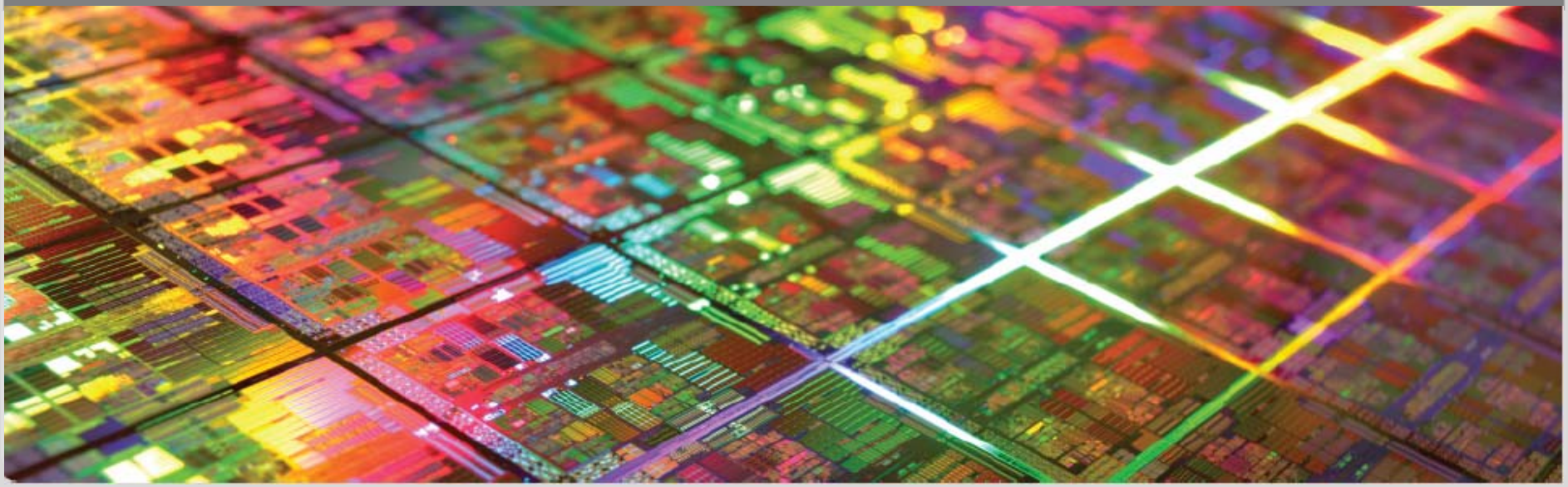
Karlsruher Institut für Technologie

# Rechnerstrukturen

## Vorlesung im Sommersemester 2010

**Prof. Dr. Wolfgang Karl**

Fakultät für Informatik – Lehrstuhl für Rechnerarchitektur und Parallelverarbeitung



# Vorlesung Rechnerstrukturen

- Kapitel 1: Grundlagen
  - 1.5 Parallelverarbeitung

# Parallelverarbeitung

- **Formen des Parallelismus**
- Nebenläufigkeit
  - Eine Maschine arbeitet nebenläufig, wenn die Objekte vollständig gleichzeitig abgearbeitet werden.
- Pipelining
  - 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)

# Parallelverarbeitung

- **Ebenen der Parallelität**
- Programmebene
  - Parallele Verarbeitung verschiedener Programme
  - Vollständig unabhängige Einheiten
    - ohne gemeinsame Daten
    - wenig oder keine Kommunikation und Synchronisation
  - Parallelverarbeitung wird vom Betriebssystem organisiert

# Parallelverarbeitung

- **Ebenen der Parallelität**
- Prozessebene (Task-Ebene)
  - Programm wird in Anzahl parallel ausführbarer Prozesse zerlegt
  - Prozess: schwergewichtiger Prozess (heavy-weighted process), Beispiel: UNIX-Prozesse
    - Besteht aus vielen sequentiell ausgeführten Befehlen und umfasst eigene Datenbereiche
  - Synchronisation und Kommunikationsaufwand
  - Betriebssysteme unterstützt Parallelverarbeitung durch Primitive zur Prozessverwaltung, Prozess-Synchronisation, Prozesskommunikation

# Parallelverarbeitung

- **Ebenen der Parallelität**
- Blockebene
  - leichtgewichtige Prozesse (Threads)
    - Bestehen jeweils aus sequentiell ausgeführten Befehlen teilen sich gemeinsamen Adressraum
    - Beispiel: Threads gemäß POSIX 1003.a Standard in mehrfädigen (multithreaded) Betriebssystemen
    - Synchronisation über Schlossvariablen (mutex), und Bedingungsvariablen (condition variables) oder darauf aufbauenden Synchronisationsmechanismen
    - Kommunikation über gemeinsame Daten
    - Aufwand für Thread-Erzeugung und-Beendigung, Thread-Wechsel geringer
  - Anweisungsblöcke
    - Innere und äußere parallele Schleifen in FORTRAN-Dialekten
    - Verwendung von Microtasking
    - Hohes Parallelitätspotential durch parallel ausführbare Schleifeninterationen

# Parallelverarbeitung

- **Ebenen der Parallelität**
- Anweisungs- oder Befehlsebene
  - Parallele Ausführung einzelner Maschinenbefehle oder elementarer Anweisungen
  - Optimierende (parallelisierende) Compiler für VLIW-Prozessoren oder Anwendung der Superskalartechnik in superskalaren Mikroprozessoren
  - Analyse der sequentiellen Befehlsfolge
  - Umordnen und Parallelisieren der Befehle
  - Datenflusssprachen und funktionale Programmiersprachen erlauben explizite Spezifikation der Parallelität

# Parallelverarbeitung

- **Ebenen der Parallelität**
- Suboperationsebene
  - Elementare Anweisung wird durch Compiler oder durch die Maschine in Suboperationen aufgebrochen, die parallel ausgeführt werden
  - Vektoroperationen
    - Überlappte Ausführung auf Vektorrechner in Vektorpipeline
    - Komplexe Datenstrukturen (Matrizen, Vektoren, Datenströme) sind in höherer Programmiersprache verfügbar oder werden von einem parallelisierenden, vektorisierenden Compiler aus einer sequentiellen Programmiersprache generiert
    - Beispiel: Vektoraddition C = A + B statt Abarbeitung einer Schleife

# Parallelverarbeitung

- **Ebenen der Parallelität**
- Körnigkeit der Parallelität
  - Die Körnigkeit oder Granularität (grain size) ergibt sich aus dem Verhältnis von Rechenaufwand zu Kommunikations- oder Synchronisationsaufwand. Sie bemisst sich nach der Anzahl der Befehle in einer sequentiellen Befehlsfolge.
  - Programm-, Prozess- und Blockebene werden häufig auch als grobkörnige (large grained) Parallelität,
  - die Anweisungsebene als feinkörnige (finely grained) Parallelität bezeichnet.
  - Seltener wird auch von mittelkörniger (medium grained) Parallelität gesprochen, dann ist meist die Blockebene gemeint.

# Parallelverarbeitung

## Techniken der Parallelarbeit vs. Parallelitätsebenen

| Parallelarbeitstechniken | Programmebene | Prozessebene | Blockebene | Anweisungsebene | Suboperationsebene |
|---|---|---|---|---|---|
| **Techniken der Parallelarbeit durch Rechnerkopplung** | | | | | |
| Grid-Computing | X | X | | | |
| Cluster | X | X | | | |
| **Techniken der Parallelarbeit durch Prozessorkopplung** | | | | | |
| Nachrichtenkopplung | X | X | | | |
| Speicherkopplung (SMP) | X | X | X | | |
| Speicherkopplung (DSM) | X | X | X | | |
| Grobkörniges Datenflußprinzip | | X | X | | |
| **Techniken der Parallelarbeit in der Prozessorarchitektur** | | | | | |
| Befehlspipelining | | | | X | |
| Superskalar | | | | X | |
| VLIW | | | | X | |
| Überlappung von E/A- mit CPU-Operationen | | | | X | |
| Feinkörniges Datenflußprinzip | | | | X | |
| **SIMD-Techniken** | | | | | |
| Vektorrechnerprinzip | | | | | X |
| Feldrechnerprinzip | | | | | X |
| SIMD-Operationen | | | | | |

Quelle: Ungerer, T. : Skript Rechnerstrukturen, SS 2000

# Einführung: Literatur

- Theo Ungerer: Parallelrechner und parallele Programmierung, Kap.1.1 – 1.3

# Vorlesung Rechnerstrukturen

- Kapitel 1: Grundlagen
  - 1.6 Klassifikation von Rechnerarchitekturen

# Grundlagen

- **Klassifikationen**
  - Aufspannen von Entwurfsräumen
  - Aufzeigen von Entwurfsalternativen
  - Klassifikationsschemata versuchen, der Vielfalt von Rechnerarchitekturen eine Ordnungsstruktur zu geben
  - Frühe Klassifikationen konzentrieren sich auf die Hardware-Struktur
    - Anordnung und Organisation der Verarbeitungselemente
    - Operationsprinzip

# Klassifikation von Rechnerarchitekturen

- **Klassifizierung nach M. Flynn**
  - Zweidimensionale Klassifizierung
  - Hauptkriterien:
    - Zahl der Befehlsströme und
    - Zahl der Datenströme sind.
  - Merkmale:
    - Ein Rechner bearbeitet zu einem gegebenen Zeitpunkt einen oder mehr als einen Befehl.
    - Ein Rechner bearbeitet zu einem gegebenen Zeitpunkt einen oder mehr als einen Datenwert.

# Klassifikation von Rechnerarchitekturen

- **Klassifizierung nach M. Flynn**
- Vier Klassen von Rechnerarchitekturen
  - SISD Single Instruction – Single Data
    - Uniprozessor
  - SIMD Single Instruction – Multiple Data
    - Vektorrechner, Feldrechner
  - MISD Multiple Instructions – Single Data
    - ?
  - MIMD Multiple Instructions – Multiple Data
    - Multiprozessor

# Klassifikation von Rechnerarchitekturen

- **Kennzeichnend für moderne Rechnerstrukturen:**
- Prinzip der Virtualität:
  - Mit verschiedenen Techniken und Mechanismen in der Hardware können auf einer Maschine verschiedene parallele Programmiermodelle unterstützt werden
  - Die zugrunde liegende Organisation und Architektur ist weitgehend transparent
- Allgemeiner Trend in der Rechnerarchitektur
  - Verwendung von Standardkomponenten
  - Verständnis über die Implementierungstechniken zur Virtualität
  - Keine allgemeingültige Klassifikation!

# Vorlesung Rechnerstrukturen

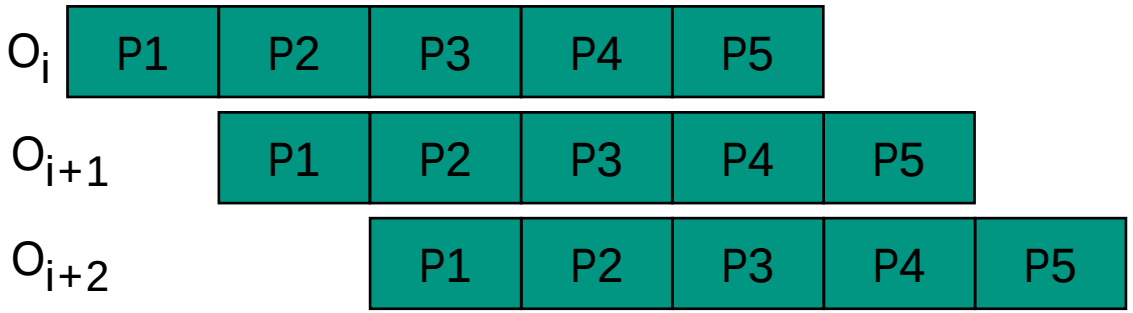
- **Kapitel 2: Parallelismus auf Maschinenbefehlsebene**
- Überblick
  - Pipelining
    - Überlappte Ausführung der Phasen des Maschinenbefehlszyklus
    - Nützen alle Prozessoren seit 1985 aus
  - Nebenläufigkeit
    - Zu einem Zeitpunkt gleichzeitige Ausführung mehrerer Maschinenbefehle zu
    - Dynamische Ansätze
      - Superskalare Mikroprozessoren
    - Statische Ansätze
      - VLIW, EPIC

# Vorlesung Rechnerstrukturen

- **Kapitel 2: Parallelismus auf Maschinenbefehlsebene**
- 2.1 Pipelining

# Parallelismus auf Befehlsebene

- **Pipelining**
  - *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)*



# Parallelismus auf Befehlsebene

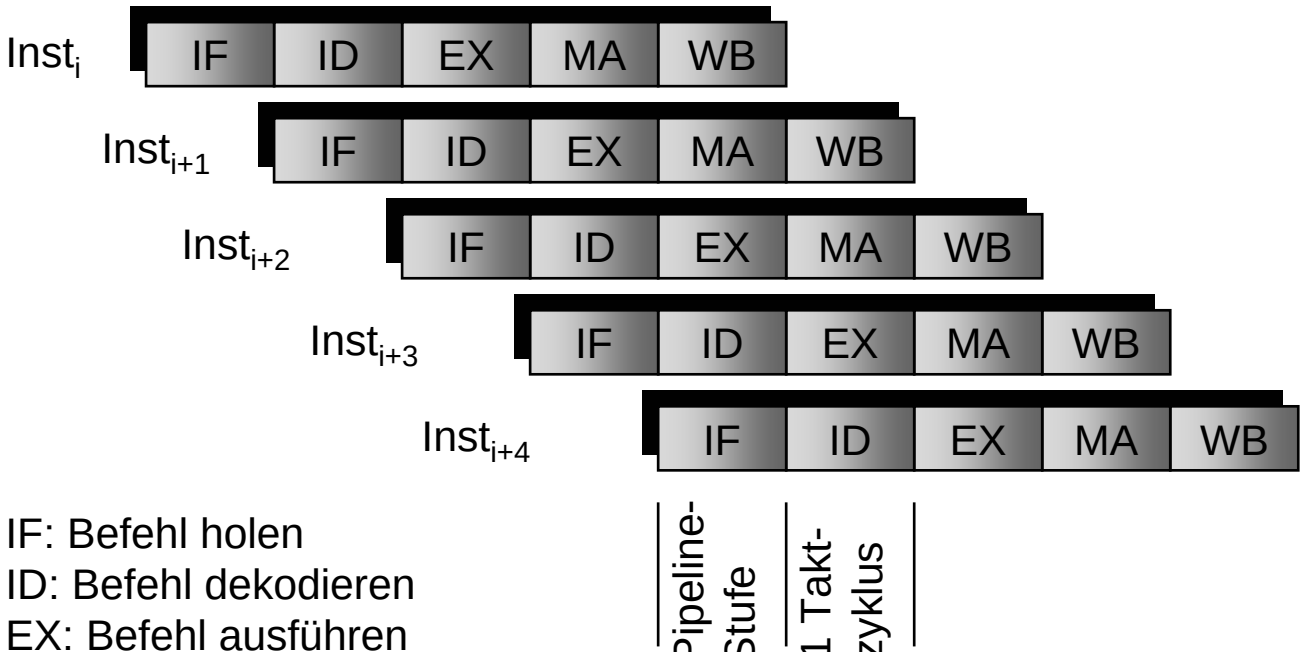
- **Pipelining**
- Befehlspipelining (Instruction Pipelining):
  - Zerlegung der Ausführung einer Maschinenoperation in Teilphasen, die dann von hintereinander geschalteten Verarbeitungseinheiten taktsynchron bearbeitet werden, wobei jede Einheit genau eine spezielle Teiloperation ausführt.
- Pipeline:
  - Gesamtheit der Verarbeitungseinheiten
- Pipeline-Stufe:
  - Stufen der Pipeline, die jeweils durch Pipeline-Register getrennt sind

# Pipelining

- **RISC (Reduced Instruction Set Computers)**
- Einfache Maschinenbefehle
  - Einheitliches und festes Befehlsformat
- Load/Store Architektur
  - Befehle arbeiten auf Registeroperanden
  - Lade- und Speicherbefehle greifen auf Speicher zu
- Einzyklus-Maschinenbefehle
  - Effizientes Pipelining des Maschinenbefehlszyklus
  - Einheitliches Zeitverhalten der Maschinenbefehle, wovon nur Lade- und Speicherbefehle sowie die Verzweigungsbefehle abweichen
- Optimierende Compiler
  - Reduzierung der Befehle im Programm

# Pipelining

- **RISC (Reduced Instruction Set Computers)**
- Implementierung eines RISC- Befehlssatzes
  - k-stufige Befehlspipeline (k=5)




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

# Pipelining

- **RISC (Reduced Instruction Set Computers)**
- Leistungsaspekte
  - Ausführungszeit eines Befehls:
    - Zeit, die zum Durchlaufen der Pipeline benötigt wird
    - Ausführung eines Befehls in *k* Taktzyklen (ideale Verhältnisse)
    - Gleichzeitige Behandlung von *k* Befehlen (ideale Verhältnisse)
- Latenz:
  - Anzahl der Zyklen zwischen einer Operation, die ein Ergebnis produziert, und einer Operation, die das Ergebnis verwendet

# Pipelining

- **RISC (Reduced Instruction Set Computers)**
- Leistungsaspekte
  - Laufzeit T:
    - *T = n+k-1,* mit *n*: Anzahl der Befehle in einem Programm (Annahme: ideale Verhältnisse!)
  - Beschleunigung S
    - S = n * k / (k + n - 1)

# Pipelining

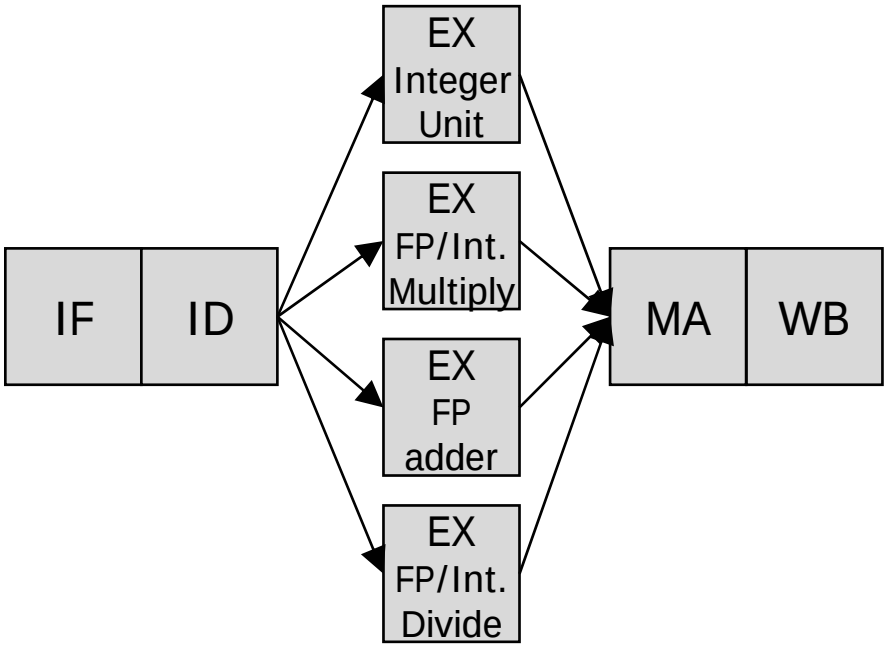
- **RISC (Reduced Instruction Set Computers)**
- Diskussion
  - Alle Pipelinestufen benützen unterschiedliche Ressourcen
  - Pipelining erhöht den Durchsatz
    - Mit jedem Takt wird unter Annahme idealer Verhältnisse ein Befehl geholt bzw. beendet.
    - Im eingeschwungenen Zustand der Pipeline: Durchsatz = 1 Befehl / Taktzyklus
    - Aber, reduziert nicht die Ausführungszeit einer individuellen Instruktion
  - Zykluszeit, Taktzyklus:
    - Abhängig vom kritischen Pfad, der langsamsten Pipelinestufe

# Pipelining

- **RISC (Reduced Instruction Set Computers)**
- Diskussion
  - Ausführungsphase
  - Integer-Verarbeitung
    - Ausführung von arithmetischen und logischen Befehlen dauert einen Taktzyklus (Ausnahme: Division)
  - Gleitkomma-Verarbeitung:
    - Zerlegung in weitere Stufen
    - Eingliederung an der Stelle der Ausführungsstufe in der Befehlspipeline
    - Mehrere Gleitkommarechenwerke (Floating-Point Units)

# Pipelining

- **Diskussion**
- Ausführungsphase
  - Gleitkomma-Verarbeitung: weitere Rechenwerke



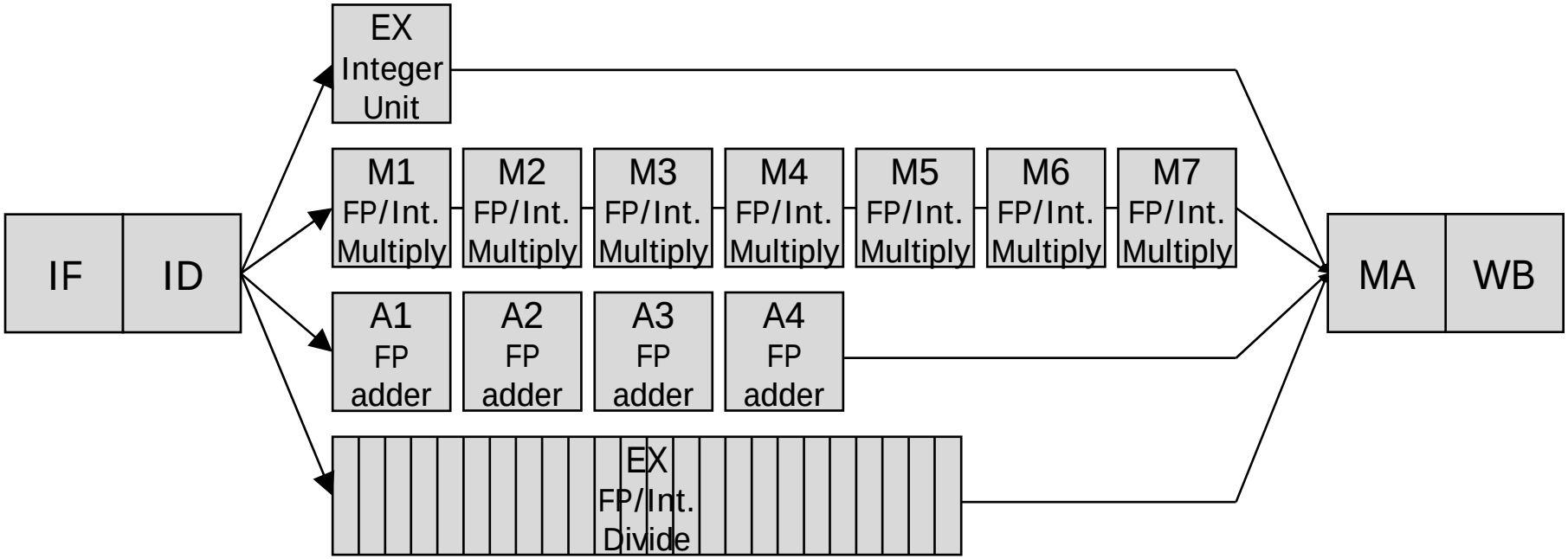

| Rechenwerk | Latenz | Initiierungs-intervall |
|---|---|---|
| Integer ALU | 0 | 1 |
| FP Add | 3 | 1 |
| FP Multiply | 6 | 1 |
| FP Divide | 24 | 25 |

**Latenz**: Anzahl der Zyklen zwischen einer Operation, die ein Ergebnis produziert und einer Operation, die das Ergebnis verwendet

**Initiierungsintervall**: Anzahl der Zyklen zwischen zwei Operationen

# Pipelining

- **Diskussion**
- Ausführungsphase
  - Gleitkomma-Verarbeitung: weitere Rechenwerke



# Pipelining

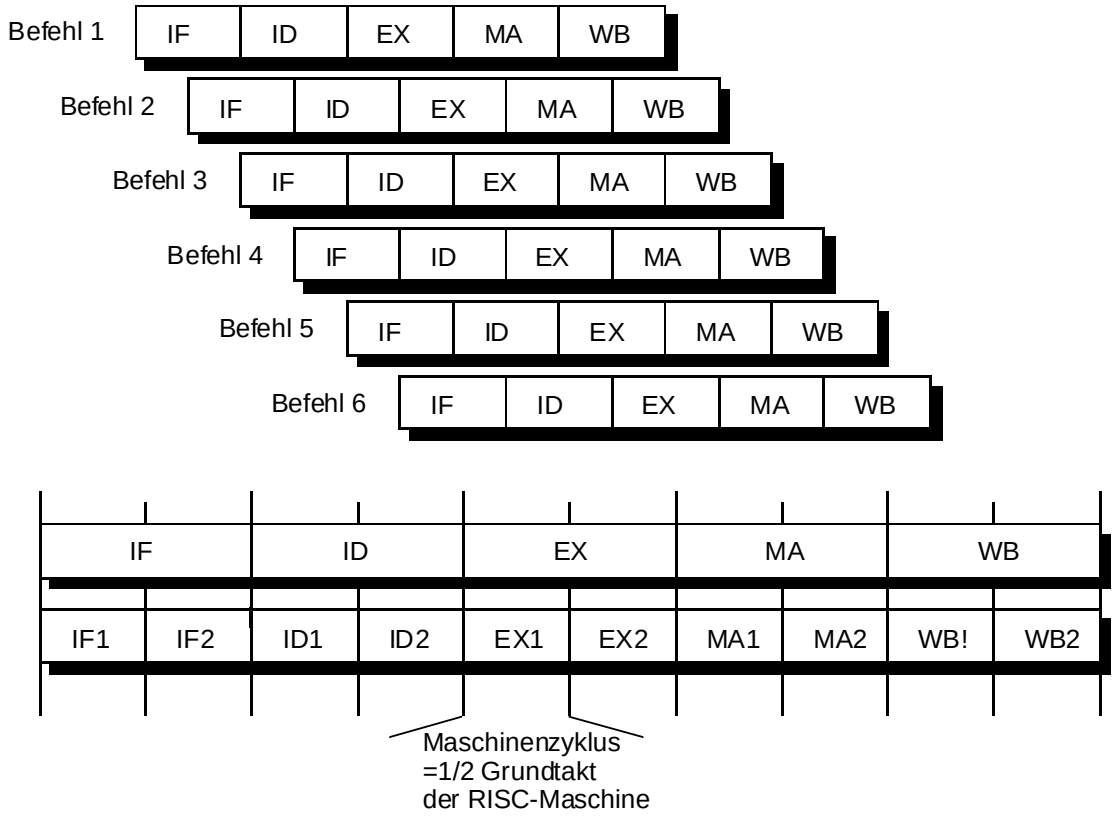
- **Diskussion**
- Ausführungsphase
  - Gleitkomma-Verarbeitung: Pipelining der Rechenwerke
    - Latenz: 1 Zyklus weniger als die Anzahl der Pipelinestufen
    - Beispiel:
      - 4 ausstehende FP add Operationen
      - 7 ausstehende FP multiply Operationen
      - 1 FP Divide Operation, da kein Pipelining

# Pipelining

- **Diskussion**
- Verfeinerung der Pipeline-Stufen
  - Weitere Unterteilung der Pipeline-Stufen
  - Weniger Logik-Ebenen pro Pipeline-Stufe
  - Erhöhung der Taktrate
  - Führt aber auch zu einer Erhöhung der Ausführungszeit pro Instruktion
- „Superpipelining“

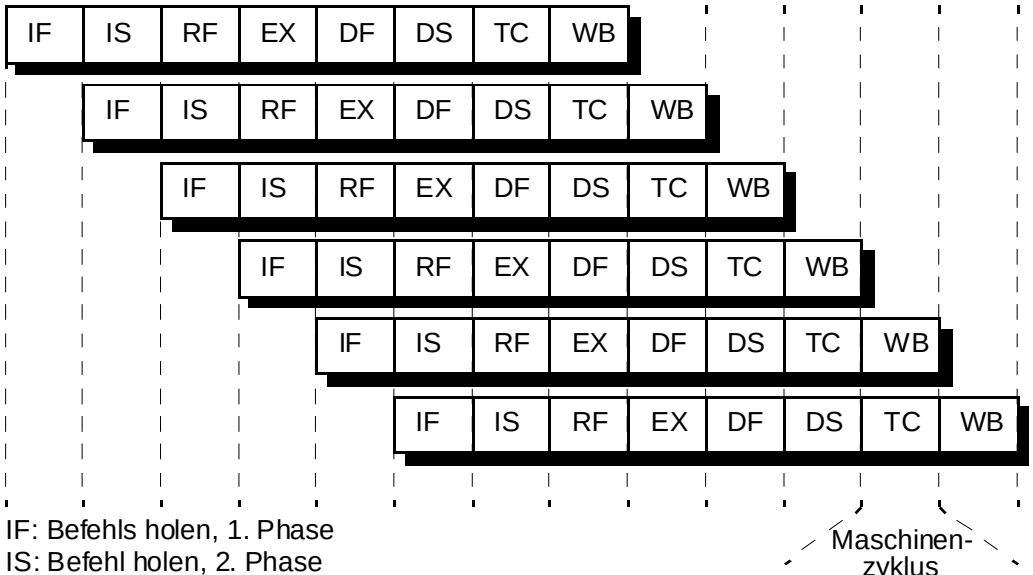
# Pipelining

- **Diskussion**
- Verfeinerung der Pipeline-Stufen



# Pipelining

- **Diskussion**
- Verfeinerung der Pipeline-Stufen: 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

# Pipelining

- **RISC (Reduced Instruction Set Computers)**
- Pipeline-Konflikte (Pipeline Hazards, Pipeline-Hemmnisse)
  - Situationen, die verhindern, dass die nächste Instruktion im Befehlsstrom im zugewiesenen Taktzyklus ausgeführt wird
    - Unterbrechung des taktsynchronen Durchlaufs durch die einzelnen Stufen der Pipeline
  - Verursachen Leistungseinbußen im Vergleich zum idealen Speedup
  - Erfordern ein Anhalten der Pipeline (Pipeline stall)
  - Bei einfacher Pipeline:
    - Wenn eine Instruktion angehalten wird, werden auch alle Befehle, die nach dieser Instruktion zur Ausführung angestoßen wurden, angehalten
    - Alle Befehle, die vor dieser Instruktion zur Ausführung angestoßen wurden, durchlaufen weiter die Pipeline

# Pipelining

- **RISC (Reduced Instruction Set Computers)**
- Pipeline-Konflikte (Pipeline Hazards, Pipeline-Hemmnisse)
  - Strukturkonflikte
    - Ergeben sich aus Ressourcenkonflikten
    - Die Hardware kann nicht alle möglichen Kombinationen von Befehlen unterstützen, die sich in der Pipeline befinden können
    - Beispiel:
      - Gleichzeitiger Schreibzugriff zweier Befehle auf eine Registerdatei mit nur einem Schreibeingang
  - Datenkonflikte
    - Ergeben sich aus Datenabhängigkeiten zwischen Befehlen im Programm
    - Instruktion benötigt das Ergebnis einer vorangehenden und noch nicht abgeschlossenen Instruktion in der Pipeline
      - D.h. ein Operand ist noch nicht verfügbar
  - Steuerkonflikte
    - Treten bei Verzweigungsbefehlen und anderen Instruktionen auf, die den Befehlszähler verändern

# Pipelining

- **RISC (Reduced Instruction Set Computers)**
- Pipeline-Konflikte (Pipeline Hazards, Pipeline-Hemmnisse)
  - Auflösung der Pipeline-Konflikte
    - Einfache Lösung: Anhalten der Pipeline
    - Einfügen eines Leerzyklus (Pipeline Bubble)
    - Führt zu Leistungseinbußen
    - Verschiedene Maßnahmen in der Hardware und in der Software, um Auswirkungen auf die Leistungsfähigkeit möglichst zu vermeiden

# Pipelining

- **RISC (Reduced Instruction Set Computers)**
- Pipeline-Konflikte (Pipeline Hazards, Pipeline-Hemmnisse)
  - Ursachen für Datenkonflikte: Datenabhängigkeiten zwischen Befehlen im Programm
  - Beispiel:

```
add R1,R2,R3
sub R4,R5,R6
and R6,R1,R8
xor R9,R1,R11
```

# Pipelining

- **RISC (Reduced Instruction Set Computers)**
- Pipeline-Konflikte (Pipeline Hazards, Pipeline-Hemmnisse)
  - Ursachen für Datenkonflikte:
  - Datenabhängigkeiten zwischen Befehlen im Programm
    - sind Eigenschaften des Programms!
      - Es hängt von der Pipeline-Organisation ab, ob eine gegebene Anhängigkeit zu einem Konflikt führt und ob Konflikte zu einem Anhalten der Pipeline führen!
    - in einem Programm zeigen die Möglichkeit eines Konflikts an!
    - legen die Programmordnung fest, d.h. die Reihenfolge in der die Ergebnisse berechnet werden müssen.
    - legen eine obere Grenze für den Grad des Parallelismus fest, der ausgenützt werden kann

# Pipelining

- **RISC (Reduced Instruction Set Computers)**
- Pipeline-Konflikte (Pipeline Hazards, Pipeline-Hemmnisse)
  - Ursachen für Datenkonflikte:
  - Echte Datenabhängigkeit (true dependence, flow dependence)
    - Ein Befehl *j* ist datenabhängig von einem Befehl *i*, wenn eine der folgenden Bedingungen gilt:
      - Befehl *i* produziert ein Ergebnis, das von Befehl *j* verwendet wird, oder
      - Befehl *j* ist datenabhängig von Befehl *k* und Befehl *k* ist datenabhängig von Befehl i (Abhängigkeitskette)
    - Beispiel:

```
LOOP:   L.D      F0,0(R1)
        ADD.D    F4,F0,F2
        S.D      F4,0(R1)
        D.ADDUI  R1,R1,#-8
        BNE      R1,R2,LOOP
```

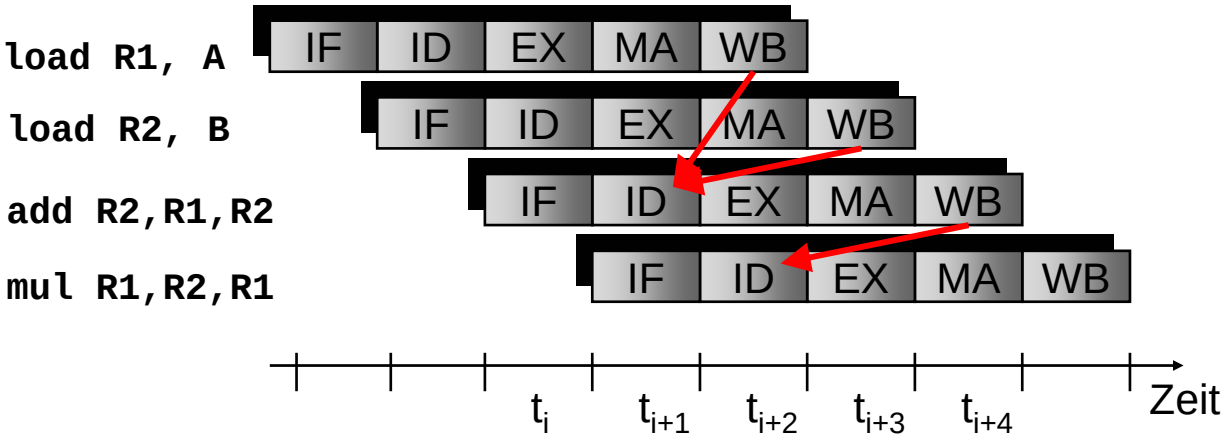
Abhängigkeit o tritt in Pipeline auf, in der der Vergleich in der ID Phase stattfindet

# Pipelining

- **RISC (Reduced Instruction Set Computers)**
- Pipeline-Konflikte (Pipeline Hazards, Pipeline-Hemmnisse)
  - Ursachen für Datenkonflikte:
  - Namensabhängigkeiten
    - Treten auf, wenn zwei Instruktionen dasselbe Register dieselbe Speicherzelle (den Namen) verwenden, aber kein Datenfluss zwischen den Befehlen mit dem Namen verbunden ist.
    - Es gibt zwei Arten von Namensabhängigkeiten zwischen zwei Befehlen *i* und *j:*
      - Gegenabhängigkeit (Anti dependence)
        - `ADD R2,R3,R4`
        - `XOR R3,R5,R6`
      - Ausgabeabhängigkeit (Output dependence)
        - `ADD R2,R3,R4`
        - `XOR R2,R5,R6`

# Pipelining

- **RISC (Reduced Instruction Set Computers)**
- Pipeline-Konflikte (Pipeline Hazards, Pipeline-Hemmnisse)
  - 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



# Pipelining

- **RISC (Reduced Instruction Set Computers)**
- Pipeline-Konflikte (Pipeline Hazards, Pipeline-Hemmnisse)
  - Datenabhängigkeiten können folgende Konflikte verursachen:
    - Lese-nach-Schreib-Konflikt (Read-After-Write, RAW)
      - Tritt auf, wenn Befehl j sein Quellregister liest, bevor Befehl i das Ergebnis geschrieben hat.
    - Lese-nach-Schreib-Konflikt (Write-After-Read, WAR)
      - Tritt auf, wenn Befehl j sein Zielregister beschreibt, bevor Befehl i den Operanden gelesen hat.
      - D.h. der Befehl i liest einen falschen Wert
    - Schreib-nach-Schreib-Konflikt (Write-After-Write, WAW)
      - Tritt auf, wenn Befehl j sein Zielregister beschreibt, bevor Befehl i das Ergebnis geschrieben hat.
      - D.h. Der Befehl i liefert den Wert für das Zielregister, anstelle von j

# Pipelining

- **RISC (Reduced Instruction Set Computers)**
- Pipeline-Konflikte (Pipeline Hazards, Pipeline-Hemmnisse)
  - Auflösen von Konflikten
    - Hardware-Lösungen (Dynamische Verfahren)
      - Erkennen von Konflikten
        - Entsprechende Konflikterkennungslogik notwendig!
      - Techniken:
        - Leerlauf der Pipeline (Interlocking, Stalling)
        - Forwarding
        - Forwarding mit Interlocking