
Rechnerstrukturen

Vorlesung im Sommersemester 2006

Prof. Dr. Wolfgang Karl

Universität Karlsruhe (TH)

Fakultät für Informatik

Institut für Technische Informatik

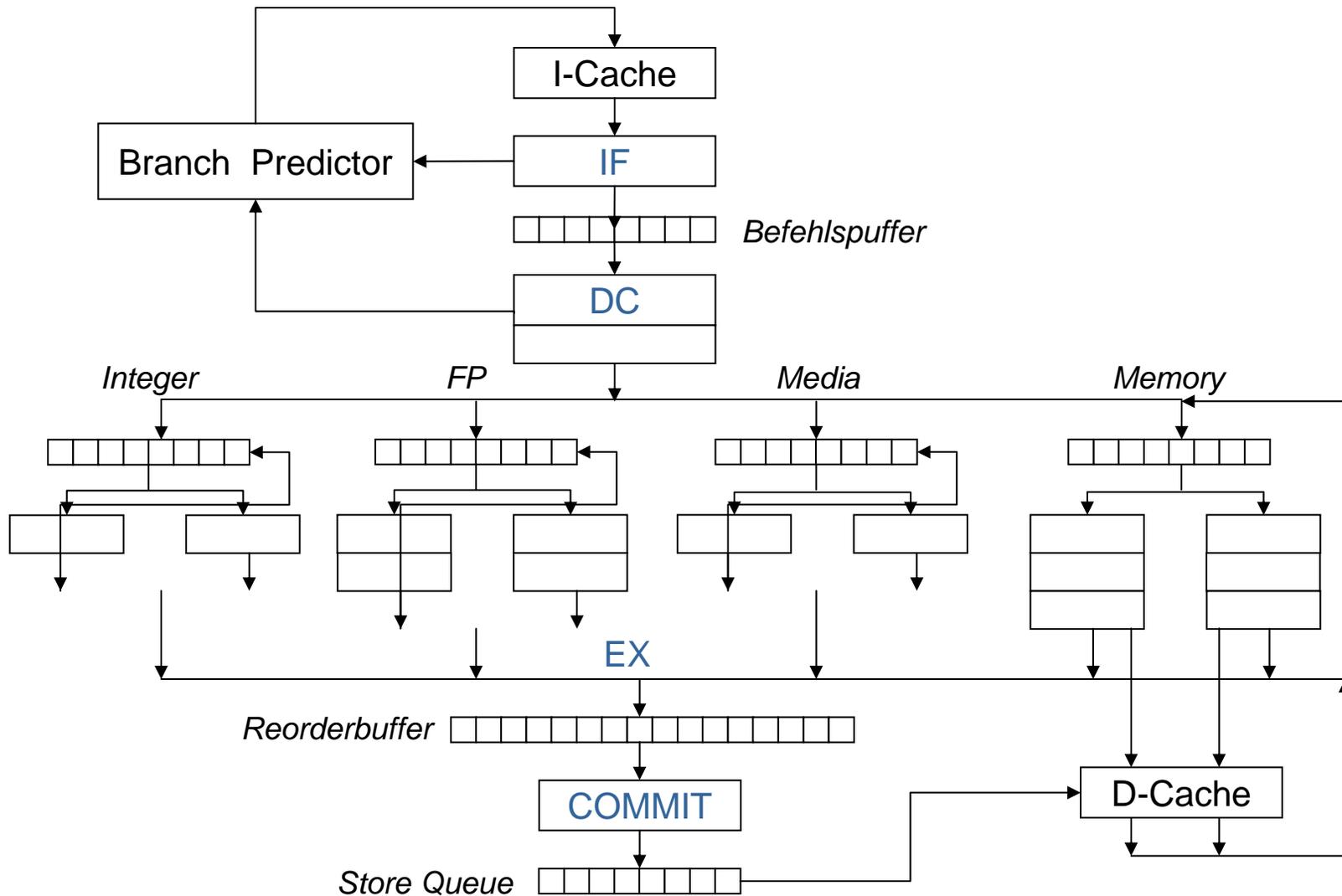


- **Kapitel 2: Parallelismus auf Befehlsebene**

2.1: Nebenläufigkeit, Superskalartechnik

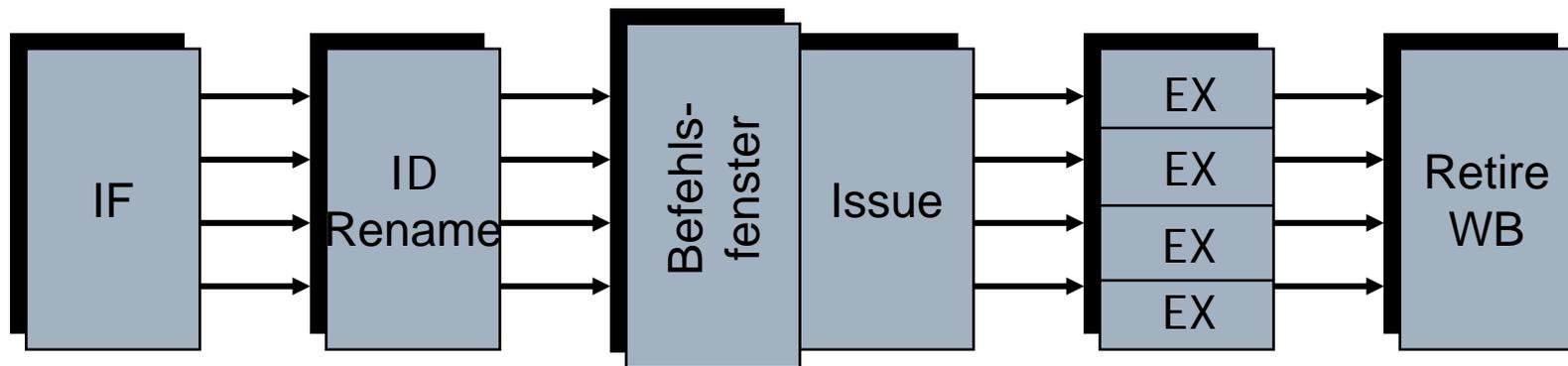


- Superskalarer Prozessor



• Superskalare Prozessor-Pipeline

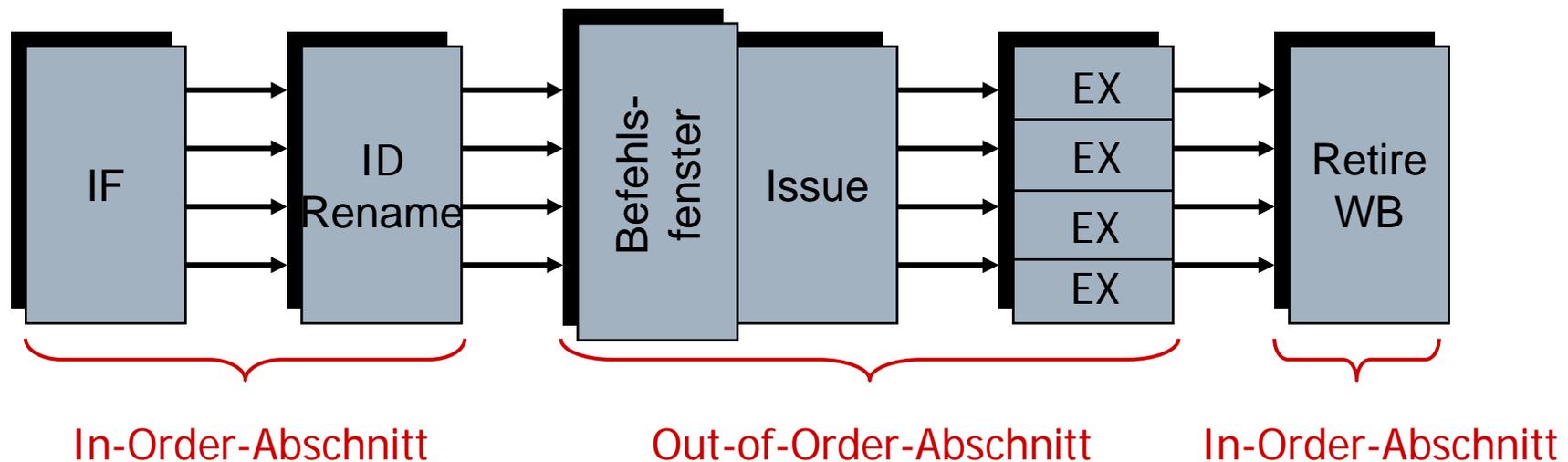
- Mehrere Maschinenbefehle werden gleichzeitig geholt, dekodiert und ausgeführt; Rückschreiben der Ergebnisse
- Zusätzlich:
 - Zuordnungstufe
 - Rückordnungsstufe
 - Weitere Puffer zur Entkopplung der Pipelinestufen



• 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
 - Behandlung von Sprungbefehlen
 - Verwaltung der Komponenten für die Sprungvorhersage
 - » Sprungverlaufstabelle
 - Sprungzieladress-Cache
 - 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

- Sprungzieladress-Cache (Branch Target Address Cache, BTAC)

- » Enthält die Adressen der Sprungbefehle mit den jeweiligen Sprungzielen



- **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**

- Befehlsholphase (IF Phase)

- Dynamische Sprungvorhersagetechniken

- Ein- und Zwei-Bit-Prädiktoren (siehe Vorlesung)

- Komplexere Techniken für superskalare Prozessoren:
(hier nur genannt, siehe Übungen):

- » (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**

- Dekodierphase (ID Phase)

- Dekodierung der im Befehlspeicher 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
 - 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 Befehlspeicherplätze 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 ausführbaren 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

- 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

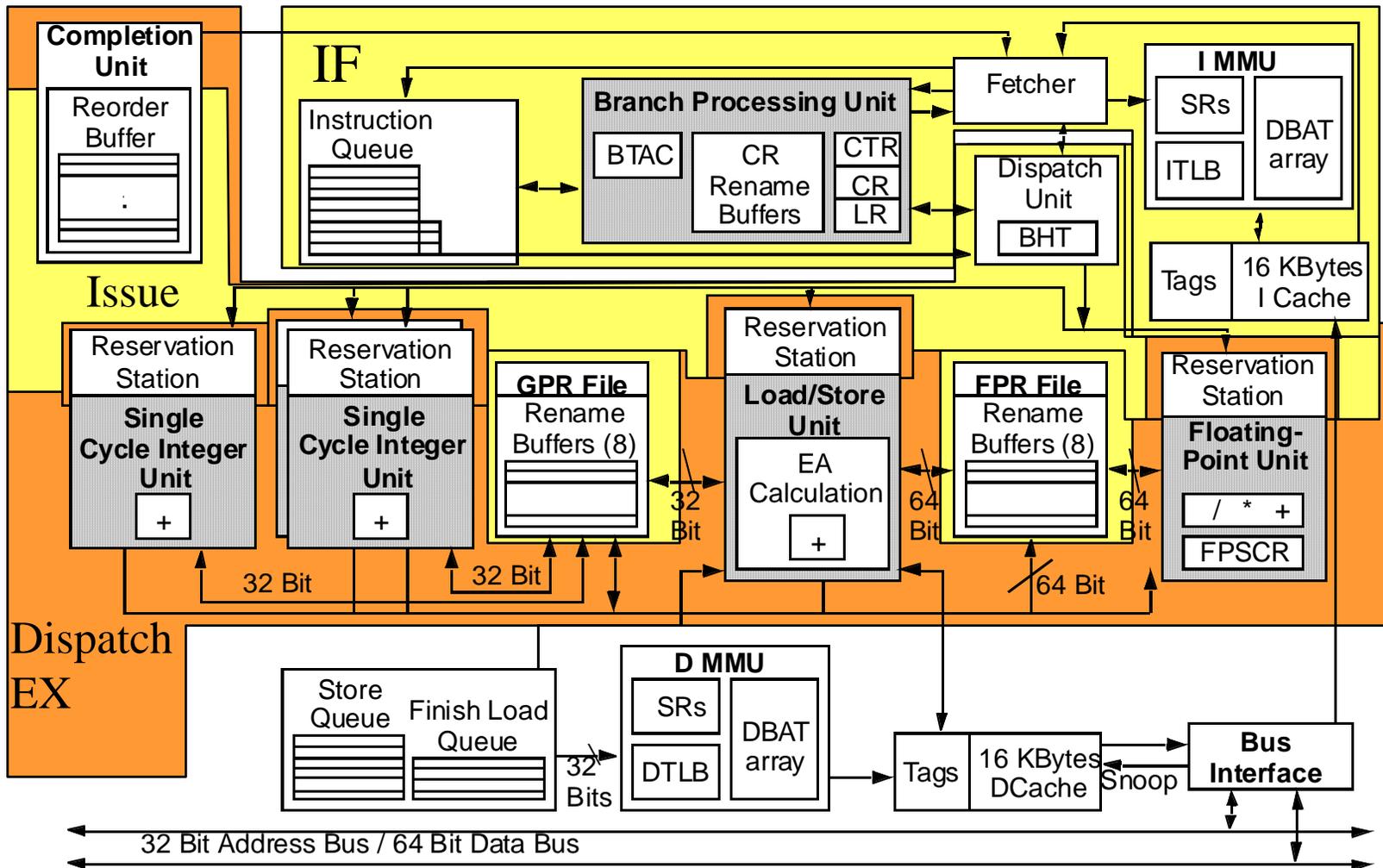


- **Superskalare Prozessor-Pipeline**
 - Rückordnungsstufe
 - Retirement
 - Freigeben des Platzes im Umordnungspuffer (retirement)



- Fallstudie: Motorola PowerPC 604

Commitment



- **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
    
```

		Empty	InFU	Op	Dest	Src1	Vld1	RS1	Src2	Vld2	RS2
reservation stations	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
    
```

		Empty	InFU	Op	Dest	Src1	Vld1	RS1	Src2	Vld2	RS2
reservation stations	S_{add} {	1									
	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
    
```

		Empty	InFU	Op	Dest	Src1	Vld1	RS1	Src2	Vld2	RS2	
reservation stations	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)

		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
    
```

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

RS status

cycle 3

token.tag
token.data

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

		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

```

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

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

RS status

cycle 4

token.tag 1
 token.data (R4)-(R3)



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

		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

```

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

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

		Empty	InFU	Op	Dest	Src1	Vld1	RS1	Src2	Vld2	RS2	
reservation stations	S_{add}	1	1	sub	2	(R4)	1	0	(R3)	1	0	
		2	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

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)	$\frac{(R4)}{(R3)+(R3)}$	(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

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

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		$(R3) \cdot (R5)$	$(R4) - (R3)$	$(R3)$	$\frac{(R4) - (R3)}{(R3) + (R3)}$	$(R5)$	-
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
    
```

		Empty	InFU	Op	Dest	Src1	Vld1	RS1	Src2	Vld2	RS2
reservation stations	S_{add}	1	1	sub	2	(R4)	1	0	(R3)	1	0
		2	1	add	4	$(R4) - (R3)$	1	0	(R3)	1	0
	S_{mul}	3	1	mul	1	(R3)	1	0	(R5)	1	0
	S_{div}	4	0	div	6	$(R3) \cdot (R5)$	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)	$\frac{(R4)}{(R3)+(R3)}$	(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
    
```

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

RS status

cycle 9

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
    
```

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

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	$(R3) \cdot (R5)$	$(R4) - (R3)$	$(R3)$	$\frac{(R4)}{(R3) + (R3)}$	$(R5)$	-
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
    
```

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

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
    
```

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