Rechnerstrukturen

Vorlesung im Sommersemester 2009

Prof. Dr. Wolfgang Karl

Universität Karlsruhe (TH)

Fakultät für Informatik

Institut für Technische Informatik

- **Kapitel 2: Parallelismus auf Befehlsebene**

2.1: Nebenläufigkeit, Superskalartechnik

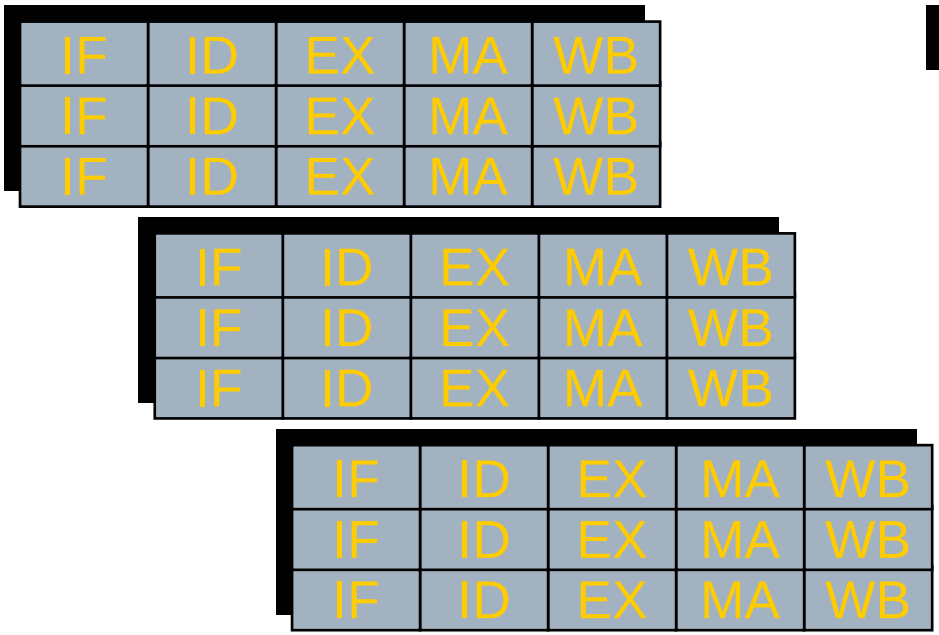
Nebenläufigkeit

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

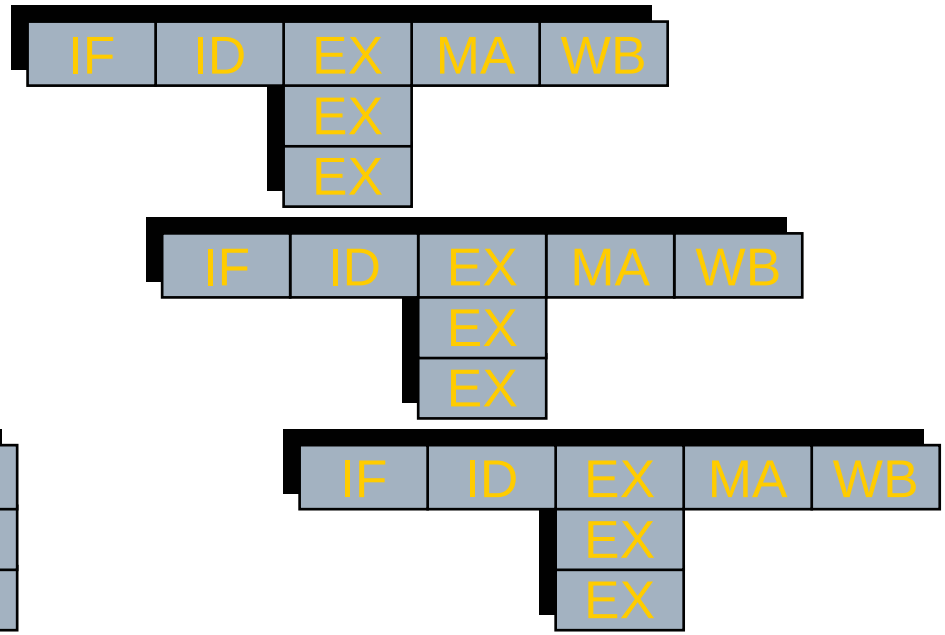
Nebenläufigkeit

- 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



Superskalartechnik

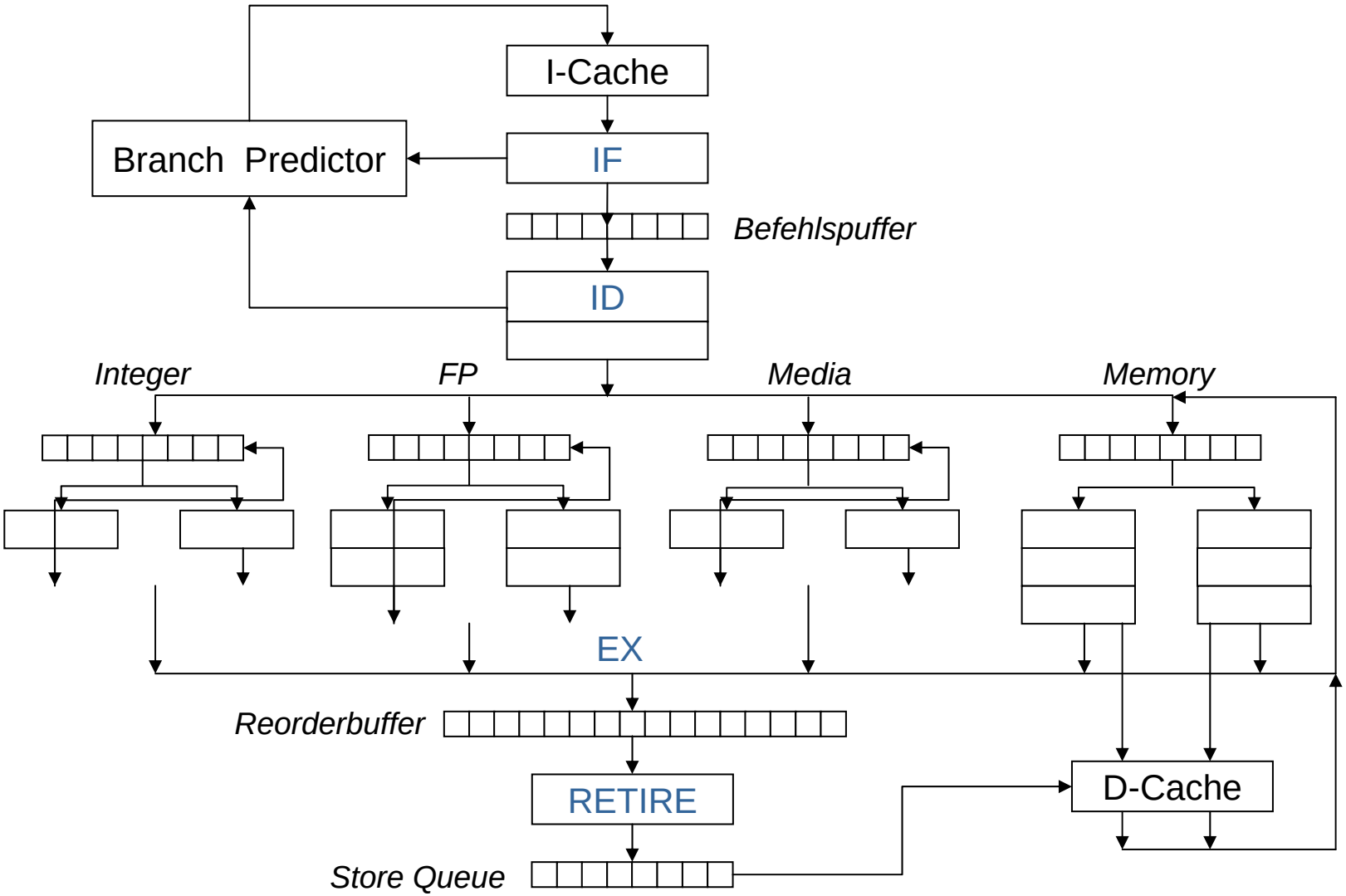
- RISC → Superskalar
  - Mehrfachzuweisungsmethoden (multiple issue)
    - Die Superskalar-Technik ermöglicht es heute, pro Takt mehrere Befehle den Ausführungseinheiten zuzuordnen und eine gleiche Anzahl von Befehlsausführungen pro Takt zu beenden.
  - Superskalare RISC-Prozessoren:
    - RISC-Charakteristika 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

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

Superskalartechnik

- Superskalarer Prozessor

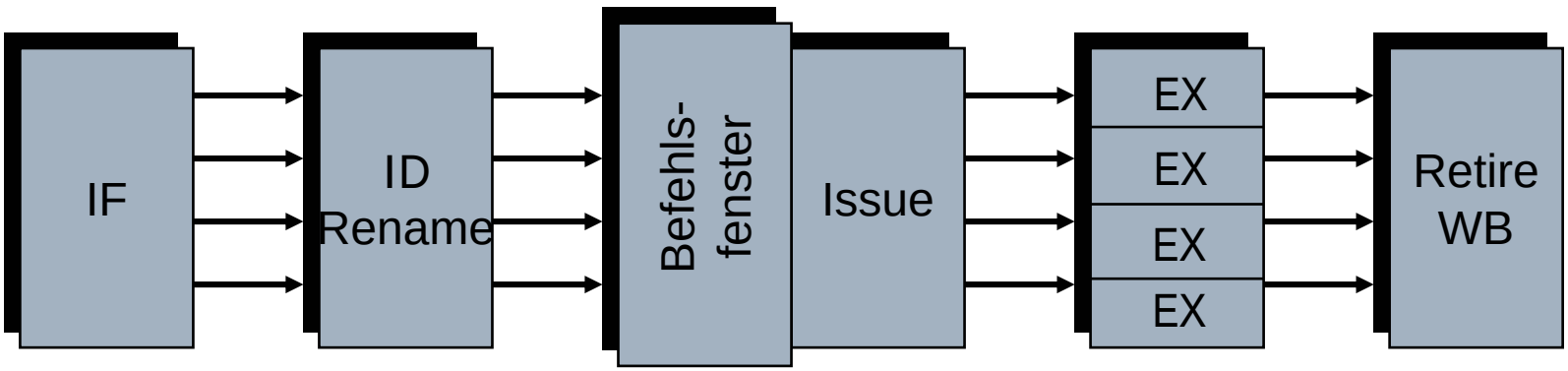


Superskalartechnik

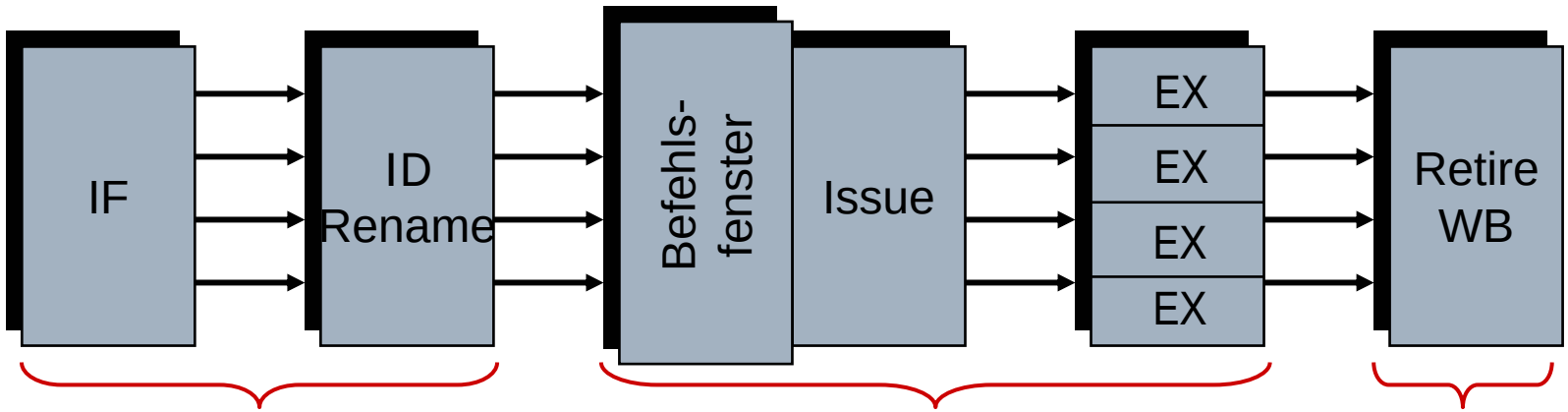
- Superskalarer Prozessor
  - Komponenten
    - Befehlsholeinheit (Instruction Fetch)
    - Dekodiereinheit (Instruction Decode) mit Registerumbenennung (Register renaming)
    - Zuordnungseinheit (Instruction Issue)
    - Unabhängige Verarbeitungseinheiten (Functional Units)
    - Rückordnungseinheit (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

Superskalartechnik

- 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

Superskalartechnik

- 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

Superskalartechnik

- 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)

Superskalartechnik

- 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.

Superskalartechnik

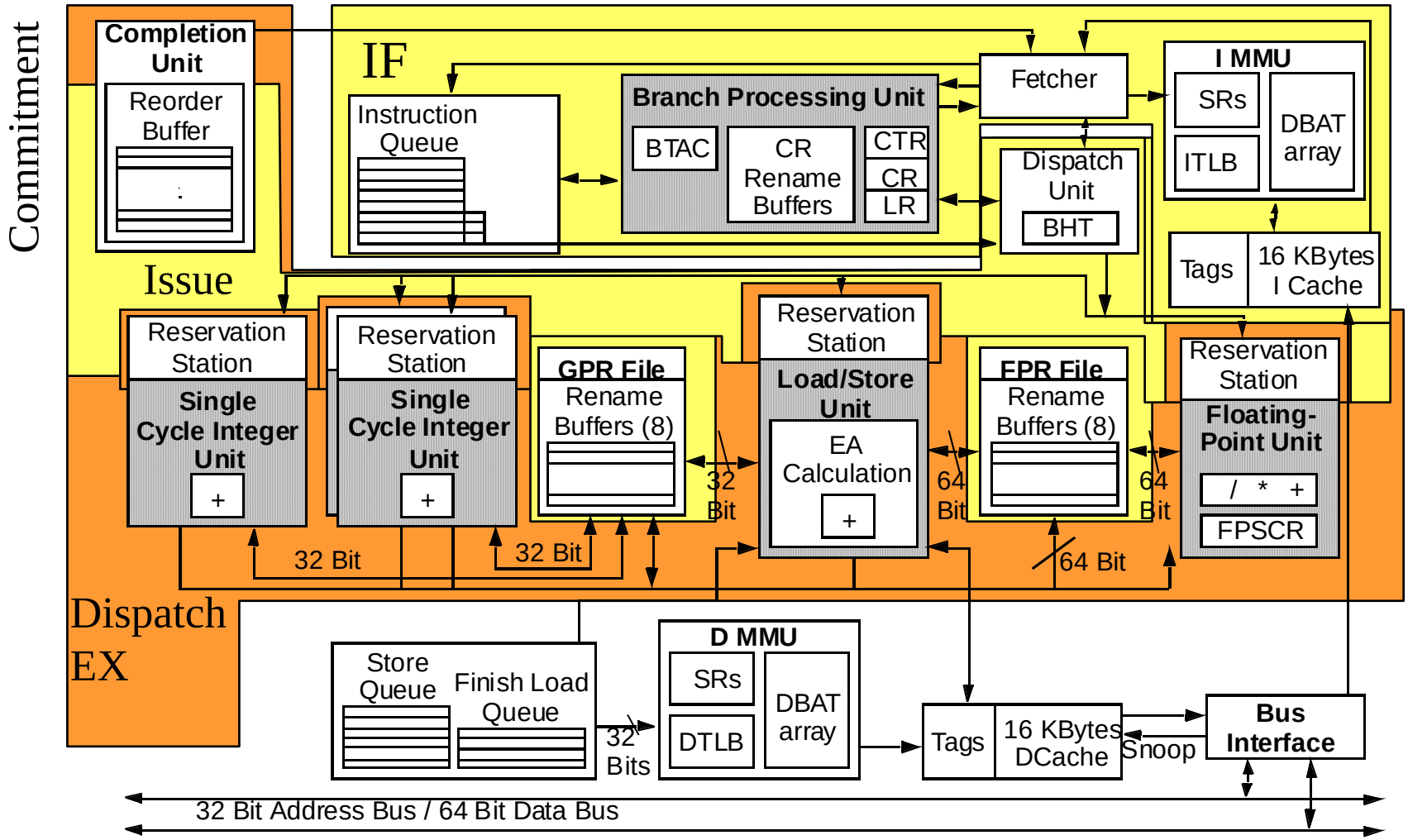
- Superskalare Prozessor-Pipeline
  - Rückordnungsstufe
    - Commitment:
      - Nach der Vervollständigung beenden die Befehle ihre Bearbeitung (Commitment), d.h. die Befehlsresultate werden in der Programmreihenfolge gültig gemacht
        - » Ergebnisse werden in den Architekturregistern dauerhaft 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)

Superskalartechnik

- Fallstudie: Motorola PowerPC 604



Superskalartechnik

- 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.

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 (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									
S_{add}	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)

```
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	1	1	1	1	1
RS	3	0	0	0	0	0

register status

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)

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
S_{add}	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)

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

```
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	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

- 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	
S_{add}	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)

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

```
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	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)

- 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	
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)

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	1

RS status

cycle 11

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)	(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

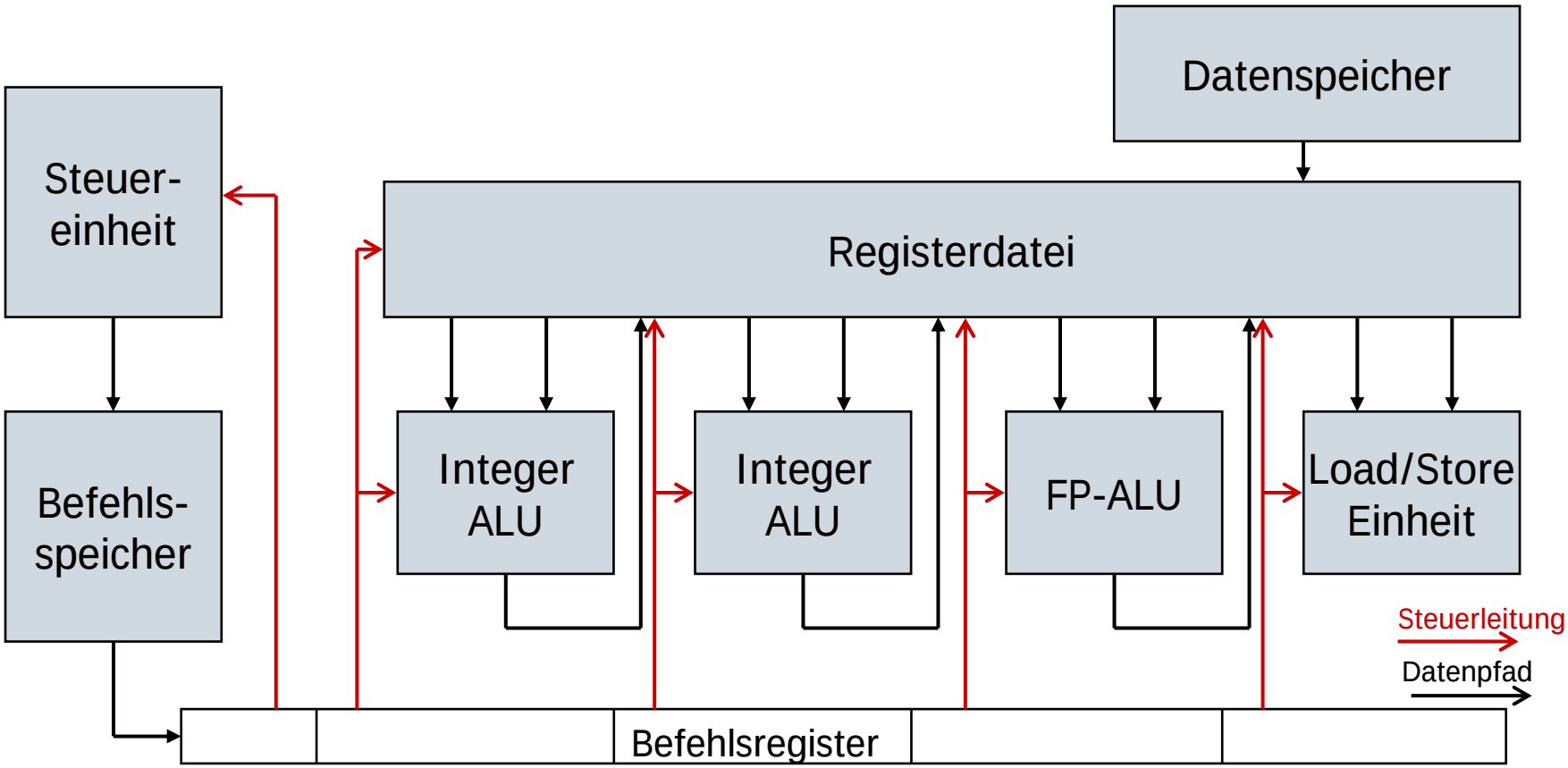
- **Kapitel 2: Parallelismus auf Befehlsebene**

2.3: Nebenläufigkeit, VLIW, EPIC

- Grundprinzip: VLIW
  - VLIW: Very Long Instruction Word
  - 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-Architektur
  - Steuerung der parallelen Abarbeitung zur Übersetzungszeit (automatisch parallelisierender Compiler)

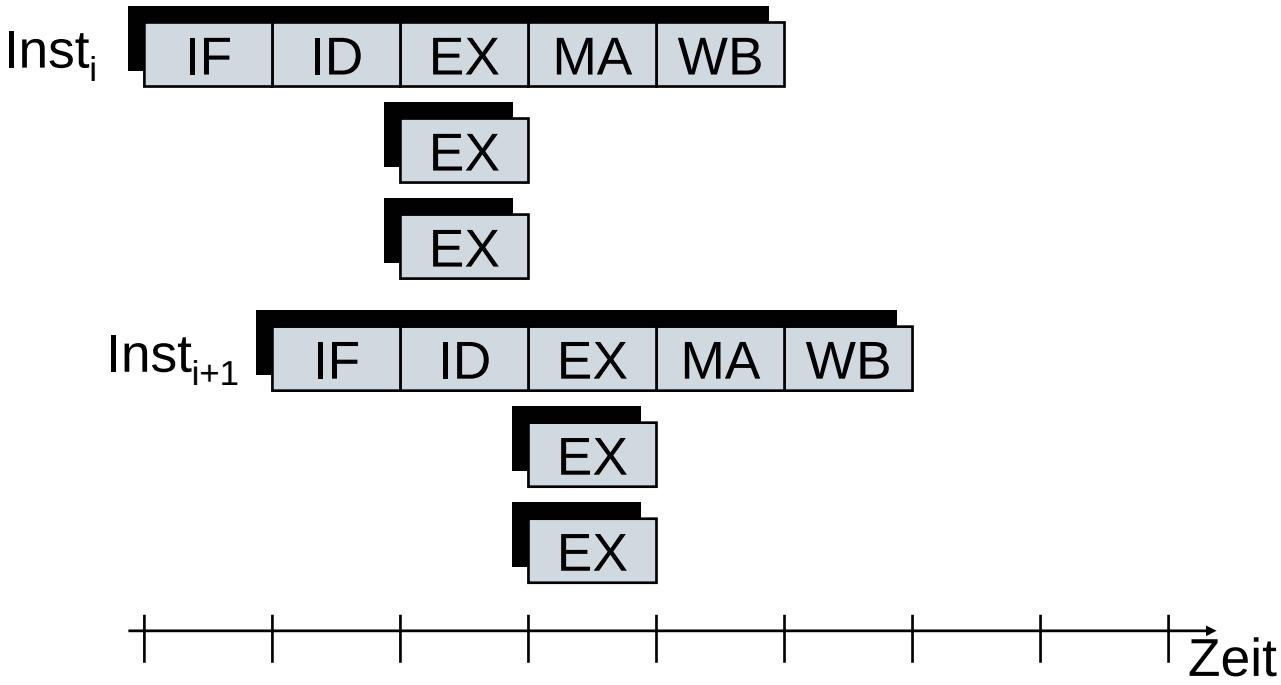
VLIW-Prinzip

- Grundprinzip VLIW
  - Prinzipieller Aufbau



VLIW-Prinzip

- Grundprinzip VLIW
  - Prinzipielle Pipeline-Struktur



VLIW-Prinzip

- 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

VLIW-Prinzip

- Statische Steuerung der parallelen Abarbeitung
  - 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
```

VLIW-Prinzip

- Statische Steuerung der parallelen Abarbeitung
  - Scheduling (Fortsetzung Beispiel)

```
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
```

Scheduling

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

VLIW-Prinzip

- Statische Steuerung der parallelen Abarbeitung
  - Scheduling
    - 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
}
```

VLIW-Prinzip

- Statische Steuerung der parallelen Abarbeitung
  - 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
```

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

VLIW-Prinzip

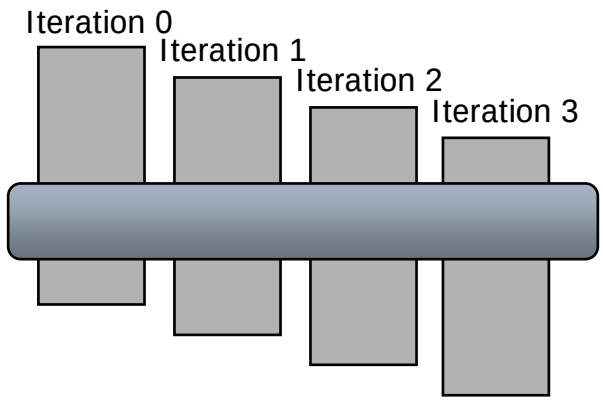
- Statische Steuerung der parallelen Abarbeitung
  - Software-Pipelining

```
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		

VLIW-Prinzip

- Statische Steuerung der parallelen Abarbeitung
  - Software-Pipelining
    - Technik zur Reorganisation von Schleifen
      - Jede Iteration in dem mit Software-Pipelining generierten Code enthält Befehle aus verschiedenen Iterationen der ursprünglichen Schleife

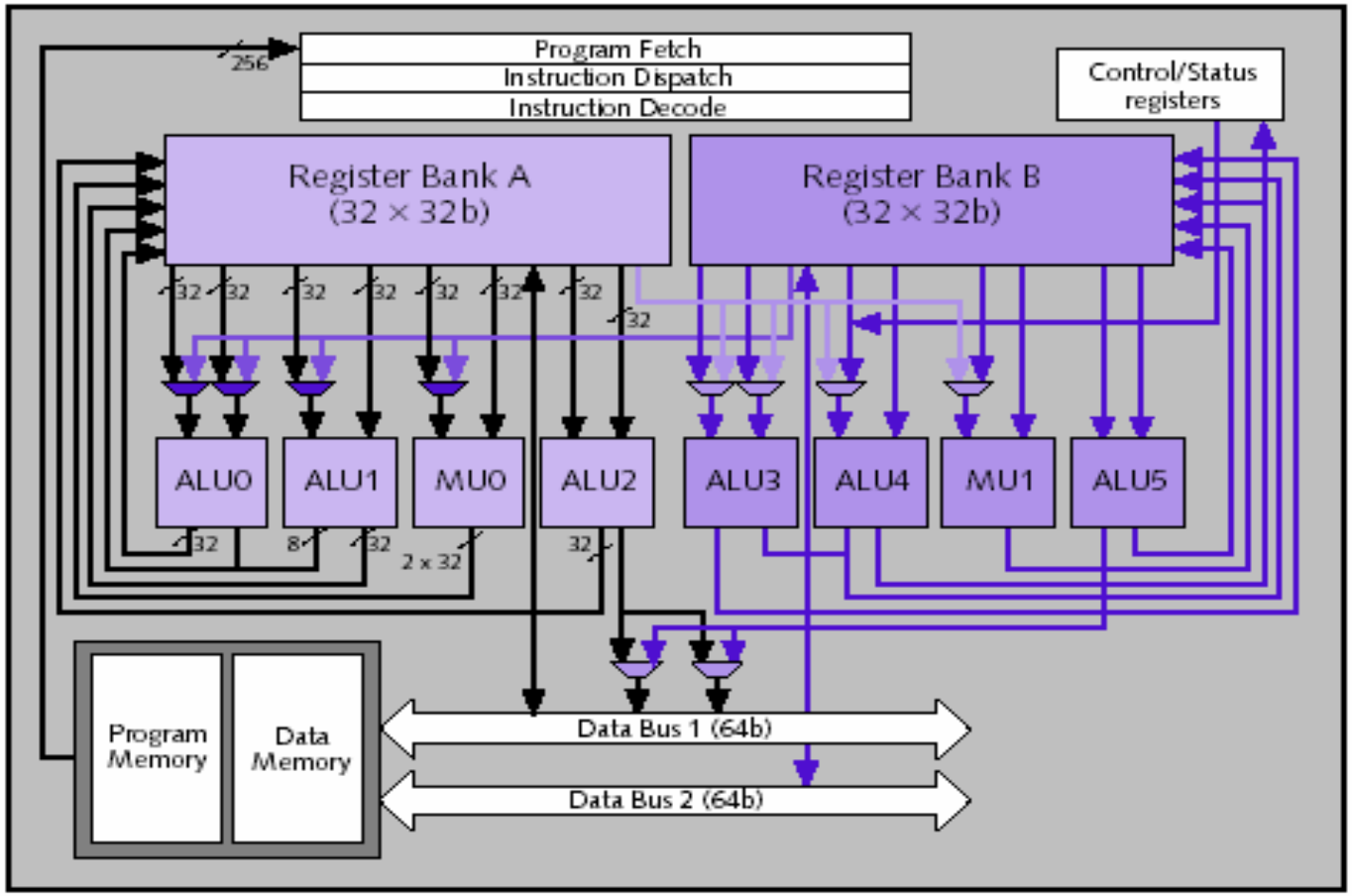


VLIW-Prinzip

- 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:
  - Allzweck-Mikroprozessoren
    - Transmeta Crusoe
  - Bereich eingebetteter Prozessoren (DSP)
    - Beispiel: Trimedia TM32 Architecture
    - Agere/Motorola Star Core

VLIW-Prinzip: Beispiel

- TI TMS320C6400



VIIW-Prinzip: Beispiel

- TI TMS320C6000 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

VIIW-Prinzip: Beispiel

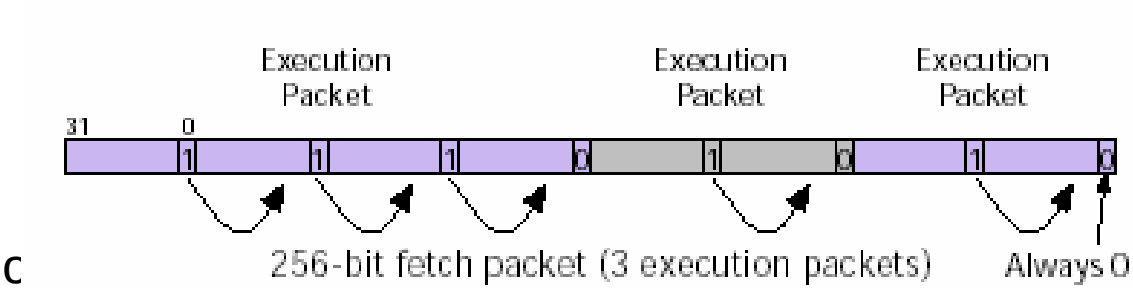
- TI TMS320C6000 Architekturmerkmale
  - 2 x 4 Funktionseinheiten (A und B Seite)
    - 40 Bit Schiebeeinheit
      - Arithmetische Operationen, Verzweigung und Verzweigungsadressgenerierung
    - 32 Bit Addierer
      - Adressgenerierung

VlIW-Prinzip: Beispiel

- TI TMS320C6000 Architekturmerkmale
  - VLIW-Prinzip
    - 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

VlIW-Prinzip: Beispiel

- TI TMS320C6000 Architekturmerkmale
  - VLIW-Prinzip
    - Bis zu 8 Befehle können gleichzeitig ausgeführt werden: Gruppierung von gleichzeitig ausführbaren Befehlen (Execution Packets)
    - Werden im niedrigstwertigen Bit gekennzeichnet: alle nachfolgenden Befehle werden gleichzeitig ausgeführt
    - Prc
      ver



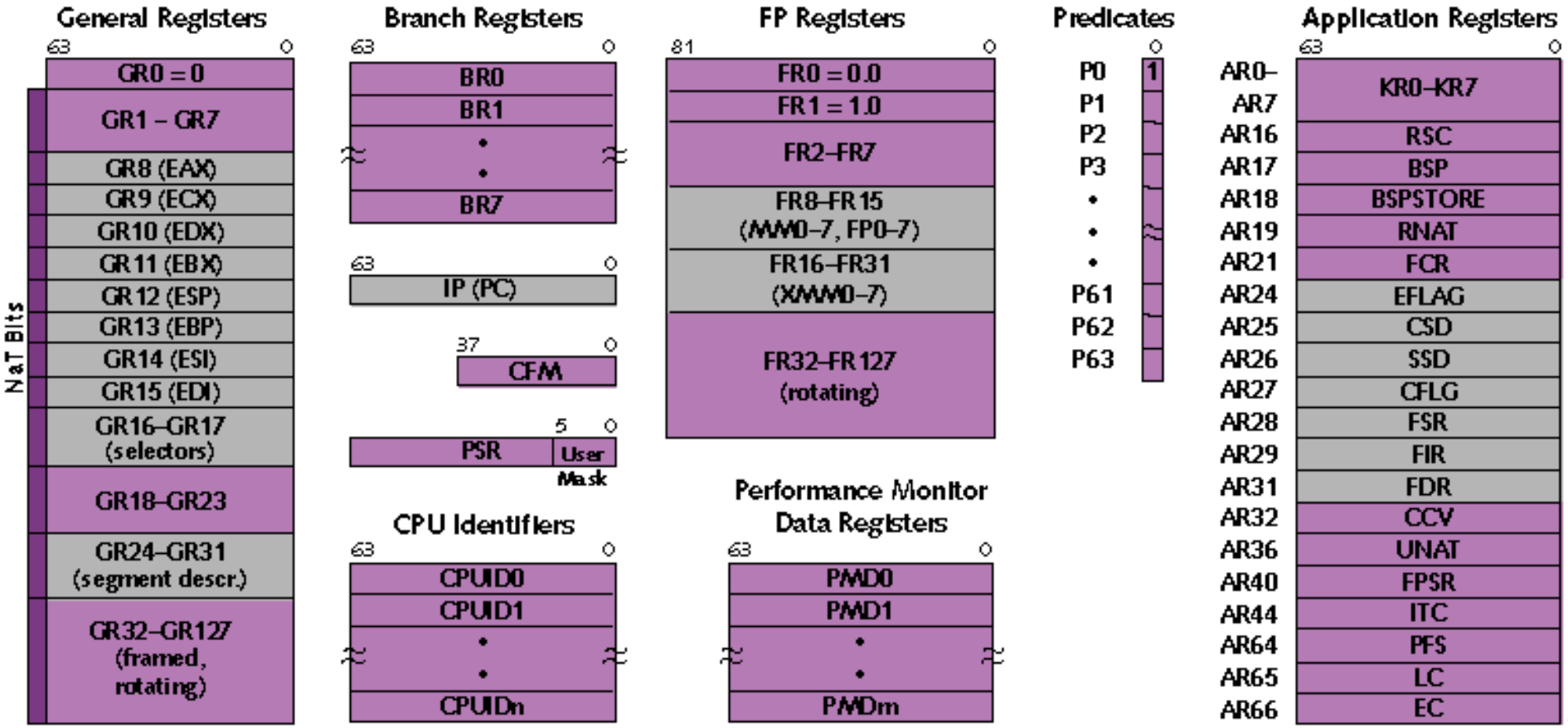
VLIW-Prinzip

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

- EPIC: Explicitly Parallel Instruction Computing
  - Gemeinsames Projekt von Hewlett-Packard und Intel (1994 angekündigt)
  - Ziele:
    - 64 Bit Architektur: IA-64
    - Explizite Spezifikation des Parallelismus im Maschinencode: EPIC-Format, (entspricht VLIW-Prinzip)
    - Bedingte Ausführung von Befehlen (Predication)
    - Spekulative Ausführung von Ladeoperationen (Data Speculation)
    - Großer Registersatz
    - Skalierbarer Befehlssatz
    - Sinnvolles Zusammenwirken zwischen Compiler und Hardware
  - Itanium: erster Prozessor der P7-Generation (Code-Name Merced)

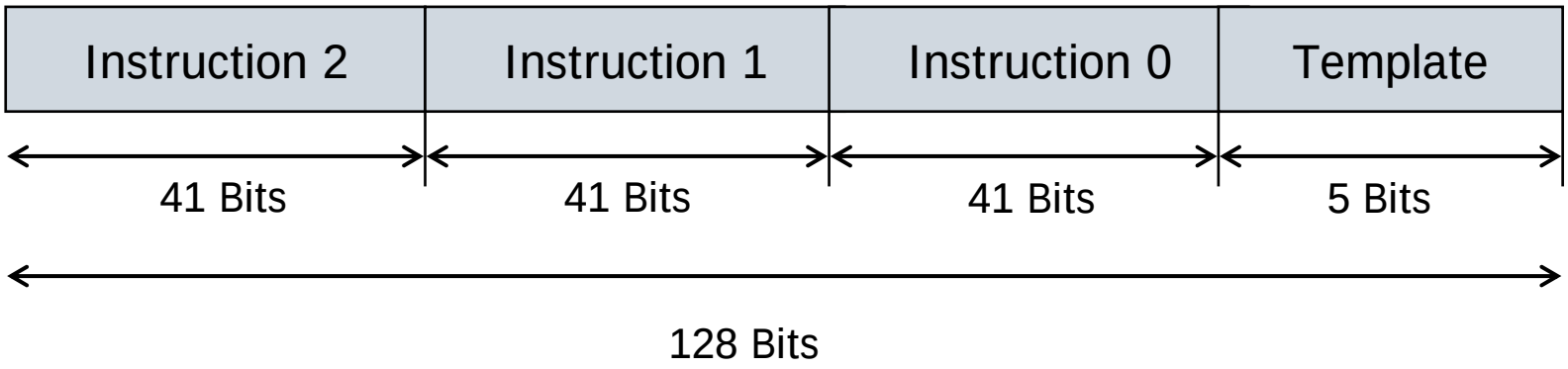
- Intel IA-64
  - Registersatz

Quelle: Microprocessor Report, Vol.13, Nr.7, 1999



- IA-64 ISA: Befehlsformat
  - Opcode
  - Predicate
  - 2 Felder für Quelloperandenregister
  - 1 Feld für Zielregister

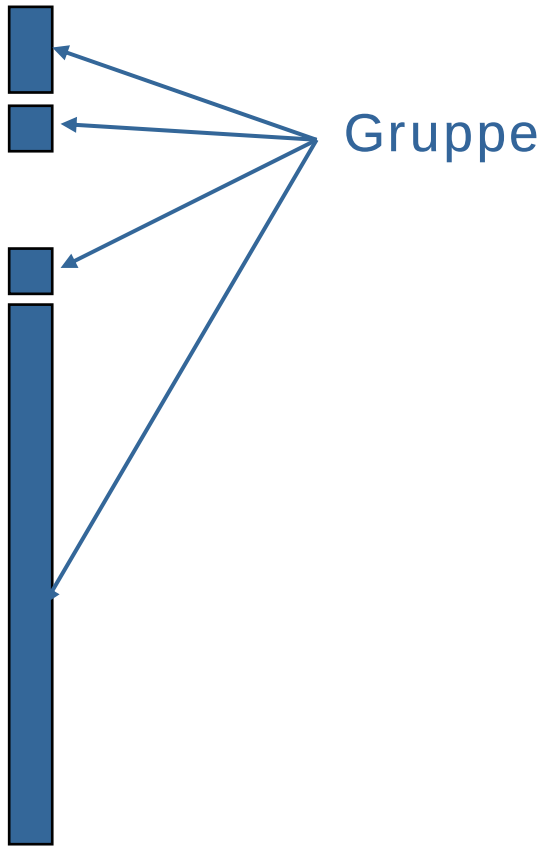
- IA-64 ISA
  - IA-64 Instruktionen werden vom Compiler in so genannte Bundles gepackt



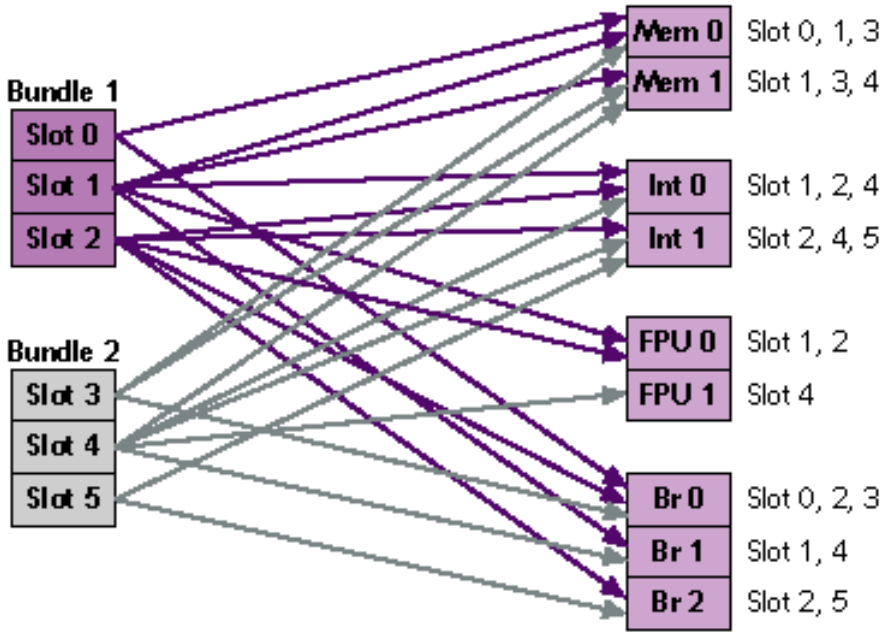
- IA-64 ISA
  - Bundles:
    - Template: zeigt explizit an, ob
      - die Instruktionen im Bundle gleichzeitig ausgeführt werden dürfen, oder
      - eine oder mehrere Instruktionen sequentiell auszuführen sind, oder
      - benachbarte Bundles parallel ausgeführt werden können.

- Beispiele von Befehlsgruppen:

```
{ .mii
        add r1 = r2,r3
        sub r4 = r4, r5;;
        shr r7 = r4, r12;;
}
{ .mmi
        ld8 r2 = [r1];;
        st8 [r1] = r23
        tbit p1,p2 = r4,5
}
{ .mbb
        ld8 r45 = [r55]
(p3)    br.call b1  =func1
(p4)    br.cond Label1
}
{ .mfi
        st4 [r45] = r6
        fmac f1 = f2,f3
        add r3 = r3,8;;
```



- Anstoßen der Befehle zur Ausführung
  - Beispiel: Itanium



  - Es können bis zu sechs Instruktionen pro Takt zur Ausführung angestoßen werden. Die Instruktionen kommen von zwei Bundles. Jeder Bundle hat 3 Slots.

- IA-64: Skalierbarkeit
  - Jedes Bundle enthält drei Instruktionen für eine Menge von drei Funktionseinheiten.
  - Ein IA-64 Prozessor kann *n* Mengen von jeweils drei Funktionseinheiten haben, welche die Informationen im Template nutzen, dann können mehrere Bundles in ein Instruktionswort mit der Länge *n* Bundles gepackt werden.

→ Skalierbarkeit bezüglich der Anzahl der Funktionseinheiten

- Predication, bedingte Befehlsausführung
  - Beispiel:

```
if (r1 == r2)
        r9 = r10 - r11;
else
        r5 = r6 + r7;
```

Bedingte Befehlsfolge

```
        cmp.eq p1,p2 = r1, r2;;
(p1)    sub r9 = r10, r11
(p2)    add r5 = r6, r7
```

- Evaluierung der bedingten Ausdrücke mit Hilfe von Compare-Operationen.
- Jeder Befehl hat ein 6 Bit breites Predicate Feld zur Angabe des Predicate-Registers
- Elimination von Sprüngen

- Predication (Bedingte Befehlsausführung)
  - Zur Laufzeit werden die voneinander unabhängigen Befehle zur Ausführung angestoßen.
  - Der Prozessor führt die Befehle auf den möglichen Programmverzweigungen aus, aber speichert die Ergebnisse nicht endgültig.
  - Überprüfen der Predicate Register
    - Falls das Register eine 1 enthält, dann ist wird die Ausführung der Instruktion abgeschlossen.
    - Falls das Register eine 0 enthält, dann wird das Ergebnis verworfen.

- IA-64 Control Speculation
  - Problem: Verzweigungen schränken Code-Verschiebungen ein

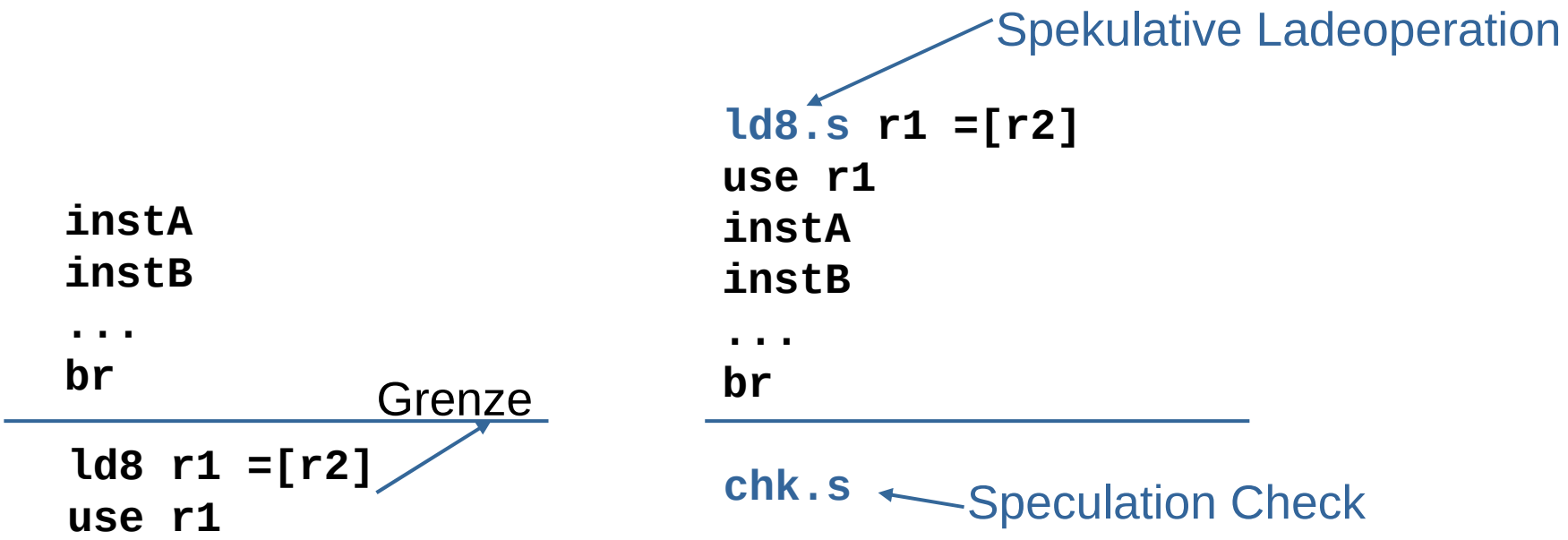
```
instA
instB
...
br
```

Grenze

```
ld8 r1 =[r2]
use r1
```

Ladeoperation kann nicht über den Sprung verschoben werden, denn es könnten Alarme ausgelöst werden.

- IA-64 Control Speculation
  - Lösung: Spekulative Operationen, die keine Alarme auslösen



- IA-64 Control Speculation
  - Einführung von spekulativen Ladeoperationen
    - Verursachen keinen Alarm:
      - Falls zur Laufzeit die Operation einen Alarm auslöst, dann wird dieser Alarm verzögert
      - Setzen von NaT-Bit (Deferred Exception Token, Not-a-Thing Bit) in Zielregister
    - Spekulative Ladeoperation kann über Verzweigungen hinweg verschoben werden
    - Einfügen von Speculation Check Instruktion (chk.s) anstelle der Ladeoperation
      - Zur Laufzeit überprüft die Check-Operation das Zielregister, ob NaT gesetzt ist.Wenn ja, dann erfolgt eine Verzweigung zu einem speziellen Fix-up-Code.

- IA-64 Data Speculation
  - Problem: Zeiger können Compiler zu konservativen Annahmen über die Referenzen zwingen, was eine Code-Verschiebung verhindert.

```
instA
instB
...
store
```

```
ld8 r1 =[r2]
use r1
```

Ladeoperation kann nicht über den die Speicheroperation verschoben werden, da beide dieselbe Adresse referenzieren können.

- IA-64 Data Speculation
  - Lösung: Vorgezogene Ladeoperationen

```
instA
instB
...
store
```

```
ld8 r1 =[r2]
use r1
```

Vorgezogene Ladeoperation

```
ld8.a r1 =[r2]
use r1
instA
instB
...
store
```

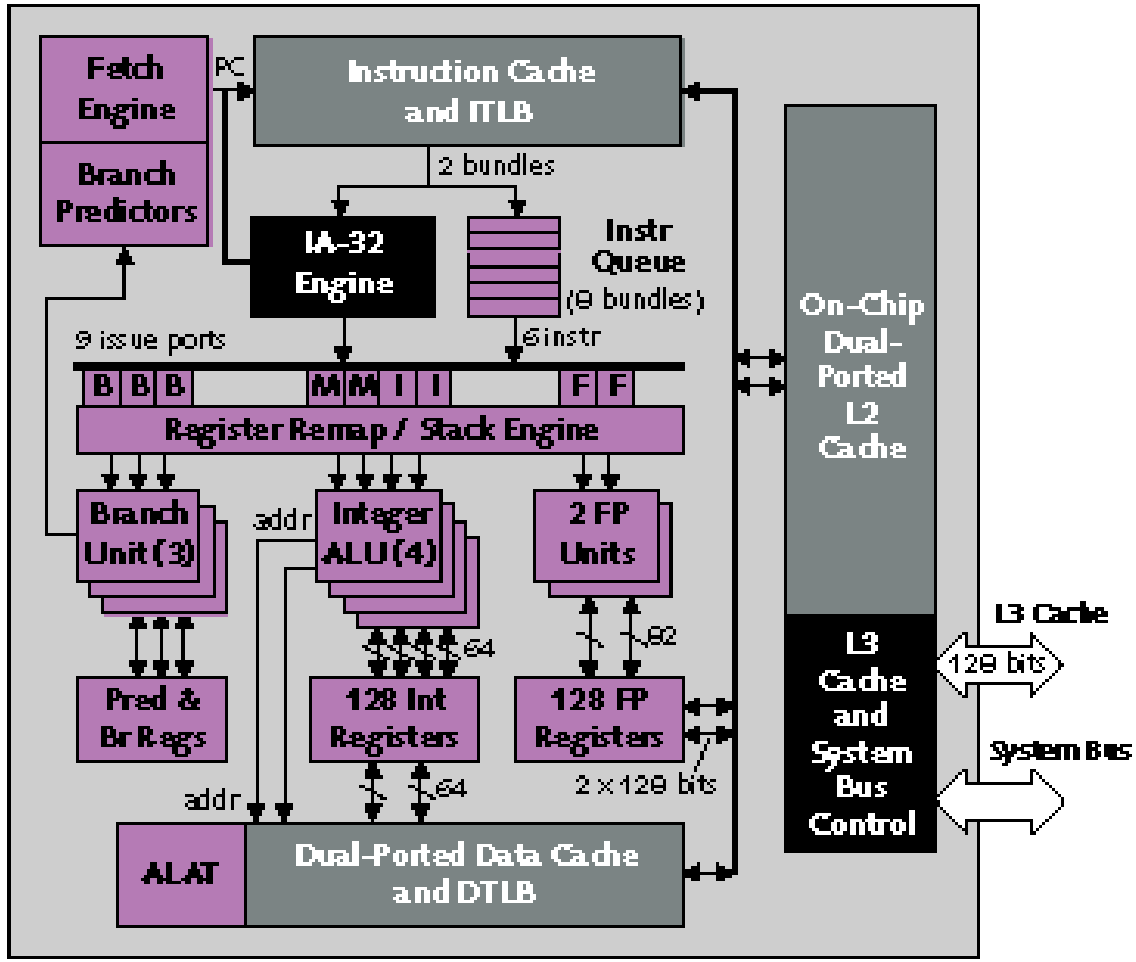
```
chk.a
```

Speculation Check

- IA-64 Data Speculation
  - Lösung: Vorgezogene Ladeoperationen
    - Verschieben von Lade-Operationen auch vor Speicher-Operationen.
      - Vorgezogene Lade-Operation (ld.a)
      - Zur Laufzeit werden Informationen (Zielregister, Speicheradresse, auf die zugegriffen wird, Zugriffsgröße) in Advanced Load Address Table (ALAT) festgehalten.
      - Zur Laufzeit prüft die Hardware, wenn eine Speicheroperation ausgeführt wird, ob eine Adresse in der ALAT mit der Zugriffsadresse übereinstimmt. Wenn ja, dann wird der Eintrag in ALAT gelöscht

- IA-64 Data Speculation
  - Lösung: Vorgezogene Ladeoperationen
    - Einfügen der Check-Operation (**`chk.a`**)
      - Prüft, ob Eintrag von der entsprechenden vorgezogenen Ladeoperation in ALAT steht.
      - Wenn nicht, dann hat es eine Kollission mit einer Speicheroperation gegeben und es erfolgt eine Verzweigung zu einem Fix-up-Code.

- Intel Itanium



Quelle: Microprocessor Report, Vol.13, Nr.13, October 5, 1999

- Intel Itanium
  - 64-Bit Prozessor der P7 Generation
  - EPIC
  - 10-stufige Pipeline mit 6 Befehlen, die gleichzeitig zur Ausführung angestoßen werden können.
  - 128 GP- und 128 FP-Register - keine Registerumbenennung: rotierende Registerfenster
  - Bedingte Befehlsausführung: 64 1 Bit Predicate Register
  - 9 Funktionseinheiten
  - Misprediction Penalty: 9 Zyklen

- Intel Itanium
  - Spekulative Ladeopationen
  - 4-fach mengenassoziative L1 Befehls- und Daten-Cache-Speicher (Write-through)
  - 6-fach mengenassoziativer L2 Cache (Copy-back)
  - Informationen
    - Intel: http://www.intel.com/
      - Intel Technology Journal: http://developer.intel.com/technology/itj
    - IEEE Micro: Sonderheft im Spe./Okt. 2000
    - Henn./Patt.: Computer Architecture: Kap. 4.7

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