben des Werts im ALU-Ausgaberegister nach ALU-Ergebnisregister
 - ◆ *Laden*
Lesen des Datums aus dem Datenspeicher (Cache) in das Load Memory Data Register (Adresse steht im ALU-Ausgaberegister)
 - ◆ *Speichern*
Schreiben der Daten in den Datenspeicher (Cache)
 - ◆ *Steuerungstransfer*
Sprünge: Ersetzen des PCs in IF durch den Inhalt im ALU-Ausgaberegister; bei False-Bedingung: Adresse aus PC

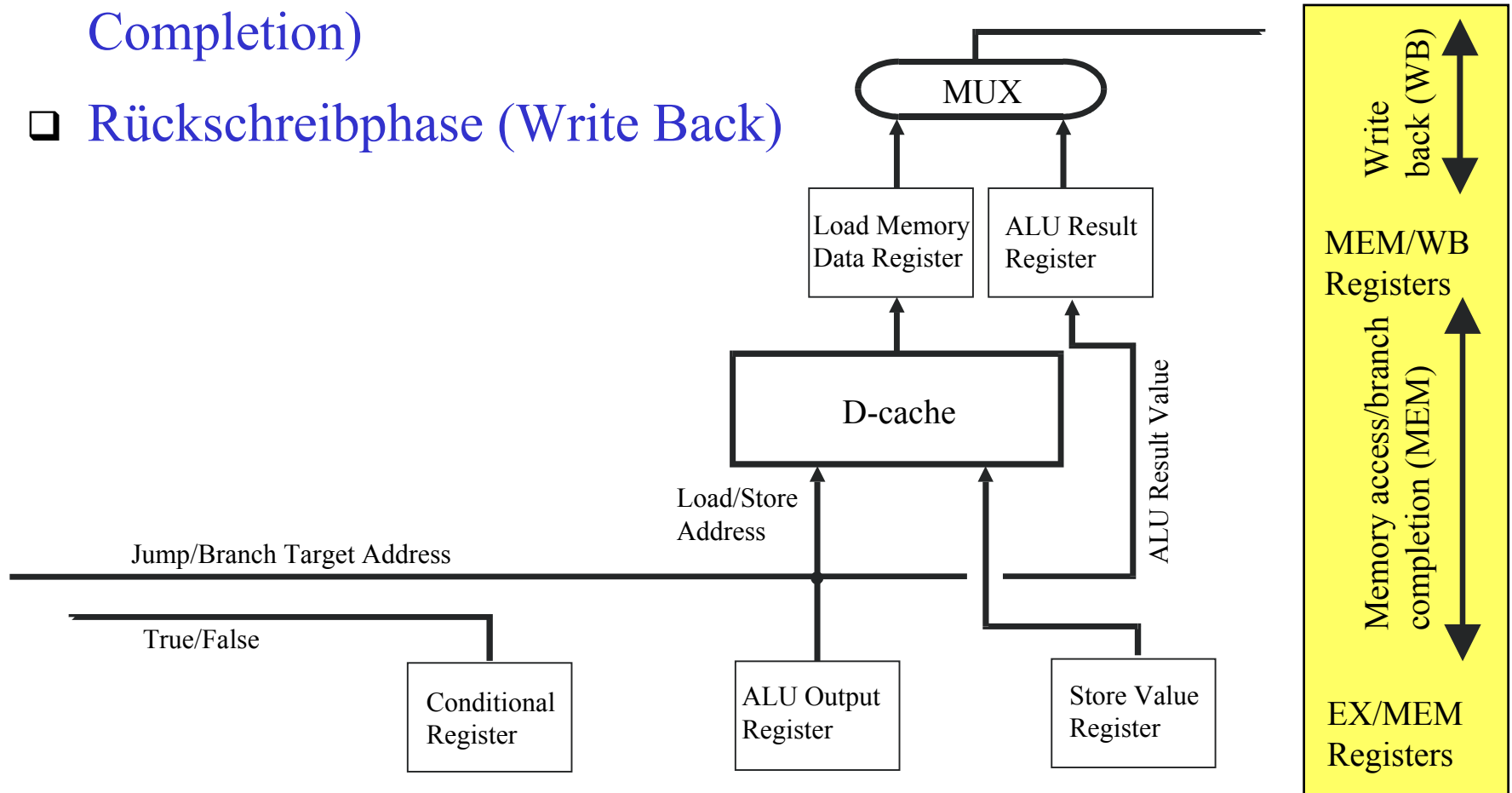
Phasen der Befehlspipeline

□ Rückschreibphase (Write Back, WB)

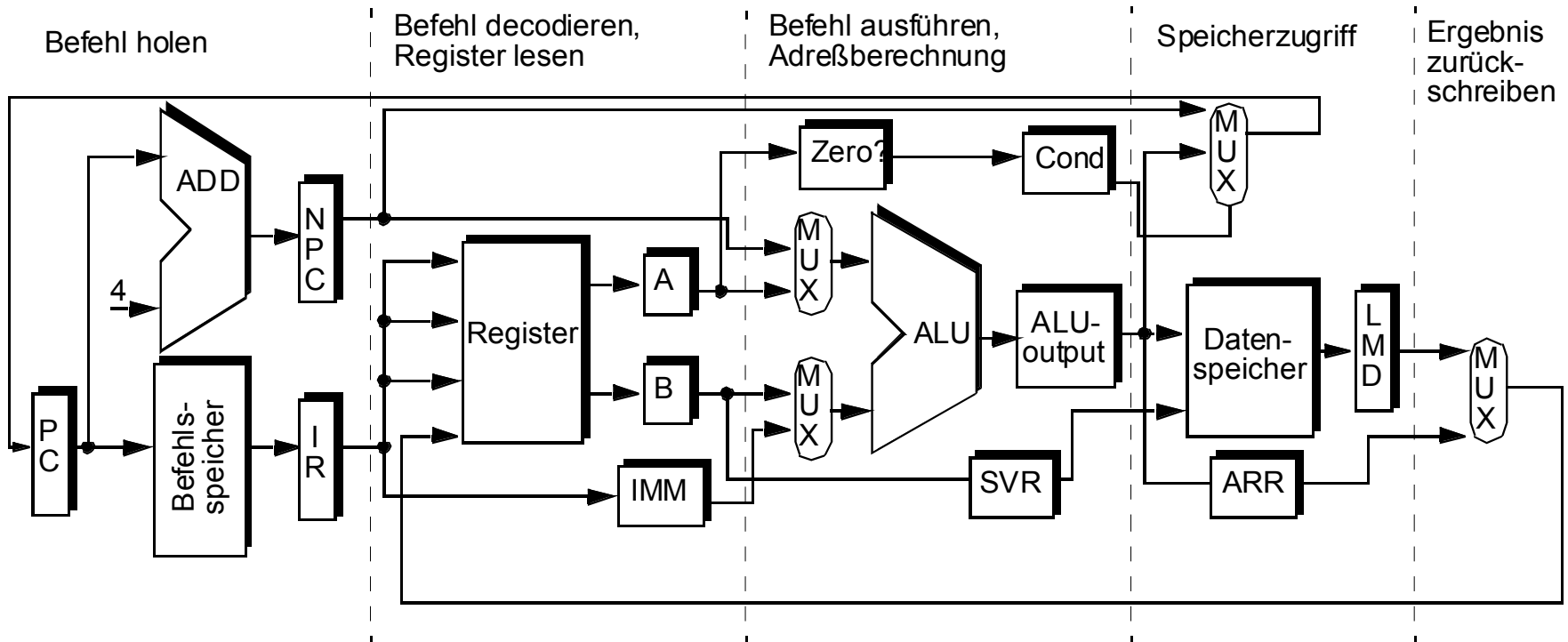
- ◆ Speichern des Ergebnisses der Ausführung von Register-Register- oder einem Ladebefehl in die Register
 - ◆ Schreiben der Inhalte des Load Memory Data Register und des ALU-Ausgaberegisters

Phasen der Befehlspipeline

- ❑ Speicherzugriff / Sprungvollendung (Memory Access / Branch Completion)
- ❑ Rückschreibphase (Write Back)



Befehlspipeline: Übersicht



Befehlspipeline: Diskussion

- ❑ Zykluszeit, Taktzyklus:
 - ◆ Abhängig vom kritischen Pfad, der langsamsten Pipelinestufe
- ❑ Alle Pipelinestufen benützen unterschiedliche Ressourcen:
 - ◆ keine Ressourcenkonflikte!
- ❑ Mit jedem Takt wird unter Annahme idealer Verhältnisse ein Befehl geholt bzw. beendet.
 - ◆ Im eingeschwungenen Zustand der Pipeline: Durchsatz = 1

Befehlspipeline: Diskussion

□ Ausführungsphase

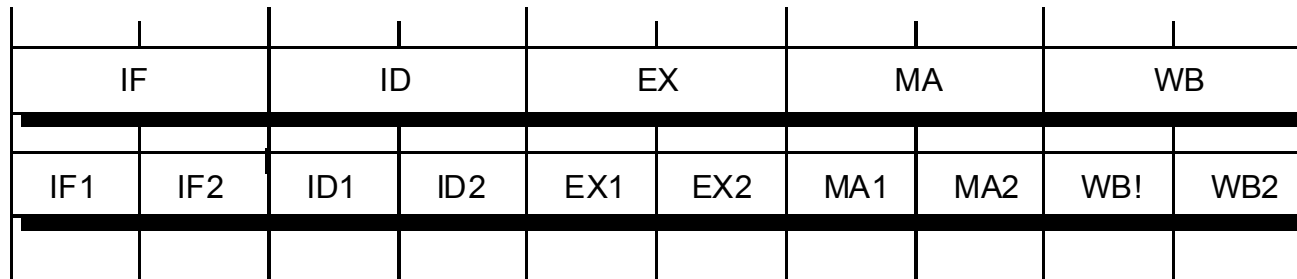
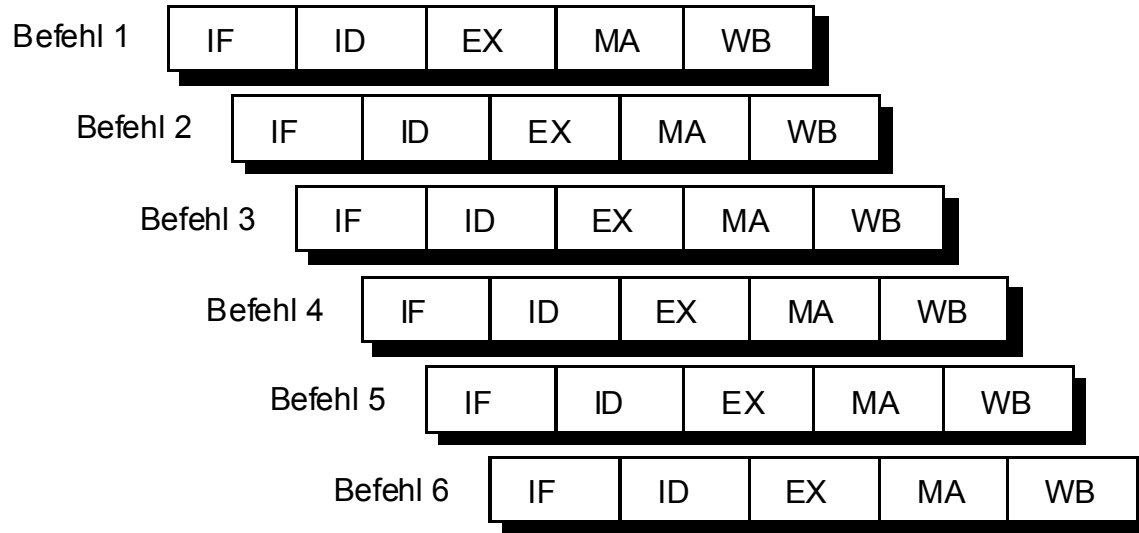
- ◆ Bei Integer-Operanden dauert die Ausführung bei arithmetischen und logischen Befehlen einen Taktzyklus (Ausnahme: Division)
- ◆ Gleitkomma-Verarbeitung:
 - ◆ Zerlegung in weitere Stufen
 - ◆ Eingliederung an der Stelle der Ausführungsstufe in der Befehlspipeline
 - ◆ Gleitkommaverarbeitungseinheiten (Floating-Point Units)

Befehlspipeline: Diskussion

- Verfeinerung der Pipelinestufen
 - ◆ Weitere Unterteilung der Pipelinestufen
 - ◆ Erhöhung der Taktrate
 - ◆ „Superpipelining“

Befehlspipeline: Diskussion

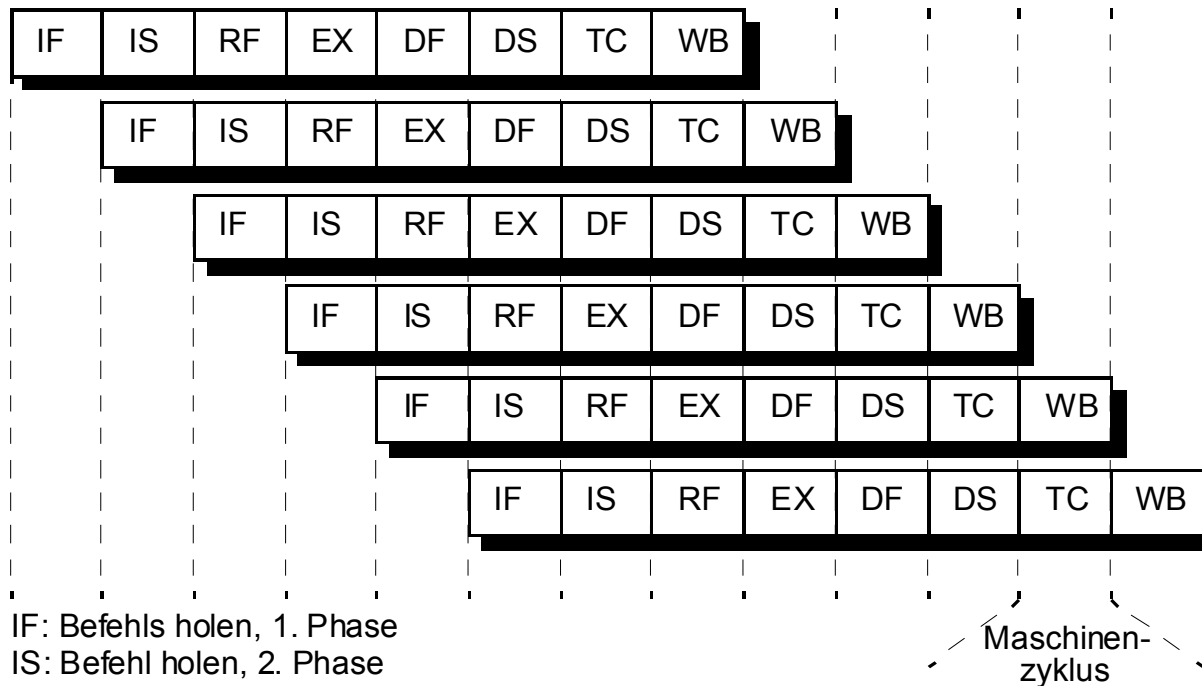
□ Verfeinerung der Befehlspipeline (k=10)



Maschinenzyklus
= 1/2 Grundtakt
der RISC-Maschine

Befehlspipeline: Diskussion

- Verfeinerung der Befehlspipeline: Beispiel MIPS R4000 (~1991)



IF: Befehls holen, 1. Phase
IS: Befehl holen, 2. Phase
RF: Holen der Daten aus der Registerdatei
EX: Befehl ausführen
DF: Holen der Daten, 1. Zyklus (für Load- und Store-Befehle,
DS: Holen der Daten, 2. Zyklus
TC Tag-Check
WB: Ergebnis zurückschreiben

Befehlspipeline: Beispiele

- ◆ RISC-Mikroprozessoren der 2. Generation: 4 – 6 Stufen der Befehlspipeline
 - ◆ AM29000, MIPS R2000, MIPS R3000, SPARC
- ◆ Moderne Mikroprozessoren: 7 – 24 Stufen
 - ◆ AMD Athlon: 9/11 Stufen
 - ◆ HP PA 8600: 7/9 Stufen
 - ◆ IBM Power3-II: 7/8 Stufen
 - ◆ Intel Pentium III: 12/14 Stufen
 - ◆ Intel Pentium 4: 22/24 Stufen
 - ◆ MIPS R12000: 6 Stufen
 - ◆ Sun Ultra-III: 14/15 Stufen

Befehlspipeline: Diskussion

□ Pipeline-Konflikte (Pipeline Hazards)

- ◆ Unterbrechung des taktsynchronen Durchlaufs eines Befehls durch die einzelnen Stufen der Pipeline
- ◆ Beispiel:
 - ◆ Unter der Annahme eines gemeinsamen Befehls- und Datenspeichers und nur einem Ausgang tritt ein Speicherlesekonflikt zwischen den Phasen IF und MA auf.
 - ◆ Auflösung des Konflikts durch Anhalten des Befehls

□ Pipeline-Leerzyklus, Pipelineblase (Pipeline Bubble)

- ◆ Entsteht bei Anhalten des Befehls

□ Pipeline-Hemmnis

- ◆ Situationen, die zu Pipeline-Konflikten führen.

Vorlesung Rechnerstrukturen

- **Kapitel 2: Architektur und Mikroarchitektur von Mikroprozessoren**
 - ◆ **2.7 Pipeline-Konflikte und Lösungen**

Pipeline-Konflikte

□ Datenkonflikte

- ♦ Treten auf, wenn ein Operand nicht verfügbar ist;
- ♦ Beispiel: Befehl benötigt das Ergebnis eines vorhergehenden und noch nicht abgeschlossenen Befehls;
- ♦ Datenabhängigkeiten zwischen Befehlen im Befehlsstrom

□ Strukturkonflikte

- ♦ Treten bei einigen Befehlskombinationen aufgrund von Ressourcenkonflikten auf;
- ♦ Beispiel: Gleichzeitiger Schreibzugriff zweier Befehle auf Registerdatei mit nur einem Schreibeingang.

Pipeline-Konflikte

□ Steuerkonflikte

- ◆ Bei Sprung- und Steuerflussbefehlen.
- ◆ Beispiel: Bei einem bedingten Sprung muss der Befehlsstrom in der Pipeline unterbrochen und das Sprungziel geholt werden.

Pipeline-Konflikte

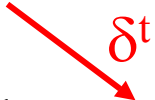
□ Lösung der Konflikte

- ◆ Einfache Lösung: Anhalten der Pipeline bis der Konflikt gelöst ist (Pipeline Stall):
- ◆ Einfügen von Leerzyklen (Pipeline Bubbles)
- ◆ Einfluss auf die Leistungsfähigkeit
- ◆ Verschiedene Maßnahmen in der Hardware und der Software, um Auswirkungen auf die Leistungsfähigkeit möglichst zu vermeiden.

Datenabhängigkeiten

- Datenabhängigkeiten zwischen zwei Befehlen *Inst1* und *Inst2*, wobei *Inst1* vor *Inst2* im Programm steht.
- Echte Datenabhängigkeit δ^t (true dependence, flow dependence)
 - ♦ Der Befehl *Inst1* schreibt sein Ergebnis in ein Register (Speicher), das von *Inst2* als Eingabe gelesen wird.

Inst1: $a := b + c$
Inst2: $d := a + e$



Datenabhängigkeiten

□ Gegenabhängigkeit δ^a (anti-dependence)

- ♦ Der Befehl *Inst1* liest einen Operanden aus einem Register, (Speicher), das von *Inst2* anschließend überschrieben wird.

Inst1: $b := a + c$

Inst2: $a := d + e$

□ Ausgabeabhängigkeit δ^o (output dependence)

- ♦ Beide Befehle beschreiben das gleiche Register.

Inst1: $a := b + c$

Inst2: $a := d + e$

Abhängigkeiten

- ❑ Steuerflussabhängigkeit (control dependence)
 - ♦ Der Befehl Inst1 muss abgeschlossen sein, bevor die Entscheidung getroffen werden kann, ob Inst2 ausgeführt werden soll oder nicht.

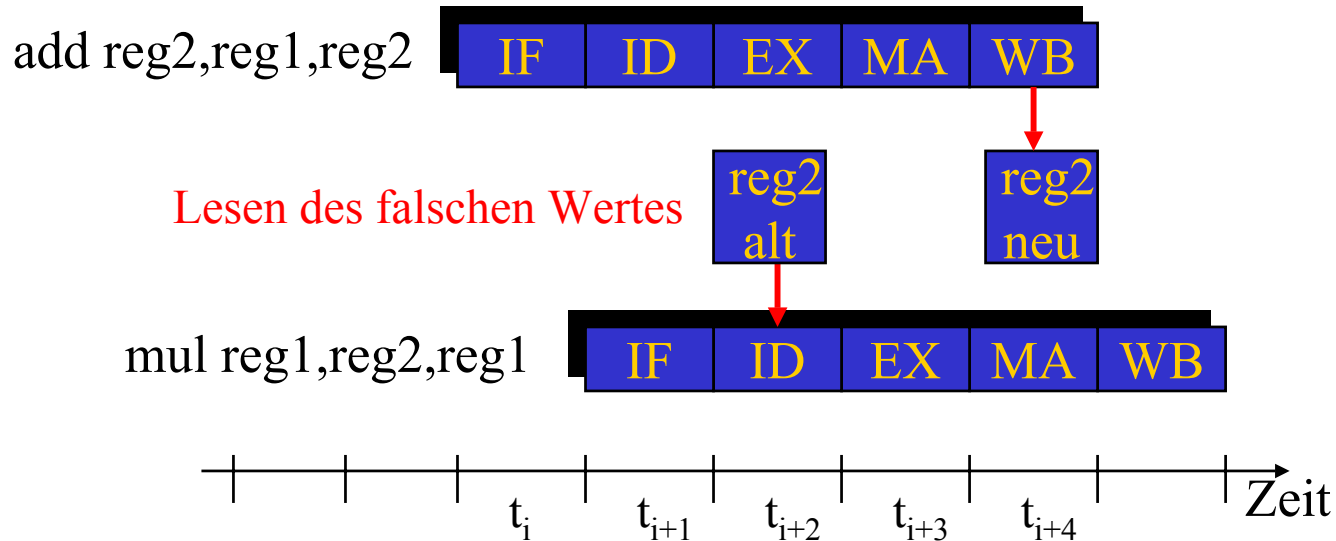
Datenkonflikte

- Zwischen datenabhängigen Befehlen können Datenkonflikte auftreten, wenn sie so nahe im Programm stehen, dass ihre Überlappung innerhalb der Pipeline die Zugriffsreihenfolge auf die Register verändern würde.
- Beispiel: Echte Datenabhängigkeit



Datenkonflikte

□ Beispiel: Echte Datenabhängigkeit



Datenkonflikte

- Datenabhängigkeiten können folgende Datenkonflikte verursachen:
 - ◆ **Lese-nach-Schreib-Konflikte (read-after-write, RAW)**
 - ◆ Verursacht durch echte Datenabhängigkeit
 - ◆ **Schreib-nach-Lese-Konflikt (write-after-read, WAR)**
 - ◆ Versursacht durch Gegenabhängigkeit
 - ◆ Tritt dann auf, wenn in einer Pipeline die Schreibstufe der Lesestufe vorangeht;
 - ◆ **Schreib-nach-Schreib-Konflikt (wrtite-after-write, WAW)**
 - ◆ Verursacht durch Ausgabeabhängigkeit
 - ◆ Tritt in Pipelines auf, die in mehr als einer Stufe schreiben, oder es erlauben, dass ein Befehl mit seiner Ausführung fortfahren darf, obwohl ein vorhergehender Befehl angehalten worden ist.

Auflösen von Konflikten

□ Software-Lösung

◆ Aufgabe des Compilers:

- ◆ Erkennen von Datenkonflikten
- ◆ Einfügen von Leeroperationen nach jedem Befehl, der einen Konflikt verursacht oder verursachen kann.
- ◆ Statische Verfahren:
 - ◆ Instruction Scheduling, Pipeline Scheduling
 - ◆ Eliminieren von Leeroperationen
 - ◆ Umordnen der Befehle des Programms (Code-Optimierung)

Dynamische Verfahren

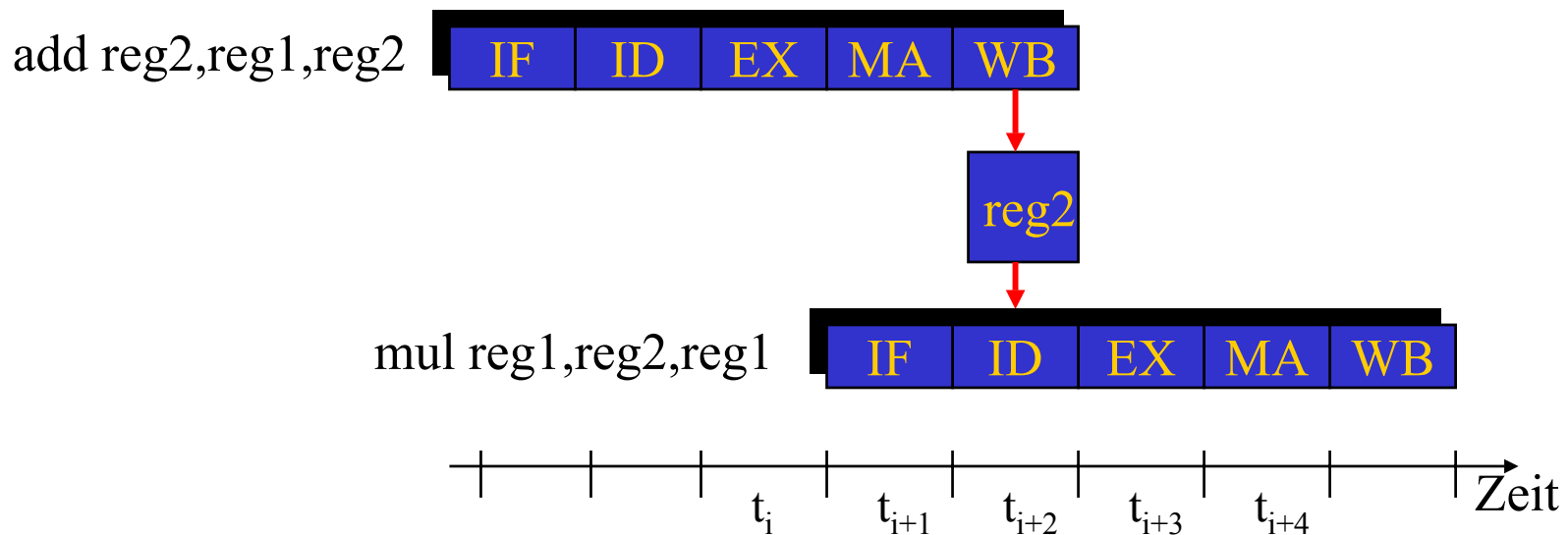
□ Hardware-Lösungen

- ◆ Erkennen von Konflikten
 - ◆ Entsprechende Konflikterkennungslogik notwendig!
- ◆ Techniken
 - ◆ Leerlauf der Pipeline (Interlocking, Stalling)
 - ◆ Forwarding
 - ◆ Forwarding mit Interlocking

Dynamische Verfahren

❑ Leerlauf der Pipeline: Interlocking

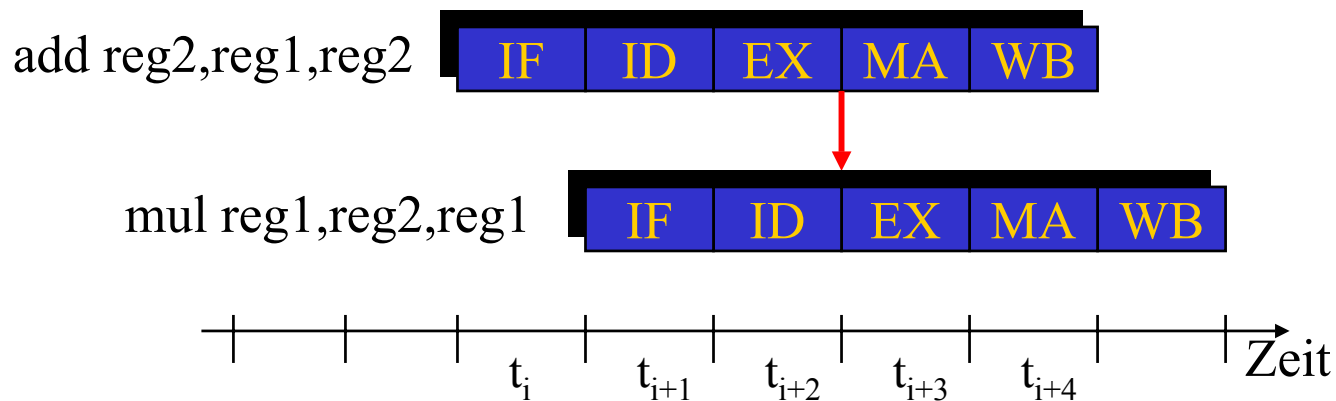
- ◆ Anhalten des Befehls Inst2 in der Pipeline für mehrere Takte
- ◆ Erkennen von Konflikten und Auflösen des Konflikts durch Anhalten der Pipeline



Dynamische Verfahren

□ Forwarding

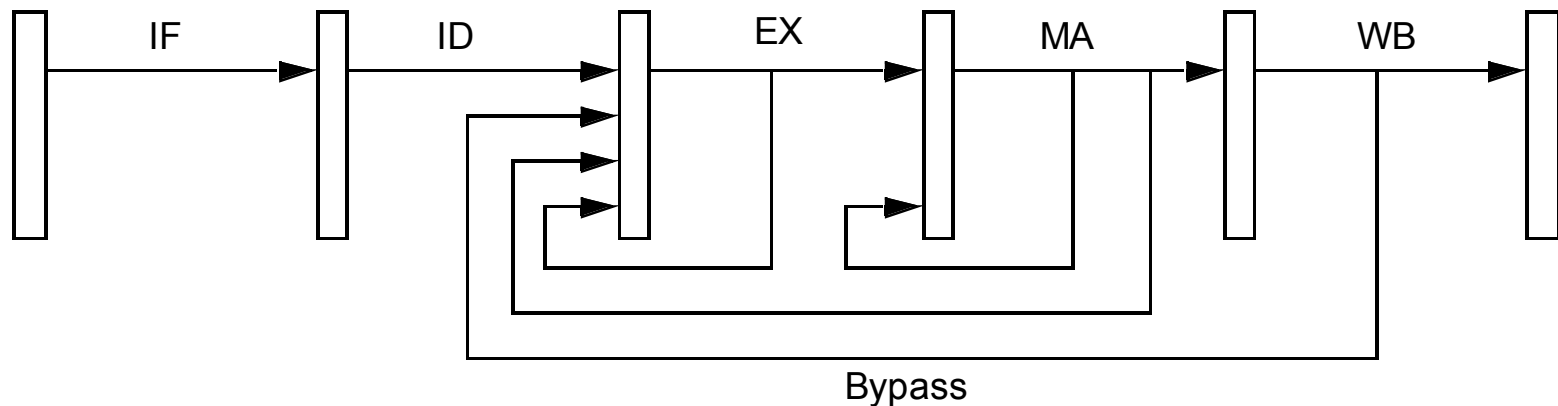
- ◆ Erhöhter Hardware-Aufwand
- ◆ Rückführung des ALU-Ergebnisses zur Eingabe
- ◆ Kein Warten notwendig!



Dynamische Verfahren

□ Forwarding

- ◆ Hardware-Aufwand
- ◆ Forwarding-Logik und zusätzliche Datenpfade



Dynamische Verfahren

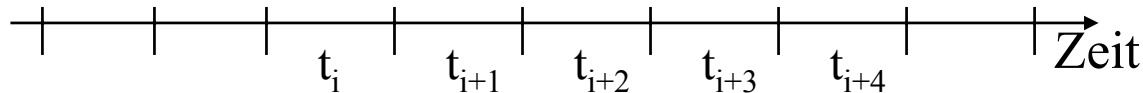
□ Forwarding mit Interlocking

- ◆ Nicht alle Konflikte lassen sich mit Forwarding auflösen
- ◆ Speicherzugriff: Ladeoperation
- ◆ Problem:

load reg2,B



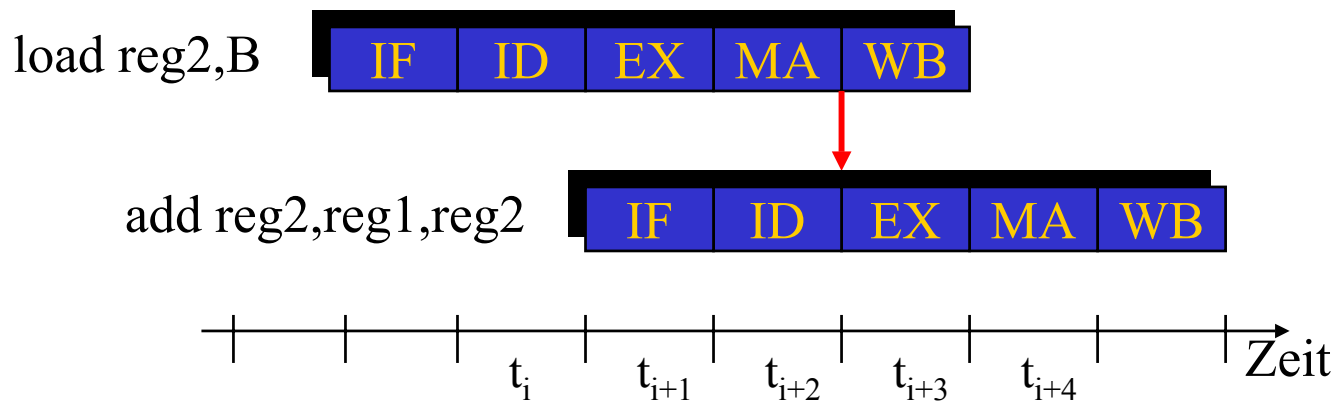
add reg2,reg1,reg2



Dynamische Verfahren

□ Forwarding mit Interlocking

- ◆ Nicht alle Konflikte lassen sich mit Forwarding auflösen
- ◆ Speicherzugriff: Ladeoperation
- ◆ Lösung: Leerbefehl (Interlocking)



Strukturkonflikte

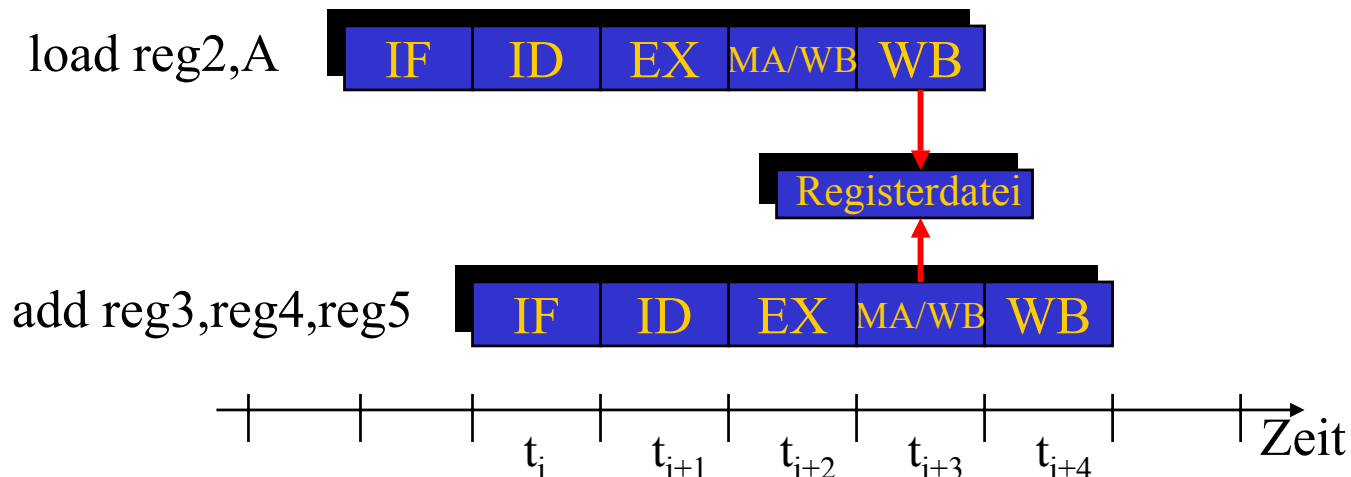
□ Ressourcenkonflikt:

- ♦ bei der Abarbeitung einer Folge von Maschinenbefehlen auf einer parallelen Maschine treten genau dann Ressourcenkonflikte auf, wenn zwei oder mehr Maschinenbefehle (Operationen) auf ein nur einfach vorhandenes Betriebsmittel (ohne Speicherelemente) zugreifen.
- ♦ Treten bei einer einfachen Pipeline, wie vorgestellt, nicht auf

Strukturkonflikte

□ Ressourcenkonflikt:

- ◆ Beispiel: Beispielmachine mit einfacher Pipeline
 - ◆ Annahme : Die MA-Stufe ist in der Lage, bei einem Register-Register-Befehl die Ausgabe der ALU durch einen Schreibkanal zurückzuschreiben.
 - ◆ Im Beispiel greifen zwei Befehle gleichzeitig auf einen nureinfach vorhandenen Schreibeingang zu.



Strukturkonflikte

□ Ressourcenkonflikt: Lösungen

◆ Arbitrierung mit Interlocking

- ◆ Auflösung durch Hardware
- ◆ Anhalten eines Befehls, der den Konflikt verursacht
- ◆ Einfügen von Leerzyklen

◆ Ressourcenreplizierung

- ◆ Vervielfachung der Ressourcen
 - ◆ Beispiel: mehrere Schreibeingänge für Registerdatei