Zentralübung Rechnerstrukturen im SS2009

Prozessorarchitektur und Parallelismus auf Befehlsebene

Fabian Nowak

Universität Karlsruhe (TH)
Forschungsuniversität • gegründet 1825

Institut für Technische Informatik
Lehrstuhl für Rechnerarchitektur

3. Juni 2009

Inhalt

- Ausführungseinheiten, Leistung
- Pipelining
- Sprungvorhersage
- Befehlsausführung mit Prädikaten
- Superskalartechnik: Algorithmus von Tomasulo

Aufgabe 1.1: Leistungsbewertung

Moderne Mikroprozessoren – Leistungsbewertung I

„Wie definieren sich Latenz und Durchsatz? Geben Sie Latenz und maximalen Durchsatz bei einer 7-stufigen Pipeline und 5 maximal gleichzeitig in der Pipeline befindlichen Befehlen an.“

Latenz und Durchsatz

- Latenz = Wartezeit aus der Sicht eines darauffolgenden Befehls = Bearbeitungsdauer - 1
  $T_{\text{Latenz}} = \#\text{Pipeline-Stufen} - 1$
  $\text{Durchsatz} = \frac{\#\text{Befehle}}{\text{Zeit}}$
- In diesem Fall:
  $T_{\text{Latenz}} = 6$
  $\text{Durchsatz} = \frac{5\text{Befehle}}{7\text{Takte}}$
  $\Rightarrow$ IPC-Wert von 0,71 bzw. CPI-Wert von 1,4 bei voll ausgelasteter Pipeline

Moderne Mikroprozessoren – Leistungsbewertung II

„Berechnen Sie den Speedup S zwischen der sequentiellen Ausführung von 95 Befehlen ohne Pipelining und der Ausführung in einer 6-stufigen Pipeline.“

Speedup

- Sei k die Anzahl der Pipelinestufen, n die Anzahl der ausgeführten Befehle.
- Speedup $S = \frac{n*k}{n+k-1}$
- $S = \frac{95*6}{95+6-1} = \frac{570}{100} = 5,7$

Moderne Mikroprozessoren – Leistungsbewertung III

„Es treten nun Konflikte in der Pipeline auf: Jeder 6te Befehl sei ein Sprungbefehl, der mittels Stalling über die Dauer von 3 Takten behandelt werde. Wie verändert sich die Ausführungszeit?“

Lösung

- Alle 6 Befehle werden 3 zusätzliche Takte benötigt
- Verlängerung der Ausführungszeit um $3\ \text{Takte} * 95/6$
- Neue Ausführungszeit: $95 + 6 - 1 + 3 * 95/6 = 100 + 47,5 = 147,5$
- Neuer Speedup $S = \frac{95*6}{147,5} = \frac{570}{147,5} = 3,86$

Aufgabe 1.2: Pipelining

Moderne Mikroprozessoren – Pipelining

„Betrachten Sie das Codestück in Listing 1. Führen Sie dieses zunächst sequentiell durch. Welchen Wert erhalten Sie am Ende in Register 2, welchen in 4? Welche Funktionalität erbringt dieses Programmstück?“

Listing 1

```
 1: INIT:   XOR   R0,R0,R0   ; R0=0
 2:         ADDI  R2,R0,#24  ; R2=24
 3:         ADDI  R3,R0,#8   ; R3=8
 4:
 5: START:  SUB   R4,R2,R3   ; R4=R2-R3?
 6:         BLTZ  R4,L1      ; if (R2<R3) goto L1
 7:         BGTZ  R4,L2      ; if (R2>R3) goto L2
 8:         J     END        ; jump to END
 9:
10: L1:     SUB   R3,R3,R2   ; R3 = R3 - R2
11:         J     START      ; goto START
12:
13: L2:     SUB   R2,R2,R3   ; R2=R2-R3
14:         J     START      ; goto START
15:
16: END:
```

Moderne Mikroprozessoren – Pipelining I

Zeile	Aktion	Zeile	Aktion
1	R0 = 0	6	–
2	R2 = 24	7	B 13
3	R3 = 8	13	R2 = 8
5	R4 = 16	14	B 5
6	–	5	R4 = 0
7	B 13	6	–
13	R2 = 16	7	–
14	B 5	8	B 16
5	R4 = 8	16	

Funktionalität

- Register R2 = R3 = 8
- Register R4 = 0
- Codestück berechnet **größten gemeinsamen Teiler** von R2 und R3

Moderne Mikroprozessoren – Pipelining II

„Bilden Sie die ersten 6 Befehle des Codes auf eine einfache 5-stufige Pipeline ab, in der Konflikte mittels Einfügen von Leerzyklen aufgelöst werden.“

Zeile	Befehl	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	XOR R0,R0,R0	IF	ID	EX	MEM	WB											
2	ADDI R2,R0,#24				IF	ID	EX	MEM	WB								
3	ADDI R3,R0,#8					IF	ID	EX	MEM	WB							
5	SUB R4,R2,R3								IF	ID	EX	MEM	WB				
6	BLTZ R4,L1											IF	ID	EX	MEM	WB	
7	BGTZ R4,L2															IF	ID

Moderne Mikroprozessoren – Pipelining III

„Welche Abhängigkeiten führen in dem Codefragment unter Verwendung der bekannten 5-stufigen Pipeline zu Konflikten und wie lassen sich diese Konflikte vermeiden?“

Zeile	Befehl	Abhängigkeiten			Konflikte
		true	anti	output	
1	XOR R0,R0,R0				
2	ADDI R2,R0,#24	R0 → 1			RAW
3	ADDI R3,R0,#8	R0 → 1			RAW
5	SUB R4,R2,R3	R3 → 3			RAW
6	BLTZ R4,L1	R4 → 5			RAW, Steuerkonflikt
7	BGTZ R4,L2	R4 → 5			RAW, Steuerkonflikt
13	SUB R2,R2,R3	R2 → 2, R3 → 3	R2 → 5	R2 → 2	(2x RAW, 1xWAR, 1xWAW)
14	J START				Steuerkonflikt
5	SUB R4,R2,R3	R2 → 2&13, R3 → 3	R4 → 6&7	R4 → 5	RAW (WAR, WAW)
6	BLTZ R4,L1	R4 → 5			RAW, Steuerkonflikt
7	BGTZ R4,L2	R4 → 5			RAW, Steuerkonflikt
13	SUB R2,R2,R3	R2 → 2, R3 → 3	R2 → 5	R2 → 2&13	(2x RAW, 1xWAR, 1xWAW)
14	J START				Steuerkonflikt
5	SUB R4,R2,R3	R2 → 2&13, R3 → 3	R4 → 6&7	R4 → 5	RAW (WAR, WAW)
6	BLTZ R4,L1	R4 → 5			RAW, Steuerkonflikt
7	BGTZ R4,L2	R4 → 5			RAW, Steuerkonflikt
8	J END				Steuerkonflikt
16					

Moderne Mikroprozessoren – Pipelining IV

Konfliktvermeidung

- Per Hardware: **Forwarding**, **Stalling/Interlocking**
- **vorgezogene Berechnung des Sprungziels**: in Abhängigkeit der verwendeten Stufe ein oder zwei weitere Stall Cycles benötigt
- Softwarelösung: Compiler fügt **Leeroperationen** ein oder **Codeumordnung**
- weitere Möglichkeit: Ressourcenreplizierung zur Vermeidung von Strukturkonflikten

Moderne Mikroprozessoren – Pipelining V

„Wie lassen sich sogenannte *Stall Cycles* zwischen den Befehlen vermeiden?“

Vermeidung von Stall Cycles

- Konfliktbehandlung häufig mittels Leerzyklen (Stall Cycles)
- Vermeidung per Hardwaretechnik: **Forwarding**, vorgezogene Sprungzielberechnung
- **Result Forwarding**: Ergebnis einer arithmetisch-logischen Operation aus der Execute-Stufe
- **Load Forwarding** eines geladenen Datums
- Vermeidung per Softwaretechnik: **Codeumordnung**

Moderne Mikroprozessoren – Pipelining VI

„Führen Sie den Code nun unter Berücksichtigung der Hardware-Techniken zur Vermeidung von Konflikten aus bei stets korrekter Sprungvorhersage.“

Techniken

- Interlocking, Result-Forwarding, Load-Forwarding mit Interlocking
- Berechnung der Sprungrichtung des Sprungziels während der Dekodierphase in einer separaten Adressberechnungseinheit (Address Generation Unit, AGU) nach Auswertung der Bedingung
- Sprungvorhersage in der Befehlsholstufe
- Sprungzieladressen noch unbekannt $\Rightarrow$ Berechnung in Dekodierstufe

Moderne Mikroprozessoren – Pipelining VII

Zeile	Befehl	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		Auflösung der RAW-Konflikte mittels **Forwarding**.														
1	XOR R0,R0,R0	IF	ID	EX	MEM	WB										
2	ADDI R2,R0,#24		IF	ID	EX	MEM	WB									
3	ADDI R3,R0,#8			IF	ID	EX	MEM	WB								
5	SUB R4,R2,R3				IF	ID	EX	MEM	WB							
		Sprungvorhersage sagt NOT TAKEN vorher.														
6	BLTZ R4,L1					IF	ID	EX	MEM	WB						
		Sprungvorhersage sagt zwar TAKEN vorher, aber Zieladresse muss berechnet														
		und Befehlszähler geschrieben werden, daher **1 Leerzyklus**.														
		R4 kann weitergeleitet werden aus Zeile 5.														
7	BGTZ R4,L2						IF	ID	EX	MEM	WB					
13	SUB R2,R2,R3								IF	ID	EX	MEM	WB			
		Ziel in Befehlswort enthalten, aber **nicht in BTAC**, daher muss														
		Befehlszähler neu geschrieben werden, daher **1 Leerzyklus**.														
14	J START									IF	ID	EX	MEM	WB		
5	SUB R4,R2,R3											IF	ID	EX	MEM	WB

Moderne Mikroprozessoren – Pipelining VIII

Zeile	Befehl	12	13	14	15	16	17	18	19	20	21	22	23	24
6	BLTZ R4,L1	IF	ID	EX	MEM	WB								
	Sprungvorhersage sagt TAKEN vorher, jetzt **Ziel bekannt**, daher **kein Leerzyklus**.													
7	BGTZ R4,L2		IF	ID	EX	MEM	WB							
13	SUB R2,R2,R3			IF	ID	EX	MEM	WB						
	Sprungziel jetzt bekannt, daher kein Leerzyklus.													
14	J START				IF	ID	EX	MEM	WB					
5	SUB R4,R2,R3					IF	ID	EX	MEM	WB				
6	BLTZ R4,L1						IF	ID	EX	MEM	WB			
	Sprungvorhersage sagt NOT TAKEN vorher, aber **Sprungziel unbekannt**, daher **1 Leerzyklus**.													
7	BGTZ R4,L2							IF	ID	EX	MEM	WB		
	Ziel in Befehlswort enthalten, aber neuer Befehlszähler muss geschrieben werden, daher **1 Leerzyklus**.													
8	J END									IF	ID	EX	MEM	WB
16												IF	ID	EX

Die Berechnung des Sprungziels könnte auch mit sehr viel Hardwareaufwand in der Befehlsholstufe erfolgen, dadurch würden weitere Leerzyklen vermieden.

Moderne Mikroprozessoren – Pipelining IX

„Nehmen Sie an, die lezte Sprungvorhersage sagt TAKEN voraus, obwohl der Sprung nicht genommen wird. Welche Aktionen sind nun nötig, um die Pipeline in einen gültigen Zustand zu überführen?“

- Sprungvorhersage lädt Befehle von vorhergesagter Adresse
- Ausführung der Befehle nur **spekulativ**, **kein Rückschreiben** in Register

- Statt Zeile 8 würde Zeile 13 nach letztem Branch-Befehl **unmittelbar** geladen
- Auswertung der Branch-Bedingung in Dekodierstufe als NOT TAKEN; Befehl aus Zeile 13 bereits in Befehlsholstufe
- Befehl aus Zeile 13 wird **verworfen**, Leeren der betroffenen Pipelinestufen (**Flushing**)
- Laden des korrekten Befehls aus Zeile 8, während Sprungbefehl in Ausführungsstufe ist
  $\Rightarrow$ **1 Takt Verzögerung**

Pipeline-Flushing

Zeile	Befehl	18	19	20	21	22	23	24
Sprungvorhersage sagt TAKEN vorher, Sprungziel bekannt, lädt Befehl aus Zeile 13.								
7	BGTZ R4,L2	IF	ID	EX	MEM	WB		
13	SUB R2,R2,R3		IF	–	–	–	–	
8	J END			IF	ID	EX	MEM	WB

Moderne Mikroprozessoren – Pipelining XI

„Vergleichen Sie die Ausführungszeiten zwischen sequentieller, einfacher 5-stufiger, und 5-stufiger Pipeline mit Result Forwarding und Pipeline Interlocking. Wie groß ist die Geschwindigkeitszunahme? Was ist am effizientesten?“

Sequentielle Ausführung

- Sequentielle Ausführung: **ein Befehl in einem Taktzyklus** abgearbeitet
- Allerdings **Zyklusdauer sehr groß**
- Zyklusdauer entspricht etwa **allen k Stufen** einer Pipeline zusammen (abzüglich Verzögerungen für Pipelineregister)
- $T_{\text{seq}} = 17 * t_{\text{long\_cycle}} \approx 17 * 5 * t_{\text{short\_cycle}} = 85 * t_{\text{short\_cycle}}$

Moderne Mikroprozessoren – Pipelining XII

Einfache Pipeline

- **Leerzyklen** nötig (Compiler: Leeroperationen (NOPs) oder Hardware: Interlocking)
- Zusätzlich Zeit bis zur Vollendung des letzten Befehls addieren: $(k-1)$ **addieren**
- $T_{\text{einfach}} = (17+2+2+2+3+3+3+2+3+3+3+2+3+3+3+5-1) * t_{\text{short\_cycle}} = 58 * t_{\text{short\_cycle}}$

Pipeline mit Konfliktbehandlung

- 17 Befehle in der Pipeline
- Forwarding, Sprungvorhersage und eigene Adressrechnungseinheit
- Ausführungsdauer:
  $T_{\text{komplex}} = (17+3+5-1) * t_{\text{short\_cycle}} = 24 * t_{\text{short\_cycle}}$
- Etwa ein Drittel der sequentiellen Ausführungsdauer

Vergleich und Bewertung

Den größten Durchsatz hat die „komplexe“ Pipeline.

- $\text{Durchsatz} = \frac{n\,\text{Befehle}}{T(n)}$
- $\text{Durchsatz}_{\text{seq}} = \frac{17\,\text{Befehle}}{85\,\text{Takte}}$, IPC-Wert von 0,2.
- $\text{Durchsatz}_{\text{einfach}} = \frac{17\,\text{Befehle}}{58\,\text{Takte}}$, IPC-Wert von 0,3, Speedup $S = \frac{85}{58} = 1,47$.
- $\text{Durchsatz}_{\text{komplex}} = \frac{17\,\text{Befehle}}{24\,\text{Takten}}$, IPC-Wert von 0,7, Speedup $S = \frac{85}{24} = 3,54$
- Zum Vergleich: theoretisch maximaler Speedup $S(17) = \frac{85}{21} = 4.05$, IPC-Wert von 0,81

Aufgabe 1.3: Sprungvorhersage

Sprungvorhersage I

Ziel: Anpassung der Sprungvorhersage an laufendes Programm
Mittel: Berücksichtigung der Sprunghistorie

Dynamische Sprungvorhersage

- 1-Bit-Prädiktor
- 2-Bit-Prädiktor (Sättigungszähler und Hysteresezähler), Aufgabe 1.3 a)
- Korrelationsprädiktor, Aufgabe 1.3 b)
- Zweistufig adaptive Prädiktoren
- gshare, gselect
- Hybridprädiktoren

Sprungvorhersage II

Führen Sie den Codeabschnitt 2 (R0 sei das „Zero“-Register) auf einem Zwei-Bit-Prädiktor mit Hysteresezähler aus, wobei für jeden Sprung ein eigener Prädiktor zur Verfügung stehe. Diese Prädiktoren seien mit *Predict Weakly Taken* (WT) initialisiert.

Codeabschnitt 2

```
 1: INIT:   ADD  R1,R0,#0 ; R1=0
 2:         ADD  R2,R0,#2 ; R2=2
 3:
 4: START:  BNE  R1,R0,L1 ; if (R1!=0) goto L1
 5:         ADD  R1,R0,#1 ; R1=1
 6:
 7: L1:     SUB  R3,R1,R2 ; R3=R1-R2
 8:         BNE  R3,R0,L2 ; if (R1!=R2) goto L2
 9:         ADD  R1,R0,#0 ; R1=0
10:         J    START    ; goto START
11:
12: L2:     ADD  R1,R0,#2 ; R1=2
13:         J    START    ; goto START
```

Sprungvorhersage III

Sprungverläufe

Sprung 1 Zeile 4 BNE L1	Sprung 2 Zeile 8 BNE L2	Sprung 3 Zeile 10 J START	Sprung 4 Zeile 13 J START
NT (R1=0)	T (R1=1)	–	T
T (R1=2)	NT (R1=2)	T	–
NT (R1=0)	T (R1=1)	–	T
T (R1=2)	NT (R1=2)	T	–

Die Werte von `R1` in Klammern gelten **vor** dem Sprung.
Hierbei müssen die letzten zwei Sprünge nicht berücksichtigt werden:
Ihre Bedingung ist immer wahr und somit wird immer gesprungen.

Sprungvorhersage IV

2-Bit-Prädiktor mit Hysteresezähler

Initialisierung der **sprungeigenen** (lokalen) Prädiktoren mit *Predict Weakly Taken* (WT).

Es werden **5** Fehlannahmen gemacht:

Sprung 1			Sprung 2		
Prädiktion	Sprung	Neue Vorh.	Prädiktion	Sprung	Neue Vorh.
WT	**NT**	NT	WT	T	T
NT	**T**	WNT	T	**NT**	WT
WNT	NT	NT	WT	T	T
NT	**T**	WNT	T	**NT**	WT

Sprungvorhersage V

2-Bit-Prädiktor mit Sättigungszähler

Jeweils initialisiert mit WNT.

Es werden **6** Fehlannahmen gemacht:

Sprung 1			Sprung 2		
Prädiktion	Sprung	Neue Vorh.	Prädiktion	Sprung	Neue Vorh.
WNT	NT	NT	WNT	**T**	WT
NT	**T**	WNT	WT	**NT**	WNT
WNT	NT	NT	WNT	**T**	WT
NT	**T**	WNT	WT	**NT**	WNT

Sprungvorhersage VI

(m,n)-Korrelationsprädiktoren

- m Sprünge umfassenden Historie
- Speicherung in Sprungverlaufsregister (Branch History Register, BHR)
- Auswahl eines n-Bit-Prädiktors aus Sprungverlaufstabelle (Pattern History Table, PHT) anhand von Sprungadresse und BHR

- (Teil der) Sprungadresse selektiert Reihe in PHT
- BHR wählt eine von 2^{m} Spalten
- Ausgewähltes Speicherfeld enthält n-Bit-Prädiktor

- (1,1)-Korrelationsprädiktor: Globales 1-Bit-BHR, pro Sprung zwei separate 1-Bit-Prädiktoren

Sprungvorhersage VII

„Vergleichen Sie anschließend mit der Ausführung auf einem (1,1)-Korrelationsprädiktor. Das globale Schieberegister sei dabei mit (NT) gefüllt, die zwei Prädiktoren seien mit T vorbelegt.“

(1,1)-Korrelationsprädiktor

Auswahl eines sprungeigenen (lokalen) 1-Bit-Prädiktors gemäß globalem 1-Bit-Sprungverlaufsregister (BHR).
Es werden lediglich zwei Fehlannahmen gemacht (Auswahl und Fehlvorhersage hervorgehoben):

Zeile	Richtung	Aktuelle Vorhersage			Neue Vorhersage	
		BHR	Prädiktor	Vorh.	BHR	Prädiktoren
4	NT	NT	(**T**, T)	**T**	NT	(**NT**, T)
7	T	NT	(**T**, T)	T	T	(T, T)
4	T	T	(NT, **T**)	T	T	(NT, T)
7	NT	T	(T, **T**)	**T**	NT	(T, **NT**)
4	NT	NT	(**NT**, T)	NT	NT	(NT, T)
7	T	NT	(**T**, NT)	T	T	(T, NT)
4	T	T	(NT, **T**)	T	T	(NT, T)
7	NT	T	(T, **NT**)	NT	NT	(T, NT)

Sprungvorhersage VIII

Weiterentwicklungen der Korrelationsprädiktoren

- Zweistufig adaptive Prädiktoren: Verschiedenerlei Kombinationen von lokalen und globalen BHR/BHT und Sprungadressen zur Auswahl eines Prädiktors
- McFarling, 1994: **gselect** und **gshare** für geeignetere Wahl der Prädiktoren
- McFarling, 1993: **Hybridprädiktoren**

Sprungvorhersage IX

gselect und gshare

- Wunsch: Verringerung der Interferenzen bei **Korrelationsprädiktoren**
- Verwendung von BHR und PHT wie bekannt
- **gselect**: Adressierung durch Konkatenation von BHR und Sprungbefehlsadresse (bzw. Teilen hiervon)
- **gshare**
  - Statt Konkatenation bitweise Verknüpfung von BHR und Sprungbefehlsadresse mit XOR
  - Resultat: weniger Interferenzen bei gleicher Länge

Adressteil	BHR	gselect 4/4	gshare 8/8
00000000	00000001	00000001	00000001
00000000	00000000	00000000	00000000
11111111	00000000	11110000	11111111
11111111	10000000	11110000	01111111

Aufgabe 1.4: Befehlsausführung mit Prädikaten

Befehlsausführung mit Prädikaten

Prädikation

- Prädikation = Ansage, Ankündigung mittels Prädikaten
- Wesentliches Ziel: Sprungvermeidung
- Ergebnis wird nur gültig gemacht, wenn Prädikat wahr
- Kein Rückrollen nötig wegen zusätzlicher Register für prädikative Pfade
- Eingesetzt bei VLIW-Architekturen, z.B. Intel IA64, sowie ARM-Architekturen (Prädikation einzelner Befehle)

Befehlsausführung mit Prädikaten

„Betrachten Sie nochmals Codeabschnitt 1. Wie könnte der Assembler-Code für die IA64- Architektur aussehen?“

Überlegungen

- Zunächst ein Bundle für die beiden MOV-Befehle, werden übersetzt in Additionsbefehl mit Null-Register
- Codeumstellung nötig für die gleichzeitige Ausführung der zwei Subtraktionen entsprechend Prädikaten

$\Rightarrow$ erst Überprüfung auf Gleichheit

$\Rightarrow$ dann ggf. Überprüfung auf „kleiner“ oder Abbruch

- Abschließend Rücksprung

Befehlsausführung mit Prädikaten

Programmcode

```
.implicit

INIT:   MOV     R2 = 24
        MOV     R3 = 8              ;;

START:  CMP.EQ  P1, P2 = R2, R3     ;;
  (P1)  JMP     END
  (P2)  CMP.LT  P3, P4 = R2, R3     ;;
  (P3)  SUB     R3, R3 = R2
  (P4)  SUB     R2, R2 = R3         ;;
        JMP     START               ;;

END:
```

Die Gruppen sind durch Strichpunkte voneinander abgetrennt. So entstehen maximal 5 Bundles mit insgesamt 8 Befehlen statt 11 Befehlen

Befehlsausführung mit Prädikaten

Bundling

Compiler kann mehrere Gruppen in einem Bundle unterbringen:

MOV R2 = 24	MOV R3 = 8 ;;	NOP
CMP.EQ P1, P2 = R2, R3 ;;	(P1) JMP END	(P2) CMP.LT P3, P4 = R2, R3 ;;
(P3) SUB R3, R3 = R2	(P4) SUB R2, R2 = R3 ;;	JMP START ;;

$\Rightarrow$ 3 Bundles, ein NOP eingefügt.

Befehlsausführung mit Prädikaten

Explizites Bundling

Bundling ist auch explizit spezifizierbar:

```
.explicit
{
INIT:  MOV    R2 = 24
       MOV    R3 = 8          ;;
}
{
START: CMP.EQ P1, P2 = R2, R3 ;;
  (P1) JMP    END
  (P2) CMP.LT P3, P4 = R2, R3 ;;
}
{
  (P3) SUB    R3, R3 = R2
  (P4) SUB    R2, R2 = R3     ;;
       JMP    START           ;;
}
END:
```

Führt zu Bundling wie oben beschrieben.

Aufgabe 2: Superskalartechnik

Superskalartechnik I

Superskalartechnik

- Pro Takt werden ein oder mehrere Befehle aus dem Programmspeicher geholt
- *Reservation Stations* und Registertabellen geben Aufschluss über Belegung, Nutzung und Zuweisung der Ausführungseinheiten
- **Konfliktauflösung:** Parallele Ausführbarkeit der Befehle dadurch von Hardware ermittelbar
- **Dynamisches** Konzept
- Weit verbreitet
- **Sequentielles** Erscheinungsbild

Superskalare Pipelines – Stufen I

Zunächst: **Befehl holen**

Je nach Auslegung dann 4 bis 5 Phasen:

- Brinkschulte/Ungerer: 5 Phasen bestehend aus
  - **Dekodierung**: Erkennen der Befehlsklasse, der Adressierungsmodi usw.
  - **Registerumbenennung**: Belegen eines physischen Registers mit dem virtuellen
  - **Befehlszuordnung**: Zuordnung eines Befehls zu einer Einheitenfamilie und Anstoßen bei Verfügbarkeit der Operandenwerte; Eintragung in die Reservation Station
  - **Ausführung**: Ausführung der Rechenoperation oder des Speicherzugriffs
  - **Rückordnung**: Schreiben des physikalischen Registers
- **Dekodierung** und **Registerumbenennung** können dabei zusammengefasst werden.
- Befehlszuordnung kann weiter unterteilt werden: Issue und Dispatch

Superskalare Pipelines – Stufen II

- Hennessy/Patterson: 4 Phasen bestehend aus
  - **Issue**: Zuweisung eines Befehls zu einer Einheitenfamilie mit mindestens einer freien Reservation Station; gleichzeitig Dekodierung und natürlich Eintragung in Reservation Station
  - **Read Operands**: Warten auf gültige Operandenwerte und Anstoßen der Operation
  - **Execute**: Rechenoperation bzw. Speicherzugriff
  - **Write Results**: Schreiben des physikalischen Registers
- Prinzipiell sind die Auslegungen also nicht sonderlich unterschiedlich.
- Wir verwenden die Definition nach Brinkschulte/Ungerer mit den zusammengefassten ersten zwei Phasen.

Algorithmus von Tomasulo I

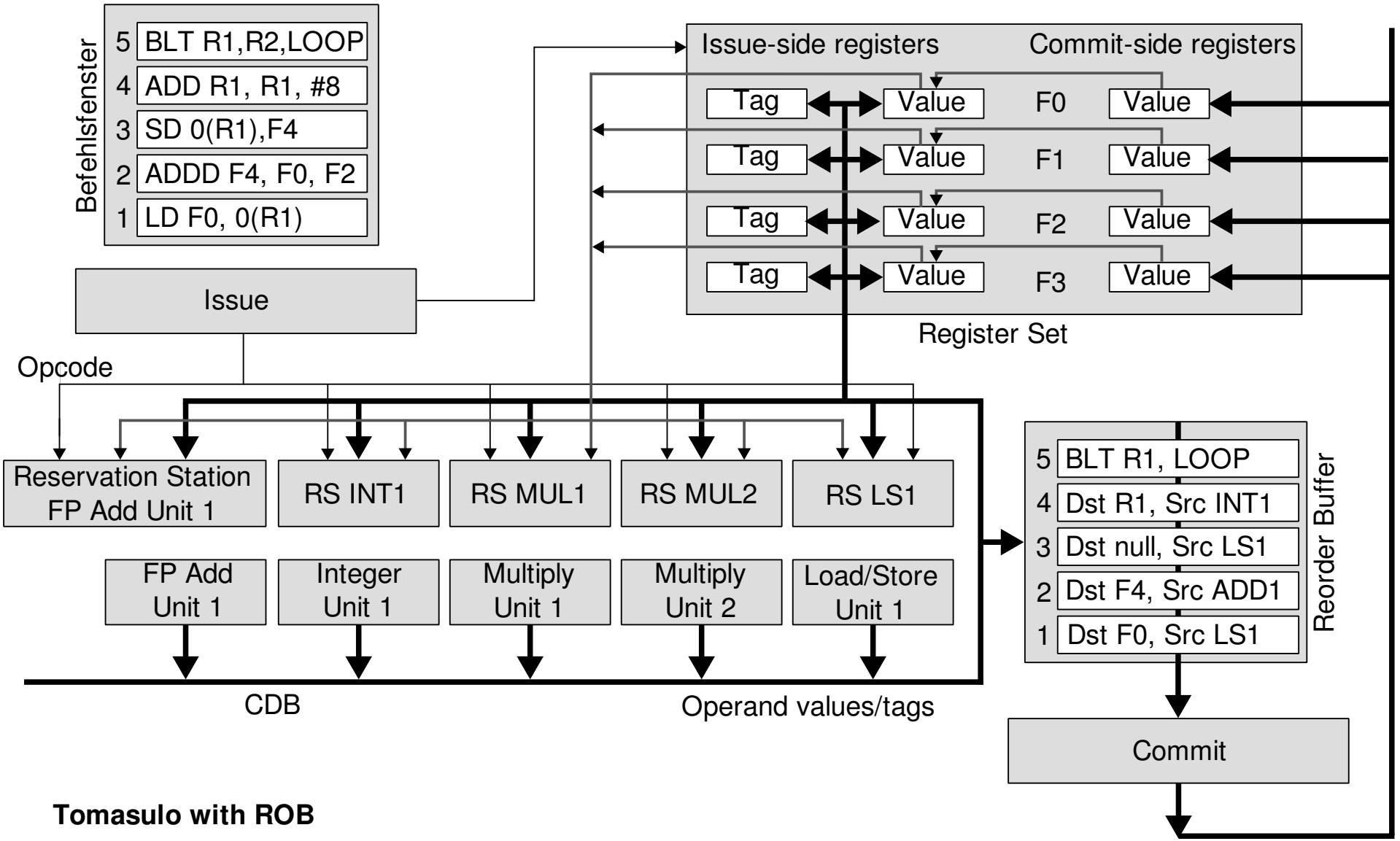
Tomasulo

- Anstoßen der Befehle **dezentral** aus den Reservation Stations

Aktiv	Befehl	Value j	Vk	Qj (Producer)	Qk	Address A

- Einlesen der Operanden **dezentral** in den Reservation Stations
- Übertragung der Operandenwerte über den **Common Data Bus** (CDB):
  Tupel aus Tag/Wert
- Result Forwarding ebenfalls **dezentral**
- Einheiten können zu **Einheitenfamilien** zusammengefasst werden mit einer gemeinsamen, mehrzeiligen Reservation Station Table
- Reorder Buffer für die Befehlsrückordnung (Wiederherstellung der sequentiellen Programmsemantik)

Algorithmus von Tomasulo II



Tomasulo with ROB

Bestandteile einer Implementierung des Tomasulo-Algorithmus I

Befehlsfenster

Nummer	Befehl	Station
1	`Op  Operands`	–

- Zuständig für die Bereitstellung dekodierter Befehle an Zuweisungseinheit (**Instruction Issue**)
- Begrenzte Größe: 16 bis 46 Einträge heutzutage
- Zuweisung von bis zu sechs Anweisungen pro Takt
- Hier erweitert um Feld der derzeitigen Station; dadurch auch mehr Einträge als in echter Hardware

Bestandteile einer Implementierung des Tomasulo-Algorithmus II

Registerstatustabelle

Feld	**PhR0**	**PhR1**	**PhR2**	**PhF0**	**PhF2**	...
Architektur-Register						...
Qi		R7	R3			
Valid		✓				

- Gibt Auskunft über Abbildung von Architekturregistern auf physische Register
- Des Platzes wegen Valid-Bit später nicht eingezeichnet, wird durch * angegeben

Bestandteile einer Implementierung des Tomasulo-Algorithmus III

Reservation Stations (Umordnungspuffer)

In tabellarischer Form hier zusammengefasst (sonst separat, damit nicht große Tabelle durchsucht werden muss von mehreren Dispatchern):

Einheit	Aktiv	Befehl	Vj	Vk	Qj	Qk	A
L/S 1	n						
L/S 2	n						
Int-Add	n						
Int-Mul	n						
FP-Add	n						
FP-Mul	n						

Bei Tomasulo dezentral an jeder Einheit oder Familie von Einheiten.

Bestandteile einer Implementierung des Tomasulo-Algorithmus IV

Rückordnungspuffer

Befehlsnr.	Ziel	Quelle
2	Phys. Reg.	Einheit
1	Phys. Reg.	Einheit

- Wird bei Eintragen eines Befehls in Reservation Station gefüllt und enthält Zielregister sowie produzierende Einheit
- Wird von **Retirement Unit** (Rückordnungsstufe) benutzt, um **Commitment** oder **Removement** durchzuführen

$\Rightarrow$ Rückordnung der Befehle in Programmreihenfolge

Tomasulo – Beispiel I

Annahmen & Voraussetzungen

- Zwei Lade-Speichereinheiten (Load/Store Unit), eine Integer-Additionseinheit, eine Integer-Multiplikationseinheit, eine FP-Additionseinheit
- Delayed Branches, also kein Anhalten (Stalling) und keine Vorhersage, sondern fortwährendes Füllen der Pipeline; dafür sei ein Sprungzieladresscache vorhanden und die Vorhersage laute auf Taken
- Volles Bypassing
- FP-Register und normale Register können gleichzeitig in der WB-Stufe beschrieben werden
- Schreiben in Speicher geschehe nicht über CDB

Tomasulo – Beispiel II

Annahmen & Voraussetzungen (Forts.)

- Die Befehlszuordnungs- und Rückordnungsbandbreite betrage 4 Befehle; zwei Befehle werden pro Takt maximal geholt
- Entsprechend gibt es zwei Dekodiereinheiten, die gleichzeitig arbeiten können
- Die Auswertung der Sprungzieladresse erfolge in der Stufe Execute der Int-Add-Einheit; das Schreiben des Befehlszählers (Instruction Counter, Program Counter) in der WB-Stufe
- Speicherlesezugriffe erfolgen analog zu normalen Rechenoperationen in der Ausführungsstufe, das Rückschreiben geschieht dabei als separater Schritt (WB)
- Speicherschreibzugriffe haben eine Ausführungsdauer von nur einem Takt

Tomasulo – Beispiel III

Annahmen & Voraussetzungen (Forts.)

- Kein Pipeling in den Ausführungseinheiten
- Dekodierte Befehle werden nur dann in die Issue-Stufe gebracht, wenn Einheit frei ist und Operanden verfügbar sind (entspricht also *Read Operands*)
- Hier zur Vereinfachung der Darstellung: Nach Dekodierung benennen wir die Stufe vor der Befehlsausführung mit Issue (IS) und zeichen sie erst, wenn die Operanden verfügbar sind. Analog für die Belegung der Reservation Stations.
- Dadurch Modellierung von Einheitenfamilien erreicht
- **Achtung:** Für den Einsatz der Registerumbenennung müssen Eintragungen in RSs eigentlich direkt nach Dekodierung erfolgen!
- Folgende Pipelinestruktur: IF ID IS EX M WB

Tomasulo – Beispiel IV

Ausführungseinheiten

Einheit	L/S	Integer-Add	Integer-Mul	FP-Add
Anzahl	2	1	1	1
Bearbeitungsdauer (in Takten)	3	1	3	2

Tomasulo – Beispiel V

Auszuführender Programmcode

```
LOOP: LD.D   F0,0(R1)    ; loads Mem[i]
      ADD.D  F4,F0,F2    ; adds to Mem[i]
      S.D    0(R1),F4    ; stores into Mem[i]
      ADD    R1,R1,#8    ;
      SUB    R3,R1,R2    ; R3 = R1-R2
      BLTZ   R3,LOOP     ; delayed branch if R1 < R2
```

- `R1` enthält eine Speicheradresse, `F2` fest.
- Die Schleife wird drei mal durchlaufen wegen `R2 = R1 + 24`

Tomasulo – Beispiel VI

Takt 1

- `LD.D` und `ADD.D` geholt

Takt 2

- `LD.D` und `ADD.D` dekodiert
- `SD.D` und `ADD` geholt
- Reservation Station noch leer

Rückordnungspuffer:

Befehlsnr.	Ziel	Quelle
2	PhF4	FP-Add
1	PhF0	L/S 1

Code

```
LOOP: LD.D   F0,0(R1)
      ADD.D  F4,F0,F2
      S.D    0(R1),F4
      ADD    R1,R1,#8
      SUB    R3,R1,R2
      BLTZ   R3,LOOP
```

Pipeline

Befehl		Stufe 1	Stufe 2
LD.D	F0,0(R1)	IF	ID
ADD.D	F4,F0,F2	IF	ID
S.D	0(R1),F4		IF
ADD	R1,R1,#8		IF

Registerstatustabelle:

Feld	PhR0	PhR1	PhF0	PhF2	PhF4
Architektur-Register	R1*	R2*	F0	F2*	F4
Qi					

Tomasulo – Beispiel VII

Befehlsfenster nach Takt 1 & 2

Nummer	Befehl		Station
1	`L.D`	`F0,0(R1)`	ID
2	`ADD.D`	`F4,F0,F2`	ID
3	`S.D`	`0(R1),F4`	IF
4	`ADD`	`R1,R1,#8`	IF

Tomasulo – Beispiel VIII

Takt 3 & 4

- SUB und BLTZ geholt, dekodiert
- LD.D und ADD.D geholt

Code

```
LOOP: LD.D   F0,0(R1)
      ADD.D  F4,F0,F2
      S.D    0(R1),F4
      ADD    R1,R1,#8
      SUB    R3,R1,R2
      BLTZ   R3,LOOP
```

Befehlsfenster:

Nummer	Befehl		Station
1	LD.D	F0,0(R1)	M
2	ADD.D	F4,F0,F2	ID[1]
3	S.D	0(R1),F4	ID[1]
4	ADD	R1,R1,#8	IS
5	SUB	R3,R1,R2	ID
6	BLTZ	R3,LOOP	ID
7	LD.D	F0,0(R1)	IF
8	ADD.D	F4,F0,F2	IF

[1]: Not yet finally issued

Pipeline

Befehl		Takt 1	2	3	4
LD.D	F0,0(R1)	IF	ID	IS	M
ADD.D	F4,F0,F2	IF	ID		
S.D	0(R1),F4		IF	ID	
ADD	R1,R1,#8		IF	ID	IS
SUB	R3,R1,R2			IF	ID
BLTZ	R3,LOOP			IF	ID
LD.D	F0,0(R1)				IF
ADD.D	F4,F0,F2				IF

Tomasulo – Beispiel IX

Rückordnungspuffer:

Befehlsnr.	Ziel	Quelle
6	PC	Int-Add
5	PhR3	Int-Add
4	PhR2	Int-Add
3	null	any L/S
2	PhF4	FP-Add
1	PhF0	L/S 1

Reservation Stations:

Einheit	Aktiv	Befehl	Vj	Vk	Qj	Qk	A
L/S 1	j	LD.D					0 + [PhR0]
Int-Add	n	ADD	[PhR0]	#8			

Registerstatustabelle:

Feld	PhR0	PhR1	PhR2	PhF0	PhF2	PhF4	PhF6
Architektur-Register	R1	R2*	R1	F0	F2*	F4	F0
Qi			Int-Add	L/S 1			

Tomasulo – Beispiel X

Takt 5 & 6

- `ADD` schreibt `R1`
- Result Forwarding an `SUB` und `LD.D`

Befehlsfenster:

Nummer	Befehl		Station
1	`LD.D`	`F0,0(R1)`	M
2	`ADD.D`	`F4,F0,F2`	ID
3	`S.D`	`0(R1),F4`	ID
4	`ADD`	`R1,R1,#8`	WB
5	`SUB`	`R3,R1,R2`	IS
6	`BLTZ`	`R3,LOOP`	ID
7	`LD.D`	`F0,0(R1)`	IS
8	`ADD.D`	`F4,F0,F2`	ID
9	`S.D`	`0(R1),F4`	ID
10	`ADD`	`R1,R1,#8`	ID
11	`SUB`	`R3,R1,R2`	IF
12	`BLTZ`	`R3,LOOP`	IF

Pipeline

Befehl		Takt			
		3	4	5	6
`LD.D`	`F0,0(R1)`	IS	M	M	M
`ADD.D`	`F4,F0,F2`				
`S.D`	`0(R1),F4`	ID			
`ADD`	`R1,R1,#8`	ID	IS	EX	WB
`SUB`	`R3,R1,R2`	IF	ID		IS
`BLTZ`	`R3,LOOP`	IF	ID		
`LD.D`	`F0,0(R1)`		IF	ID	IS
`ADD.D`	`F4,F0,F2`		IF	ID	
`S.D`	`0(R1),F4`			IF	ID
`ADD`	`R1,R1,#8`			IF	ID
`SUB`	`R3,R1,R2`				IF
`BLTZ`	`R3,LOOP`				IF

Tomasulo – Beispiel XI

Rückordnungspuffer:

Befehlsnr.	Ziel	Quelle
10	PhR0	Int-Add
9	null	any L/S Unit
8	PhF8	FP-Add
7	PhF6	L/S 2
6	PC	Int-Add
5	PhR3	Int-Add
4	PhR2	Int-Add
3	null	any L/S Unit
2	PhF4	FP-Add
1	PhF0	L/S 1

Reservation Stations:

Einheit	Aktiv	Befehl	Vj	Vk	Qj	Qk	A
L/S 1	j	LD.D		0			0 + [PhR0]
L/S 2	n	LD.D	[PhR2]	0			
Int-Add	j	SUB	[PhR2]	[PhR1]			

Registerstatustabelle:

Feld	PhR0	PhR1	PhR2	PhR3	PhF0	PhF2	PhF4	PhF6	PhF8
Architektur-Register	R1	R2*	R1*	R3	F0	F2*	F4	F0	F4
Qi				SUB	L/S 1			L/S2	

Tomasulo – Beispiel XII

Takt 7 & 8

- Erstes `LD.D` beendet
- Zweites `LD.D` führt Speicherzugriff durch
- `ADD.D` angestoßen
- Int-Einheit frei für `SUB`

Pipeline

Befehl		Takt					
		3	4	5	6	7	8
`LD.D`	`F0,0(R1)`	IS	M	M	M	WB	
`ADD.D`	`F4,F0,F2`					IS	EX
`S.D`	`0(R1),F4`	ID					
`ADD`	`R1,R1,#8`	ID	IS	EX	WB		
`SUB`	`R3,R1,R2`	IF	ID		IS	EX	WB
`BLTZ`	`R3,LOOP`	IF	ID				IS
`LD.D`	`F0,0(R1)`		IF	ID	IS	M	M
`ADD.D`	`F4,F0,F2`		IF	ID			
`S.D`	`0(R1),F4`			IF	ID		
`ADD`	`R1,R1,#8`			IF	ID		
`SUB`	`R3,R1,R2`				IF	ID	
`BLTZ`	`R3,LOOP`				IF	ID	
`LD.D`	`F0,0(R1)`					IF	ID
`ADD.D`	`F4,F0,F2`					IF	ID
`S.D`	`0(R1),F4`						IF
`ADD`	`R1,R1,#8`						IF

Tomasulo – Beispiel XIII

Befehlsfenster:

Nummer	Befehl		Station
1	ADD.D	F4,F0,F2	EX
2	S.D	0(R1),F4	ID
3	SUB	R3,R1,R2	WB
4	BLTZ	R3,LOOP	IS
5	LD.D	F0,0(R1)	M
6	ADD.D	F4,F0,F2	ID
7	S.D	0(R1),F4	ID
8	ADD	R1,R1,#8	ID
9	SUB	R3,R1,R2	ID
10	BLTZ	R3,LOOP	ID
11	LD.D	F0,0(R1)	ID
12	ADD.D	F4,F0,F2	ID
13	S.D	0(R1),F4	IF
14	ADD	R1,R1,#8	IF

Rückordnungspuffer:

Befehlsnr.	Ziel	Quelle
13	PhF12	FP-Add
12	PhF10	any L/S Unit
11	PC	Int-Add
10	PhR4	Int-Add
9	PhR0	Int-Add
8	null	any L/S Unit
7	PhF8	FP-Add
6	PhF6	L/S 2
5	PC	Int-Add
4	PhR3	Int-Add
3	PhR2	Int-Add
2	null	any L/S Unit
1	PhF4	FP-Add

Der vollendete Additionsbefehl kann noch nicht zurückgeordnet werden, da die vorherigen Befehle noch nicht alle beendet sind.

Tomasulo – Beispiel XIV

Reservation Stations:

Einheit	Aktiv	Befehl	Vj	Vk	Qj	Qk	A
L/S 1	n						
L/S 2	j	LD.D		0			0 + [PhR2]
Int-Add	n	BLTZ	[PhR3]				
Int-Mul	n						
FP-Add	j	ADD.D	[PhF0]	[PhF2]			

Registerstatustabelle:

Feld	PhR0	PhR1	PhR2	PhR3	PhR4	PhF0	PhF2
Architektur-Register	R1	R2*	R1*	R3*	R1	F0*	F2*
Qi							

PhF4	PhF6	PhF8	PhF10	PhF12
F4	F0	F4	F0	F4
FP-Add	L/S2			

Tomasulo – Beispiel XV

Befehl		Takt											
		1	2	3	4	5	6	7	8	9	10	11	12
LD.D	F0,0(R1)	IF	ID	IS	M	M	M	WB					
ADD.D	F4,F0,F2	IF	ID					IS	EX	EX	WB		
S.D	0(R1),F4		IF	ID							IS	M/WB	
ADD	R1,R1,#8		IF	ID	IS	EX	WB						
SUB	R3,R1,R2			IF	ID		IS	EX	WB				
BLTZ	R3,LOOP			IF	ID				IS	EX	WB[2]		
LD.D	F0,0(R1)				IF	ID	IS	M	M	M		WB[1]	
ADD.D	F4,F0,F2				IF	ID						IS	EX
S.D	0(R1),F4					IF	ID						
ADD	R1,R1,#8					IF	ID				IS	EX	WB
SUB	R3,R1,R2						IF	ID					IS
BLTZ	R3,LOOP						IF	ID					
LD.D	F0,0(R1)							IF	ID				IS
ADD.D	F4,F0,F2							IF	ID				
S.D	0(R1),F4								IF	ID			
ADD	R1,R1,#8								IF	ID			
SUB	R3,R1,R2									IF	ID		
BLTZ	R3,LOOP									IF	ID		

Tomasulo – Beispiel XVI

Befehl		Takt										
		12	13	14	15	16	17	18	19	20	21	22
LD.D	F0,0(R1)											
ADD.D	F4,F0,F2											
S.D	0(R1),F4											
ADD	R1,R1,#8											
SUB	R3,R1,R2											
BLTZ	R3,LOOP											
LD.D	F0,0(R1)											
ADD.D	F4,F0,F2	EX	EX	WB								
S.D	0(R1),F4			IS	M/WB							
ADD	R1,R1,#8	WB										
SUB	R3,R1,R2	IS	EX	WB								
BLTZ	R3,LOOP			IS	EX	WB^2						
LD.D	F0,0(R1)	IS	M	M	M	WB						
ADD.D	F4,F0,F2					IS	EX	EX	WB			
S.D	0(R1),F4								IS	M/WB		
ADD	R1,R1,#8					IS	EX	WB				
SUB	R3,R1,R2							IS	EX	WB		
BLTZ	R3,LOOP									IS	EX	WB

Tomasulo – Beispiel XVII

Leistung

- 18 Befehle in 22 Takten
- Durchsatz = $\frac{18}{22}$, IPC $\approx 0,82$

Erläuterungen/Fußnoten

- [1] Der CDB wird beim Speicherzugriff nicht benutzt.
- [2] Der Befehlszähler wird nicht über den CDB geschrieben.

Tomasulo – Beispiel XVIII

Hinweis

- Beispiel rein künstlich und nur zur prinzipiellen Darstellung
- Reservation Stations wurden **nicht** sofort nach Dekodierung gefüllt gezeichnet
- Implementierungen mit zu Familien zusammgefassten Einheiten gleichermaßen ohne Zusammenfassung machbar
- Reservation Stations haben einen **oder mehrere** Einträge

Fragen?