Superskalare Architekturen (2)

Parallele Ausführung

- Zwei Pipelines (U, V); abhängig
- Zwei Pipelines; unabhängig
  ⇒ out of order completion
- Superskalare Pipelines allgemein: unabhängig und evtl. mehr als zwei

  ⇒ Es kann (wie bei zwei unabhäng. Pipelines) verschiedene Datenabhängigkeiten geben
- Ziel ist immer: Ergebnis muss identisch mit dem bei sequentieller Ausführung sein

Datenabhängigkeiten (1)

- Keine Abhängigkeit:

```
30    FMUL  temp1,xk,xk      x_k^2
31    FMUL  temp2,yk,yk      y_k^2
```

  (kann man einfach vertauschen)

- **Read-after-Write**-Abhängigkeit (**RAW**):

```
50    ADD   temp2,temp2,bildx
51    STBU  k,    temp1,temp2
```

  tauschen verboten (sonst wird in #51 der Wert an der falschen Adresse gespeichert)

Datenabhängigkeiten (2)

- **Write-after-Read**-Abhängigkeit (**WAR**):

```
35    FMUL  yk,   xk,yk
36    SETH  xk,   #4000      2,0 (Gleitkommawert!)
37    FMUL  yk,   yk,xk      2 * y_k
```

  - #36 muss zwischen #35 und #37 bleiben
  - #35/#36 tauschbar, falls anderes Register verwendet wird
  - auch: **Antidependence**, „falsche" Abhängigkeit
  - außerdem: **Name Dependence**

Datenabhängigkeiten (3)

- **Write-after-Write**-Abhängigkeit (**WAW**):

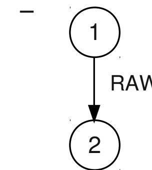
```
37      FMUL  yk,    yk,xk     2 * yk
38      FADD  yk,    yk,q
```

  - Beide Befehle schreiben yk. Würde #38 vor #37 fertiggestellt (Write-Back-Phase), wäre das Ergebnis falsch
  - auch: **Output Dependence**
  - außerdem: **Name Dependence**

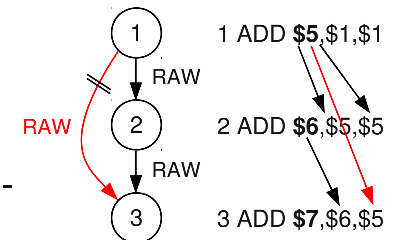
Datenabhängigkeiten, Übersicht

RAW	**Read-After-Write** true dependence	Befehl *B* verwendet das Ergebnis von Befehl *A* (Vorgänger). Konflikt, falls Ergebnis noch nicht da
WAR	**Write-After-Read** antidependence (name dependence)	*B* überschreibt Wert, den der Vorgänger *A* benötigt. Konflikt, falls *B* schreibt, bevor *A* liest
WAW	**Write-After-Write** output dependence (name dependence)	*A* und *B* schreiben in dasselbe Register (oder dieselbe Speicherzelle) Konflikt, wenn *B* vor *A* schreibt
RAR	**Read-After-Read**	*A* und *B* lesen denselben Wert; kein Konflikt

Abhängigkeitsgraph (1)

- Abhängigkeiten grafisch darstellen
  - 

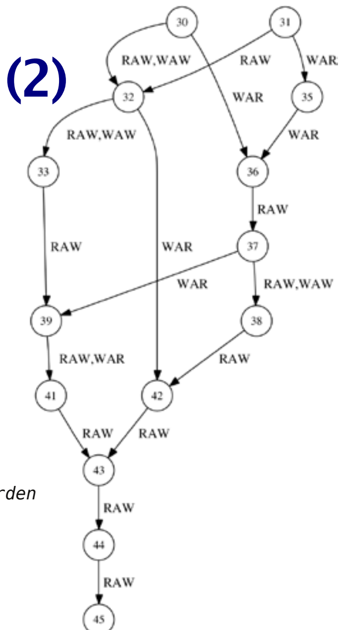
bedeutet eine RAW-Abhängigkeit zwischen Instruktionen 1 und 2, also: (2) liest Wert, den (1) schreibt
  - mehrere Abhängigkeiten gleichzeitig: alle angeben (z. B. „RAW, WAW“)
  - keine transitiven Abhängigkeiten einzeichnen



Abhängigkeitsgraph (2)

Auszug aus Mandelbrot-Programm (kompletter Code: Foliensatz 7)

```
30      FMUL   temp1,xk,xk          xk²
31      FMUL   temp2,yk,yk          yk²
32      FSUB   temp1,temp1,temp2
33      FADD   temp1,temp1,p        xk+1
34  * yk+1=2xkyk+q
35      FMUL   yk,    xk,yk
36      SETH   xk,    #4000         2,0 (Gleit-
                                    kommawert!)
37      FMUL   yk,    yk,xk         2 * yk
38      FADD   yk,    yk,q
39      SET    xk,    temp1         xk kann über-
                                    schrieben werden
40  * r=xk+1²+yk+1²
41      FMUL   temp1,xk,xk
42      FMUL   temp2,yk,yk
43      FADD   r,     temp1,temp2
44      FCMP   test,  r,:M
45      BNP           test, 2F
```

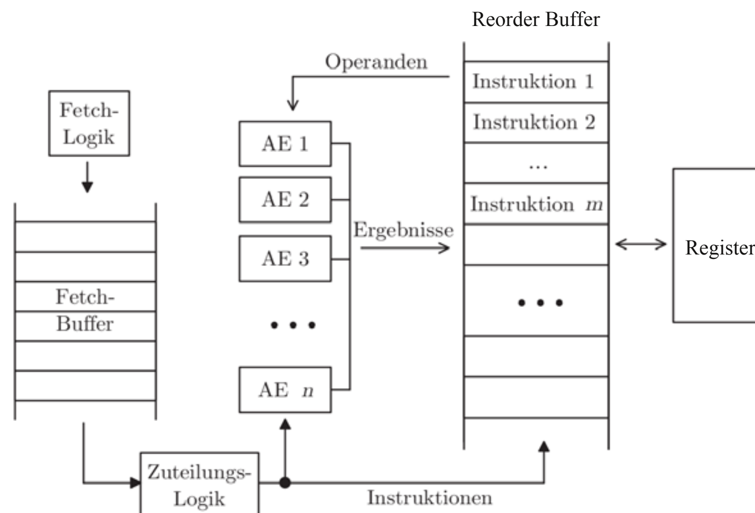


Phasen superskalarer Pipelines

- **Instruction Fetch:** Es steht ein Puffer zur Verfügung (Fetch buffer), in dem mehrere Instruktionen bereit stehen können.
- **Instruction Issue:** Instruktionen werden der Reihe nach (<u>in order</u>) auf freie Ausführungseinheiten (functional units, FUs) verteilt („issued“).

  Dieser Prozess stoppt, falls keine freien FUs vorhanden sind.
- **Read Operands:** Instruktionen warten ggf. auf ihre Operanden (RAW); ab hier ist <u>Überholen</u> möglich.
- **Execute:** <u>out of order</u> (OOO). Instruktionen warten, bis alle WAR-Vorgänger fertig sind.
- **Write Back:** Endgültiges Zurückschreiben <u>in order</u>: Write-after-Write (WAW) beachten. Ergebnisse stehen allerdings schon vorher zur Verfügung (Forwarding).

Reorder Buffer



Reorder Buffer

1. Freie Ausführungseinheit suchen.
   Falls verfügbar, durch den Befehl belegen und einen Eintrag im **Reo**rder **B**uffer (**ROB**) anlegen
2. Operanden bereitstellen. Für jeden Operanden den ROB von unten nach oben durchsuchen:
   - Wert aus Register (kein Ergebnis eines Bef. im ROB) *oder*
   - Ergebnis eines vorangehenden (fertigen) Befehls *oder*
   - vorangehenden Befehl beobachten, falls er noch nicht fertig ausgeführt ist (Pause!)
3. Befehl ausführen. Ergebnis im Eintrag des Befehls im ROB vermerken
4. Befehl bestätigen (commit): Ergebnisse in die Register übernehmen

Aufbau des Reorder Buffer

t_0			Operanden				
Befehl	unit	Status	Q_Y	Q_Z	V_Y	V_Z	Ergebnis
FMUL yk,yk,xk	MUL	▶	\$4	\$3	—	—	\$4 (ungültig)
FADD yk,yk,q	FPU	❚❚	—	\$2	FMUL	—	\$4 (ungültig)

q = \$2
xk = \$3
yk = \$4

- Befehl
- Unit (welche Ausführungseinheit?)
- Status: ▶ = läuft, ❚❚ = Pause, ✔ = fertig
- Operanden: Q_Y, Q_Z: Werte der zwei Operanden, evtl. noch nicht verfügbar
- V_Y, V_Z: ggf. Zeiger auf ROB-Eintrag ⇒ dann Q_Y bzw. Q_Z bereits (im ROB) verfügbar
- Ziel

Beispiel: Reorder Buffer (1)

- wieder unser Mandelbrot-Programm; diesmal Zeilen 34-41

```
34  * y_{k+1} = 2 x_k y_k + q
35  FMUL yk,xy,yk
36  SETH xk,#4000
37  FMUL yk,yk,xk
38  FADD yk,yk,q
39  SET  xk,temp1
40  * r = x_{k+1}^2 + y_{k+1}^2
41  FMUL temp1,xk,xk
```

(Zeilen 37–41:) das schauen wir uns an ...

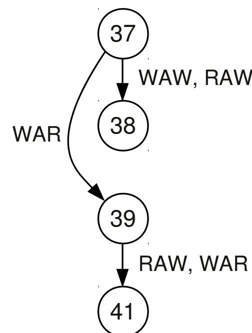
Legende

```
q     = $2
xk    = $3
yk    = $4
temp1 = $10
```


Beispiel: Reorder Buffer (2)

- Abhängigkeitsgraph:

```
37  FMUL yk,    yk,xk
38  FADD yk,    yk,q
39  SET  xk,    temp1
40
41  FMUL temp1,xk,xk
```



Beispiel: Reorder Buffer (3)

Annahmen:

- 1 FPU (alle Floating-Point-Ops außer FMUL)
- 2 Integer-Einheiten (INT1, INT2), alles außer Multiplikation
- 1 MUL (FP- und Integer-Multiplikation)
- FPU und MUL: je 4 Takte, INT: je 1 Takt
- Reorder Buffer hat Platz für 4 Einträge
- Je Takt max. 2 Befehle zuteilen (issue) und max. 2 Befehle bestätigen (commit)

(4)

`q = $2,  xk = $3,  yk = $4,  temp1 = $10`

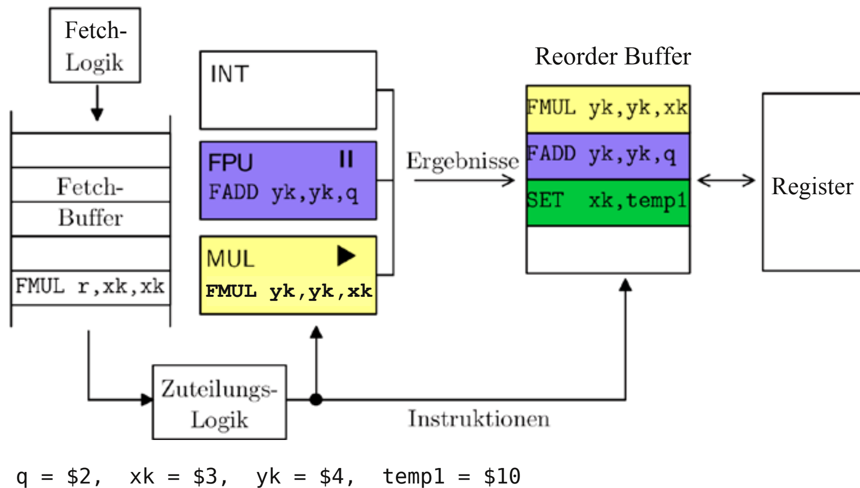
t_0			Operanden				
Befehl	unit	Status	Q_Y	Q_Z	V_Y	V_Z	Ergebnis
FMUL yk,yk,xk	MUL	▶	$4	$3	—	—	$4 (ungültig)
FADD yk,yk,q	FPU	❚❚	—	$2	FMUL	—	$4 (ungültig)
—							
—							

t_1			Operanden				
Befehl	unit	Status	Q_Y	Q_Z	V_Y	V_Z	Ergebnis
FMUL yk,yk,xk	MUL	▶	$4	$3	—	—	$4 (ungültig)
FADD yk,yk,q	FPU	❚❚	—	$2	FMUL	—	$4 (ungültig)
SET xk,temp1	INT1	▶	$10	×	—	—	$3 (ungültig)
—							

t_2 und t_3			Operanden				
Befehl	unit	Status	Q_Y	Q_Z	V_Y	V_Z	Ergebnis
FMUL yk,yk,xk	MUL	▶	$4	$3	—	—	$4 (ungültig)
FADD yk,yk,q	FPU	❚❚	—	$2	FMUL	—	$4 (ungültig)
SET xk,temp1	—	✔	$10	×	—	—	$3
—							

(5)

zu den Taktzeitpunkten t_2 und t_3:




```
q = $2,  xk = $3,  yk = $4,  temp1 = $10
```


(6)

```
q = $2,  xk = $3,  yk = $4,  temp1 = $10
```

t_4			Operanden				
Befehl	unit	Status	Q_Y	Q_Z	V_Y	V_Z	Ergebnis
FMUL yk,yk,xk	—	✔	$4	$3	—	—	$4
FADD yk,yk,q	FPU	▶	$4*	$2	—	—	$4 (ung.)
SET xk,temp1	INT1	✔	$10	×	—	—	$3
FMUL temp1,xk,xk	MUL	▶	$3*	$3*	—	—	—

t_5			Operanden				
Befehl	unit	Status	Q_Y	Q_Z	V_Y	V_Z	Ergebnis
FADD yk,yk,q	FPU	▶	$4*	$2	—	—	$4 (ung.)
SET xk,temp1	—	✔	$10	×	—	—	$3
FMUL temp1,xk,xk	MUL	▶	$3*	$3*	—	—	$10 (ung.)
—							

*) Wert aus ROB übernommen (Forwarding)

Beispiel: Reorder Buffer (7)

Klassisch:

			0	1	2	3	4	5	6	7	8	
FMUL **yk**, yk,xk	F	D	Xm	Xm	Xm	Xm	W					(Xm: 0-3)
FADD **yk**, yk,q	F	D	-	-	-	-	Xf	Xf	Xf	Xf	W	(Xf: 4-7)
SET **xk**, temp1		F	D	Xi	W							(Xi: 1)
FMUL **temp1**,xk,xk		F	D	-	-	-	Xm	Xm	Xm	Xm	W	(Xm: 4-7)

Leicht geändertes Programm:

			0	1	2	3	4	5	6	7	8		
FMUL **yk**, yk,xk	F	D	Xm	Xm	Xm	Xm	W						(Xm: 0-3)
FADD **yk**, yk,q	F	D	-	-	-	-	Xf	Xf	Xf	Xf	W		(Xf: 4-7)
SET **yk**, temp1		F	D	-	-	-	-	-	-	-	Xi	W	(Xi: 8)
FMUL **temp1**,yk,yk		F	D	-	-	-	-	-	-	-	-	Xm	(Xm: 9-)

Register Renaming (1)

- Jeder Eintrag im Reorder Buffer benötigt Register zum Speichern von Operanden und spekulativen (= noch nicht bestätigten) Ergebnissen.
- Diese heißen **Schattenregister** oder **Rename Registers** (im Unterschied zu den **Befehls-satz-Registern**; **Architectural Registers**)
- Es ist ökonomisch sinnvoll, lediglich Pointer auf Rename Register in einem Register-Pool (für Zwischenergebnisse) zu speichern.

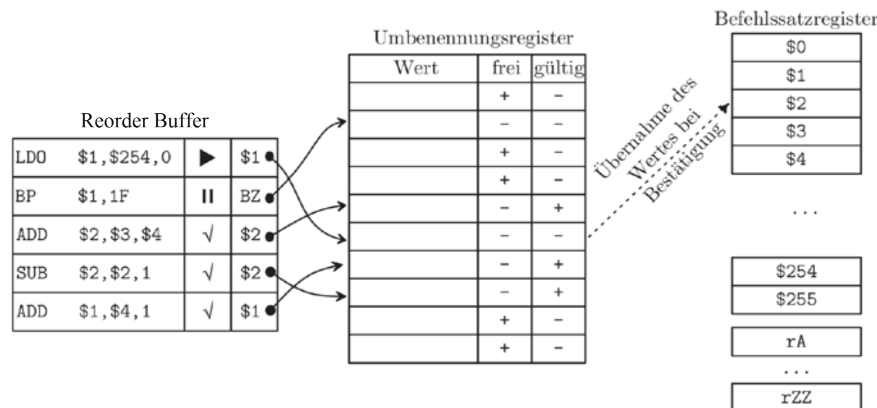
Register Renaming (2)

Implizite Umbenennung

- Zwei getrennte Sätze physischer Register für Befehlssatz-Register und Rename Register.
- Es ist nirgends explizit vermerkt, wo sich der jüngste spekulative Wert eines Registers befindet. Zum Auffinden des Wertes muss also der ROB durchsucht werden.
- Das Verwerfen spekulativer Werte (bei Interrupts oder falsch vorhergesagten Sprüungen) ist einfach.

Register Renaming (3)

Implizite Umbenennung



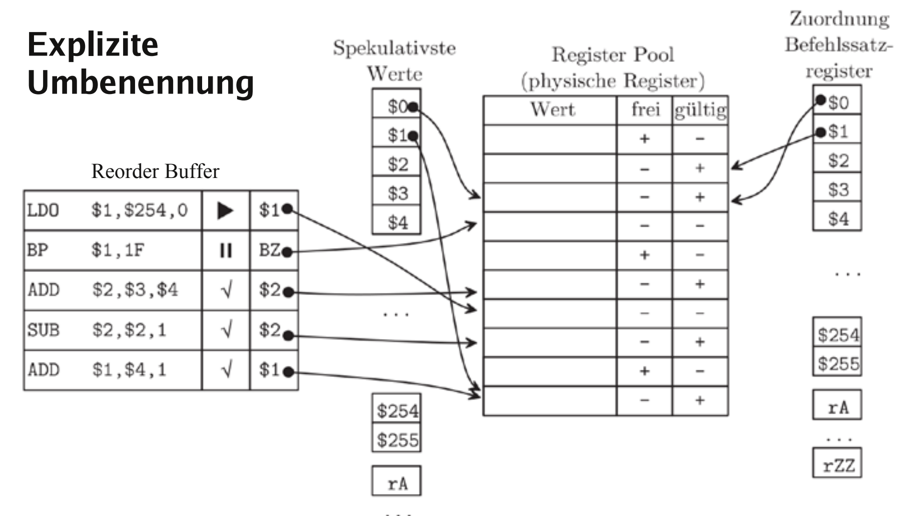
Register Renaming (4)

Explizite Umbenennung

- Es gibt nur einen Register-Pool (gemeinsam für Befehlssatz- und Rename-Register)
- Eine Tabelle gibt an, wo die jüngsten (spekulativsten) Werte der Register stehen
- Kein Umkopieren der Register nach Bestätigen eines Befehls;
  Pointer auf die physischen Register beschreiben Registerzustand

Register Renaming (5)

Explizite Umbenennung



Register Renaming (6)

Datenabhängigkeiten

- **Read-after-Write** (**RAW**):
  Führt stets zum Stillstand, bis das Ergebnis zur Verfügung steht.
- **Write-after-Read** (**WAR**) und
  Write-after-Write (**WAW**):
  Schattenregister können Stillstand verhindern (sofern genügend freie davon zur Verfügung stehen)

Vorschau 02.12.2010

- Superskalare Architekturen (Fortsetzung),
  „Real-World"-Pipeline (Pentium Pro)

- **Heute wieder Übung im Hörsaal 2.007 (Übungen ohne PC), nur 13:30 und 15:15**

- Hinweis: **Die Übungen am 02.12. entfallen wegen der studentischen Vollversammlung.**