

Rechnerstrukturen

Vorlesung im Sommersemester 2003

Wolfgang Karl

Universität Karlsruhe

Fakultät für Informatik

Vorlesung Rechnerstrukturen

□ Kapitel 2: Zentraleinheiten

Pipeline-Konflikte und Lösungen

Dynamische Sprungvorhersage

□ 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.

Dynamische Sprungvorhersage

□ Korrelations-Prediktoren (Correlation-Based Predictors)

- ♦ Auswerten des Verhaltens anderer Sprünge zusätzlich zu der Verzweigung, für welche die Vorhersage zu treffen ist.
- ♦ Neben der Auswertung der Vorgeschichte des aktuellen Sprungs, auch Auswertung der Vorgeschichte abhängiger Sprünge.

□ (m,n)-Predictors:

- ♦ Benutzt das Verhalten der letzten m Sprünge für die Auswahl aus 2^m Prediktoren, wobei jeder Prediktor ein n -Bit Prediktor für einen Sprung ist.

Dynamische Sprungvorhersage

□ (m,n)-Predictors:

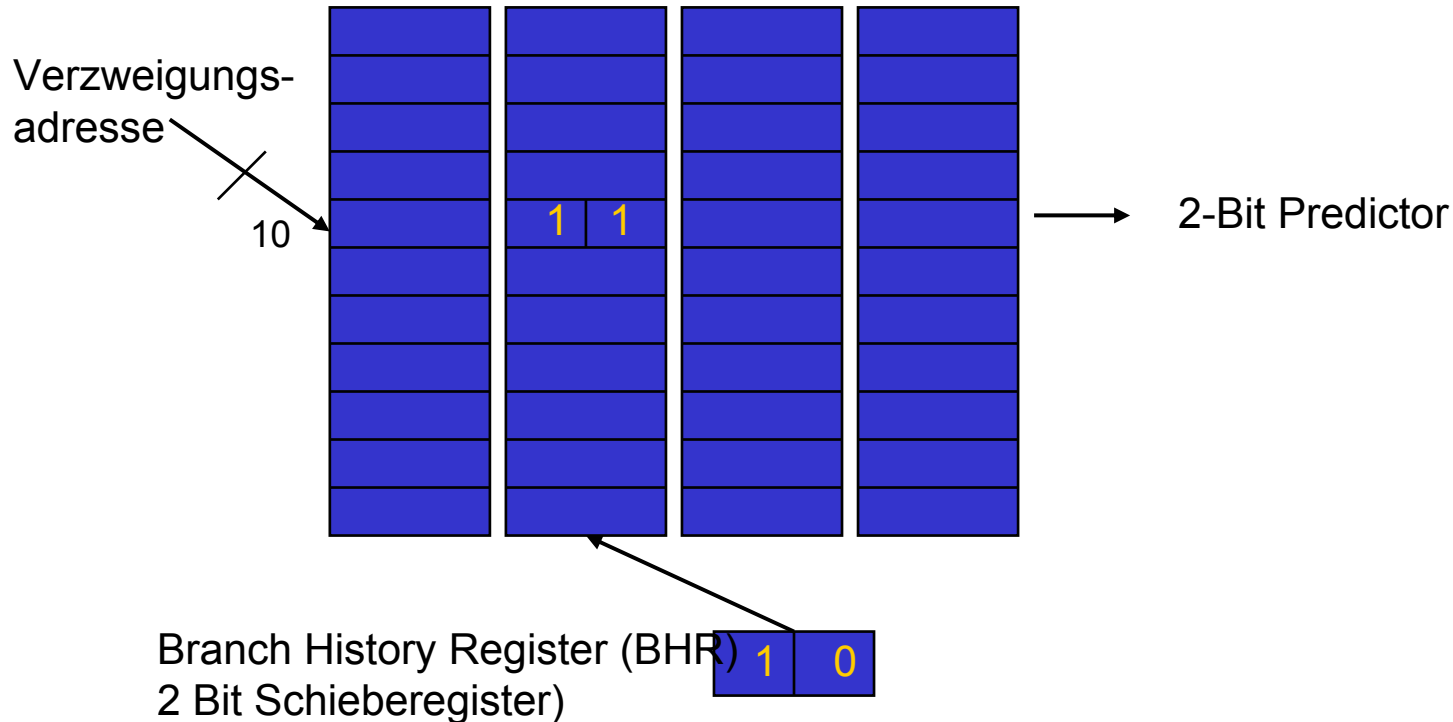
◆ Branch History Register (BHR)

- ◆ m-Bit Schieberegister
- ◆ Globale Vorgeschichte der letzten m Sprünge
- ◆ Zustand der Bits zeigt an, ob die letzten Sprünge genommen worden sind oder nicht.
- ◆ Nach jedem Sprung wird der Sprungausgang in BHR geschoben.
- ◆ Inhalt des BHR bildet Index in sogenannte **Pattern History Table (PHT)**

Dynamische Sprungvorhersage

□ Beispiel: (2,2)-Predictor:

Pattern History Tables PHTs
(2-Bit Predictors)



Zusammenfassung

❑ Statische Vorhersage

- ◆ Hardware- oder Compiler-Techniken

❑ Dynamische Vorhersage

- ◆ Berücksichtigung der Vorgeschichte
- ◆ Hohe Genauigkeit erreichbar
- ◆ Hoher Hardware-Aufwand

❑ Sprungvorhersage

- ◆ Für superskalare Prozessoren ist eine möglichst genaue Vorhersagetechnik notwendig.

Zusammenfassung

□ Dynamische Vorhersage

- ◆ Branch Target Buffer (BTB), Branch Target Address Cache (BTAC)
 - ◆ Festhalten des Sprungziels
- ◆ Branch History Table
 - ◆ Festhalten der Vorgeschichte eines Sprungbefehls
 - ◆ 1-Bit Predictor
 - ◆ 2-Bit Predictor
- ◆ (m,n) Predictors
 - ◆ Festhalten der Vorgeschichte des Sprungbefehls
 - ◆ Korrelation mit abhängigen Sprüngen

Vorlesung Rechnerarchitektur

□ Kapitel 2: Zentraleinheiten

Pipeline-Konflikte und Lösungen

Ausführung in mehreren Takten

□ Multizyklus-Befehle

- ◆ Befehle die vom einheitlichen Zeitverhalten abweichen
- ◆ Lange Ausführungszeiten
 - ◆ Beispiele: **Gleitkomma-Verarbeitung, Lade-/Speicherbefehle**
 - ◆ Forderung, dass alle Befehle in der EX-Phase einen Takt zur Ausführung benötigen, ist nicht praktikabel
 - ◆ Lösung: ein Befehl benötigt mehrere Takte in EX-Phase zur Ausführung
 - ◆ **Strukturkonflikt:**
 - ◆ Ein Befehl Inst2, der einem lange rechnenden Befehl folgt, kann die nur einfach vorhandene Ausführungseinheit nicht verwenden, bis der Konflikt aufgelöst ist.

Ausführung in mehreren Takten

□ Multizyklus-Befehle: Lösungsmöglichkeiten:

◆ Interlocking:

- ◆ Anhalten des Befehls Inst2 in der Pipeline
- ◆ Einfügen von Leerzyklen
- ◆ Zeitverlust!

◆ Organisation der Funktionseinheit in EX-Stufe als Pipeline

- ◆ Allzweck-Funktionseinheit
- ◆ Zu jedem Takt kann neuer Befehl in die Pipeline der Funktionseinheit geladen werden
- ◆ Langsame Lösung für einfache Befehle!

Ausführung in mehreren Takten

□ Multizyklus-Befehle: Lösungsmöglichkeiten:

◆ Mehrere Funktionseinheiten

- ◆ Gleichzeitige Ausführung der Befehle in voneinander unabhängigen Funktionseinheiten
- ◆ Problem: Änderung der ursprünglichen Programmordnung!
- ◆ Beispiel: WAW-Konflikt

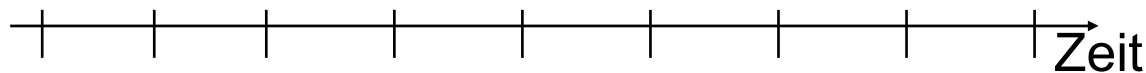
div reg3,reg11,reg12



...



mul reg3,reg1,reg2



Ausführung in mehreren Takten

□ Multizyklus-Befehle: Lösungsmöglichkeiten:

◆ Mehrere Funktionseinheiten

◆ Beispiel: Auflösung des WAW-Konflikts

- ◆ Warten des Multiplikationsbefehls mit Zurückschreiben und anschließend Überschreiben des Registers
- ◆ Verwerfen des Ergebnisses der Division
 - ◆ Achtung!
 - ◆ Gibt es eventuell weitere Abhängigkeiten
 - ◆ Präzise Interrupts (Precise Interrupts)
- ◆ Komplexe Lösungen in modernen superskalaren Mikroprozessoren

Zusammenfassung Pipelining

- ❑ Pipelining erhöht den Durchsatz
- ❑ Konflikte wirken sich auf die Leistungsfähigkeit aus!
 - ◆ Strukturkonflikte:
 - ◆ höherer Hardware-Aufwand
 - ◆ Datenkonflikte:
 - ◆ Hardwarelogik notwendig zum Erkennen und Auflösen von Konflikten
 - ◆ Steuerkonflikte:
 - ◆ Frühe Ermittlung des Verzweigungsverhaltens, Verzögerte Verzweigung, Sprungvorhersage

Zusammenfassung Pipelining

- ❑ Compiler können die Auswirkungen der Konflikte auf die Leistungsfähigkeit reduzieren: Code-Optimierung
 - ◆ Scheduling
 - ◆ Verzögerte Verzweigung
 - ◆ Statische Sprungvorhersage
- ❑ Tiefe Pipelines erhöhen den Durchsatz
 - ◆ Es ergeben sich mehr Konflikte
- ❑ Multizyklus-Operationen und Unterbrechungen erhöhen die Komplexität der Pipeline

Zusammenfassung Pipelining

❑ Merkmale moderner Mikroprozessoren:

- ◆ Ausgeprägte Pipeline-Stufen
- ◆ Superskalartechnik
- ◆ Mehrstufige Cache-Speicher
- ◆ Multiprozessor-Unterstützung

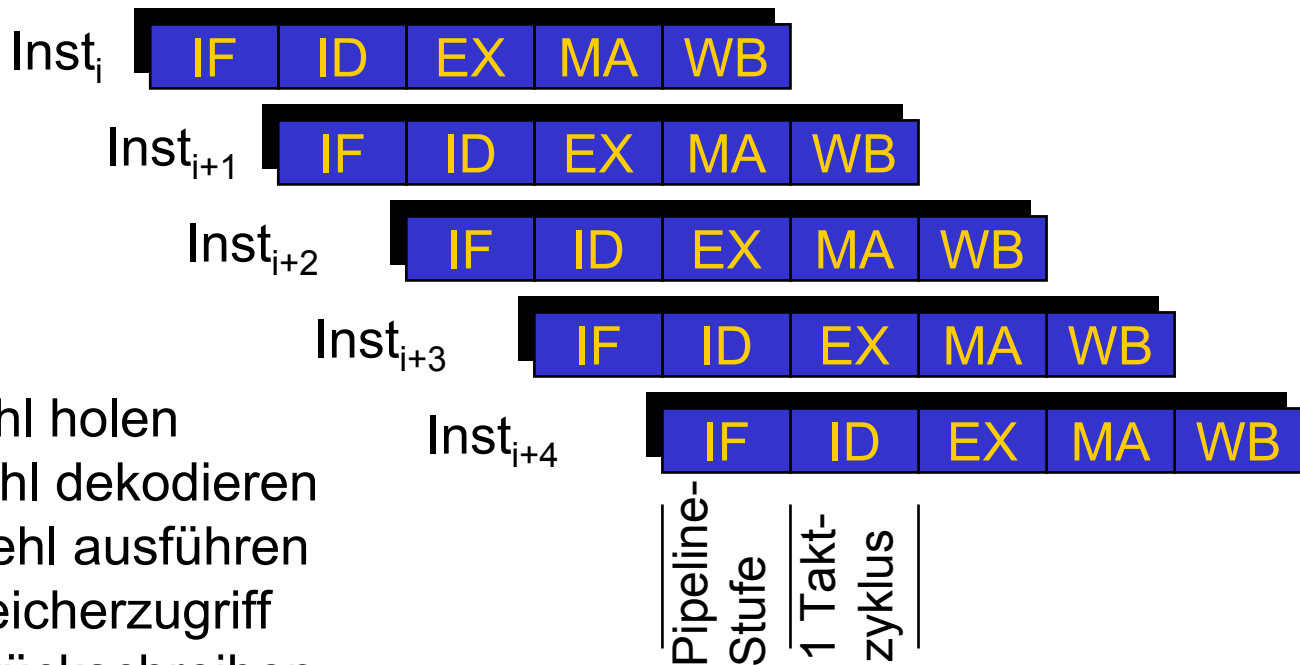
Vorlesung Rechnerarchitektur

□ Kapitel 2: Zentraleinheiten

Superskalarprinzip

Superskalarprinzip

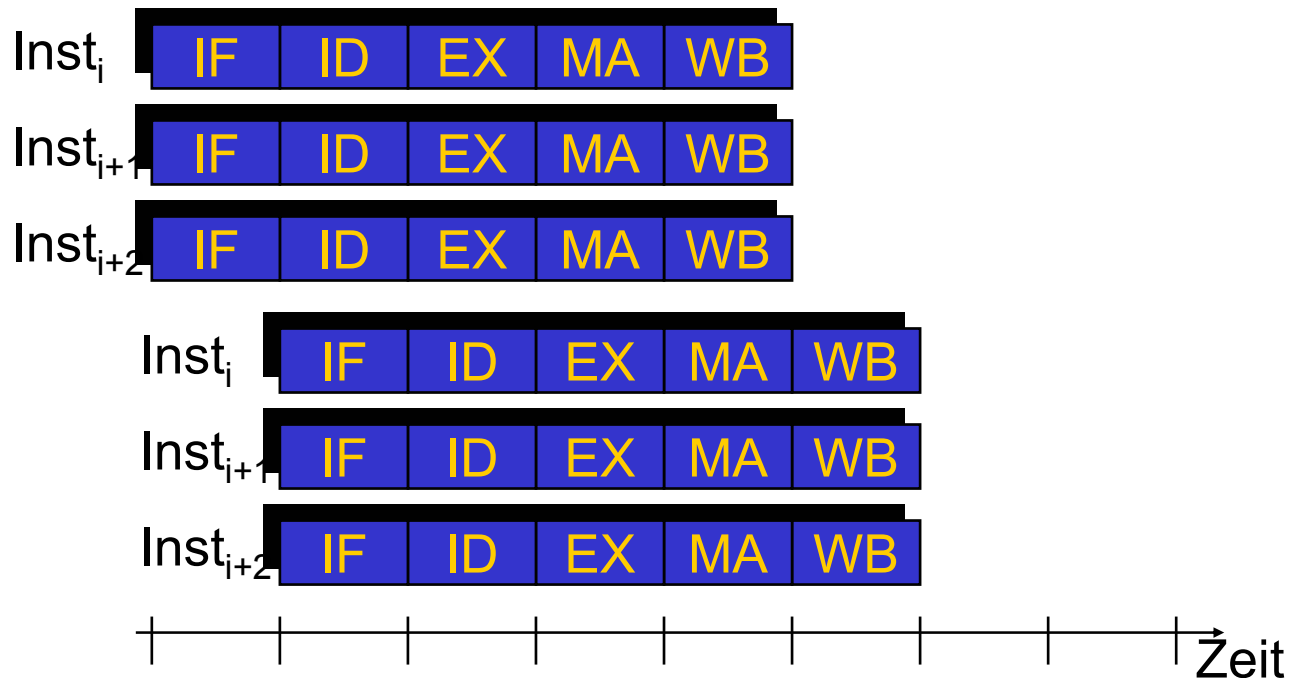
- Ausgangspunkt: Rechnerorganisation mit Merkmalen einer RISC-Architektur



Superskalarprinzip

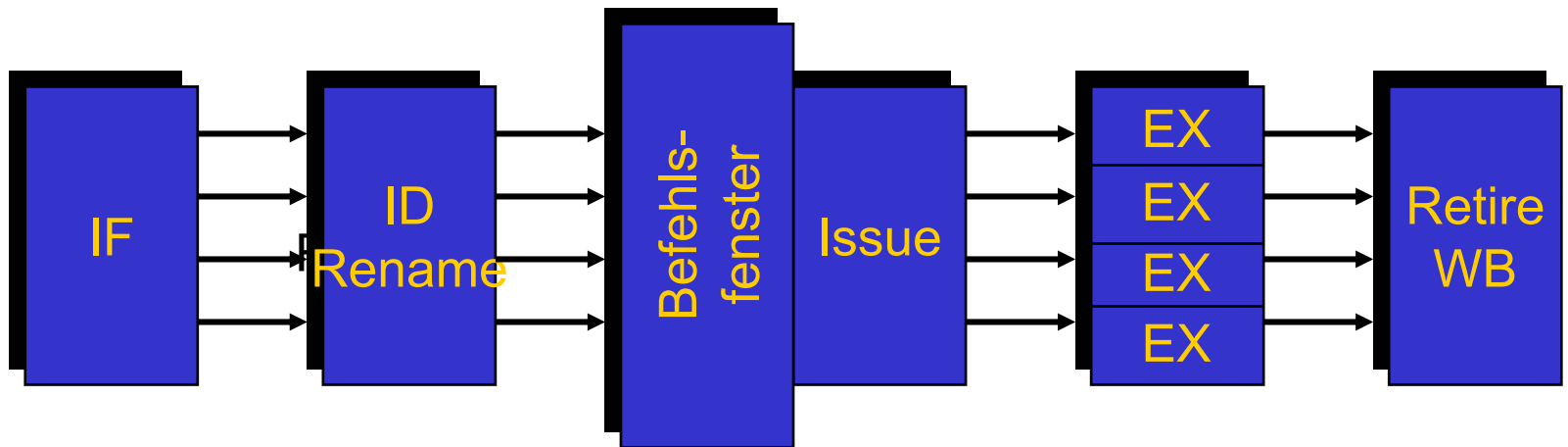
□ Erhöhung der Leistung:

- ◆ Mehrere Maschinenbefehle können gleichzeitig zur Ausführung angestoßen werden



Superskalar-Prinzip

□ Prinzipielle superskalare Prozessorpipeline



Superskalarprinzip

□ Prinzipielle superskalare Prozessorpipeline

◆ IF Phase

- ◆ Holen mehrerer Befehle in den Hole-Puffer
- ◆ Verwaltung der Komponenten für die Sprungzielvorhersage

Superskalarprinzip

□ Prinzipielle superskalare Prozessorpipeline

◆ ID Phase

- ◆ Dekodierung der Befehle
- ◆ Umbenennung von Registern
 - ◆ Abbildung von logischen Registern auf physikalische Prozessorregister
- ◆ Schreiben der Befehle in Befehlsfenster
 - ◆ Erkennen und Auflösen von Konflikten aufgrund von Daten- und Strukturabhängigkeiten

Superskalarprinzip

□ Prinzipielle superskalare Prozessorpipeline

◆ Issue

- ◆ Zuweisen von Befehlen an Ausführungseinheiten
 - ◆ In-order Issue
 - ◆ Out-of-Order Issue
- ◆ Ablegen der Befehle in Programmreihenfolge in den Reorder-Puffer
- ◆ Reservation Stations
- ◆ Weiterleiten der Befehle an Funktionseinheiten, wenn Operanden verfügbar sind.

Superskalarprinzip

□ Reservation Stations

- ◆ Puffer mit ein oder mehreren Einträgen:
 - ◆ (siehe Tomasulo Artikel, Hennessy/Patterson Buch, Beispiel folgt)
 - ◆ Jeder Eintrag kann eine Instruktion mit ihren Operanden enthalten
- ◆ Konfliktauflösung mit Hilfe von Reservation Stations
 - ◆ Befehle warten in Reservation Station bis Operanden verfügbar sind.

Superskalarprinzip

□ Dispatch

- ◆ Eine Instruktion wird aus der Reservation Station an die Funktionseinheit weitergegeben, wenn alle Operanden verfügbar sind.
- ◆ Wenn alle Operanden verfügbar sind und die Funktionseinheit nicht besetzt ist, wird die Instruktion sofort zur Ausführung angestoßen.
- ◆ Der Befehl wird in der Funktionseinheit ausgeführt.
- ◆ Eine Instruktion kann sich null oder mehrere Zyklen in einer Reservation Station aufhalten
- ◆ Dispatch und Ausführen erfolgt nicht gemäß der Programmordnung

Superskalarprinzip

□ Completion

- ◆ Eine Instruktion beendet ihre Ausführung, wenn das Ergebnis für nachfolgende Befehle bereitsteht (Forwarding, Puffer)
- ◆ Completion: Befehl 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.

Superskalarprinzip

□ Commitment

- ◆ Nach der Vervollständigung beenden die Befehle ihre Bearbeitung (Commitment).
- ◆ Bedingungen für Commitment:
 - ◆ Die Befehlsausführung ist vollständig
 - ◆ Alle Befehle, die in der Programmordnung vor dem Befehl stehen, haben bereits ihre Bearbeitung beendet (sind committed) oder beenden ihre Bearbeitung im selben Takt.
 - ◆ Der Befehl hängt von keiner Spekulation ab.
- ◆ Mit der Beendigung der Bearbeitung werden die Ergebnisse in den Architekturregistern dauerhaft gemacht, d.h. aus den internen Umbenennungsregistern (Schattenregistern) zurückgeschrieben.

Superskalarprinzip

❑ Precise Interrupts (Precise Exceptions)

- ◆ Bei einer Unterbrechung werden beenden alle Befehle, die in der Programmordnung vor dem Ereignis stehen, ihre Bearbeitung (Commitment), alle nachfolgenden werden verworfen.
- ◆ Abhängig von der Organisation und der Art der Unterbrechung beendet der die Unterbrechung verursachende Befehl die Ausführung oder er wird verworfen.
- ◆ Die verworfenen Befehle können zum gegebenen Zeitpunkt von neuem ihre Ausführung beginnen.

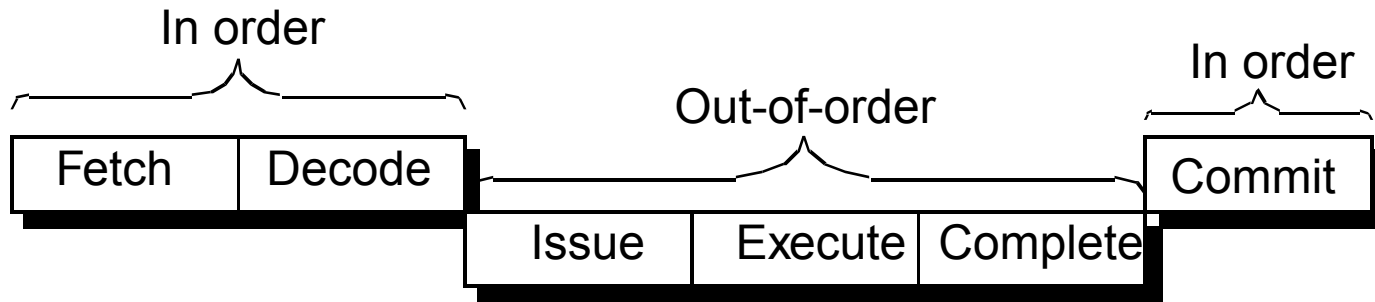
Superskalarprinzip

□ Retirement

- ◆ Freigeben eines Platzes im Reorder Buffer
 - ◆ Nach Commitment: Ergebnis ist dauerhaft
 - ◆ Nach Verwerfen eines Befehls: ohne dauerhafte Zustandsänderungen

Superskalarprinzip

□ Programmordnung

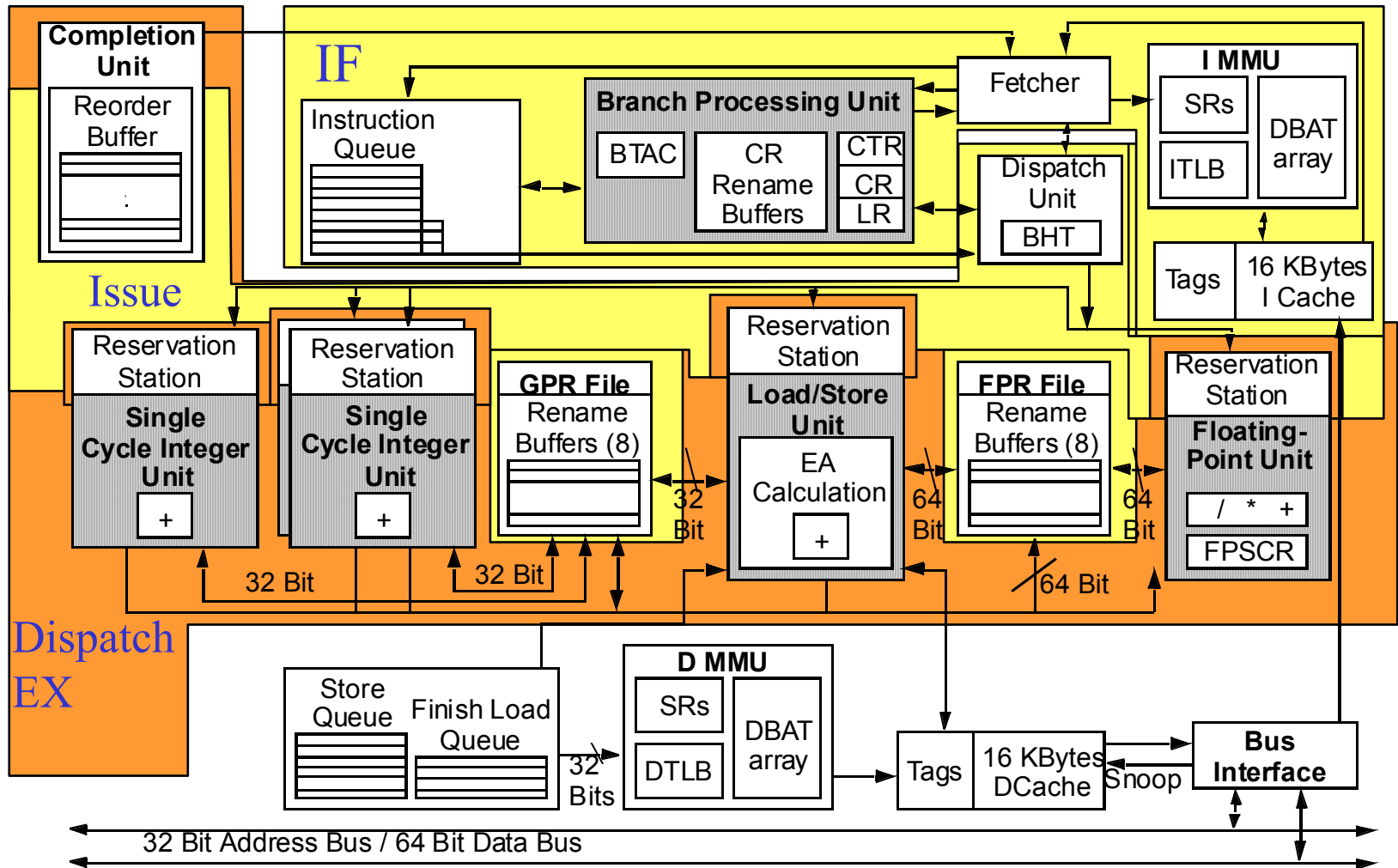


Zusammenfassung

- ❑ Aus einem sequentiellen Befehlsstrom werden Befehle zur Ausführung angestoßen (zugewiesen).
- ❑ Die Zuweisung erfolgt dynamisch durch die Hardware
- ❑ Es können 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 Funktionseinheiten sind verfügbar.
- ❑ Mikroarchitektur bestimmt superskalare Eigenschaft.

Superskalarprinzip

❑ Fallstudie: Motorola PowerPC 604



Commitment

Dispatch EX

Superskalarprinzip

- ❑ **Dynamische Methoden zur Erkennung und Auflösung von Datenkonflikten**
 - ♦ Eine effiziente Auslastung der Funktionseinheiten einer superskalaren Maschine kann nur dann erreicht werden, wenn während der einzelnen Phasen des Maschinenbefehlszyklus verschiedene Hardware-Maßnahmen bzw. vorbereitende Maßnahmen des Compilers dafür sorgen, dass genügend Maschinenbefehle (Operationen) zur gleichzeitigen Ausführung bereitstehen.

Superskalarprinzip

- ❑ **Dynamische Methoden zur Erkennung und Auflösung von Datenkonflikten**
 - ◆ Konflikte zwischen Maschinenbefehlen erzwingen
 - ◆ ein Anhalten der Pipeline und
 - ◆ die nachfolgenden Befehle müssen warten, bis sie zur Ausführung angestoßen werden können.
 - ◆ Falls der Prozessor die Möglichkeit hat, während der Bearbeitung des Konflikts weiterhin Maschinenbefehle (Operationen) zu holen und zu decodieren; können unter diesen eventuell konfliktunabhängige Befehle (Operationen) gefunden werden.

Superskalarprinzip

□ Dynamische Methoden zur Erkennung und Auflösung von Datenkonflikten

- ♦ Mit der Eigenschaft eines Prozessors über den gerade aktuellen Bearbeitungszeitpunkt vorzuschauen (**lookahead capability**) ergibt sich eine von der ursprünglich gegebene Reihenfolge geänderte Abarbeitung der Maschinenbefehle (Operationen).
- ♦ Die korrekte Semantik des Programms muss dabei gewährleistet bleiben.

Superskalarprinzip

□ Dynamische Methoden zur Erkennung und Auflösung von Datenkonflikten

◆ Scheduling:

- ◆ Verfahren, nach dem bestimmt wird, wenn eine Instruktion zur Ausführung angestoßen, die Operanden gelesen und die Ergebnisse geschrieben werden sollen.
- ◆ Ziel: Umordnen der Befehle, um Wartezyklen zu vermeiden
- ◆ Statisches und dynamisches Scheduling

Superskalarprinzip

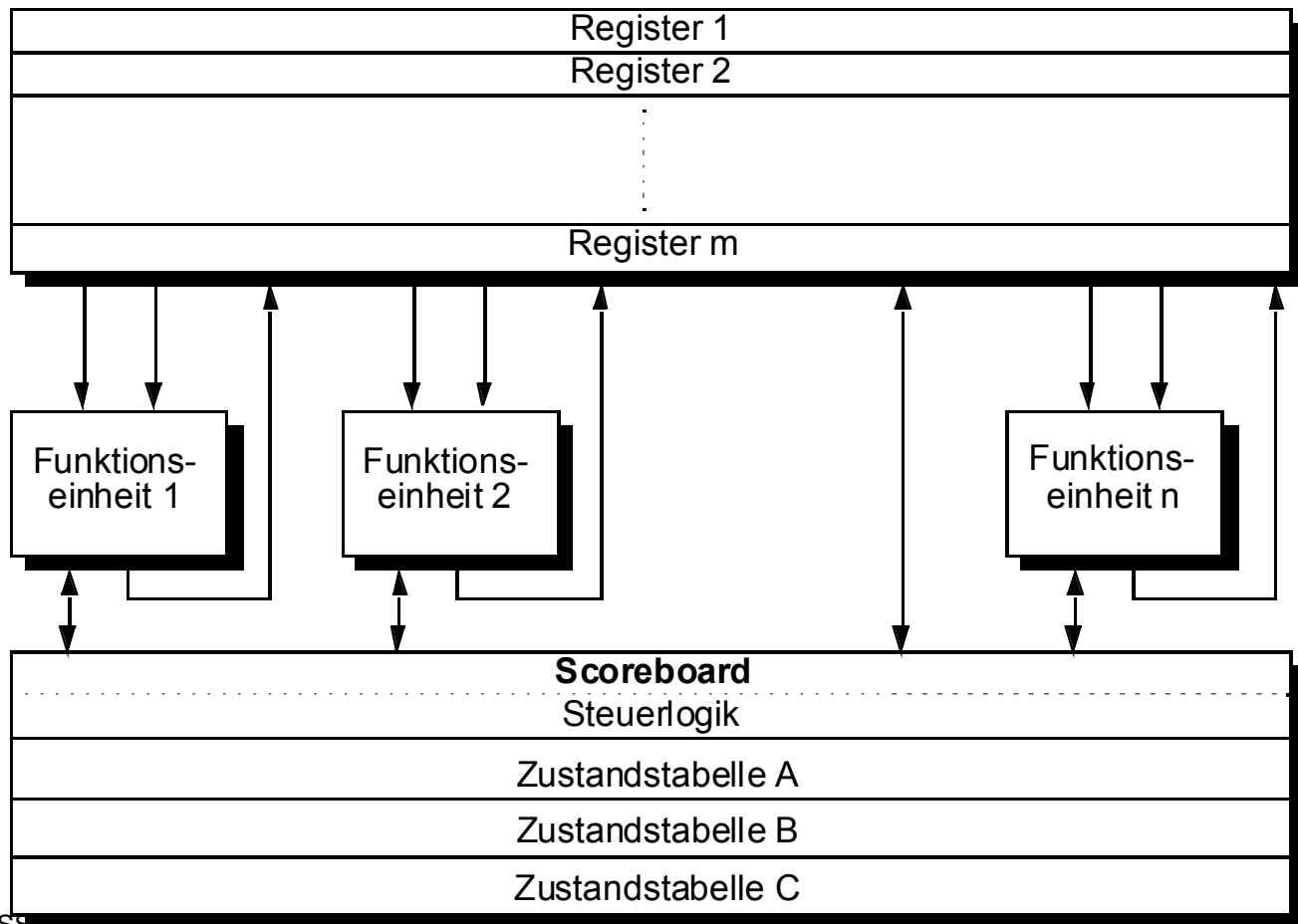
- ❑ Dynamische Methoden zur Erkennung und Auflösung von Datenkonflikten
- ❑ Fallstudie: Scoreboard (CDC 6600, Thornton 1970)
 - ◆ Idee: Maschinenbefehle möglichst früh zur Ausführung anstoßen, um so eine hohe Ausführungsrate zu erhalten.
 - ◆ Das Scoreboard erkennt mögliche Konflikte und steuert während der Decodier-, Ausführungs- und Rückschreibphasen die parallele Abarbeitung von Maschinenbefehlen.

Superskalarprinzip

- ❑ **Fallstudie: Scoreboard (CDC 6600, Thornton 1970)**
 - ◆ Für die Steuerung benötigt das Scoreboard zu jedem Zeitpunkt Informationen über:
 - ◆ den aktuellen Bearbeitungszustand der Maschinenbefehle (Phase des Maschinenbefehlszyklus, in dem sich die jeweiligen Maschinenbefehle befinden,
 - ◆ den aktuellen Ausführungszustand der einzelnen Funktionseinheiten und über
 - ◆ die Register, in welche die Funktionseinheiten ihre Ergebnisse schreiben.
 - ◆ Die Daten werden in Tabellen gehalten und nach jedem Ausführungsschritt aktualisiert.

Scoreboard

- Grundlegende Struktur einer superskalaren Maschine mit Scoreboard



Scoreboard

- ◆ Tabelle A: Bearbeitungszustand der Befehle

| | Maschinenbefehl zur Ausführung anstoßen | Operanden Lesen, RO | Ausführung Beenden (EX) | Ergebnis Zurückschreiben (WB) |
|------------|---|------------------------|----------------------------|-------------------------------------|
| Befehl i | | | | |
| Befehl i+1 | | | | |

- ◆ Tabelle B: Bearbeitungszustand der Funktionseinheiten

| | Busy | OP | Dest | Src1 | Vld1 | FU1 | Src2 | Vld2 | FU2 |
|--------------------|------|----|------|------|------|-----|------|------|-----|
| Funktionseinheit 1 | | | | | | | | | |
| Funktionseinheit 2 | | | | | | | | | |
| Funktionseinheit n | | | | | | | | | |

Scoreboard

- ◆ Tabelle C: Zustand der Ergebnisregister

| | | | | | | | | | |
|-----------------------------|----|----|--|--|--|--|--|--|----|
| Register | R1 | R2 | | | | | | | Rm |
| Nummer der Funktionseinheit | | | | | | | | | |

Scoreboard

- Informationen über den Ausführungszustand der Funktionseinheiten:
 - ♦ ein Flag, das angibt, ob die Funktionseinheit gerade mit der Ausführung einer Operation beschäftigt ist (*Busy*),
 - ♦ die auszuführende Operation (*OP*),
 - ♦ die Adresse des Zielregisters (*Dest*),
 - ♦ die Adressen der Quellregister (*Src1*, *Src2*),
 - ♦ die Nummern der Funktionseinheiten, welche die Quelloperanden liefern (*FU1*, *FU2*) und
 - ♦ zwei Flags, die anzeigen, ob die Quellregister *Src1* und *Src2* die gültigen Operanden enthalten (*Vld1*, *Vld2*)

Scoreboard

□ Funktionsweise des Scoreboards:

- ♦ Das Scoreboard beobachtet jede Veränderung in der befehlsausführenden Hardware und entscheidet mit Hilfe dieser Informationen, wann eine Funktionseinheit für einen auszuführenden Befehl die Operanden lesen, die Ausführung beginnen und die Ergebnisse zurückschreiben darf.

Scoreboard

□ Funktionsweise des Scoreboards:

- ◆ Die Ausführung eines Maschinenbefehls kann beginnen, falls
 - ◆ eine Funktionseinheit frei ist und
 - ◆ kein bereits aktiver Maschinenbefehl in dasselbe Register schreiben will.
- ◆ Ist eine der beiden Bedingungen erfüllt, können keine weiteren Maschinenbefehle zur Ausführung angestoßen werden, bis die Konfliktsituation aufgelöst ist.

Scoreboard

□ Funktionsweise des Scoreboards:

- ◆ Prüfen der Verfügbarkeit der Operanden.
- ◆ Ein Quelloperand ist verfügbar, falls kein bereits aktiver Maschinenbefehl diesen Operanden beschreiben will oder keine Funktionseinheit das Register für den Quelloperanden beschreibt.
- ◆ Das Scoreboard teilt der Funktionseinheit die Verfügbarkeit der Operanden mit und lässt diese mit der Ausführung beginnen.
- ◆ Prüfen von Anti-Datenabhängigkeiten.

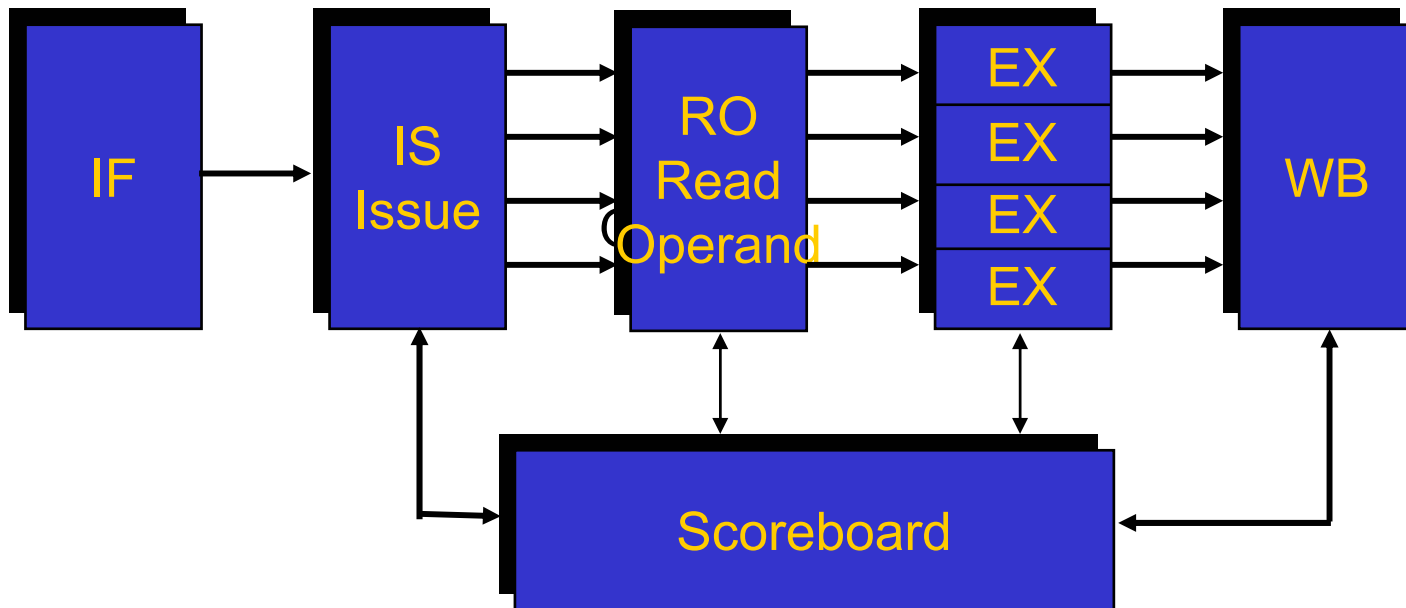
Scoreboard

□ Funktionsweise des Scoreboards:

- ◆ Dynamisches Auflösen von Konflikten aufgrund von echten Datenabhängigkeiten.
- ◆ Wenn eine Funktionseinheit ihr Ergebnis berechnet hat, teilt diese es dem Scoreboard mit und beendet seine Ausführung.
- ◆ Prüfen von Anti-Datenabhängigkeiten.

Superskalarprinzip

- ❑ Fallstudie: Scoreboard CDC 6600 (Thornton 1970)
 - ◆ Grundlegende Pipeline mit Scoreboard



Scoreboard

- ❑ **Grundlegende Struktur der Pipeline**
 - ◆ **Aufteilung der ID Phase**
 - ◆ **Issue Phase (IS)**
 - ◆ Dekodieren der Befehle und Prüfen von strukturellen und WAW Konflikten
 - ◆ **Read Operands Phase (RO)**
 - ◆ Warte bis Konflikte aufgelöst sind, dann Lesen der Operanden aus den Registern
 - ◆ **Zusätzliche Verwaltungsaufgaben für EX und WB Phasen**

Scoreboard

□ Grundlegende Struktur der Pipeline

◆ Issue Phase (IS)

- ◆ Falls kein struktureller oder WAW Konflikt zu einem Zeitpunkt gegeben ist, wird der Befehl an die Funktionseinheit weitergegeben, Aktualisieren der internen Datenstrukturen
- ◆ Sonst: Anhalten der Pipeline (in-order issue)

◆ Read Operands Phase (RO)

- ◆ Beobachten der Verfügbarkeit von Operanden
- ◆ Wenn die Operanden verfügbar sind, werden die Funktionseinheiten informiert
- ◆ Kein Forwarding
- ◆ Dynamische Auflösung von RAW Konflikten

Scoreboard

□ Grundlegende Struktur der Pipeline

◆ Ausführungsphase (EX)

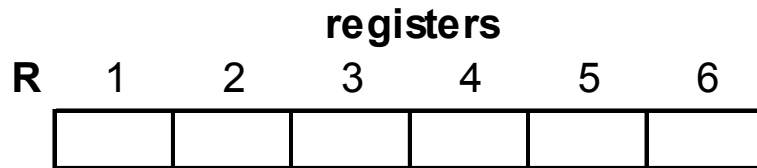
- Ausführung der Operationen und Benachrichtigung des Scoreboards, wenn das Ergebnis berechnet ist

◆ Rückschreibphase (WB)

- Wenn das Scoreboard über die Berechnung eines Operanden informiert worden ist, wird überprüft, ob ein WAW Konflikt aufgetreten ist
- Gegebenenfalls wird die Beendigung der Ausführung der Operation verzögert
- Ansonsten kann die Funktionseinheit ihr Ergebnis in das Zielregister schreiben

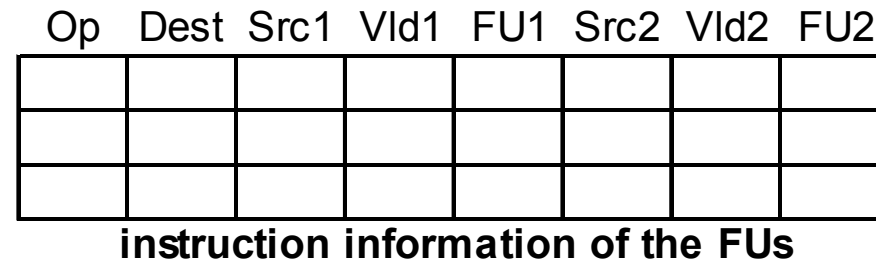
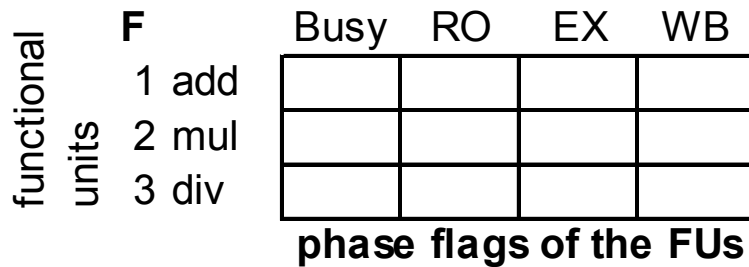
Scoreboard

□ Beispiel (T. Ungerer)



```

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



cycle 0

all fields are initialized with 0
empty fields have the value 0

| op | EX cycles |
|-----|-----------|
| add | 1 |
| sub | 1 |
| mul | 4 |
| div | 4 |

Scoreboard

□ Beispiel (T. Ungerer)

registers

| | | | | | | |
|---|---|---|---|---|---|---|
| R | 1 | 2 | 3 | 4 | 5 | 6 |
| | 2 | | | | | |

```

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

functional units

| | | | | | |
|----------|---|------|----|----|----|
| F | | Busy | RO | EX | WB |
| 1 add | | | | | |
| 2 mul | 1 | | | | |
| 3 div | | | | | |

phase flags of the FUs

| | | | | | | | |
|-----|------|------|------|-----|------|------|-----|
| Op | Dest | Src1 | Vld1 | FU1 | Src2 | Vld2 | FU2 |
| mul | 1 | 3 | 1 | | 5 | 1 | |
| | | | | | | | |

instruction information of the FUs

cycle 1

Scoreboard

□ Beispiel (T. Ungerer)

registers

| | | | | | | |
|----------|---|---|---|---|---|---|
| R | 1 | 2 | 3 | 4 | 5 | 6 |
| | 2 | 1 | | | | |

```

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

F

| | | | | | |
|-------------------------|-----|------|----|----|----|
| functional units | | Busy | RO | EX | WB |
| 1 | add | 1 | | | |
| 2 | mul | 1 | 1 | | |
| 3 | div | | | | |

phase flags of the FUs

| | | | | | | | | |
|--|-----|------|------|------|-----|------|------|-----|
| | Op | Dest | Src1 | Vld1 | FU1 | Src2 | Vld2 | FU2 |
| | sub | 2 | 4 | 1 | | 3 | 1 | |
| | mul | 1 | 3 | 1 | | 5 | 1 | |
| | | | | | | | | |

instruction information of the FUs

cycle 2

Scoreboard

□ Beispiel (T. Ungerer)

registers

| | | | | | | |
|---|---|---|---|---|---|---|
| R | 1 | 2 | 3 | 4 | 5 | 6 |
| | 2 | 1 | | | | 3 |

```

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

phase flags of the FUs

| | | | | | |
|---------------------|----------|------|----|----|----|
| functional units | F | Busy | RO | EX | WB |
| | 1 add | 1 | 1 | | |
| | 2 mul | 1 | 1 | | |
| | 3 div | 1 | | | |

instruction information of the FUs

| | | | | | | | |
|-----|------|------|------|-----|------|------|-----|
| Op | Dest | Src1 | Vld1 | FU1 | Src2 | Vld2 | FU2 |
| sub | 2 | 4 | 1 | | 3 | 1 | |
| mul | 1 | 3 | 1 | | 5 | 1 | |
| div | 6 | 1 | | 2 | 4 | 1 | |

cycle 3

remaining cycles in EX

mul takes 4 cycles in EX

Scoreboard

□ 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 |
| | 2 | 1 | | | | 3 |

functional units

| F | | Busy | RO | EX | WB |
|---|-----|------|----|----|----|
| 1 | add | 1 | 1 | 1 | |
| 2 | mul | 1 | 1 | | |
| 3 | div | 1 | | | |

phase flags of the FUs

| Op | Dest | Src1 | Vld1 | FU1 | Src2 | Vld2 | FU2 |
|-----|------|------|------|-----|------|------|-----|
| sub | 2 | 4 | 1 | | 3 | 1 | |
| mul | 1 | 3 | 1 | | 5 | 1 | |
| div | 6 | 1 | | 2 | 4 | 1 | |

instruction information of the FUs

cycle 4

Scoreboard

□ 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 |
|---|---|---|---|---|---|---|
| | 2 | | | | | 3 |

functional units

| F | Busy | RO | EX | WB |
|-------|------|----|----|----|
| 1 add | | 1 | 1 | 1 |
| 2 mul | 1 | 1 | | |
| 3 div | 1 | | | |

phase flags of the FUs

| Op | Dest | Src1 | Vld1 | FU1 | Src2 | Vld2 | FU2 |
|-----|------|------|------|-----|------|------|-----|
| | | | | | | | |
| mul | 1 | 3 | 1 | | 5 | 1 | |
| div | 6 | 1 | | 2 | 4 | 1 | |

instruction information of the FUs

cycle 5

Scoreboard

□ Beispiel (T. Ungerer)

registers

| R | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| | 2 | | | 1 | | 3 |

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

functional units

| F | Busy | RO | EX | WB |
|-------|------|----|----|----|
| 1 add | 1 | | | |
| 2 mul | 1 | 1 | 1 | |
| 3 div | 1 | | | |

phase flags of the FUs

| Op | Dest | Src1 | Vld1 | FU1 | Src2 | Vld2 | FU2 |
|-----|------|------|------|-----|------|------|-----|
| add | 4 | 2 | 1 | | 3 | 1 | |
| mul | 1 | 3 | 1 | | 5 | 1 | |
| div | 6 | 1 | | 2 | 4 | 1 | |

instruction information of the FUs

cycle

6

Scoreboard

□ 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 |
| | | | | 1 | | 3 |

functional units

| F | | Busy | RO | EX | WB |
|---|-----|------|----|----|----|
| 1 | add | 1 | 1 | | |
| 2 | mul | | 1 | 1 | 1 |
| 3 | div | 1 | | | |

phase flags of the FUs

| Op | Dest | Src1 | Vld1 | FU1 | Src2 | Vld2 | FU2 |
|-----|------|------|------|-----|------|------|-----|
| add | 4 | 2 | 1 | | 3 | 1 | |
| | | | | | | | |
| div | 6 | 1 | 1 | | 4 | 1 | |

instruction information of the FUs

cycle 7

Scoreboard

□ 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 |
| | | | | 1 | | 3 |

| | | | | | |
|------------------|----------|------|----|----|----|
| functional units | F | Busy | RO | EX | WB |
| | 1 add | 1 | 1 | 1 | |
| | 2 mul | | 1 | 1 | 1 |
| | 3 div | 1 | 1 | | |

phase flags of the FUs

| | | | | | | | |
|-----|------|------|------|-----|------|------|-----|
| Op | Dest | Src1 | Vld1 | FU1 | Src2 | Vld2 | FU2 |
| add | 4 | 2 | 1 | | 3 | 1 | |
| | | | | | | | |
| div | 6 | 1 | 1 | | 4 | 1 | |

instruction information of the FUs

cycle 8

Scoreboard

□ Beispiel (T. Ungerer)

registers

| | | | | | | |
|---|---|---|---|---|---|---|
| R | 1 | 2 | 3 | 4 | 5 | 6 |
| | | | | | | 3 |

```

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

functional units

| F | | Busy | RO | EX | WB | |
|---|-----|------|----|----|----|---|
| 1 | add | | 1 | 1 | 1 | |
| 2 | mul | | 1 | 1 | 1 | |
| 3 | div | 1 | 1 | | | 3 |

phase flags of the FUs

| Op | Dest | Src1 | Vld1 | FU1 | Src2 | Vld2 | FU2 |
|-----|------|------|------|-----|------|------|-----|
| | | | | | | | |
| | | | | | | | |
| div | 6 | 1 | 1 | | 4 | 1 | |

instruction information of the FUs

cycle 9

Scoreboard

□ Beispiel (T. Ungerer)

registers

| | | | | | | |
|---|---|---|---|---|---|---|
| R | 1 | 2 | 3 | 4 | 5 | 6 |
| | | | | | | 3 |

```

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

functional units

| F | Busy | RO | EX | WB |
|-------|------|----|----|----|
| 1 add | | 1 | 1 | 1 |
| 2 mul | | 1 | 1 | 1 |
| 3 div | 1 | 1 | | |

phase flags of the FUs

| Op | Dest | Src1 | Vld1 | FU1 | Src2 | Vld2 | FU2 |
|-----|------|------|------|-----|------|------|-----|
| | | | | | | | |
| | | | | | | | |
| div | 6 | 1 | 1 | | 4 | 1 | |

instruction information of the FUs

cycle 10

Scoreboard

□ 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 |
|---|---|---|---|---|---|---|
| | | | | | | 3 |

functional units

| F | | Busy | RO | EX | WB | |
|---|-----|------|----|----|----|---|
| 1 | add | | 1 | 1 | 1 | |
| 2 | mul | | 1 | 1 | 1 | |
| 3 | div | 1 | 1 | | | 1 |

phase flags of the FUs

| Op | Dest | Src1 | Vld1 | FU1 | Src2 | Vld2 | FU2 |
|-----|------|------|------|-----|------|------|-----|
| | | | | | | | |
| | | | | | | | |
| div | 6 | 1 | 1 | | 4 | 1 | |

instruction information of the FUs

cycle 11

Scoreboard

□ Beispiel (T. Ungerer)

registers

| R | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| | | | | | | 3 |

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

| F | Busy | RO | EX | WB |
|-------|------|----|----|----|
| 1 add | | 1 | 1 | 1 |
| 2 mul | | 1 | 1 | 1 |
| 3 div | 1 | 1 | 1 | |

phase flags of the FUs

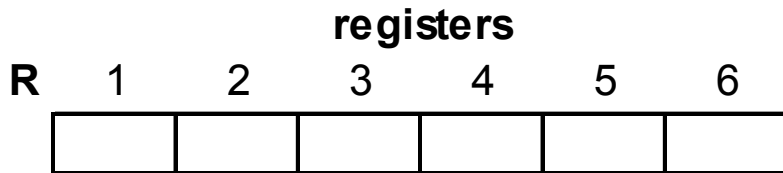
| Op | Dest | Src1 | Vld1 | FU1 | Src2 | Vld2 | FU2 |
|-----|------|------|------|-----|------|------|-----|
| | | | | | | | |
| | | | | | | | |
| div | 6 | 1 | 1 | | 4 | 1 | |

instruction information of the FUs

functional units 3
 cycle 12

Scoreboard

□ Beispiel (T. Ungerer)



```

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

| F | | Busy | RO | EX | WB |
|----------|-----|------|----|----|----|
| 1 | add | | 1 | 1 | 1 |
| 2 | mul | | 1 | 1 | 1 |
| 3 | div | | 1 | 1 | 1 |

phase flags of the FUs

| Op | Dest | Src1 | Vld1 | FU1 | Src2 | Vld2 | FU2 |
|----|------|------|------|-----|------|------|-----|
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

instruction information of the FUs

ycle 13

Reservierungstabellen

- Dynamische Methoden zur Erkennung und Auflösung von Konflikten
 - ◆ Tomaluso-Algorithmus (IBM 360/90)
 - ◆ Unterschied zu Scoreboard:
 - ◆ Konfliktauflösung und Ablaufsteuerung verteilt;
 - ◆ jede Funktionseinheit verfügt über eine Reservierungstabelle mit möglicherweise mehreren Zeilen (Einträgen);
 - ◆ Reservierungstabelle übernimmt die Kontrolle über die Abarbeitung eines Maschinenbefehls, wenn dieser von der Decodiereinheit zur Ausführung angestoßen wird;
 - ◆ 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;

Reservierungstabellen

- Tomaluso-Algorithmus (IBM 360/90)
 - ◆ Einträge in Reservierungstabellen

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

- ◆ 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

Reservierungstabellen

- Tomaluso-Algorithmus (IBM 360/90)
 - ◆ Einträge in Reservierungstabellen
 - ◆ Jeder Eintrag enthält jeweils Felder für:
 - einen Namen (destination tag) für das Ziel (**Dest**).
 - ◆ 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.

Reservierungstabellen

□ Tomaluso-Algorithmus (IBM 360/90)

◆ Einträge in Reservierungstabellen

- ◆ 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 der Reservierungstabelle kopiert.

Reservierungstabellen

□ Tomaluso-Algorithmus (IBM 360/90)

◆ Phasen der Pipeline

◆ Issue

- ◆ Holen der Befehle aus Instruction Queue
- ◆ Holen der Operanden

◆ Ausführung

- ◆ Wenn die Operanden verfügbar sind, dann Anstoßen der Ausführung auf Funktionseinheit
- ◆ Beobachten Ergebnisbus (Überprüfen von RAW Konflikten)
- ◆ Out-of-Order

◆ Rückschreibphase

- ◆ Schreiben der Ergebnisse auf Ergebnisbus
- ◆ Kennzeichnen der Reservierungseinheit als verfügbar

Reservierungstabellen

□ 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 | | 1 | 1 | 1 | 1 | 1 | 1 |
| RS | | 0 | 0 | 0 | 0 | 0 | 0 |

register status

| | | 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

Reservierungstabellen

□ 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 | | | | | | | | | | |
| | 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

Reservierungstabellen

□ 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

| | | 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