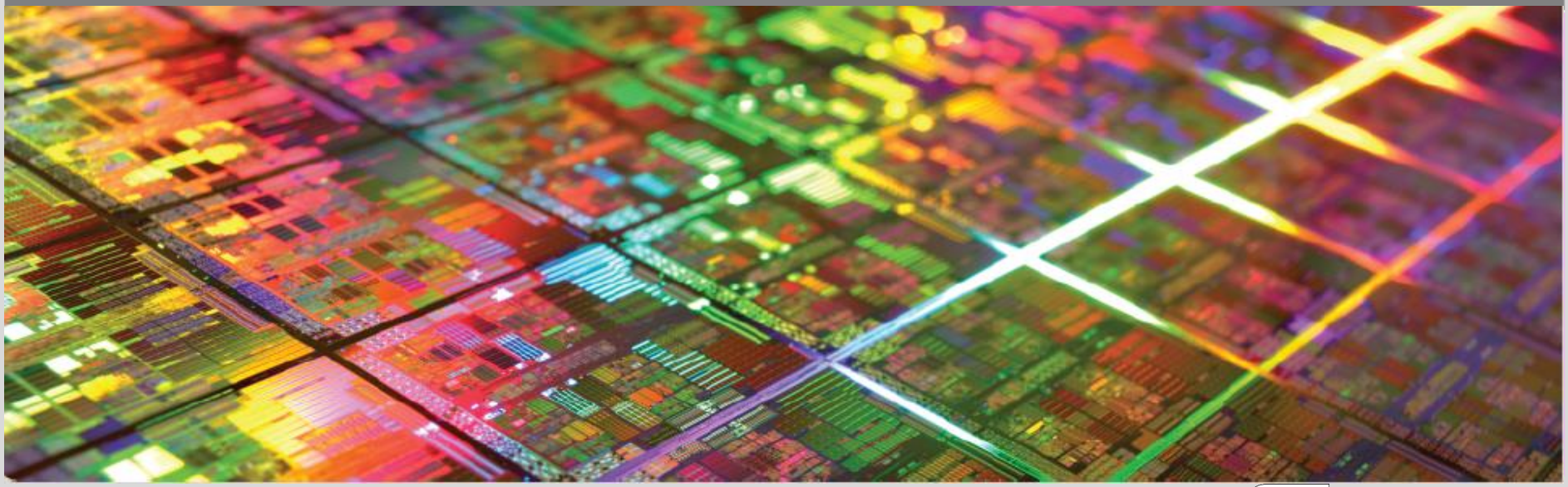
Karlsruher Institut für Technologie

Rechnerstrukturen

Vorlesung im Sommersemester 2016

Prof. Dr. Wolfgang Karl

Institut für Technische Informatik (ITEC), Lehrstuhl für Rechnerarchitektur und Parallelverarbeitung



Vorlesung Rechnerstrukturen

Kapitel 1: Grundlagen

- 1.1 Einführung, Begriffsklärung
- 1.2 Entwurf von Rechenanlagen – Entwurfsfragen
- 1.3 Einführung in den Entwurf eingebetteter Systeme
- 1.4 Energieeffizienter Entwurf – Grundlagen
- 1.5 Bewertung der Leistungsfähigkeit eines Rechners
- 1.6 Zuverlässigkeit und Fehlertoleranz
- 1.7 Grundlagen der 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 **Kommunikation**
  - Betriebssystem 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

Parallelverarbeitung

Literatur

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

Vorlesung Rechnerstrukturen

Kapitel 1: Grundlagen

- 1.1 Einführung, Begriffsklärung
- 1.2 Entwurf von Rechenanlagen – Entwurfsfragen
- 1.3 Einführung in den Entwurf eingebetteter Systeme
- 1.4 Energieeffizienter Entwurf – Grundlagen
- 1.5 Bewertung der Leistungsfähigkeit eines Rechners
- 1.6 Grundlagen der Parallelverarbeitung
- 1.7 Klassifikation von Rechnerarchitekturen

Klassifikation von Rechnerarchitekturen

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

Diskussion

- **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

- Keine allgemeingültige Klassifikation!

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

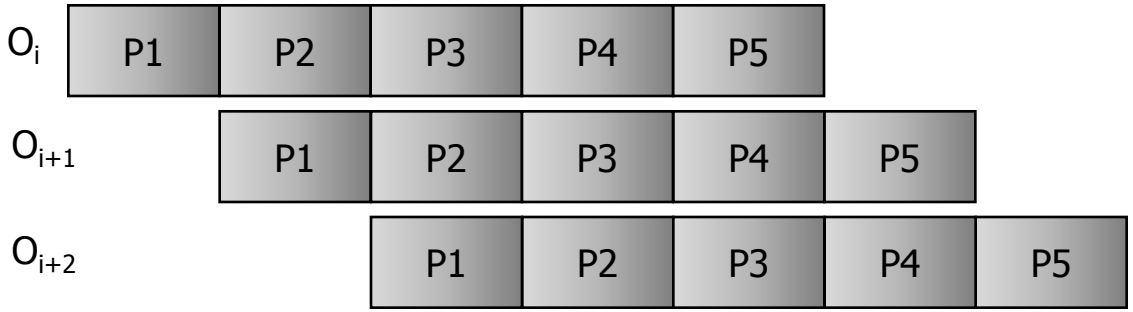
Vorlesung Rechnerstrukturen

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

Parallelismus auf Befehlsebene

Pipelining

- **Definition**
  - *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

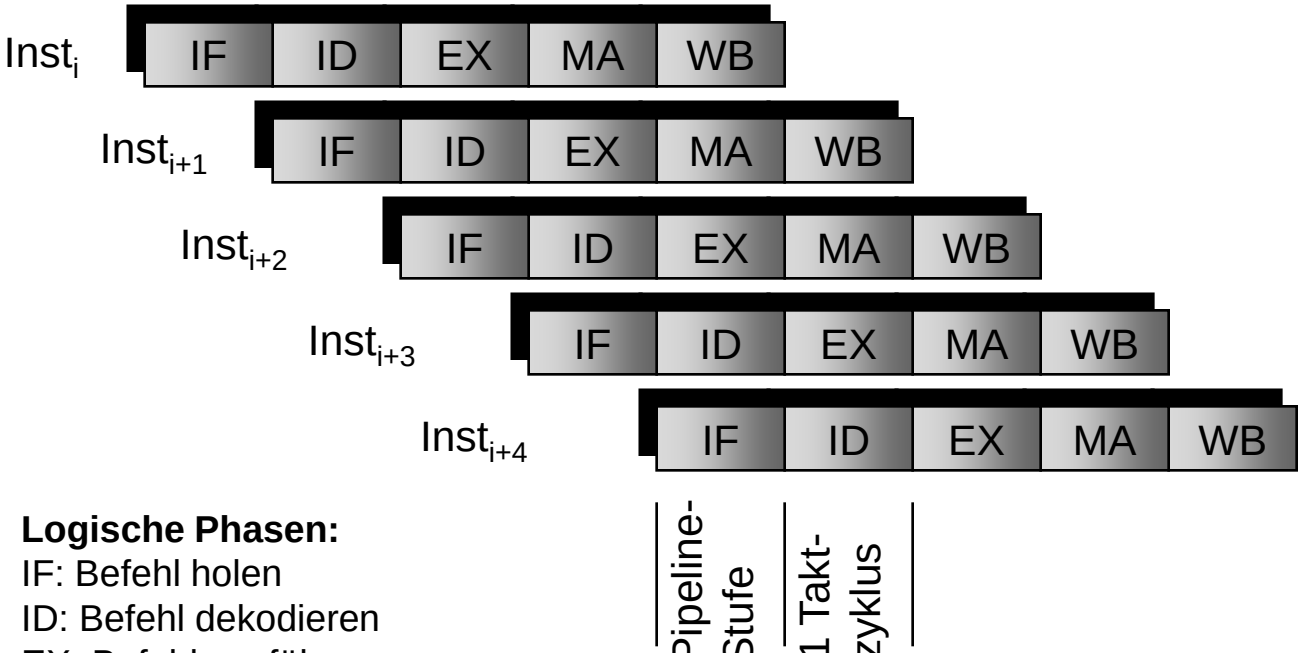
RISC (Reduced Instruction Set Computers)

- **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)

- **Pipelining des Maschinenbefehlszyklus**
  - k-stufige Befehlspipeline (k=5)



Logische Phasen:
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

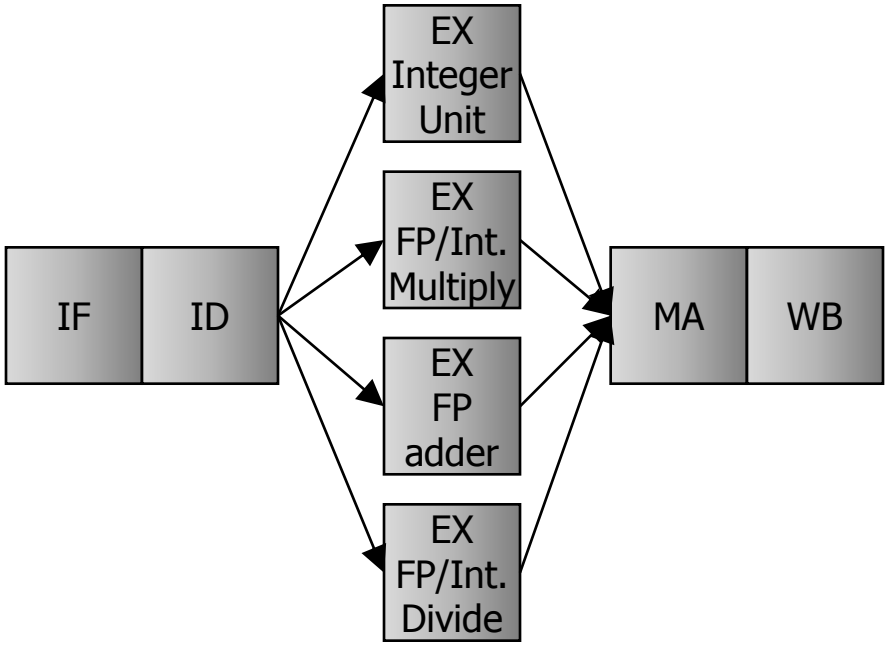
RISC (Reduced Instruction Set Computers)

- **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

RISC (Reduced Instruction Set Computers)

- **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

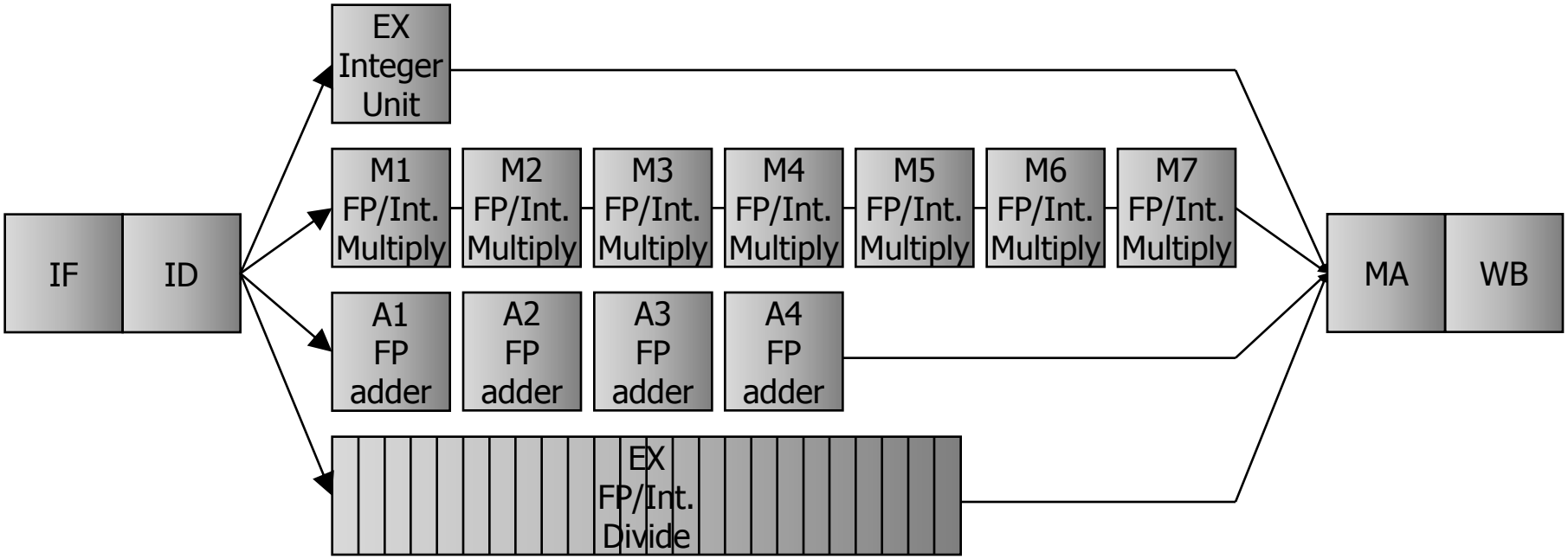
Latenz: Anzahl der Zyklen zwischen einer Operation, die ein Ergebnis produziert und einer Operation, die das Ergebnis verwendet

Initiierungsintervall: Anzahl der Zyklen zwischen zwei Operationen

Pipelining

RISC (Reduced Instruction Set Computers)

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



Pipelining

RISC (Reduced Instruction Set Computers)

- **Diskussion**
  - **Ausführungsphase**
    - **Gleitkomma-Verarbeitung**: Pipelining in den Rechenwerken – **arithmetisches Pipelining**
      - 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

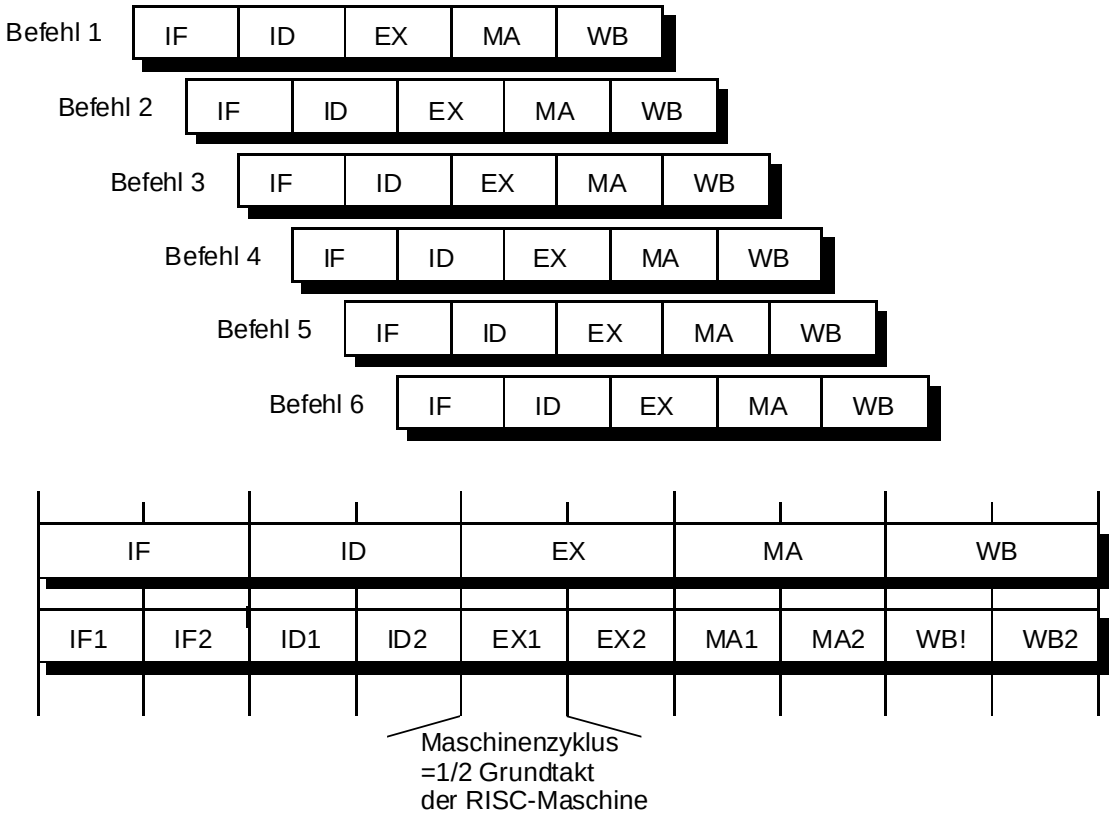
RISC (Reduced Instruction Set Computers)

- **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

RISC (Reduced Instruction Set Computers)

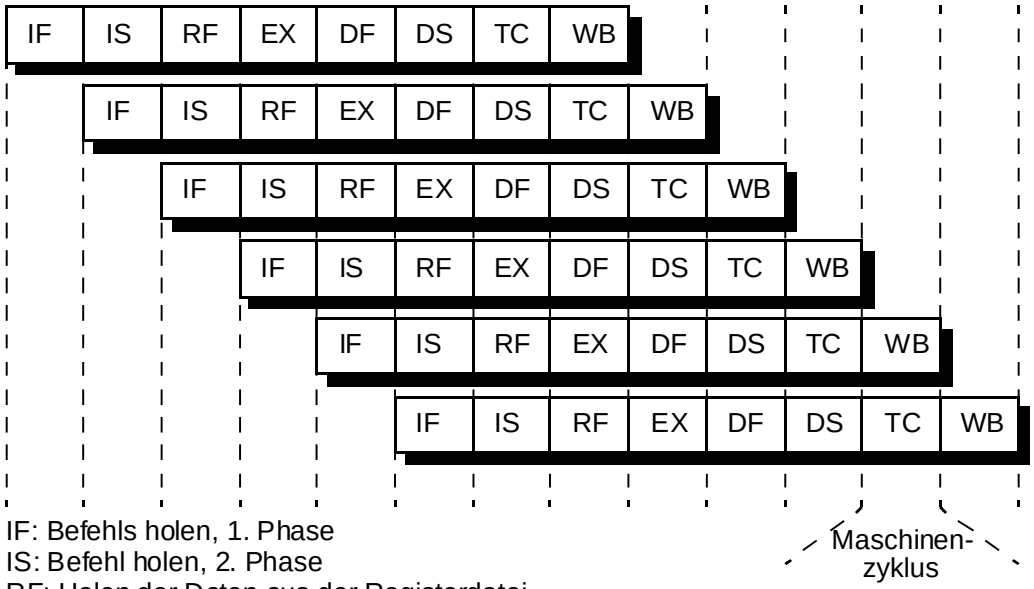
- **Verfeinerung der Pipeline-Stufen**



Pipelining

RISC (Reduced Instruction Set Computers)

- **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
  - Einfaches Verfahren zur Auflösung von Konflikten:
    - 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, die den Befehlszähler verändern,** auf

Pipelining

RISC (Reduced Instruction Set Computers)

- **Pipeline-Konflikte (Pipeline Hazards, Pipeline-Hemmnisse)**
  - **Auflösung der Pipeline-Konflikte**
    - Einfache Lösung: Anhalten der Pipeline (**Pipeline stall**)
    - 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

Vorlesung Rechnerstrukturen

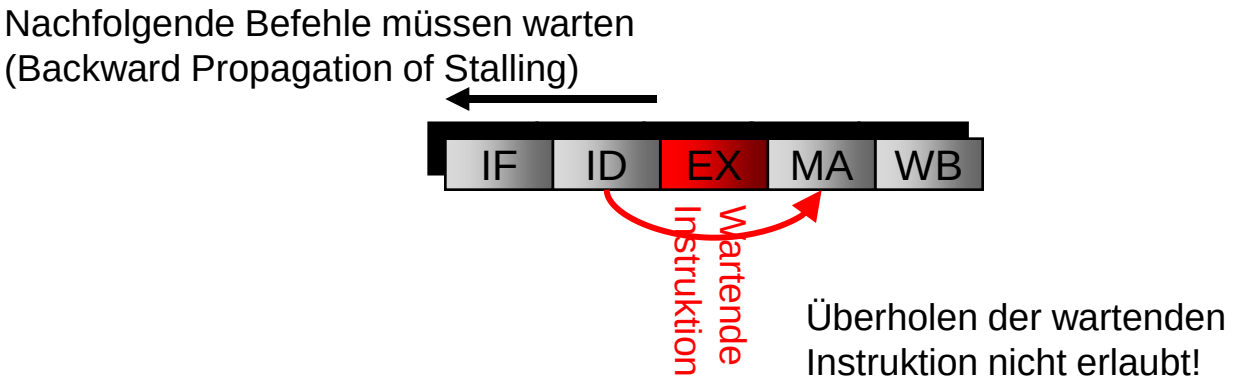
Kapitel 2: Parallelismus auf Maschinenbefehlsebene

- 2.1 Pipelining
- 2.2 Nebenläufigkeit
  - Superskalartechnik

Parallelismus auf Maschinenbefehlsebene

RISC (Reduced Instruction Set Computers)

- **Einschränkungen skalarer Pipelines**
  - Obere Grenze des Durchsatzes:
    - IPC <= 1 oder CPI >= 1
  - Ineffiziente Pipeline
    - Lange Latenzzeiten für eine Instruktion
  - Pipeline Stall Strategie
    - Anhalten der Pipeline bewirkt, dass nachfolgende Befehle ebenfalls warten müssen



Parallelismus auf Maschinenbefehlsebene

RISC (Reduced Instruction Set Computers)

- **Einschränkungen skalarer Pipelines**
  - Obere Grenze des Durchsatzes:
    - IPC <= 1 oder CPI >= 1
    - Lösung: **Nebenläufigkeit, parallele Pipelines**
  - Ineffiziente Pipeline
    - Lange Latenzzeiten für eine Instruktion
    - Lösung: **diversifizierte, spezialisierte Pipelines** (siehe Folie 2-28)
  - Pipeline Stall Strategie
    - Anhalten der Pipeline bewirkt, dass nachfolgende Befehle ebenfalls warten müssen
    - **„Out-of-Order“ Strategie, verteilte Ausführungspipelines**

Parallelismus auf Maschinenbefehlsebene

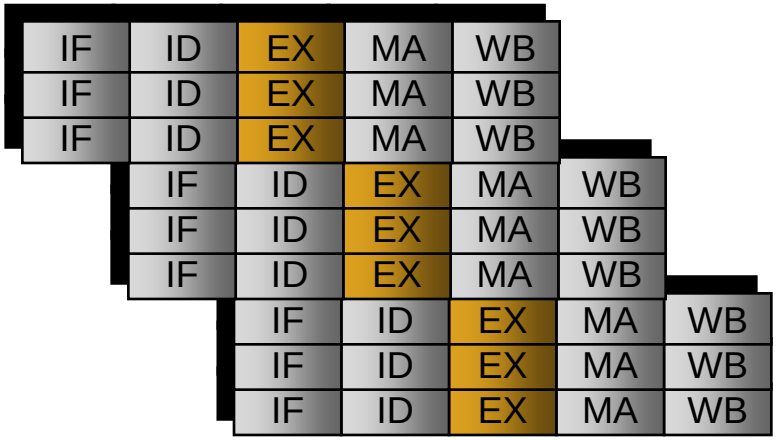
Nebenläufigkeit

- **Dynamische Ansätze**
  - Superskalartechnik
- **Statische Ansätze**
  - VLIW (Very Long Instruction Word)
  - EPIC (Explicitly Parallel Instruction Computer)

Parallelismus auf Maschinenbefehlsebene

Nebenläufigkeit

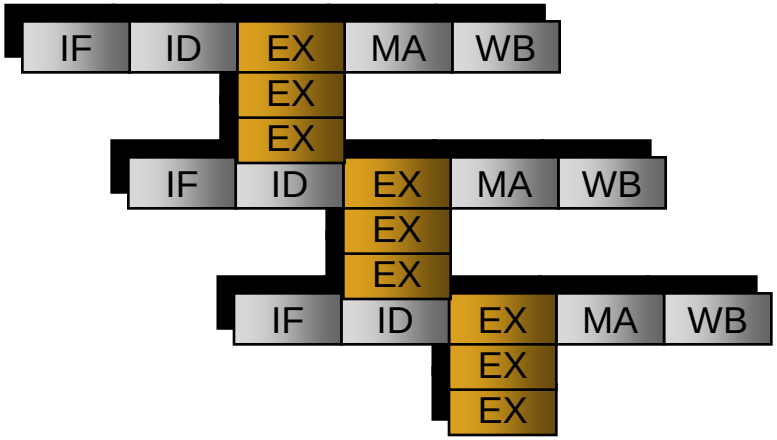
- **Dynamische Ansätze**
  - **Superskalartechnik**
    - Anstoßen (Issue) von n Befehlen pro Zyklus
    - Max. IPC = n Befehle pro Zyklus



Parallelismus auf Maschinenbefehlsebene

Nebenläufigkeit

- **Statische Ansätze**
  - **VLIW (Very Long Instruction Work)**
    - Anstoßen (Issue) von n Operationen pro Zyklus
    - Max. IPC = n Operationen pro Zyklus = 1 VLIW Befehl pro Zyklus



Parallelismus auf Maschinenbefehlsebene

Nebenläufigkeit

- **RISC → Superskalar**
  - **Mehrfachzuweisungsmethoden (multiple issue)**
    - Die Superskalar-Technik ermöglicht es, pro Takt mehrere Befehle den Ausführungseinheiten zuzuordnen und eine gleiche Anzahl von Befehlsausführungen pro Takt zu beenden.
  - **Superskalare RISC-Prozessoren:**
    - RISC-Charakteristika werden auch heute noch weitgehend beibehalten
      - Lade-/Speicher-Architektur
      - Festes Befehlsformat (z. B.: Befehlslänge: 32 Bit)
    - Entwurfsziel: Erhöhung des IPC (Instruction per Cycle)
    - Heutige Mikroprozessoren nutzen Parallelismus auf Maschinenbefehlsebene durch die Pipelining- und Superskalartechnik

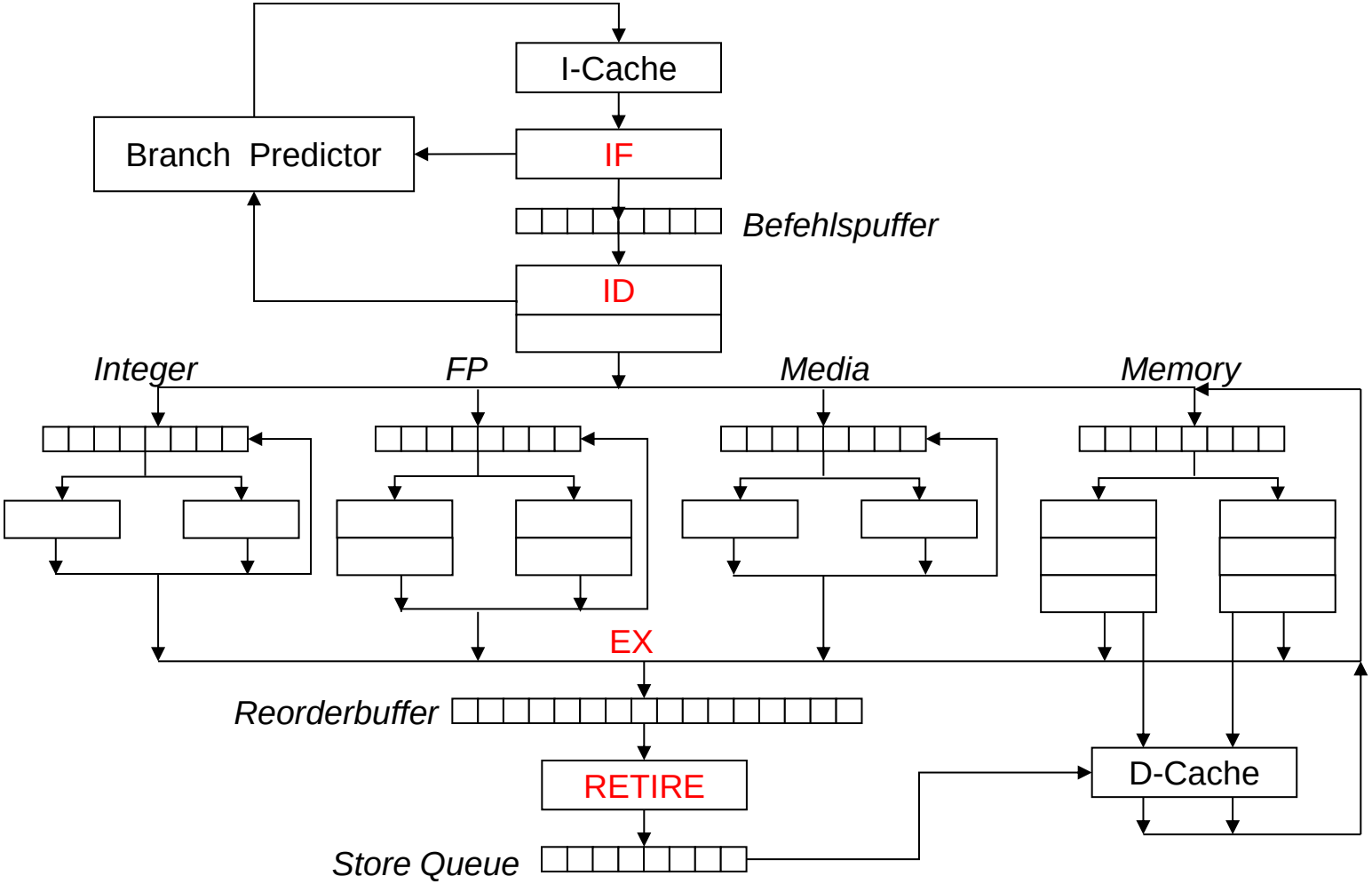
Parallelismus auf Maschinenbefehlsebene

Nebenläufigkeit

- **Superskalarer Prozessor**
  - Nützt den Parallelismus auf Befehlsebene aus
    - **Vielstufige Befehlspipeline**
    - **Superskalartechnik**
  - **Eigenschaften:**
    - Mehrere voneinander unabhängige Ausführungseinheiten
    - Zur Laufzeit werden pro Takt mehrere Befehle aus einem sequentiellen Befehlsstrom den Verarbeitungseinheiten zugeordnet und ausgeführt
    - **Dynamische Erkennung und Auflösung von Konflikten** zwischen Befehlen im Befehlsstrom ist Aufgabe der Hardware

Parallelismus auf Maschinenbefehlsebene

Superskalarer Prozessor



Parallelismus auf Maschinenbefehlsebene

Superskalarer Prozessor

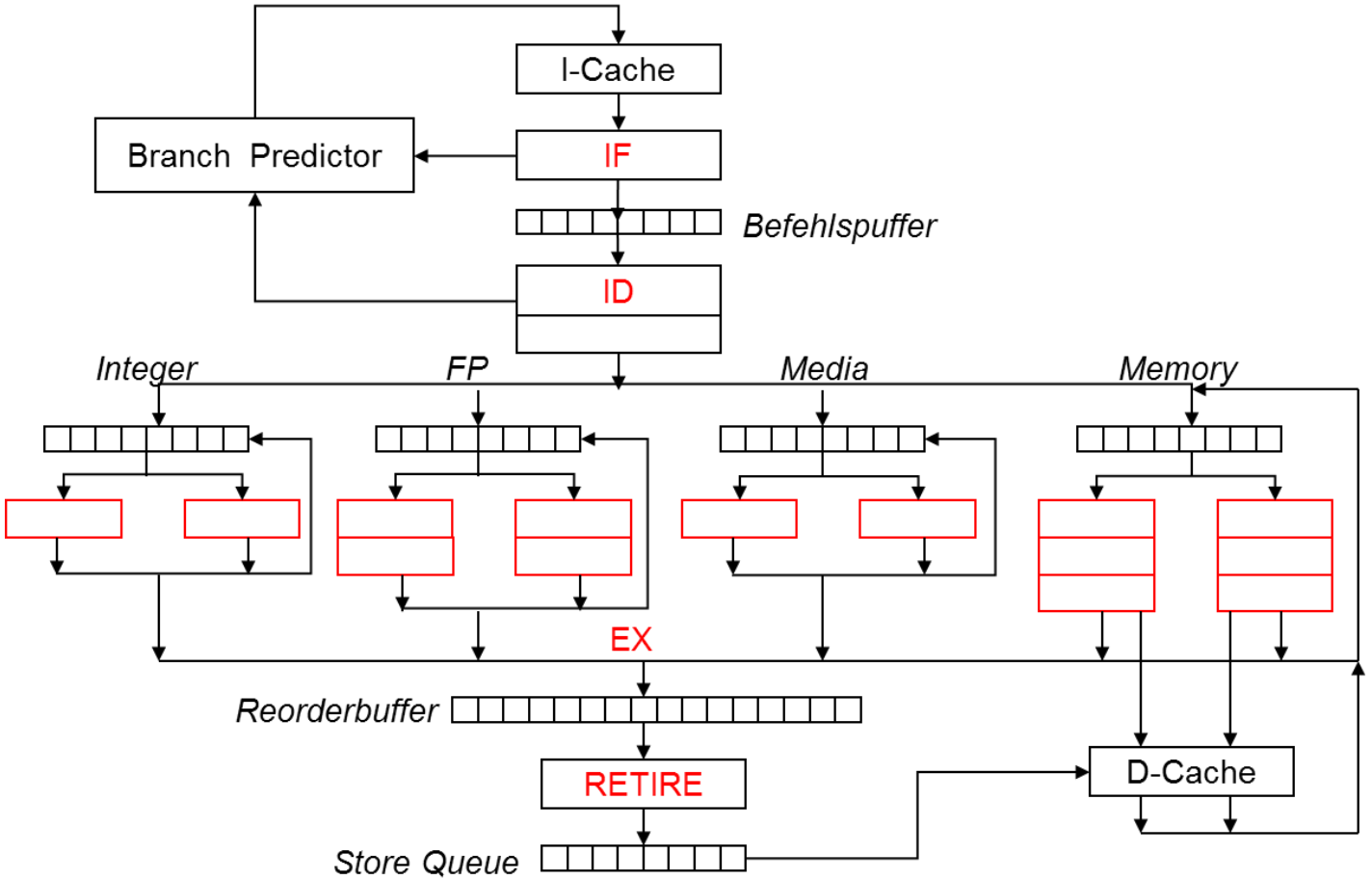
- **Komponenten**
  - **Befehlsholeinheit** (Instruction Fetch)
  - **Dekodiereinheit** (Instruction Decode) mit **Registerumbenennung** (register renaming)
  - **Zuordnungseinheit** (Instruction Issue)
  - Unabhängige **Verarbeitungseinheiten** (Functional Units)
  - **Rückordnungseinheit** (Retire Unit)
  - **Register**:
    - Allzweckregister
    - Multimediaregister
    - Spezialregister

  - *Anmerkung: Die Bezeichnungen der Einheiten sind bei den verschiedenen Prozessoren nicht einheitlich!*

Parallelismus auf Maschinenbefehlsebene

Superskalarer Prozessor

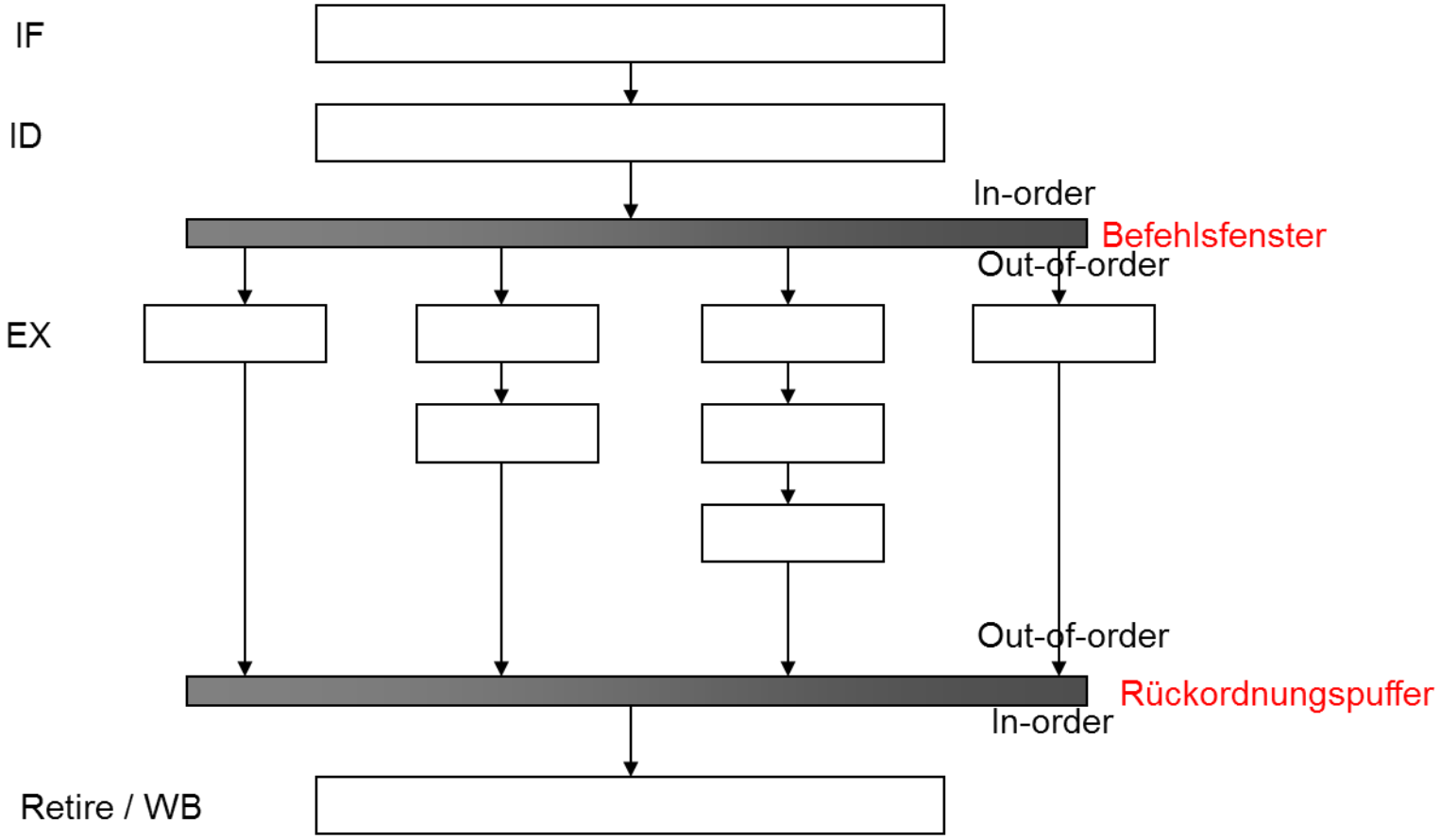
- **Spezialisierte Pipelines**



Parallelismus auf Maschinenbefehlsebene

Superskalarer Prozessor

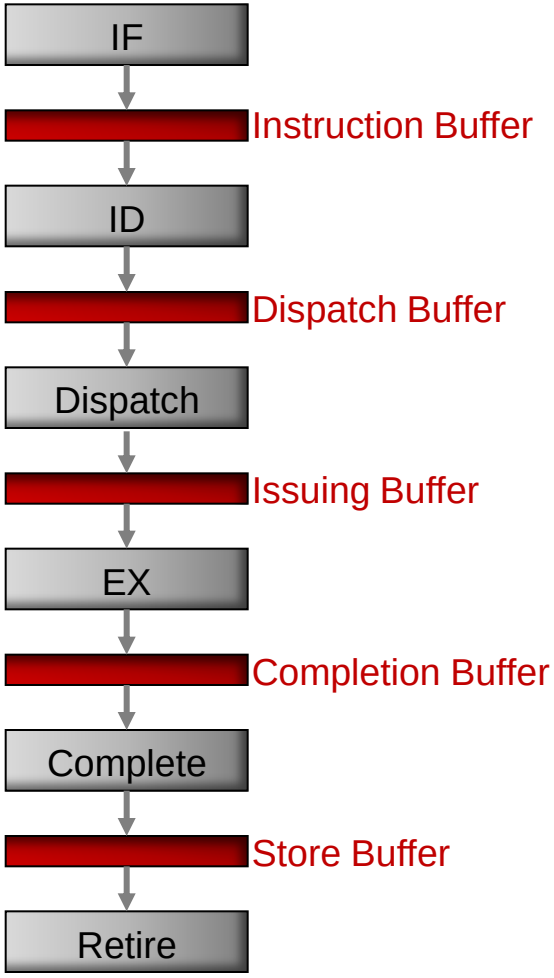
- **Dynamische Pipelines**



Parallelismus auf Maschinenbefehlsebene

Superskalarer Prozessor

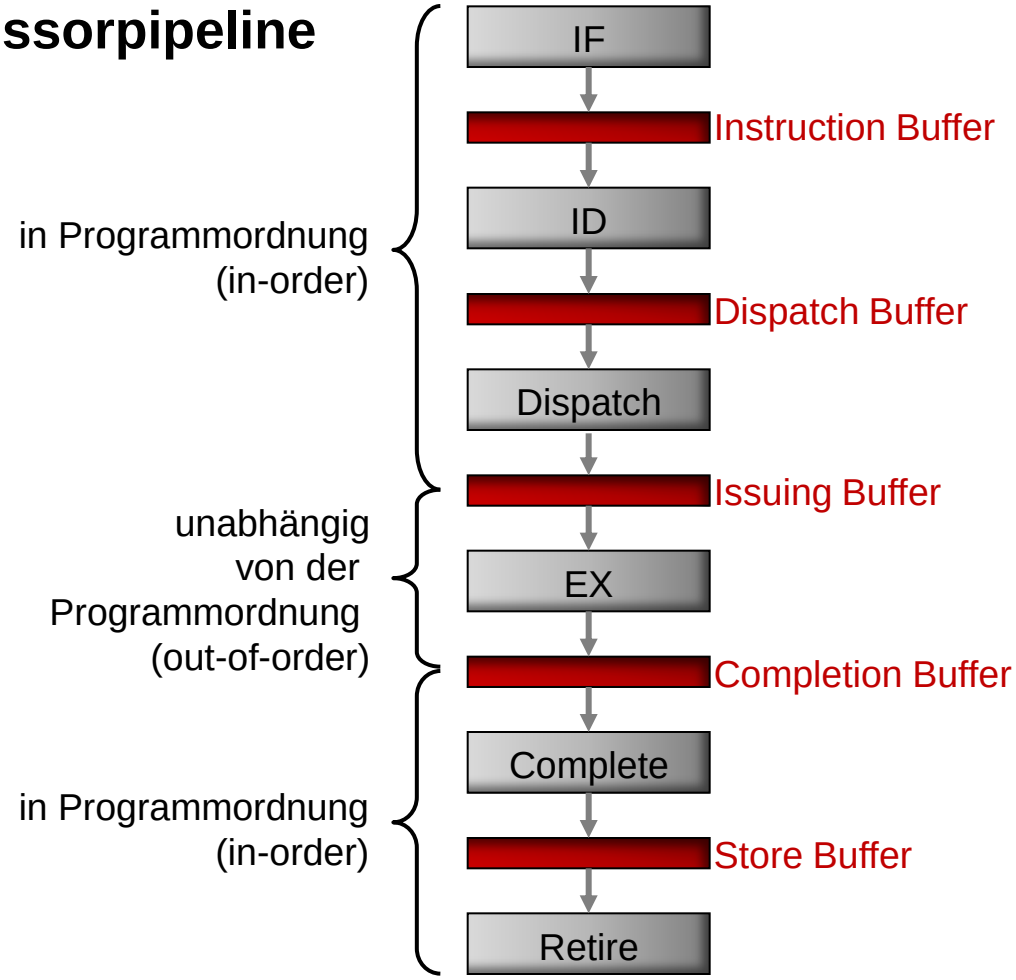
- **Prozessorpipeline**



Parallelismus auf Maschinenbefehlsebene

Superskalarer Prozessor

- **Prozessorpipeline**



Parallelismus auf Maschinenbefehlsebene

Superskalare Prozessorpipeline

- **1. In-order-Abschnitt**
  - Befehle werden entsprechend ihrer Programmordnung bearbeitet
  - Umfasst
    - die **Befehlsholphase** (IF)
    - die **Dekodierphase** (ID)
    - die **Zuordnungsstufe** (Dispatch)
      - Dynamische Zuordnung der Befehle an die Ausführungseinheiten
      - Scheduler bestimmt die Anzahl der Befehle, die im nächsten Takt zugeordnet werden können

Parallelismus auf Maschinenbefehlsebene

Superskalare Prozessorpipeline

- **Out-of-order-Abschnitt**
  - **Ausführungsphase**

- **2. In-order-Abschnitt**
  - **Gültigmachen der Ergebnisse** entsprechend der ursprünglichen Programmordnung (**Retire**)
  - Erhalten der korrekten Programmsemantik
    - Ausnahmeverarbeitung (precise interrupts)
    - Spekulation

Parallelismus auf Maschinenbefehlsebene

Superskalare Prozessorpipeline

- **Befehlsholphase (IF Phase)**
  - **Befehlsbereitstellung**
    - Holen mehrerer Befehle aus dem Befehls-Cache in den Befehlsholpuffer
    - Anzahl der Befehle, die geholt werden, entspricht typischer Weise der Zuordnungsbandbreite
    - Welche Befehle geholt werden hängt von der Sprungvorhersage ab
  - **Verzweigungseinheit**
    - Überwacht die Ausführung von Verzweigungen, Sprungbefehlen
    - Spekulatives Holen von Befehlen
    - Spekulation über weiteren Programmverlauf wird von dynamischen Sprungvorhersagetechnik entschieden
    - Verwendung der Vorgeschichte von Sprüngen
    - Gewährleistet im Falle einer Fehlspekulation die Abänderung der Tabellen sowie das Rückrollen der fälschlicherweise ausgeführten Befehle
  - **Befehlsholpuffer**
    - entkoppelt die IF Phase von der ID Phase

Parallelismus auf Maschinenbefehlsebene

Superskalare Prozessorpipeline

- **Befehlsholphase (IF Phase)**
  - **Sprungvorhersage und spekulative Ausführung**
  - **Problem**
    - Hohe Zuordnungs- und Ausführungsbandbreite
    - Etwa jeder 5.- 7. Befehl ist bedingter Sprungbefehl, der den kontinuierlichen Befehlsfluss in der Pipeline unterbrechen kann
    - Unter Berücksichtigung der spekulativen Ausführung von Befehlen können sich mehrere Sprungbefehle in der Pipeline befinden

Parallelismus auf Maschinenbefehlsebene

Superskalare Prozessorpipeline

- **Sprungvorhersage (Branch Prediction):** Vorhersage des Verhaltens bei Verzweigungen
  - Beim Auftreten einer Verzweigung: Vorhersage des Sprungziels
  - Füllen der Verzögerungsphasen spekulativ mit Befehlen, die dem Sprung folgen oder die am Sprungziel stehen
  - Nach Auswertung der Sprungbedingung:
    - Fortfahren mit der Ausführung ohne Verzögerung bei korrekter Vorhersage.
    - Verwerfen der geholten Befehle bei falscher Vorhersage

Parallelismus auf Maschinenbefehlsebene

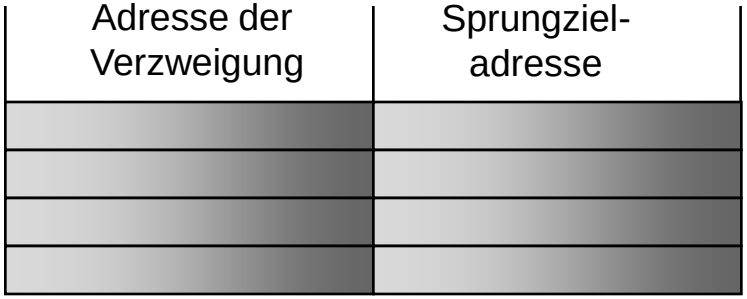
Superskalare Prozessorpipeline

- **Sprungvorhersage (Branch Prediction):** Vorhersage des Verhaltens bei Verzweigungen
  - **Statische Sprungvorhersage**
    - Die Richtung der Vorhersage ist für einen Befehl immer gleich
    - Für superskalare Prozessorarchitekturen zu unflexibel und nicht geeignet
  - **Dynamische Sprungvorhersage**
    - Die Verzweigungsrichtung hängt von der Vorgeschichte der Verzweigung ab
    - Berücksichtigung des Programmverhaltens
    - Genaue Vorhersagen möglich
    - Hoher Hardware-Aufwand!

Parallelismus auf Maschinenbefehlsebene

Superskalare Prozessorpipeline

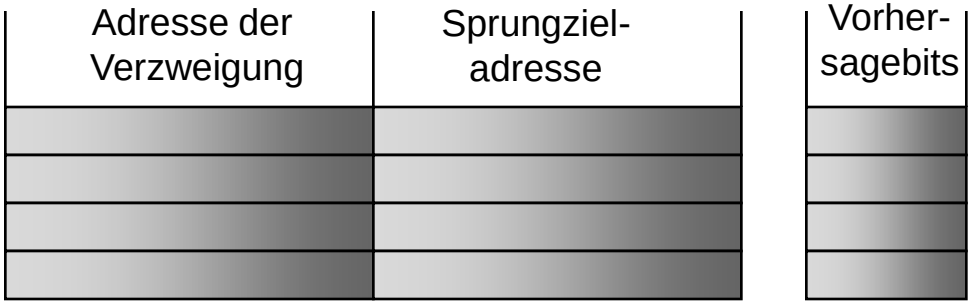
- **Dynamische Sprungvorhersage**
  - **Sprungziel-Cache**: **Branch Target Address Cache** (BTAC), **Branch Target Buffer** (BTB)
    - Speichert die Adresse der Verzweigung und das entsprechende Sprungziel



Parallelismus auf Maschinenbefehlsebene

Superskalare Prozessorpipeline

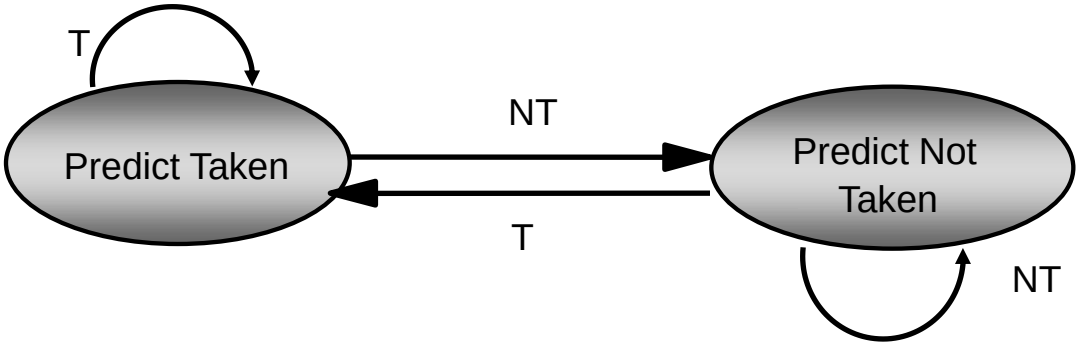
- **Dynamische Sprungvorhersage**
  - **Sprungziel-Cache**: **Branch Target Address Cache** (BTAC), **Branch Target Buffer** (BTB)
    - Speichert die Adresse der Verzweigung und das entsprechende Sprungziel
  - **Sprungverlaufstabelle**, **Branch History Table** (BHT)
    - Festhalten des Verhaltens der Sprungbefehle während der Ausführung des Programms: **Prädiktoren**
    - Vorhersage des Verhaltens eines geholten Sprungbefehls



Parallelismus auf Maschinenbefehlsebene

Superskalare Prozessorpipeline

- **Dynamische Sprungvorhersage**
  - **Sprungverlaufstabelle**, **Branch History Table** (BHT)
    - Vorhersagebit:
      - Wenn das Bit gesetzt ist, wird angenommen, dass der Sprung ausgeführt wird.
      - Wenn das Bit nicht gesetzt ist, wird angenommen, dass der Sprung nicht ausgeführt wird.
    - Bei einer Fehlannahme: Invertieren des Bits



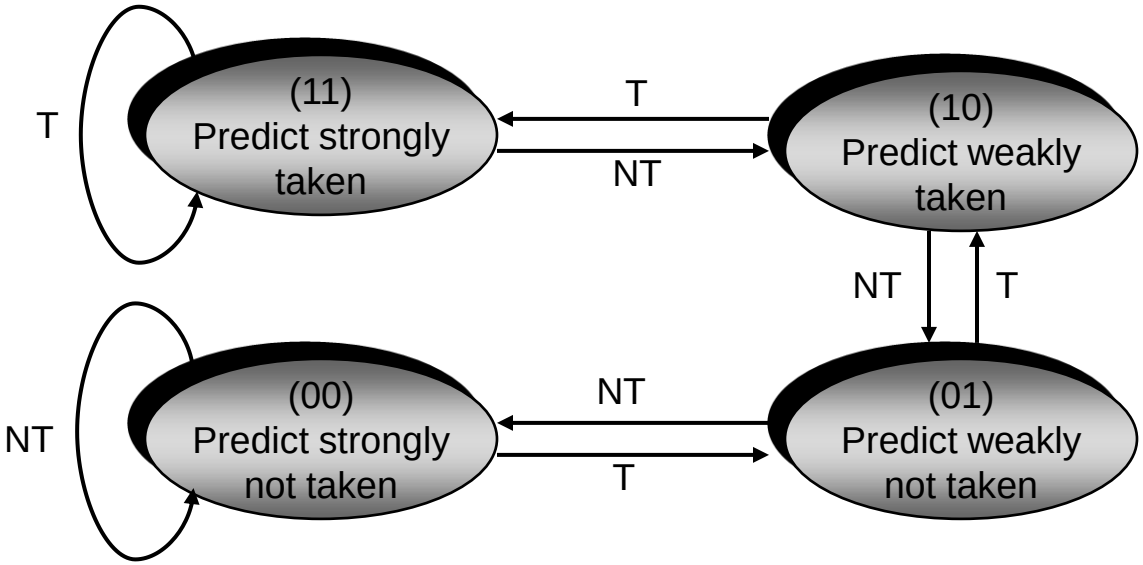
Parallelismus auf Maschinenbefehlsebene

Superskalare Prozessorpipeline

- **Dynamische Sprungvorhersage**
  - **Sprungverlaufstabelle**, **Branch History Table**: Zwei-Bit Predictor
    - Zwei Bit pro Eintrag für die Kodierung der Vorhersage → vier Zustände:
      - Sicher genommen (stronly taken)
      - Vielleicht genommen (weakly taken)
      - Vielleicht nicht genommen (weakly not taken)
      - Sicher nicht genommen (stronly not taken)
    - In einem sicheren Zustand sind zwei aufeinander folgende Fehlannahmen notwendig, um die Vorhersageannahme umzudrehen.

Superskalare Prozessorpipeline

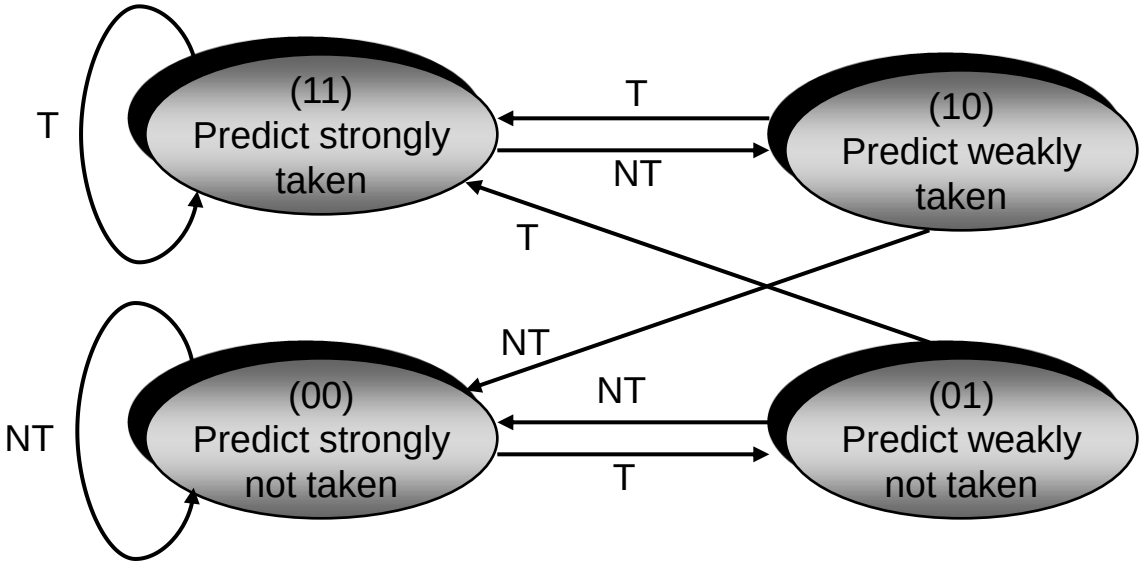
- **Dynamische Sprungvorhersage**
  - **Sprungverlaufstabelle**, **Branch History Table**: Zwei-Bit Predictor
    - **Sättigungszähler** (Two Bit Predictor with Saturation Scheme)



Parallelismus auf Maschinenbefehlsebene

Superskalare Prozessorpipeline

- **Dynamische Sprungvorhersage**
  - **Sprungverlaufstabelle**, **Branch History Table**: Zwei-Bit Predictor
    - **Hysteresemethode** (Two Bit Predictor with Hysteresis Scheme)



Parallelismus auf Maschinenbefehlsebene

Superskalare Prozessorpipeline

- **Dynamische Sprungvorhersage**
  - Sehr aufwendige Techniken für superskalare Prozessoren für die Gewährleistung einer möglichst genauen Vorhersage
  - (m,n)-Korrelationsprädiktoren
  - Zweistufige adaptive Prädiktoren
  - Gselect- und gshare-Prädiktoren
  - Hybridprädiktoren

  - Literatur zur Sprungvorhersage:
    - Brinkschulte/Ungerer: Microcontroller und Mikroprozessoren: Kap. 2.4.6, 7.2

Parallelismus auf Maschinenbefehlsebene

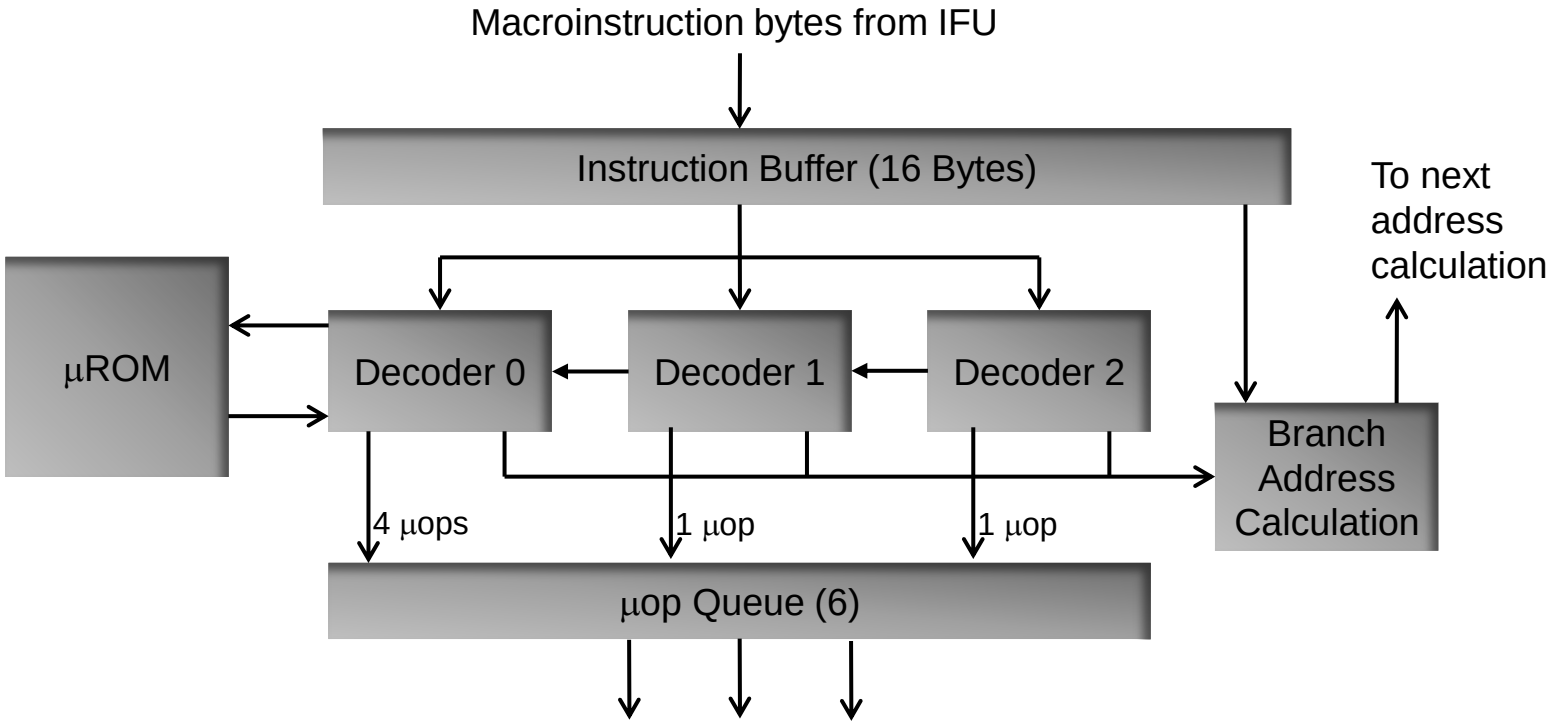
Superskalare Prozessorpipeline

- **Dekodierphase** (ID Phase)
  - Dekodierung der im Befehlspuffer abgelegten Befehle
    - Anzahl der Befehle, die dekodiert werden, entspricht typischer Weise der **Befehlsbereitstellungsbandbreite**
  - Bei **CISC-Architekturen** (IA-32)
    - Aufteilung der Dekodierung in mehrere Schritte
      - Bestimmung der Grenzen der geholten Befehle
      - Dekodierung der Befehle
      - Generierung einer Folge von RISC-ähnlichen Operationen mit Hilfe dynamischer Übersetzungstechniken
      - Ermöglicht effizientes Pipelining und superskalare Verarbeitung
      - Beispiel Intel Pentium- und AMD Athlon-Familie

Parallelismus auf Maschinenbefehlsebene

Superskalare Prozessorpipeline

- **Dekodierphase** (ID Phase)
  - Bei **CISC-Architekturen** (IA-32): Intel Pentium Pro



Parallelismus auf Maschinenbefehlsebene

Superskalare Prozessorpipeline

- **Dekodierphase** (ID Phase)
  - **Registerumbenennung**
    - Dynamische Umbenennung der Operanden- und Ergebnisregister
    - Abbildung der nach außen hin sichtbaren Architekturregister in interne physikalische Register
      - Zur Laufzeit wird für jeden Befehl das jeweils spezifizierte Zielregister auf ein noch nicht belegtes physikalisches Register abgebildet
      - Nachfolgende Befehle, die dasselbe Architekturregister als Operandenregister verwenden, erhalten das entsprechende physikalische Register
      - Anzahl der Umbenennungsregister kann die Anzahl der Architekturregister überschreiten

Parallelismus auf Maschinenbefehlsebene

Superskalare Prozessorpipeline

- **Dekodierphase** (ID Phase)
  - **Registerumbenennung**
    - **Auflösung von Konflikten aufgrund von Namensabhängigkeiten:**
    - **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

Parallelismus auf Maschinenbefehlsebene

Superskalare Prozessorpipeline

- **Dekodierphase** (ID Phase)
  - **Registerumbenennung**
  - **Ursachen** für diese 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`

Parallelismus auf Maschinenbefehlsebene

Superskalare Prozessorpipeline

- **Dekodierphase (ID Phase)**
  - Schreiben der Befehle in ein Befehlsfenster (instruction window)
  - Folge:
    - Befehle sind durch die Sprungvorhersage frei von Steuerflussabhängigkeiten
    - Befehle sind aufgrund der Registerumbenennung frei von Namensabhängigkeiten

Parallelismus auf Maschinenbefehlsebene

Superskalare Prozessorpipeline

- **Zuordnungsphase (Dispatch)**
  - Zuführung der im Befehlsfenster wartenden Befehle zu den Ausführungseinheiten
  - Zuordnung bis zur maximalen Zuordnungsbandbreite pro Takt
  - Dynamische Auflösung der Konflikte aufgrund von echten Datenabhängigkeiten und Ressourcenkonflikten

Parallelismus auf Maschinenbefehlsebene

Superskalare Prozessorpipeline

- **Zuordnungsphase (Dispatch)**
  - Zuführung der im Befehlsfenster wartenden Befehle zu den Ausführungseinheiten
  - Zuordnung bis zur maximalen Zuordnungsbandbreite pro Takt
  - **Dynamische Auflösung der Konflikte aufgrund von echten Datenabhängigkeiten und Ressourcenkonflikten**
    - **Lese-nach-Schreib-Konflikt (Read-After-Write, RAW)**
      - Tritt auf, wenn Befehl j sein Quellregister liest, bevor Befehl i das Ergebnis geschrieben hat.

Parallelismus auf Maschinenbefehlsebene

Superskalare Prozessorpipeline

- **Zuordnungsphase (Dispatch)**
  - **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

Parallelismus auf Maschinenbefehlsebene

Superskalare Prozessorpipeline

- **Zuordnungsphase (Dispatch)**
  - **Umordnungspuffer** (**Reservierungstabellen, reservation stations**)
    - Liegen vor den Verarbeitungseinheiten
    - Jede Ausführungseinheit hat seinen eigenen Umordnungspuffer oder mehrere Ausführungseinheiten teilen sich einen Umordnungspuffer
    - Zuordnung eines Befehls an Umordnungspuffer kann nur erfolgen, wenn ein freier Platz vorhanden ist, ansonsten müssen die nachfolgenden Befehle warten (Auflösen von Ressourcenkonflikten)

Parallelismus auf Maschinenbefehlsebene

Superskalare Prozessorpipeline

- **Zuordnungsphase (Dispatch)**
  - **Rückordnungspuffer (reorder buffer)**
    - Festhalten der ursprünglichen Befehlsanordnung
    - Eintragen der Befehle, die die Dekodierphase verlassen und in das Befehlsfenster eingetragen werden
    - Während der folgenden Phasen, die ein Befehl zu durchlaufen hat, wird dessen jeweiliger Ausführungsstand protokolliert.

Parallelismus auf Maschinenbefehlsebene

Superskalare Prozessorpipeline

- **Befehlsausführung**
  - Ausführung der im Opcode spezifizierten Operation und Speichern des Ergebnisses im Zielregister (Umbenennungsregister)
  - Einzyklusoperationen
    - Ausführung benötigt einen Taktzyklus
  - Mehrzyklusoperationen
    - Ausführung einer Operation auf einer Ausführungseinheit kann mehrere Zyklen dauern
    - Ausführungs-Pipeline, arithmetische Pipeline

Parallelismus auf Maschinenbefehlsebene

Superskalare Prozessorpipeline

- **Completion**
  - Eine Instruktion beendet ihre Ausführung, wenn das Ergebnis für nachfolgende Befehle bereitsteht (Forwarding, Puffer)
  - **Completion** heißt: eine Befehlsausführung ist „vollständig“
    - Erfolgt unabhängig von der Programmordnung!
  - Bereinigung der Reservierungstabellen
  - Aktualisierung des Zustands des Rückordnungspuffers (Reorder Buffer)
    - Es kann eine Unterbrechung angezeigt sein.
    - Es kann ein vollständiger Befehl angezeigt werden, der von einer Spekulation abhängt.

Parallelismus auf Maschinenbefehlsebene

Superskalare Prozessorpipeline

- **Rückordnungsstufe (Retire)**
  - **Commitment:**
    - Nach der Vervollständigung beenden die Befehle ihre Bearbeitung (Commitment), d.h. die Befehlsresultate werden in der Programmreihenfolge gültig gemacht
    - Ergebnisse werden in den Architekturregistern dauerhaft gemacht, d.h. aus den internen Umbenennungsregistern (Schattenregistern) zurück geschrieben.
  - **Bedingungen für Commitment:**
    - Die Befehlsausführung ist vollständig
    - Alle Befehle, die in der Programmordnung vor dem Befehl stehen, haben bereits ihre Bearbeitung beendet oder beenden ihre Bearbeitung im selben Takt.
    - Der Befehl hängt von keiner Spekulation ab.
    - Keine Unterbrechung ist vor oder während der Ausführung aufgetreten

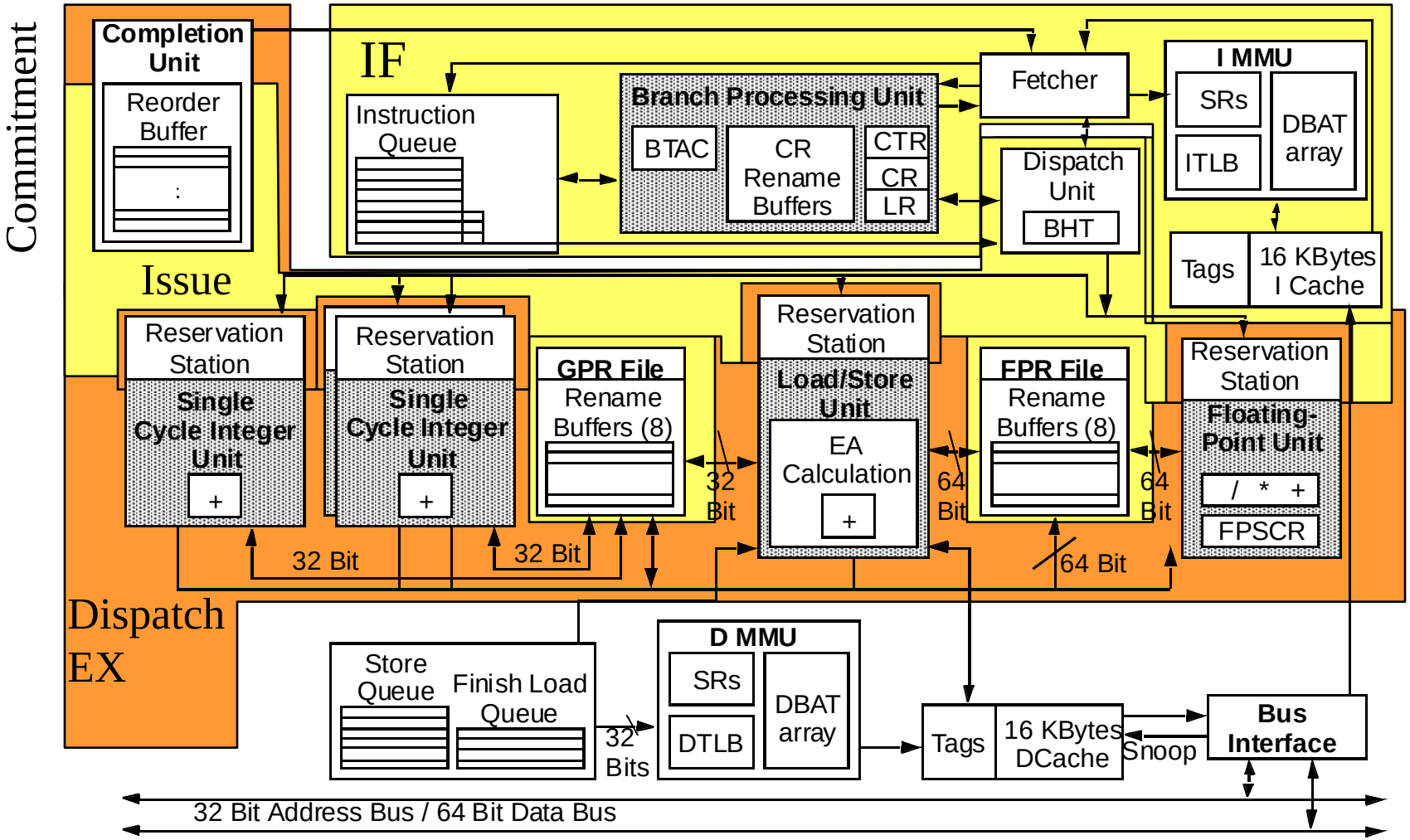
Parallelismus auf Maschinenbefehlsebene

Superskalare Prozessorpipeline

- **Rückordnungsstufe (Retire)**
  - Forderung: **Precise Interrupts** bei Auftreten einer Unterbrechung
    - Alle Resultate von Befehlen, die in der Programmreihenfolge vor dem Ereignis stehen, werden gültig gemacht
    - Die Resultate aller nachfolgenden Befehle werden verworfen
    - Das Ergebnis des verursachenden Befehls wird in Abhängigkeit der Architektur oder der Art der Unterbrechung gültig gemacht oder verworfen, ohne weitere Auswirkungen zu haben

Parallelismus auf Maschinenbefehlsebene

- **Fallstudie: Motorola PowerPC 604**



Parallelismus auf Maschinenbefehlsebene

Dynamische Methoden zur Erkennung und Auflösung von Datenkonflikten

- **Detaillierte Betrachtung der Zuordnungsphase**
- **Fallstudie: Tomasulo (IBM 360/91)**
  - Konfliktauflösung und Ablaufsteuerung verteilt;
  - jede Funktionseinheit verfügt über eine Reservierungstabelle mit möglicherweise mehreren Zeilen (Einträgen);
  - Reservierungstabelle (Umordnungspuffer, Reservation Station)
    - übernimmt die Kontrolle über die Abarbeitung eines Maschinenbefehls, wenn dieser von der Decodiereinheit zur Ausführung angestoßen wird (Issue);
  - Ergebnisbus:
    - ein von einer Funktionseinheit i produziertes Ergebnis wird direkt an die Funktionseinheit j weitergegeben, wenn diese das Ergebnis als Operand benötigt;
    - alle Funktionseinheiten, die auf einen Operanden warten, werden gleichzeitig bedient;

Superskalartechnik

Fallstudie: Tomasulo (IBM 360/91)

- **Umordnungspuffer** (**Reservierungstabelle, Reservation Station**)
  - Jeder Eintrag enthält jeweils Felder für:
    - die Werte der zwei Quelloperanden (Src1, Src2);
    - die Nummern (Namen, Tags) der Reservierungstabellen derjenigen Funktionseinheiten, welche die Quelloperanden für die auszuführende Operation liefern werden (RS1, RS2), falls der Operand noch berechnet wird
    - jeweils ein Flag für jeden Operanden, das anzeigt, ob ein Operand verfügbar ist (Vld1,Vld2) und
    - einen Namen (destination tag) für das Ziel (Dest).

	Quelloperand 1			Quelloperand 2			Ziel
	Vld1	Src1	RS1	Vld2	Src2	RS2	Dest
Befehl n							
Befehl n+1							
Befehl n+2							

Fallstudie: Tomasulo (IBM 360/91)

- **Registerdatei**
  - Jedes Register einer Registerdatei enthält
    - das Feld für den Wert (Registerinhalt)
    - einen Namen (destination tag, Dest), der mit dem Ergebnis assoziiert ist, und
    - ein Bit, das anzeigt, ob der Wert für das Register gerade berechnet wird.

Fallstudie: Tomasulo (IBM 360/91)

- **Befehlsausführung**
  - Ein Maschinenbefehl kann zur Ausführung angestoßen werden, falls ein Eintrag in der Reservierungstabelle einer Funktionseinheit frei ist.
    - Falls alle Einträge belegt sind, dann ist ein Ressourcenkonflikt gegeben, und der Maschinenbefehl muss warten, bis ein Eintrag in der Reservierungstabelle frei ist.
    - Für einen von der Decodiereinheit zur Ausführung angestoßener Maschinenbefehl werden die Inhalte seiner Quellregister und die dazugehörigen Ready-Bits von der Registerdatei in die entsprechenden Felder des Umordnungspuffers kopiert.

Superskalartechnik

Fallstudie: Tomasulo (IBM 360/91)

- **Phasen der Pipeline (vereinfacht)**
  - **Issue**
    - Holen der Befehle aus Instruction Queue
    - Holen der Operanden
  - **Ausführung**
    - Wenn die Operanden verfügbar sind, dann Anstoßen der Ausführung auf Funktionseinheit (Dispatch)
    - Beobachten des Ergebnisbusses
    - Befehlsausführung nicht in Programmreihemfolge (out-of-order execution)
  - **Rückschreibphase**
    - Schreiben der Ergebnisse auf Ergebnisbus
    - Kennzeichnen des Umordnungspuffers als verfügbar

Superskalartechnik

Fallstudie: Tomasulo (IBM 360/91)

- Beispiel (T. Ungerer)

registers

R	1	2	3	4	5	6
Value	-	-	(R3)	(R4)	(R5)	-
Vld	1	1	1	1	1	1
RS	0	0	0	0	0	0

register status

```
mul Reg1, Reg3, Reg5
sub Reg2, Reg4, Reg3
div Reg6, Reg1, Reg4
add Reg4, Reg2, Reg3
```

reservation stations

		Empty	InFU	Op	Dest	Src1	Vld1	RS1	Src2	Vld2	RS2
S_{add}	1	1									
S_{add}	2	1									
S_{mul}	3	1									
S_{div}	4	1									

RS status

cycle 0

token.tag

token.data

Superskalartechnik

Fallstudie: Tomasulo (IBM 360/91)

- Beispiel (T. Ungerer)

registers

R	1	2	3	4	5	6
Value	-	-	(R3)	(R4)	(R5)	-
Vld	0	1	1	1	1	1
RS	3	0	0	0	0	0

register status

```
mul Reg1, Reg3, Reg5
sub Reg2, Reg4, Reg3
div Reg6, Reg1, Reg4
add Reg4, Reg2, Reg3
```

reservation stations

		Empty	InFU	Op	Dest	Src1	Vld1	RS1	Src2	Vld2	RS2
S_{add}	1	1									
S_{add}	2	1									
S_{mul}	3	0	0	mul	1	(R3)	1	0	(R5)	1	0
S_{div}	4	1									

RS status

cycle 1

token.tag

token.data

Superskalartechnik

Fallstudie: Tomasulo (IBM 360/91)

- Beispiel (T. Ungerer)

registers

R	1	2	3	4	5	6
Value	-	-	(R3)	(R4)	(R5)	-
Vld	0	0	1	1	1	1
RS	3	1	0	0	0	0

register status

```
mul Reg1, Reg3, Reg5
sub Reg2, Reg4, Reg3
div Reg6, Reg1, Reg4
add Reg4, Reg2, Reg3
```

reservation stations

		Empty	InFU	Op	Dest	Src1	Vld1	RS1	Src2	Vld2	RS2
S_{add}	1	0	0	sub	2	(R4)	1	0	(R3)	1	0
	2	1									
S_{mul}	3	0	1	mul	1	(R3)	1	0	(R5)	1	0
S_{div}	4	1									

RS status

cycle 2

token.tag

token.data

Fallstudie: Tomasulo (IBM 360/91)

- Beispiel (T. Ungerer)

```
mul Reg1, Reg3, Reg5
sub Reg2, Reg4, Reg3
div Reg6, Reg1, Reg4
add Reg4, Reg2, Reg3
```

registers

R	1	2	3	4	5	6
Value	-	-	(R3)	(R4)	(R5)	-
Vld	0	0	1	1	1	0
RS	3	1	0	0	0	4

register status

reservation stations

		Empty	InFU	Op	Dest	Src1	Vld1	RS1	Src2	Vld2	RS2	
S_{add}	1	0	1	sub	2	(R4)	1	0	(R3)	1	0	
	2	1										
S_{mul}	3	0	1	mul	1	(R3)	1	0	(R5)	1	0	3
S_{div}	4	0	0	div	6		0	3	(R4)	1	0	

RS status

cycle 3

token.tag

token.data

remaining cycles in FU

Superskalartechnik

Fallstudie: Tomasulo (IBM 360/91)

- Beispiel (T. Ungerer)

```
mul Reg1, Reg3, Reg5
sub Reg2, Reg4, Reg3
div Reg6, Reg1, Reg4
add Reg4, Reg2, Reg3
```

registers

R	1	2	3	4	5	6
Value	-	(R4)-(R3)	(R3)	-	(R5)	-
Vld	0	1	1	0	1	0
RS	3	0	0	2	0	4

register status

reservation stations

		Empty	InFU	Op	Dest	Src1	Vld1	RS1	Src2	Vld2	RS2	
S_{add}	1	1	1	sub	2	(R4)	1	0	(R3)	1	0	
	2	0	0	add	4	(R4)-(R3)	1	0	(R3)	1	0	
S_{mul}	3	0	1	mul	1	(R3)	1	0	(R5)	1	0	2
S_{div}	4	0	0	div	6		0	3	(R4)	1	0	

RS status

cycle 4

token.tag 1

token.data (R4)-(R3)

Superskalartechnik

Fallstudie: Tomasulo (IBM 360/91)

- Beispiel (T. Ungerer)

```
mul Reg1, Reg3, Reg5
sub Reg2, Reg4, Reg3
div Reg6, Reg1, Reg4
add Reg4, Reg2, Reg3
```

registers

R	1	2	3	4	5	6
Value	-	(R4)-(R3)	(R3)	-	(R5)	-
Vld	0	1	1	0	1	0
RS	3	0	0	2	0	4

register status

reservation stations

		Empty	InFU	Op	Dest	Src1	Vld1	RS1	Src2	Vld2	RS2
S_{add}	1	1	1	sub	2	(R4)	1	0	(R3)	1	0
	2	0	1	add	4	(R4)-(R3)	1	0	(R3)	1	0
S_{mul}	3	0	1	mul	1	(R3)	1	0	(R5)	1	0
S_{div}	4	0	0	div	6		0	3	(R4)	1	0

1

RS status

cycle 5

token.tag

token.data

Superskalartechnik

Fallstudie: Tomasulo (IBM 360/91)

- Beispiel (T. Ungerer)

```
mul Reg1, Reg3, Reg5
sub Reg2, Reg4, Reg3
div Reg6, Reg1, Reg4
add Reg4, Reg2, Reg3
```

registers

R	1	2	3	4	5	6
Value	-	(R4)-(R3)	(R3)	(R4)-(R3)+(R3)	(R5)	-
Vld	0	1	1	1	1	0
RS	3	0	0	0	0	4

register status

reservation stations

		Empty	InFU	Op	Dest	Src1	Vld1	RS1	Src2	Vld2	RS2	
S_{add}	1	1	1	sub	2	(R4)	1	0	(R3)	1	0	
S_{add}	2	1	1	add	4	(R4)-(R3)	1	0	(R3)	1	0	
S_{mul}	3	0	1	mul	1	(R3)	1	0	(R5)	1	0	0
S_{div}	4	0	0	div	6		0	3	(R4)	1	0	

RS status

cycle 6

token.tag 2

token.data (R4)-(R3)+(R3)

Fallstudie: Tomasulo (IBM 360/91)

- Beispiel (T. Ungerer)

registers

R	1	2	3	4	5	6
Value	(R3)*(R5)	(R4)-(R3)	(R3)	(R4)-(R3)+(R3)	(R5)	-
Vld	1	1	1	1	1	0
RS	0	0	0	0	0	4

register status

```
mul Reg1, Reg3, Reg5
sub Reg2, Reg4, Reg3
div Reg6, Reg1, Reg4
add Reg4, Reg2, Reg3
```

reservation stations

		Empty	InFU	Op	Dest	Src1	Vld1	RS1	Src2	Vld2	RS2
S_{add}	1	1	1	sub	2	(R4)	1	0	(R3)	1	0
S_{add}	2	1	1	add	4	(R4)-(R3)	1	0	(R3)	1	0
S_{mul}	3	1	1	mul	1	(R3)	1	0	(R5)	1	0
S_{div}	4	0	0	div	6	(R3)*(R5)	1	0	(R4)	1	0

RS status

cycle 7

token.tag 3

token.data (R3)*(R5)

Superskalartechnik

Fallstudie: Tomasulo (IBM 360/91)

- Beispiel (T. Ungerer)

registers

R	1	2	3	4	5	6
Value	(R3)*(R5)	(R4)-(R3)	(R3)	(R4)-(R3)+(R3)	(R5)	-
Vld	1	1	1	1	1	0
RS	0	0	0	0	0	4

register status

```
mul Reg1, Reg3, Reg5
sub Reg2, Reg4, Reg3
div Reg6, Reg1, Reg4
add Reg4, Reg2, Reg3
```

reservation stations

		Empty	InFU	Op	Dest	Src1	Vld1	RS1	Src2	Vld2	RS2
S_{add}	1	1	1	sub	2	(R4)	1	0	(R3)	1	0
S_{add}	2	1	1	add	4	(R4)-(R3)	1	0	(R3)	1	0
S_{mul}	3	1	1	mul	1	(R3)	1	0	(R5)	1	0
S_{div}	4	0	1	div	6	(R3)*(R5)	1	0	(R4)	1	0

RS status

cycle 8

token.tag
token.data

Fallstudie: Tomasulo (IBM 360/91)

- Beispiel (T. Ungerer)

registers

R	1	2	3	4	5	6
Value	(R3)*(R5)	(R4)-(R3)	(R3)	(R4)-(R3)+(R3)	(R5)	-
Vld	1	1	1	1	1	0
RS	0	0	0	0	0	4

register status

```
mul Reg1, Reg3, Reg5
sub Reg2, Reg4, Reg3
div Reg6, Reg1, Reg4
add Reg4, Reg2, Reg3
```

reservation stations

		Empty	InFU	Op	Dest	Src1	Vld1	RS1	Src2	Vld2	RS2	
S_{add}	1	1	1	sub	2	(R4)	1	0	(R3)	1	0	
	2	1	1	add	4	(R4)-(R3)	1	0	(R3)	1	0	
S_{mul}	3	1	1	mul	1	(R3)	1	0	(R5)	1	0	
S_{div}	4	0	1	div	6	(R3)*(R5)	1	0	(R4)	1	0	3

RS status

cycle 9

token.tag
token.data

Superskalartechnik

Fallstudie: Tomasulo (IBM 360/91)

- Beispiel (T. Ungerer)

registers

R	1	2	3	4	5	6
Value	(R3)*(R5)	(R4)-(R3)	(R3)	(R4)-(R3)+(R3)	(R5)	-
Vld	1	1	1	1	1	0
RS	0	0	0	0	0	4

register status

```
mul Reg1, Reg3, Reg5
sub Reg2, Reg4, Reg3
div Reg6, Reg1, Reg4
add Reg4, Reg2, Reg3
```

reservation stations

		Empty	InFU	Op	Dest	Src1	Vld1	RS1	Src2	Vld2	RS2	
S_{add}	1	1	1	sub	2	(R4)	1	0	(R3)	1	0	
S_{add}	2	1	1	add	4	(R4)-(R3)	1	0	(R3)	1	0	
S_{mul}	3	1	1	mul	1	(R3)	1	0	(R5)	1	0	
S_{div}	4	0	1	div	6	(R3)*(R5)	1	0	(R4)	1	0	2

RS status

cycle 10

token.tag
token.data

Superskalartechnik

Fallstudie: Tomasulo (IBM 360/91)

- Beispiel (T. Ungerer)

```
mul Reg1, Reg3, Reg5
sub Reg2, Reg4, Reg3
div Reg6, Reg1, Reg4
add Reg4, Reg2, Reg3
```

registers

R	1	2	3	4	5	6
Value	(R3)*(R5)	(R4)-(R3)	(R3)	(R4)-(R3)+(R3)	(R5)	-
Vld	1	1	1	1	1	0
RS	0	0	0	0	0	4

register status

reservation stations

		Empty	InFU	Op	Dest	Src1	Vld1	RS1	Src2	Vld2	RS2	
S_{add}	1	1	1	sub	2	(R4)	1	0	(R3)	1	0	
S_{add}	2	1	1	add	4	(R4)-(R3)	1	0	(R3)	1	0	
S_{mul}	3	1	1	mul	1	(R3)	1	0	(R5)	1	0	
S_{div}	4	0	1	div	6	(R3)*(R5)	1	0	(R4)	1	0	1

RS status

cycle 11

token.tag

token.data

Superskalartechnik

Fallstudie: Tomasulo (IBM 360/91)

- Beispiel (T. Ungerer)

registers

R	1	2	3	4	5	6
Value	(R3)*(R5)	(R4)-(R3)	(R3)	(R4)-(R3)+(R3)	(R5)	(R3)*(R5) /(R4)
Vld	1	1	1	1	1	1
RS	0	0	0	0	0	0

register status

```
mul Reg1, Reg3, Reg5
sub Reg2, Reg4, Reg3
div Reg6, Reg1, Reg4
add Reg4, Reg2, Reg3
```

reservation stations

		Empty	InFU	Op	Dest	Src1	Vld1	RS1	Src2	Vld2	RS2	
S_{add}	1	1	1	sub	2	(R4)	1	0	(R3)	1	0	
S_{add}	2	1	1	add	4	(R4)-(R3)	1	0	(R3)	1	0	
S_{mul}	3	1	1	mul	1	(R3)	1	0	(R5)	1	0	
S_{div}	4	1	1	div	6	(R3)*(R5)	1	0	(R4)	1	0	0

RS status

cycle 12

token.tag 4

token.data (R3)*(R5) /(R4)

Superskalartechnik

Zusammenfassung

- Aus einem sequentiellen Befehlsstrom werden Befehle zur Ausführung angestoßen (zugewiesen).
- Die Zuweisung erfolgt dynamisch durch die Hardware
- Es kann mehr als ein Befehl zugewiesen werden.
- Die Anzahl der zugewiesenen Befehle pro Takt wird dynamisch von der Hardware bestimmt und liegt zwischen Null und der maximalen Zuweisungsbreite.
- Komplexe Hardware-Logik für dynamische Zuweisung notwendig.
- Mehrere von einander unabhängige Funktionsanweisungen sind verfügbar.
- Mikroarchitektur bestimmt superskalare Eigenschaft.

Superskalartechnik

Literatur:

- Brinkschulte/Ungerer: Mikrocontroller und Mikroprozessoren. Springer-Verlag, 2002: Kap. 6.1-6.4, Kap. 7
- Hennessy/Patterson: Computer Architecture – A Quantative Approach. 3. Auflage: Kap. 3

Vorlesung Rechnerstrukturen

Kapitel 2: Parallelismus auf Maschinenbefehlsebene

- 2.1 Pipelining
- 2.2 Nebenläufigkeit
  - VLIW

Vorlesung Rechnerstrukturen

Kapitel 2: Parallelismus auf Maschinenbefehlsebene

- 2.1 Pipelining
- 2.2 Nebenläufigkeit
  - VLIW

Parallelismus auf Maschinenbefehlsebene

VLIW (Very Long Instruction Word)

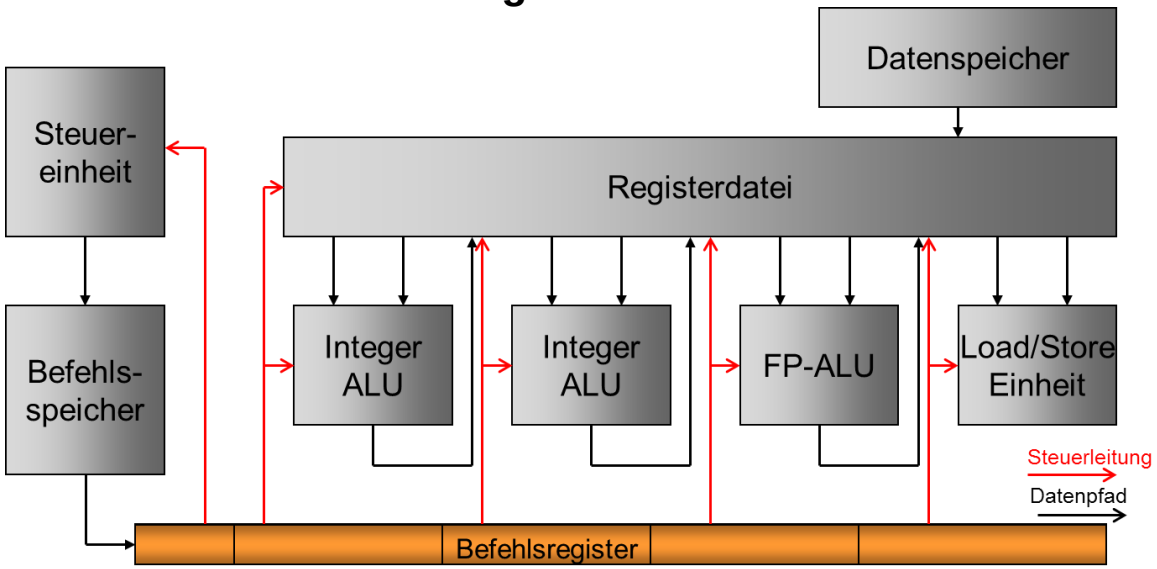
- **Grundprinzip**
  - Eine **VLIW-Architektur** (Very Long Instruction Word) ist gekennzeichnet durch ein breites Befehlsformat, das in mehrere Felder aufgeteilt ist, aus denen die Funktionseinheiten gesteuert werden;
  - Eine VLIW-Architektur mit *n* voneinander unabhängigen Funktionseinheiten kann bis zu *n* Operationen gleichzeitig ausführen;
  - Operationen: RISC-ähnliche Befehlssatzarchitekturr
  - Steuerung der parallelen Abarbeitung zur Übersetzungszeit (**automatisch parallelisierender Compiler**)

Parallelismus auf Maschinenbefehlsebene

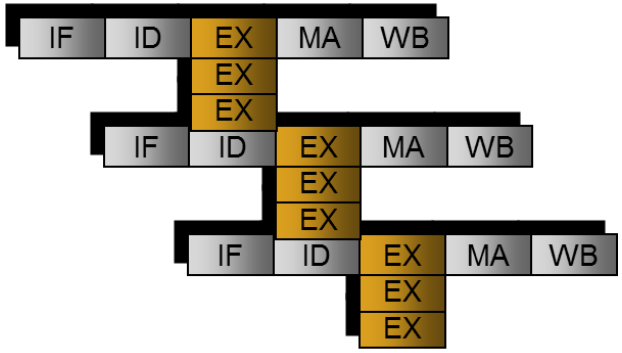
VLIW (Very Long Instruction Word)

- **Grundprinzip**

Blockdiagramm



Pipelinestruktur



Parallelismus auf Maschinenbefehlsebene

VLIW (Very Long Instruction Word)

- **Statische Steuerung der parallelen Abarbeitung**
  - **Aufgaben des Compilers**
  - **Frontend**:
    - Lexikalische, syntaktische und semantische Analyse
  - **Code-Generierung / Parallelisierung**
    - Kontrollflussanalyse
    - Datenflussanalyse
    - Datenabhängigkeitsanalyse
  - **Schleifenparallelisierung**
    - Loop Unrolling
    - Software-Pipelining
  - **Scheduling**
    - Packen der voneinander unabhängigen Befehle in breite Befehlswörter

Parallelismus auf Maschinenbefehlsebene

VLIW (Very Long Instruction Word)

- **Scheduling**
  - Beispiel (Quelle: K. Asanovic, MIT)

```
for (i=0;i<N;i++)
B[i]=A[i]+C
```

Übersetzung ↓

```
Loop:  ld f1,0(r1)
       add r1,8
       fadd f2,f0,f1
       sd f2, 0(r2)
       add r2,8
       bne r1,r3,loop
```

Parallelismus auf Maschinenbefehlsebene

VLIW (Very Long Instruction Word)

- **Scheduling**
  - Beispiel (Quelle: K. Asanovic, MIT)

```
for (i=0;i<N;i++)
B[i]=A[i]+C
```

Übersetzung

Scheduling

FE Int1	FE Int2	FE Mem1	FE Mem2	FE FP+	FE FPx
add r1		ld			
				fadd	
add r2	bne	sd			

Parallelismus auf Maschinenbefehlsebene

VLIW (Very Long Instruction Word)

- **Scheduling**
  - Beispiel (Quelle: K. Asanovic, MIT)
  - **Loop Unrolling**

```
for (i=0;i<N;i++)
B[i]=A[i]+C
```

Abrollen der inneren Schleife

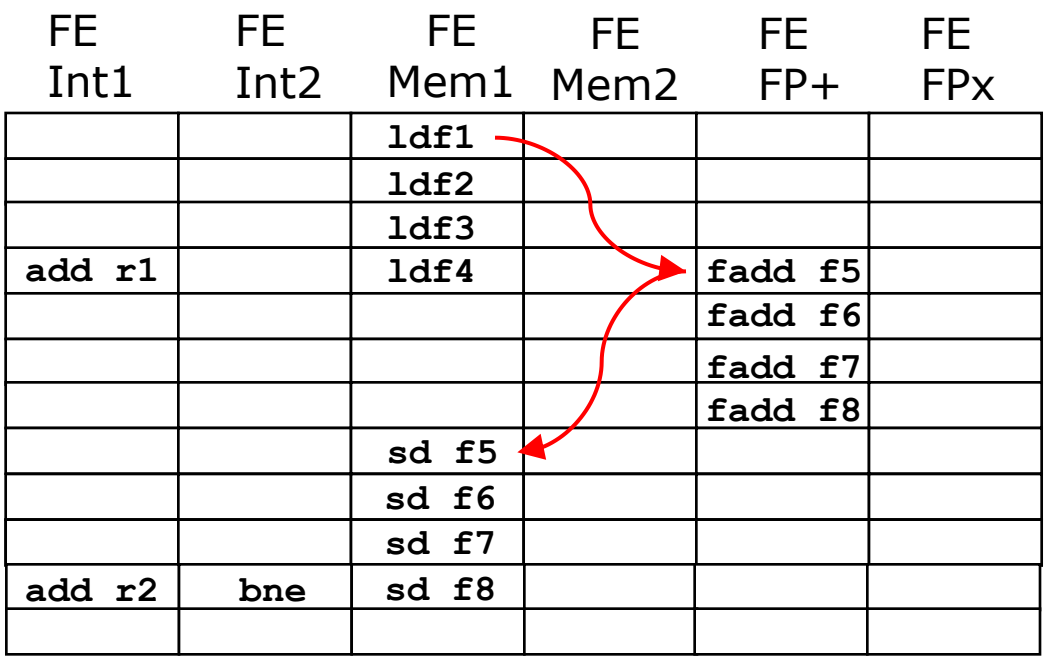
```
for (i=0;i<N-3;i+4)
{
B[i]=A[i]+C
B[i+1]=A[i+1]+C
B[i+2]=A[i+2]+C
B[i+3]=A[i+3]+C
}
```

Parallelismus auf Maschinenbefehlsebene

VLIW (Very Long Instruction Word)

- **Scheduling**
  - Beispiel (Quelle: K. Asanovic, MIT)
  - **Loop Unrolling**

```
Loop:   ld f1,0(r1)
        ld f2,8(r1)
        ld f3,16(r1)
        ld f4,24(r1)
        add r1,32
        fadd f5,f0,f1
        fadd f6,f0,f2
        fadd f7,f0,f3
        fadd f8,f0,f4
        sd f5,0(r2)
        sd f6,8(r2)
        sd f7,16(r2)
        sd f8,24(r2)
        add r2,32
        bne r1,r3,loop
```



Parallelismus auf Maschinenbefehlsebene

VLIW (Very Long Instruction Word)

- **Scheduling**
  - Beispiel (Quelle: K. Asanovic, MIT)
  - **Loop Unrolling**

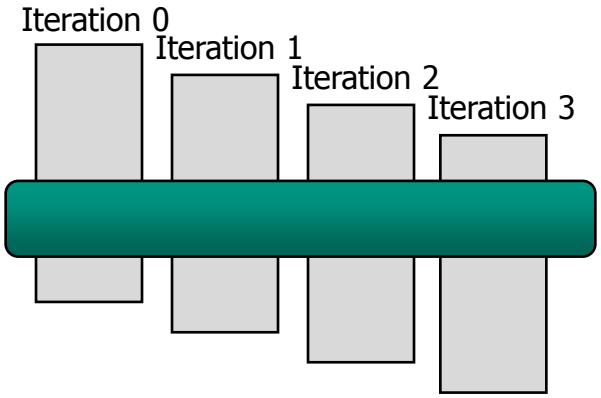
```
Loop:   ld f1,0(r1)
        ld f2,8(r1)
        ld f3,16(r1)
        ld f4,24(r1)
        add r1,32
        fadd f5,f0,f1
        fadd f6,f0,f2
        fadd f7,f0,f3
        fadd f8,f0,f4
        sd f5,0(r2)
        sd f6,8(r2)
        sd f7,16(r2)
        add r2,32
        sd f8,-8(r2)
        bne r1,r3,loop
```

	FE Int1	FE Int2	FE Mem1	FE Mem2	FE FP+	FE FPx
Prolog			ldf1			
			ldf2			
			ldf3			
	add r1		ldf4			
			ldf1		fadd f5	
			ldf2		fadd f6	
			ldf3		fadd f7	
	add r1		ldf4		fadd f8	
Schleife			ldf1	sd f5	fadd f5	
			ldf2	sd f6	fadd f6	
		add r2	ldf3	sd f7	fadd f7	
	add r1	bne	ldf4	sd f8	fadd f8	
Epilog				sd f5	fadd f5	
				sd f6	fadd f6	
				sd f7	fadd f7	
				sd f8	fadd f8	
				sd f5		

Parallelismus auf Maschinenbefehlsebene

VLIW (Very Long Instruction Word)

- **Scheduling**
  - **Software-Pipelining**
    - Technik zur Reorganisation von Schleifen
    - Jede Iteration in dem mit Software-Pipelining generierten Code enthält Befehle aus verschiedenen Iterationen der ursprünglichen Schleife



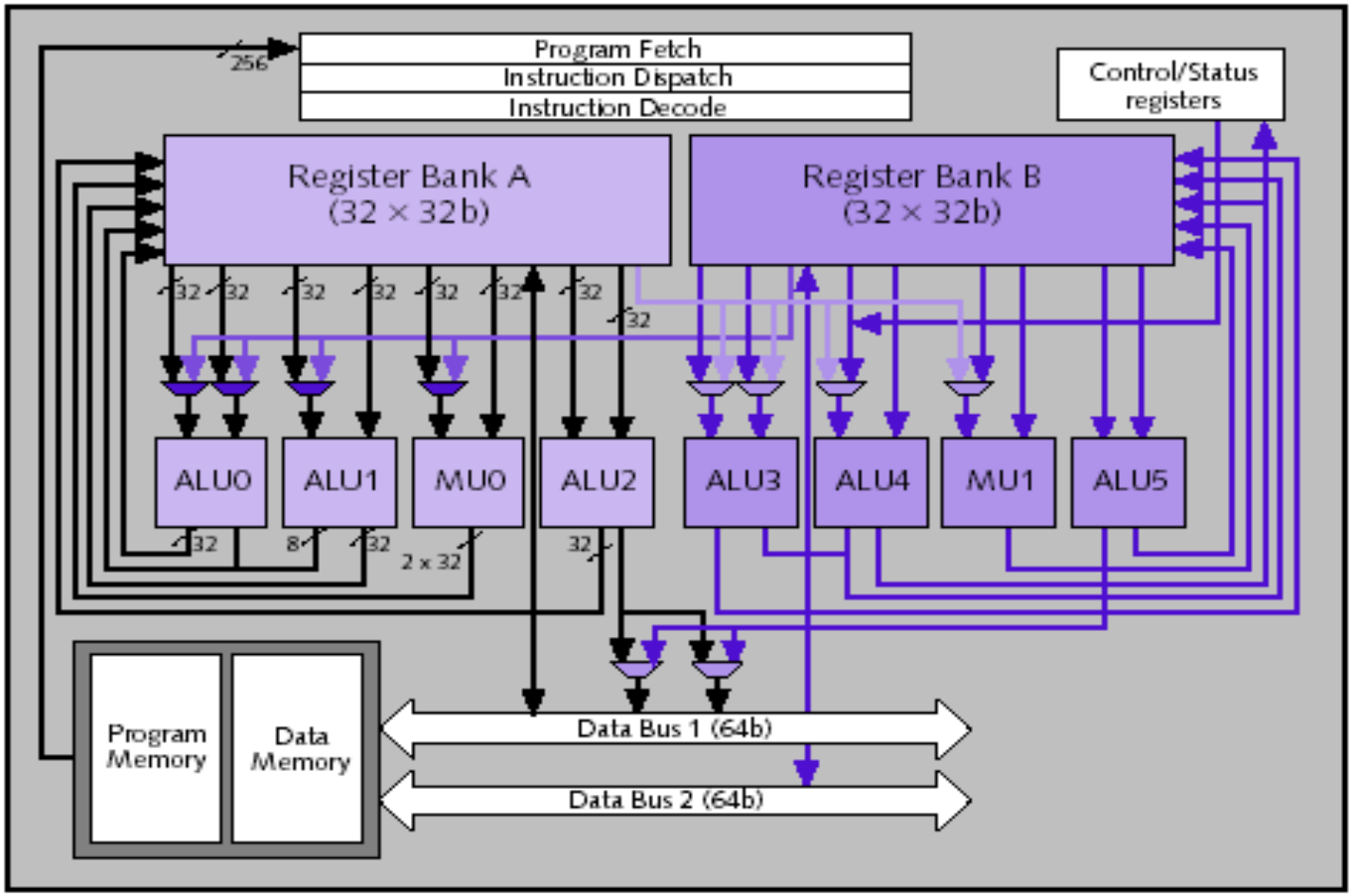
Parallelismus auf Maschinenbefehlsebene

VLIW (Very Long Instruction Word)

- **Frühe VLIW-Rechner**
  - Multiflow Trace (J. Fisher)
    - Rechnerkonfigurationen mit 7, 14 und 28 Operationen/VLIW-Instruktion
    - 28 Operationen in einem 1024 Bit Wort
    - Numerische Anwendungen
    - Trace-Scheduling
  - Cydrome Cydra-5 (B. Rau)
    - 7 Operationen/256Bit Wort
    - Rotating Register File
- **Einsatz VLIW-Technik heute:**
  - Vorwiegend im Bereich eingebetteter Prozessoren (DSP)
    - Beispiel: Trimedia TM32 Architecture
    - Agere/Motorola Star Core
  - Allzweck-Mikroprozessoren
    - Transmeta Crusoe (bis etwa 2005)

Parallelismus auf Maschinenbefehlsebene

Fallstudie TI TMS320C6400



Parallelismus auf Maschinenbefehlsebene

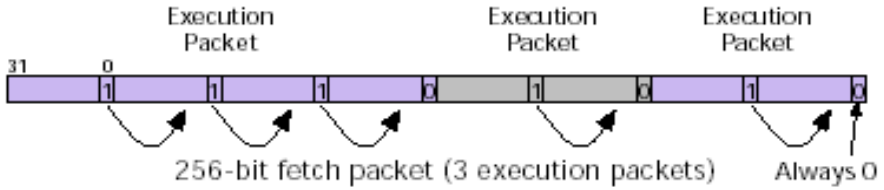
Fallstudie TI TMS320C6400

- **Architekturmerkmale**
  - 2 x 4 Funktionseinheiten (A und B Seite)
    - Jede Seite enthält 16 Register
    - Programmierer sieht 32 Register A0 – A15, B0 – B15
    - Ausgewählte Register für Boole‘sche Ergebnisse von bedingten Befehlen
    - 9 32 Bit Lese- und 6 32 Bit Schreibports
    - Jeweils ein Crossover-Pfad: Beschränkter wechselseitiger Zugriff
  - Jede Seite enthält
    - Eine 40 Bit Integer-ALU (L Unit)
      - Arithmetische und logische Operationen, Vergleiche, Normalisierung, Bit-Count,
    - 16 Bit Multiplizierer
      - 16 x 16 → 32 Bit Multiplikation
    - 40 Bit Schiebeeinheit
    - 32 Bit Addierer
      - Adressgenerierung

Parallelismus auf Maschinenbefehlsebene

Fallstudie TI TMS320C6400

- **Operationsprinzip**
  - Holen von 8 32 Bit Befehlen über 256 Bit Befehlsbus (Fetch Packet)
    - Geholte Befehle müssen nicht unbedingt gleichzeitig ausgeführt werden
    - Ein Befehl in einem Fetch Packet ist nicht auf eine Ausführungseinheit beschränkt, jeder Befehl enthält Kodierung, mit der spezifiziert wird, auf welche Einheit der Befehl ausgeführt werden soll
    - Befehle sind nicht positionsabhängig
    - Programmierer / Compiler bestimmt Bindung
  - Bis zu 8 Befehle können gleichzeitig ausgeführt werden: Gruppierung von gleichzeitig ausführbaren Befehlen (Execution Packets)
    - Wird im niedrigstwertigen Bit eines Feldes gekennzeichnet: alle nachfolgenden Befehle werden gleichzeitig ausgeführt



VLIW-Prinzip

- Literatur:
  - Hennessy/Patterson: Computer Architecture A Quantative Approach. Kap. 4.1-4.4, 4.8

Vorlesung Rechnerstrukturen

Kapitel 2: Parallelismus auf Maschinenbefehlsebene

- 2.1 Pipelining
- 2.2 Nebenläufigkeit
- 2.3 Mehrfädigkeit

Mehrfädigkeit (Multithreading)

Grundsätzliche Aufgabe beim Prozessorentwurf:

- Reduzierung der Untätigkeits- oder Latenzzeiten
  - Entstehen bei Speicherzugriffen, insbesondere bei Cache-Fehlzugriffen

Mehrfädigkeit (Multithreading)

Mehrfädige Prozessortechnik

- Gegeben mehrere ausführbereite **Kontrollfäden, Threads**
  - **Ziel**: Parallele Ausführung mehrerer Kontrollfäden
  - **Prinzip**
    - Mehrere Kontrollfäden sind geladen
    - Kontext muss für jeden Thread gesichert werden können
    - Mehrere getrennte Registersätze auf Prozessorchip
    - Mehrere Befehlszähler
    - Getrennte Seitentabellen
    - Threadwechsel, wenn gewartet werden muss

Mehrfädigkeit (Multithreading)

Mehrfädige Prozessortechnik

- **Cycle-by-cycle Interleaving** (**feingranulares Multithreading**)
  - Eine Anzahl von Kontrollfäden ist geladen.
  - Der Prozessor wählt in jedem Takt einen der ausführungsbereiten Kontrollfäden aus.
  - Der nächste Befehle in der Befehlsreihenfolge des ausgewählten Kontrollfadens wird zur Ausführung ausgewählt.

  - Beispiele
    - Multiprozessorsysteme HEP, Tera
  - Nachteil:
    - Die Verarbeitung eines Threads kann erheblich verlangsamt werden, wenn er ohne Wartezeiten ausgeführt werden kann

Mehrfädigkeit (Multithreading)

Mehrfädige Prozessortechnik

- **Block Interleaving**
  - Befehle eines Kontrollfadens werden so lange ausgeführt, bis eine Instruktion mit einer langen Latenzzeit ausgeführt wird. Dann wird zu einem anderen ausführbaren Kontrollfaden gewechselt.
  - Vorteil:
    - Die Bearbeitung eines Threads wird nicht verlangsamt, da beim Warten ausführungsbereiter Thread gestartet wird
  - Nachteil:
    - Bei Thread-Wechsel Leeren und Neustarten der Pipeline,
    - Nur bei langen Wartezeiten sinnvoll

Mehrfädigkeit (Multithreading)

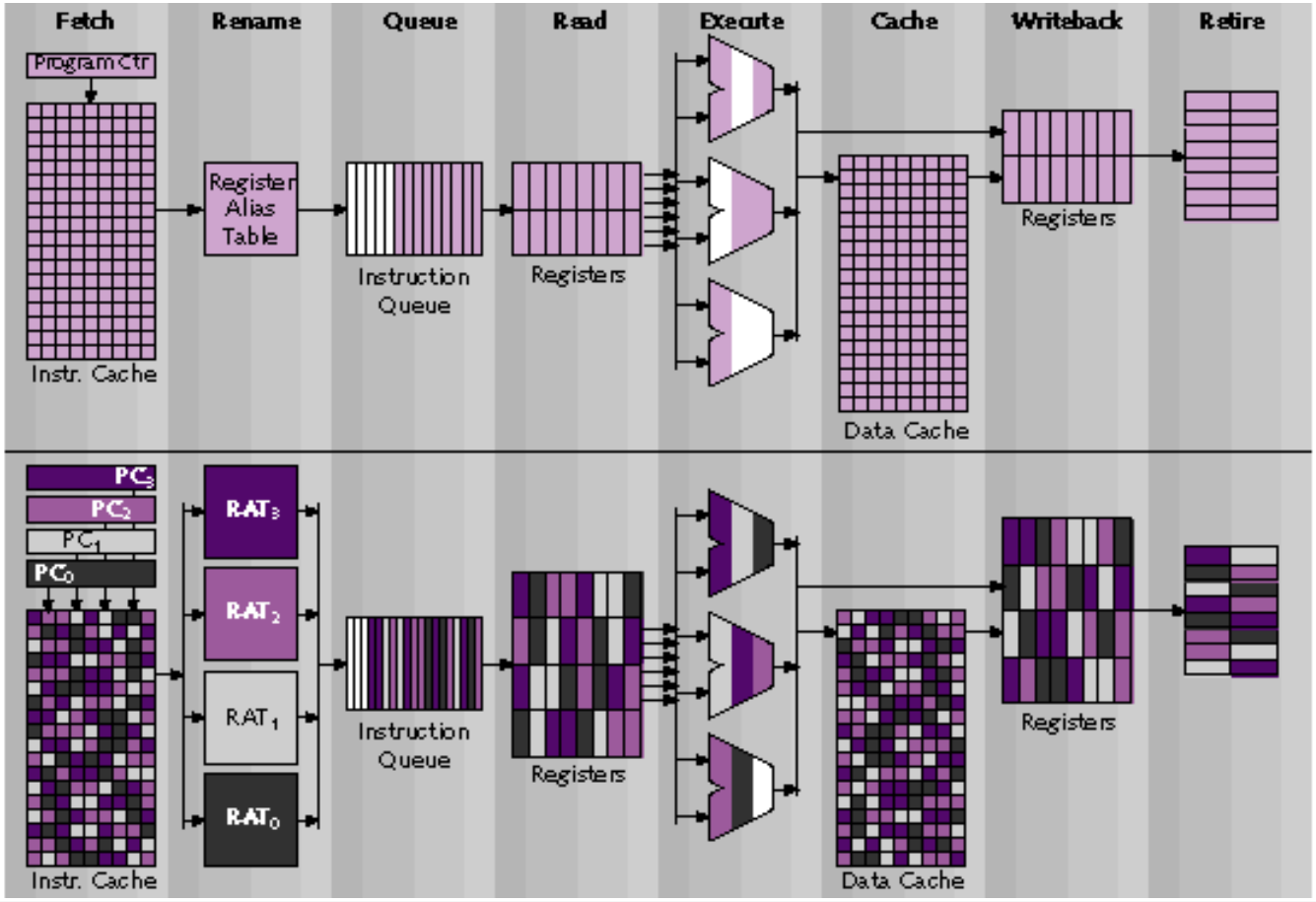
Mehrfädige Prozessortechnik

- **Simultaneous Multithreading**
  - Mehrfach superskalarer Prozessor
  - Die Ausführungseinheiten werden über eine Zuordnungseinheit aus mehreren Befehlspuffern versorgt.
  - Jeder Befehlspuffer stellt einen anderen Befehlsstrom dar.
  - Jedem Befehlsstrom ist eigener Registersatz zugeordnet.

Mehrfädigkeit (Multithreading)

Mehrfädige Prozessortechnik

- **Simultaneous Multithreading**



Mehrfädigkeit (Multithreading)

Mehrfädige Prozessortechnik

- **Simultaneous Multithreading: Diskussion**
  - Abwägen zwischen Geschwindigkeit eines Threads und dem Durchsatz vieler Threads
  - Ein bevorzugter Thread
  - Allerdings kann dies auf Kosten des Durchsatzes gehen, da Befehle anderer Threads möglicherweise nicht bereit stehen
  - Mischen vieler Threads:
  - Geht möglicherweise zu Lasten der Leistung der einzelnen Threads

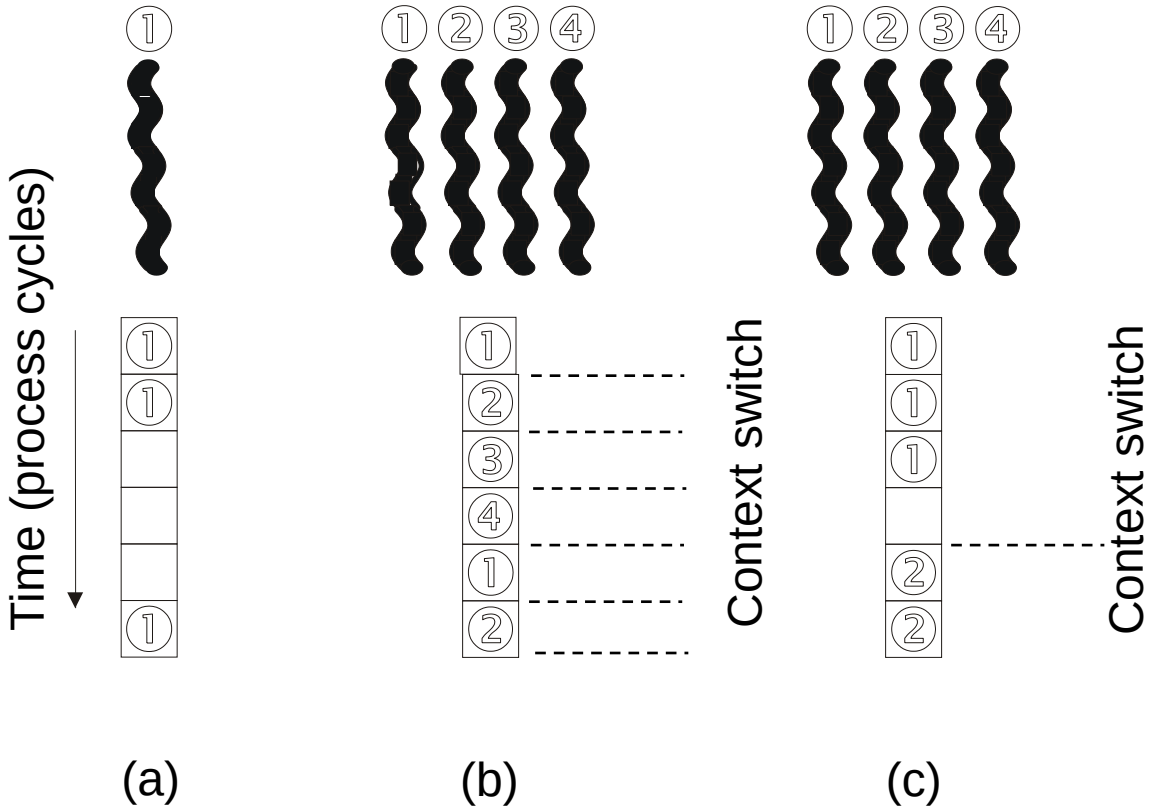
Mehrfädigkeit (Multithreading)

Mehrfädige Prozessortechnik

- **Simultaneous Multithreading: Beispiele**
  - **Compaq Alpha 21464** (EV8), ursprünglich angekündigt für 2002/2003, Entwicklung aber eingestellt! Entwicklergruppe teilweise zu Intel
  - Intel P4: **Hyperthreading**
  - **Sun Ultra SPARC IV**: Chip Multithreading

Mehrfädigkeit (Multithreading)

Vergleich von Prozessortechniken



(a): single-threaded scalar

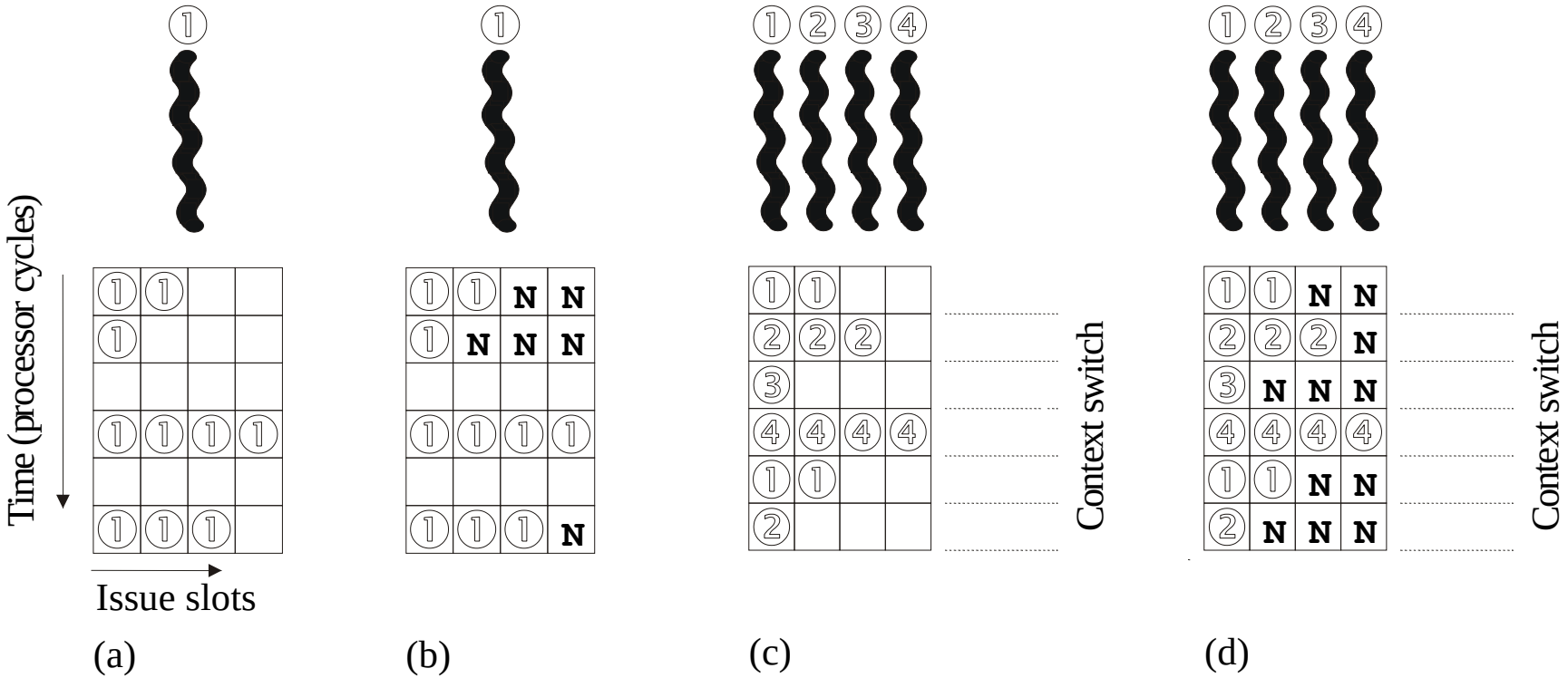
(b) cycle-by-cycle interleaving multithreaded scalar

(c) block interleaving multithreaded scalar

(siehe Brinkschulte, Ungerer: Mikrocontroller und Mikroprozessoren: Kap. 10.4.3)

Mehrfädigkeit (Multithreading)

Vergleich von Prozessortechniken

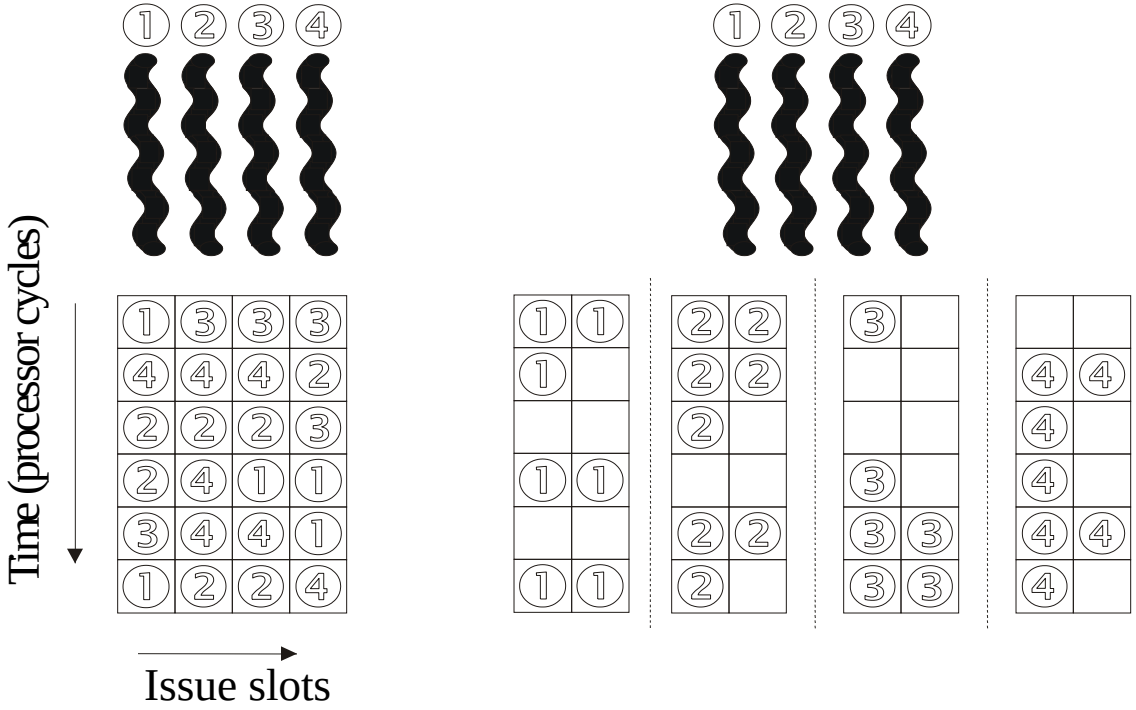


(a) Suparskalare Technik
(b) VLIW-Technik
(c) Cycle-by-cycle Interleaving
(d) Cycle-by-cycle VLIW

(siehe Brinkschulte, Ungerer: Mikrocontroller und Mikroprozessoren: Kap. 10.4.3)

Mehrfädigkeit (Multithreading)

Vergleich von Prozessortechniken



(a) Simultaneous Multithreading
(b) Single Chip Multiprocessing

(siehe Brinkschulte, Ungerer: Mikrocontroller und Mikroprozessoren: Kap. 10.4.3)

Multithreading

- **Literatur:**
  - Brinkschulte, U.; Ungerer, T.: Microcontroller und Mikroprozessoren. Springer, Heidelberg, 2002: Kap.: 10.4.3