Rechnerstrukturen

Vorlesung im Sommersemester 2007

Prof. Dr. Wolfgang Karl

Universität Karlsruhe (TH)

Fakultät für Informatik

Institut für Technische Informatik

- **Kapitel 2: Parallelismus auf Befehlsebene**

2.1: Pipelining

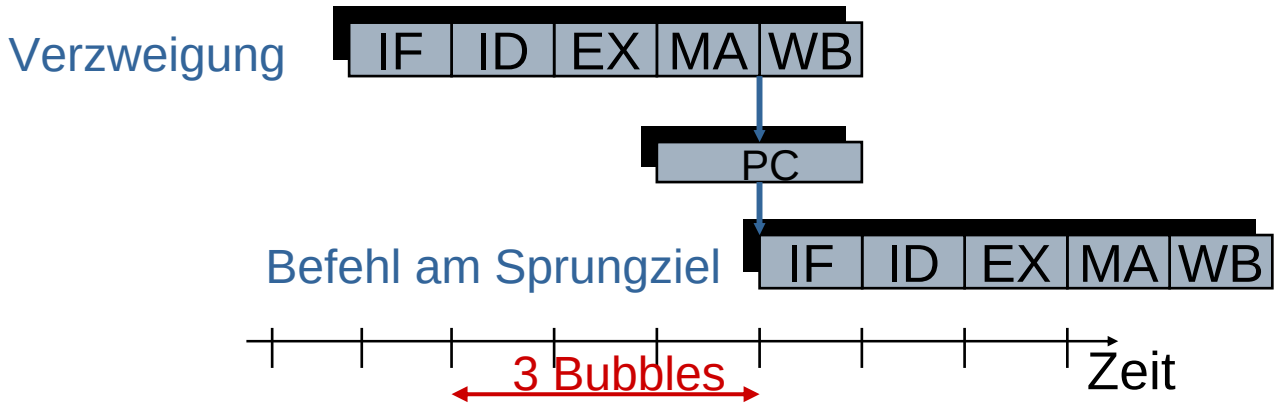
- Pipeline-Konflikte (Pipeline Hazards, Pipeline-Hemmnisse)
  - Situationen, die verhindern, dass die nächste Instruktion im Befehlsstrom im zugewiesenen Taktzyklus ausgeführt wird
    - Unterbrechung des taktsynchronen Durchlaufs durch die einzelnen Stufen der Pipeline
  - Verursachen Leistungseinbußen 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

- Pipeline-Konflikte
  - Steuerflusskonflikte
    - Lösung des Konflikts: Einfügen von Verzögerungsphasen
    - Problem: Vermeiden langer Wartezeiten
      - Möglichst frühe Auswertung der Bedingung und frühe Berechnung des Sprungziels: ID Phase ist geeignet
      - Aber
        - » Strukturkonflikt: ALU kann nicht für Berechnung des Sprungziels verwendet werden, weshalb eine zusätzliche ALU zur Sprungzielberechnung in ID-Phase notwendig ist.
        - » Datenabhängigkeit: zwischen arithmetischer Operation und nachfolgender Verzweigung; Konflikt mit Verzögerung
        - » Dekodieren, Zieladresse berechnen und PC schreiben sind in einer Pipeline-Stufe: Kritischer Pfad in der Dekodierphase, d.h. Verlängerung der Zykluszeit!

- Pipeline-Konflikte
  - Steuerflusskonflikte
    - Lösung des Konflikts: Einfügen von Verzögerungsphasen
    - Problem: Vermeiden langer Wartezeiten
      - Mit einer Pipeline-Reorganisation wie beschrieben bleibt eine Verzögerungsphase!

      ➔Lösung: Statische und dynamische Techniken zur Konfliktauflösung!

- Pipeline-Konflikte
  - Steuerflusskonflikte
    - Dynamische Technik zur Auflösung von Konflikten
      - Pipeline-Interlock
        - » Hardware zum Erkennen einer Verzweigung
        - » Hardware-Interlocking zum Anhalten des der Verzweigung folgenden Befehls



- Pipeline-Konflikte
  - Steuerflusskonflikte
    - 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

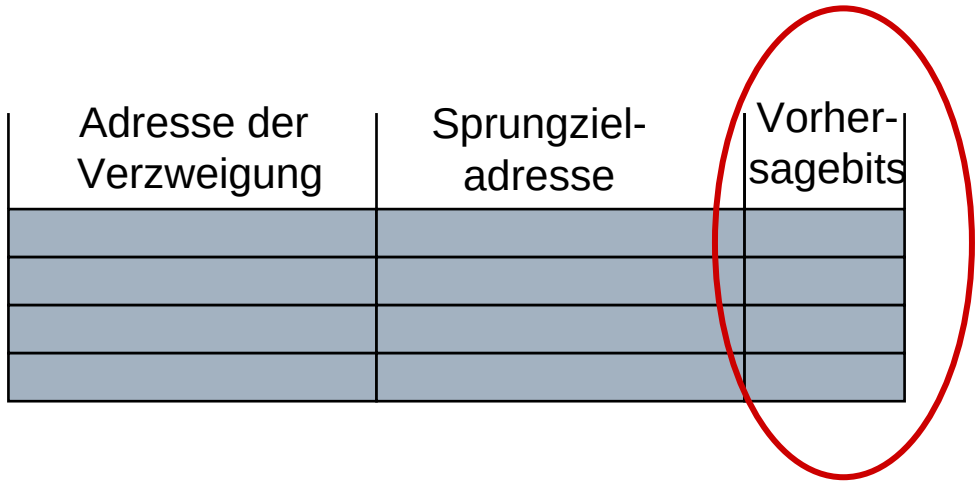
- Pipeline-Konflikte
  - Sprungvorhersage
    - Statische Sprungvorhersage
      - Die Richtung der Vorhersage ist für einen Befehl immer gleich
    - Dynamische Sprungvorhersage
      - Die Vorhersage hängt von der Vorgeschichte ab.

- Pipeline-Konflikte
  - Sprungvorhersage (Branch Prediction)
    - Dynamische Vorhersage
      - Berücksichtigung des Programmverhaltens
        - » Die Richtung hängt von der Vorgeschichte der Verzweigung ab
      - Genauere Vorhersage möglich
      - Hoher Hardware-Aufwand!

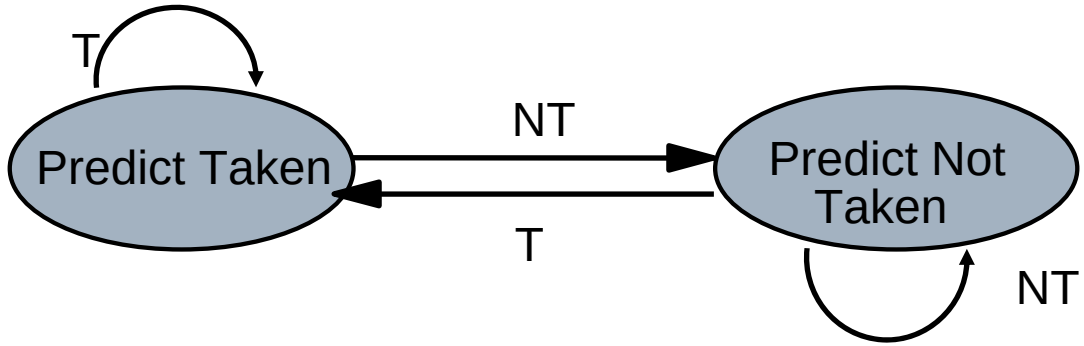
- Pipeline-Konflikte
  - Sprungvorhersage (Branch Prediction)
    - Dynamische Vorhersage
      - Sprungziel-Cache: Branch Target Address Cache (BTAC), Branch Target Buffer (BTB)
        - » Speichert die Adresse der Verzweigung und das entsprechende Sprungziel
        - » Steht in Verbindung mit IF-Phase

Adresse der Verzweigung	Sprungziel-adresse

- Pipeline-Konflikte
  - Sprungvorhersage (Branch Prediction)
    - Dynamische Vorhersage
      - Sprungziel-Cache: Branch Target Address Cache (BTAC), Branch Target Buffer (BTB)
      - In Verbindung mit Branch Prediction Buffer, Branch History Table (BHT)
        - » Weiteres Feld für Vorhersagebit

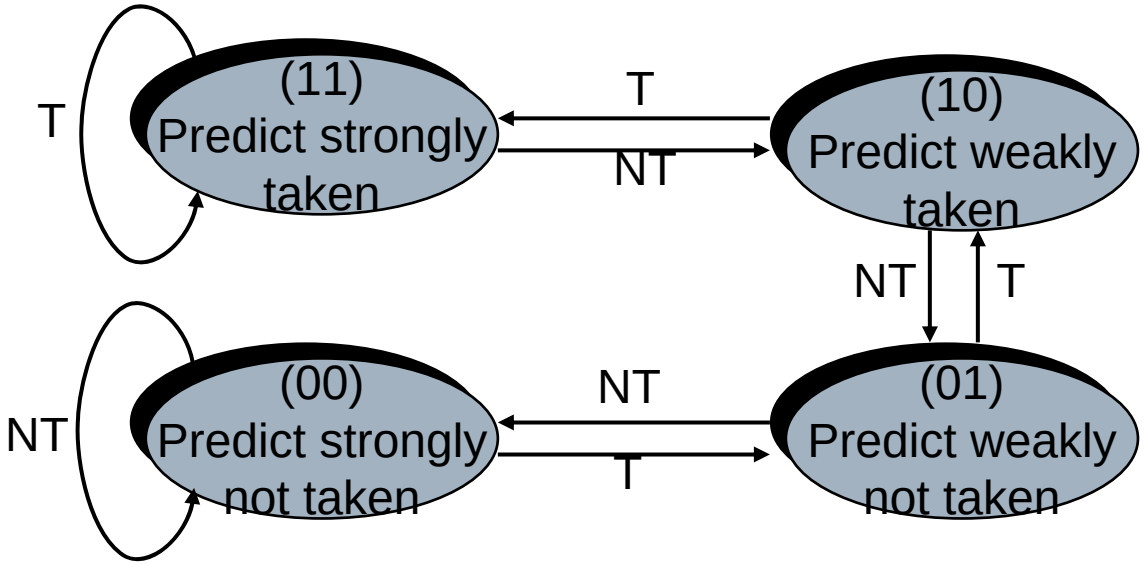


- Pipeline-Konflikte
  - Sprungvorhersage (Branch Prediction)
    - Branch Prediction Buffer, Branch History Table
      - Vorhersagebit:
        - » Wenn das Bit gesetzt ist, wird angenommen, dass der Sprung ausgeführt wird.
        - » Wenn das Bit nicht gesetzt ist, wird angenommen, dass der Sprung nicht ausgeführt wird.
        - » Bei einer Fehlannahme: Invertieren des Bits

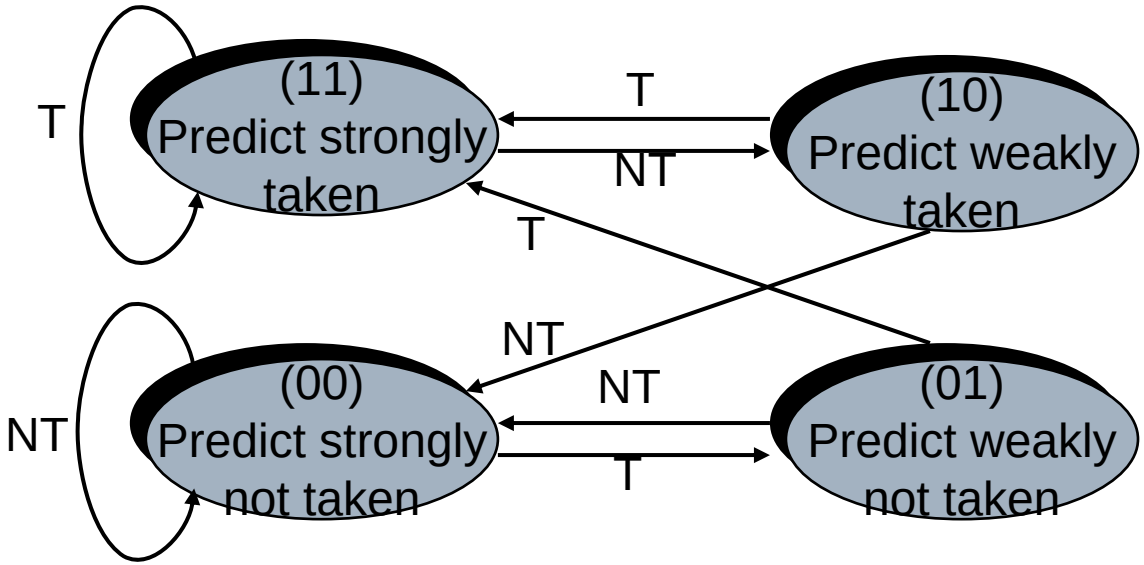


- Pipeline-Konflikte
  - Sprungvorhersage (Branch Prediction)
    - Branch Prediction Buffer, Branch History Table: Zwei-Bit Predictor
      - Zwei Bit pro Eintrag für die Kodierung der Vorhersage → vier Zustände:
        - » *Sicher genommen (stronly taken)*
        - » *Vielleicht genommen (weakly 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.

- Pipeline-Konflikte
  - Sprungvorhersage (Branch Prediction)
    - Branch Prediction Buffer, Branch History Table: Zwei-Bit Predictor mit Sättigungszähler (Two Bit Predictor with Saturation Scheme)



- Pipeline-Konflikte
  - Sprungvorhersage (Branch Prediction)
    - Branch Prediction Buffer, Branch History Table: Zwei-Bit Predictor mit Hysteresemethode (Two Bit Predictor with Hysteresis Scheme)



- Pipeline-Konflikte
  - Sprungvorhersage (Branch Prediction)
    - Branch Prediction Buffer, Branch History Table: Zwei-Bit Predictor
      - Erweiterbar auf n Bit
        - » Experimente haben gezeigt, dass kaum Verbesserungen erzielbar sind.
      - Implementierbar im Branch Target Address Cache
      - Neben BTAC Verwendung einer Branch History Table als Vorhersagetabelle

- Pipeline-Konflikte
  - Sprungvorhersage (Branch Prediction)
    - Branch Prediction Buffer, Branch History Table: Zwei-Bit Predictor
      - Fehlannahmen:
        - » Falsche Annahme für Verzweigung
        - » Durch die Indizierung wurde die Vergangenheit eines anderen Sprungbefehls betrachtet.
      - Gute Vorhersagen in technisch-wissenschaftlichen Programmen (Schleifen).
      - Hohe Fehlannahmerate bei Programmen, in denen die Sprünge miteinander in Beziehung stehen.

      ➔Führen zu komplexen Sprungvorhersagetechniken!

- Pipeline-Konflikte
  - Sprungvorhersage (Branch Prediction)
    - Zusammenfassung
      - Statische Vorhersage
        - » Hardware- oder Compiler-Techniken
      - Dynamische Vorhersage
        - » Berücksichtigung der Vorgeschichte
        - » Hohe Genauigkeit erreichbar
        - » Hoher Hardware-Aufwand
        - » Für superskalare Prozessoren ist eine möglichst genaue Vorhersagetechnik notwendig: komplexe Techniken

- Literatur:
  - Patterson/Hennessy: Rechnerorganisation und –entwurf - Die Hardware/Software-Schnittstelle. Deutsche Ausgabe herausgegeben von Arndt Bode, Wolfgang Karl, Theo Ungerer; Spektrum Akademischer Verlag, Heidelberg, 2005:
    - Kapitel 6
  - Binkschulte/Ungerer: Microcontroller und Mikroprozessoren. Springer-Verlag, Heidelberg, 2002:
    - Kapitel 2, insbesondere Abschnitte 2.3 und 2.4

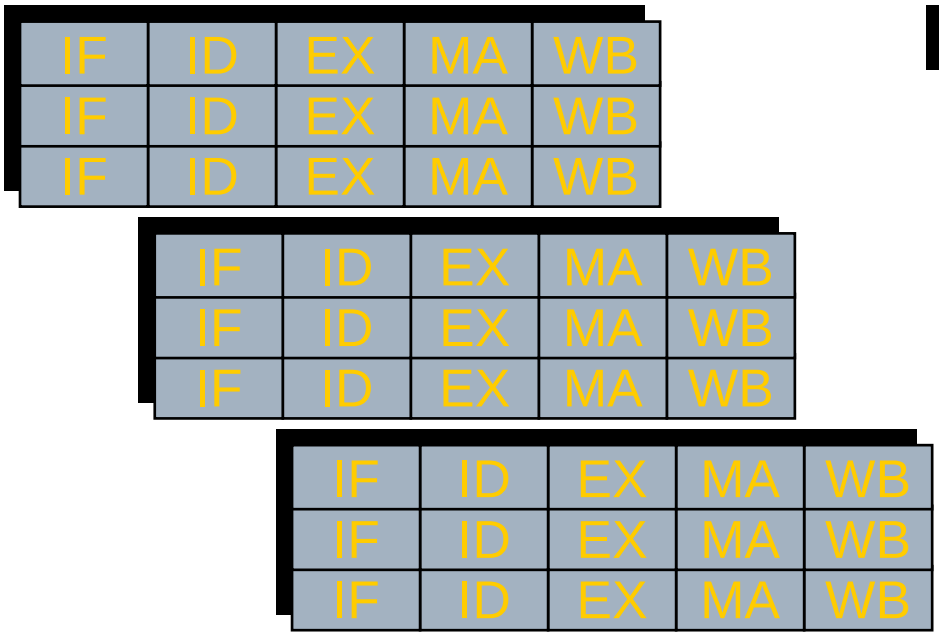
- **Kapitel 2: Parallelismus auf Befehlsebene**

2.1: Nebenläufigkeit, Superskalartechnik

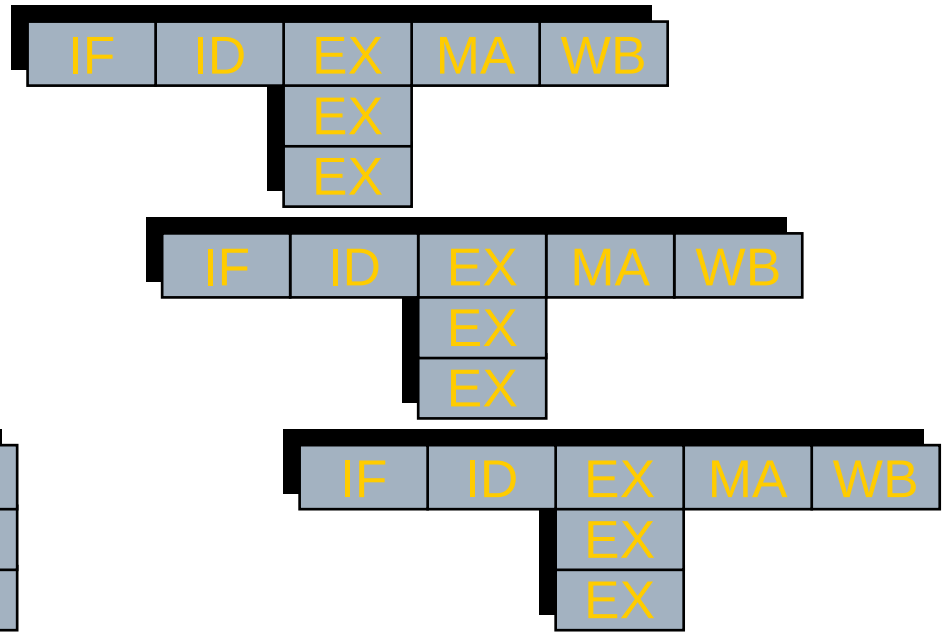
- Parallelismus auf Befehlsebene
  - Überblick
    - Nebenläufigkeit
      - Zu einem Zeitpunkt gleichzeitige Ausführung mehrerer Maschinenbefehle zu
      - Dynamische Ansätze
        - » Superskalare Mikroprozessoren
      - Statische Ansätze
        - » VLIW, EPIC

- Parallelismus auf Befehlsebene
  - Überblick
    - Nebenläufigkeit
      - Anzahl Befehle, die gleichzeitig zur Ausführung angestoßen werden (Issue parallelism IP): IP = n Befehle pro Zyklus

Dynamische Ansätze:
Superskalartechnik



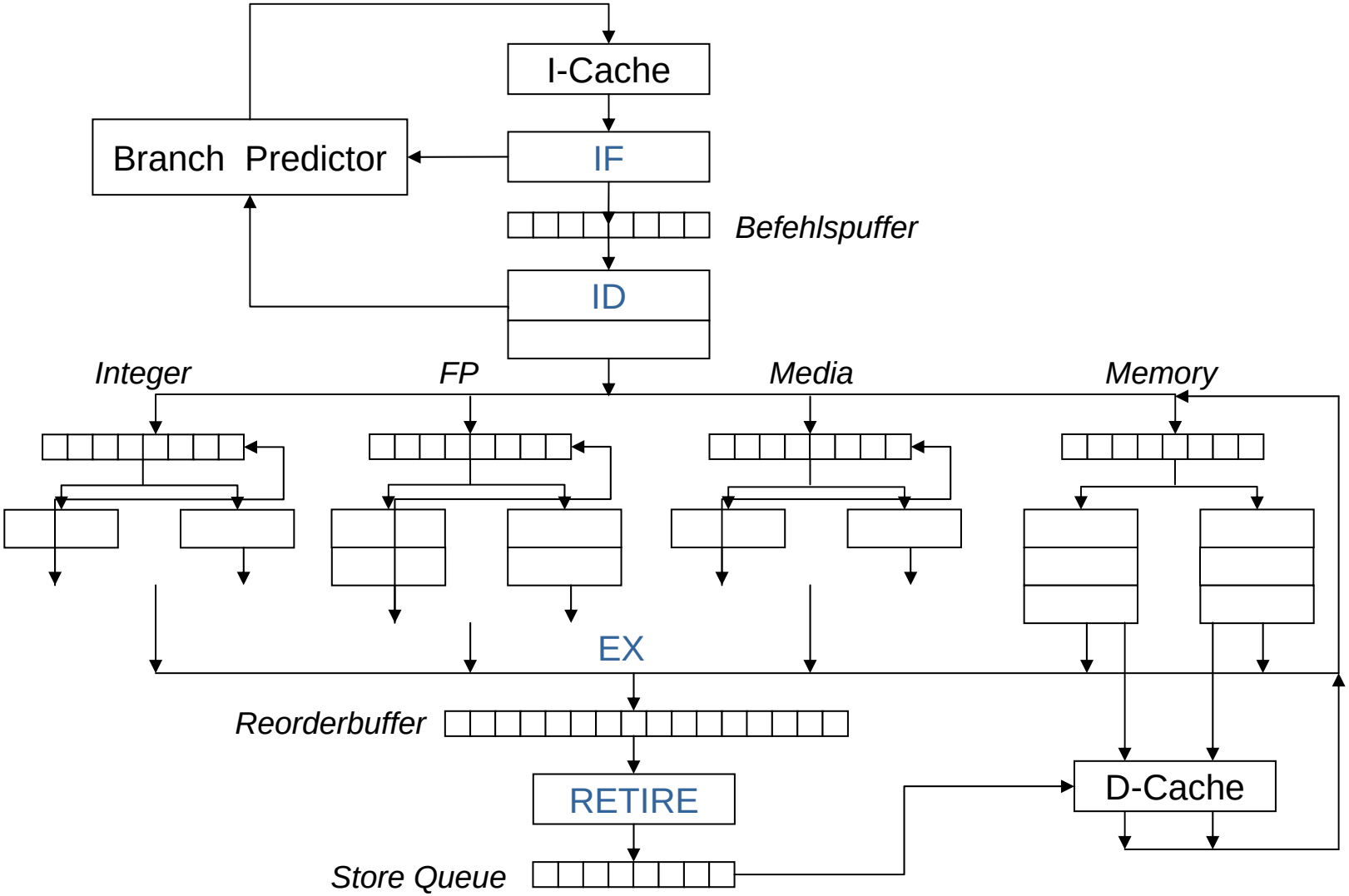
Statische Ansätze:
VLIW-Technik



- RISC → Superskalar
  - Mehrfachzuweisungsmethoden (multiple issue)
    - Die Superskalar-Technik ermöglicht es heute, 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 Befehlsebenenparallelität durch die Pipelining- und Superskalartechnik

- 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

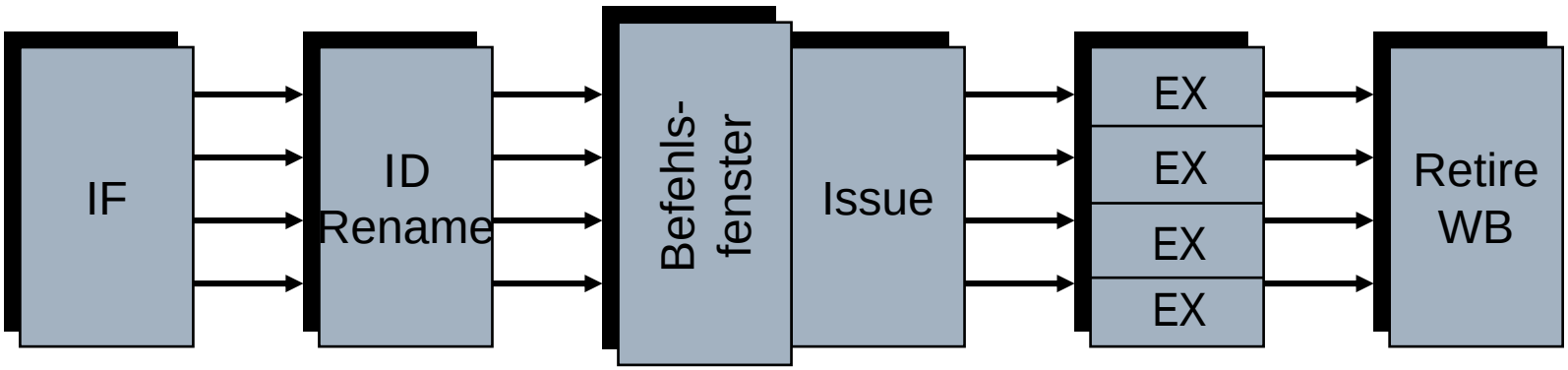
- Superskalarer Prozessor



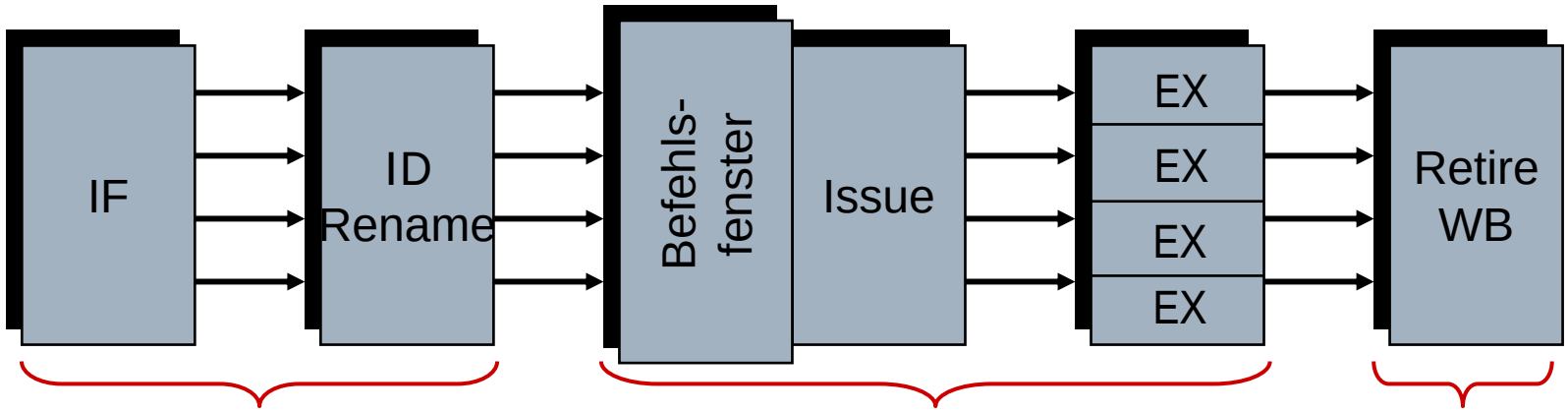
- Superskalarer Prozessor
  - Komponenten
    - Befehlsholeinheit (Instruction Fetch)
    - Dekodiereinheit (Instruction Decode) mit Registerumbenennung (Register renaming)
    - Zuordnungseinheit (Instruction Issue)
    - Unabhängige Verarbeitungseinheiten (Functional Units)
    - Rückordnungseinheit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

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

- Superskalare Prozessor-Pipeline
  - Mehrere Maschinenbefehle werden gleichzeitig geholt, dekodiert und ausgeführt; Rückschreiben der Ergebnisse
  - Zusätzlich:
    - Zuordnungstufe
    - Rückordnungsstufe
    - Weitere Puffer zur Entkopplung der Pipelinestufen



- Superskalare Prozessor-Pipeline
  - Ausführen der Befehle unabhängig ihrer Reihenfolge im Programm
    - Unter Beachtung der Ressourcenbeschränkungen werden zu einem Zeitpunkt alle Befehle ausgeführt, deren Operanden verfügbar sind
    - Ausführung erfolgt gemäß Datenfluss und nicht gemäß dem vom Programm vorgegebenen Programmfluss



- Superskalare Prozessor-Pipeline
  - 1. In-order-Abschnitt
    - Befehle werden entsprechend ihrer Programmordnung bearbeitet
    - Umfasst die Befehlshol- und Dekodierphase
    - Zuordnungsstufe
      - bei Zuordnung in Programmreihenfolge
        - » Dynamische Zuordnung der Befehle an die Ausführungseinheiten
        - » Scheduler bestimmt die Anzahl der Befehle, die im nächsten Takt zugeordnet werden können

- Superskalare Prozessor-Pipeline
  - Out-of-order-Abschnitt
    - Zuordnungsstufe
      - Falls die Zuordnung der Befehle an die Ausführungseinheiten nicht der Programmreihenfolge entspricht
    - Ausführungsphase
  - 2. In-order-Abschnitt
    - Gültigmachen der Ergebnisse entsprechend der ursprünglichen Programmordnung
    - Erhalten der korrekten Programmsemantik auch bei Ausnahmeverarbeitung

- Superskalare Prozessor-Pipeline
  - 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

- Superskalare Prozessor-Pipeline
  - 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

- Superskalare Prozessor-Pipeline
  - Befehlsholphase (IF Phase)
    - Sprungvorhersage
      - Sprungverlaufstabellen (Branch History Table)
        - » Festhalten des Verhaltens der Sprungbefehle während der Ausführung des Programms: Prädiktoren
        - » Vorhersage des Verhaltens eines geholten Sprungbefehls
      - Sprungzieladressen-Cache (Branch Target Address Cache, BTAC)
        - » Enthält die Adressen der Sprungbefehle mit den jeweiligen Sprungzielen

- Superskalare Prozessor-Pipeline
  - Befehlsholphase (IF Phase)
    - Dynamische Sprungvorhersagetechniken
      - Ein- und Zwei-Bit-Prädiktoren (siehe Vorlesung)
      - 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

- Superskalare Prozessor-Pipeline
  - Befehlsholphase (IF Phase)
    - Spekulative Ausführung
      - Unter der Annahme, dass die Vorhersage dem tatsächlichen Verlauf entspricht, werden die Befehle zunächst spekulativ den nachfolgenden Stufen der Pipeline weitergegeben
      - Erweist sich nach der Auswertung der entsprechenden Sprungbedingung die Vorhersage als richtig, dann werden die Ergebnisse gültig
      - Bei Fehlspekulation muss die Sprungverlaufstabelle aktualisiert werden und die Auswirkungen der spekulativ ausgeführten Befehle rückgängig gemacht werden

- Superskalare Prozessor-Pipeline
  - 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

- Superskalare Prozessor-Pipeline
  - Dekodierphase (ID Phase)
    - Registerumbenennung
      - Dynamische Umbenennung der Operanden- und Resultatsregister
      - 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
      - Auflösung von Konflikten aufgrund von Namensabhängigkeiten

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

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

- Superskalare Prozessor-Pipeline
  - Zuordnungsphase (Instruction Issue)
    - Befehlsfenster
      - Entkoppelt die Befehlsbereitstellung von der Ausführung
      - Fasst konzeptionell alle zwischen der Befehlsdecodier-/Umbenennungsstufe und der Ausführungsstufe liegenden Befehlspufferplätze zusammen
      - Pool von Befehlen
        - » Sammeln der geholten und dekodierten Befehle
        - » Prüfen auf Daten- und Ressourcenkonflikte
      - Eintrag im Befehlsfenster
        - » Felder für Opcode, Operanden, Informationen zur Konflikterkennung und Auflösung

- Superskalare Prozessor-Pipeline
  - Zuordnungsphase (Instruction Issue)
    - Befehlsfenster
      - Auswahl der ausführbereiten Befehle und Zuordnung zu Verarbeitungseinheit
      - Befehl ist ausführungsbereit:
        - » alle Operanden sind verfügbar
      - Befehl muss warten:
        - » mindestens einer seiner Operanden wird von einem anderen Befehl geliefert und dieser hat das Ergebnis noch nicht bereitgestellt (Auflösen eines Datenkonflikts)
        - » Befehl kann mehrere Takte warten
        - » Zuordnung kann nur erfolgen, wenn gewählte Ausführungseinheit nicht beschäftigt ist (Auflösen eines Ressourcenkonflikts)

- Superskalare Prozessor-Pipeline
  - Zuordnungsphase (Instruction Issue)
    - 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.

- Superskalare Prozessor-Pipeline
  - Zuordnungsphase (Instruction Issue)
    - Zuordnungsstrategie (instruction issue policy)
      - Bestimmt, ob die Zuordnung entsprechend der Reihenfolge (in-order-issue) oder unabhängig von dieser (out-of-order) erfolgt
      - Heutige superskalare Prozessoren setzen weitgehend die out-of-order-Strategie ein.

- Superskalare Prozessor-Pipeline
  - Zuordnungsphase (Instruction Issue)
    - Zweistufige Zuweisung:
      - 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
      - Dispatch:
        - » Ausführungsbereiter Befehl wird zur Ausführung angestoßen, falls diese frei ist

- Superskalare Prozessor-Pipeline
  - 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

- Superskalare Prozessor-Pipeline
  - Befehlsausführung
    - Typische Ausführungseinheiten:
      - Lade-/Speichereinheit
        - » Zugriff auf den Daten-Cache
        - » Lesen bzw. Schreiben von Werten aus bzw. in den Daten-Cache
        - » Bei Fehlzugriff: Laden eines neuen Blocks über die Busschnittstelle
        - » Speicherverwaltungseinheit (Memory-Management Unit)

- Superskalare Prozessor-Pipeline
  - Befehlsausführung
    - Typische Ausführungseinheiten:
      - Eine oder mehrere Integer-Einheiten:
        - » Einstufige Einheiten
        - » Organisiert als mehrstufige Pipeline
      - Multimedia-Einheiten
        - » Datenparallele Ausführung von Multimedia-Befehlen auf 8-Bit, 16-Bit oder 32-Bit Werten
        - » Verwendung der Multimediaregister (64- und 128 Bit)
      - Gleitkomma-Einheiten
        - » Ausführung von Gleitkomma-Befehlen
        - » 64 Bit Gleitkomma-Arithmetik nach IEEE-754-Standard
        - » Mehrstufige Pipeline (arithmetisches Pipelining)

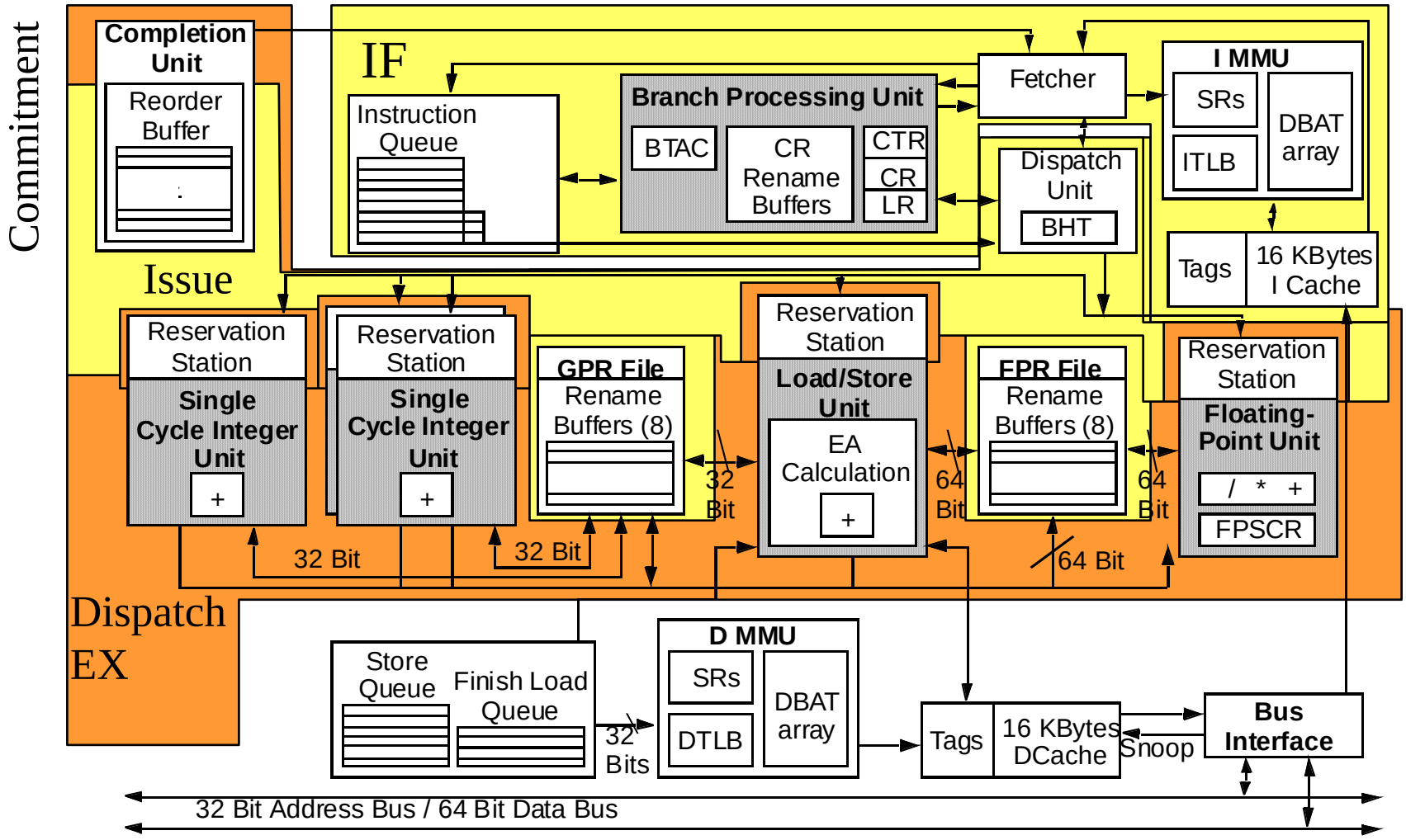
- Superskalare Prozessor-Pipeline
  - Befehlsausführung
    - Completion
      - Eine Instruktion beendet ihre Ausführung, wenn das Ergebnis für nachfolgende Befehle bereitsteht (Forwarding, Puffer)
      - Completion heißt: eine Befehlsausführung ist „vollständig“
        - » Erfolgt unabhängig von 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.

- Superskalare Prozessor-Pipeline
  - Rückordnungsstufe
    - 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

- Superskalare Prozessor-Pipeline
  - Rückordnungsstufe
    - 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

- Superskalare Prozessor-Pipeline
  - Rückordnungsstufe
    - Retirement
      - Freigeben des Platzes im Umordnungspuffer (retirement)

- Fallstudie: Motorola PowerPC 604



- 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);
      - ein von einer Funktionseinheit i produziertes Ergebnis wird direkt an die Funktionseinheit j weitergegeben, wenn diese das Ergebnis als Operand benötigt;
    - Ergebnisbus:
      - alle Funktionseinheiten, die auf einen Operanden warten, werden gleichzeitig bedient;

- Fallstudie: Tomasulo (IBM 360/91)
  - Umordnungspuffer

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

- Fallstudie: Tomasulo (IBM 360/91)
  - Registerdatei
    - Jedes Register einer Registerdatei enthält neben dem Wert ein Feld für
      - 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 in die entsprechenden Felder des Umordnungspuffers kopiert.

- 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 (Überprüfen von RAW Konflikten)
      - Out-of-Order
    - Rückschreibphase
      - Schreiben der Ergebnisse auf Ergebnisbus
      - Kennzeichnen des Umordnungspuffers als verfügbar

- 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									
	2	1									
S_{mul}	3	1									
S_{div}	4	1									

RS status

cycle 0

token.tag

token.data

- 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

- 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	1
RS	3	1	0	0	0	0

register status

reservation stations

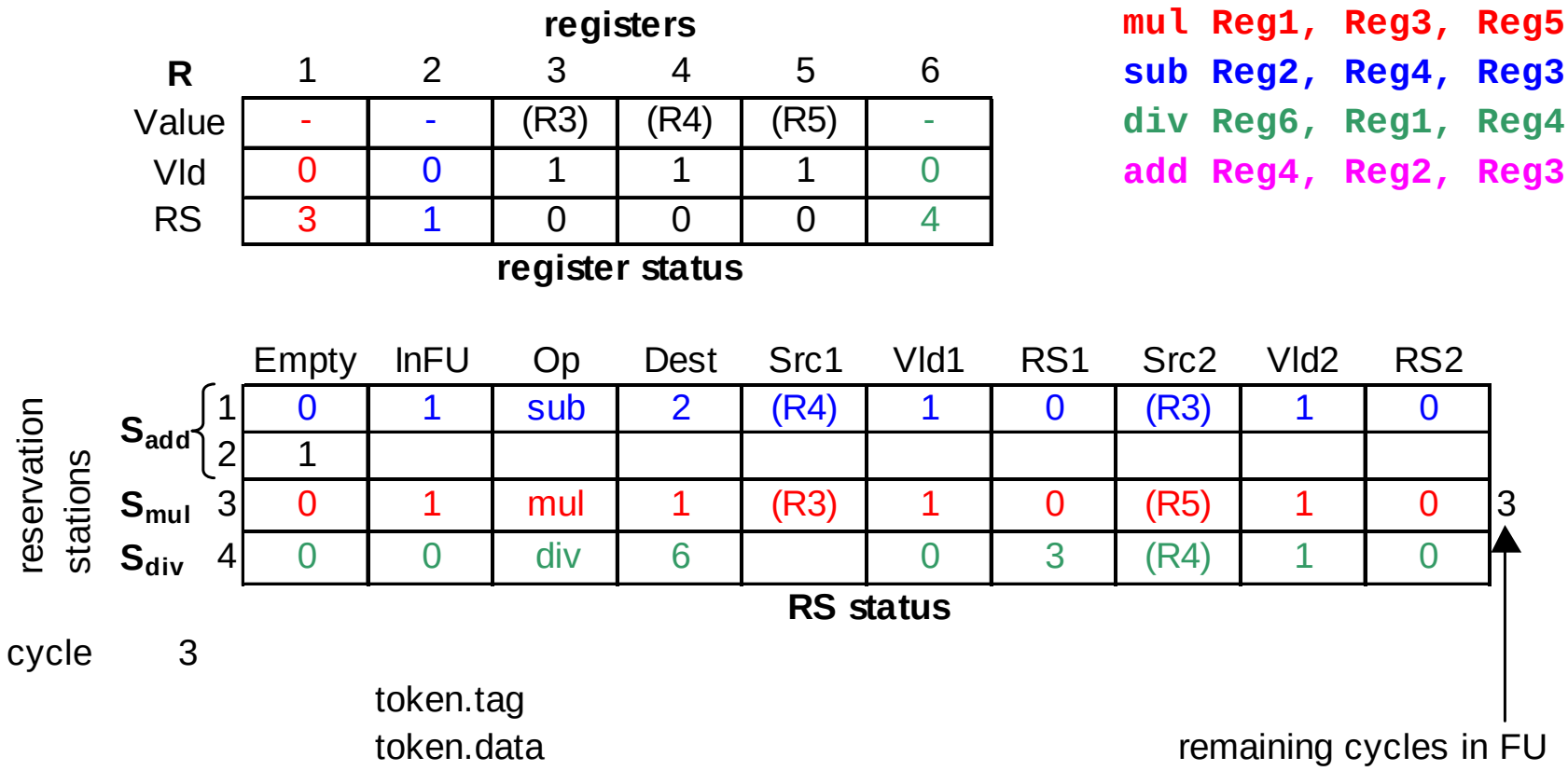
		Empty	InFU	Op	Dest	Src1	Vld1	RS1	Src2	Vld2	RS2
S_{add}	1	0	0	sub	2	(R4)	1	0	(R3)	1	0
	2	1									
S_{mul}	3	0	1	mul	1	(R3)	1	0	(R5)	1	0
S_{div}	4	1									

RS status

cycle 2

token.tag
token.data

- Fallstudie: Tomasulo (IBM 360/91)
  - Beispiel (T. Ungerer)



- 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
S_{add}	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
S_{div}	4	0	0	div	6		0	3	(R4)	1	0

2

RS status

cycle 4

token.tag 1

token.data (R4)-(R3)

- 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	1
S_{div}	4	0	0	div	6		0	3	(R4)	1	0	

RS status

cycle 5

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	-	(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)

```
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
	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)

- 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

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	
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	3

RS status

cycle 9

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)*(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	2

RS status

cycle 10

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)*(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

- 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)	(R3)*(R5) /(R4)
Vld	1	1	1	1	1	1
RS	0	0	0	0	0	0

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	1	1	div	6	(R3)*(R5)	1	0	(R4)	1	0	0

RS status

cycle 12

token.tag 4

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

- **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.

- 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