Universität Karlsruhe (TH)
Forschungsuniversität • gegründet 1825

Institut für Technische Informatik
Prof. Dr. rer. nat. Wolfgang Karl

Zentralübung Rechnerstrukturen

Lösungsblatt 3: Parallelismus auf Befehlsebene

1 Moderne Mikroprozessoren

1.1 Leistungsbewertung

a) 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 = 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:

$T_{\text{Latenz}} = 6$

$\text{Durchsatz} = \frac{5\text{Befehle}}{7\text{Takten}}$

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

Sei k die Anzahl der Pipelinestufen, und n die Anzahl der ausgeführten Befehle. Dann ist der Speedup $S = \frac{n*k}{n+k-1}$

In diesem Fall: $S = \frac{95*6}{95+6-1} = \frac{570}{100} = 5,7$

1.2 Pipelining

a) 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?

Sequentielle Ausführung:

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	

In Register 2 sowie Register 3 steht der größte gemeinsame Teiler von 24 und 8, Register 4 ist Null.

b) 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	Takt 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
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	EX	MEM	WB

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

In der nachfolgenden Tabelle ist der ausgeführte Code mitsamt der vorhandenen Abhängigkeiten eingetragen. In der letzten Spalte ist der verursachte Konflikt für die DLX-Pipeline angegeben, in Klammern die in anderen Architekturen möglichen Konflikte.

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					

Die Konflikte lassen sich per Hardware durch Forwarding, Stalling/Interlocking und vorgezogene Berechnung des Sprungziels vermeiden. In Abhängigkeit der verwendeten Stufe zur Berechnung des Sprungziels sind ein oder zwei weitere Stall Cycles nötig. Als Softwarelösung kann der Compiler Leeroperationen einfügen oder den Code umordnen.

Eine weitere Möglichkeit ist die Ressourcenreplizierung zur Vermeidung von Strukturkonflikten.

d) Wie lassen sich sogenannte *Stall Cycles* zwischen den Befehlen vermeiden?

Als Hardwaretechnik verhindert Forwarding unnötige Stall Cycles. Dabei wird zwischen Result Forwarding des Ergebnisses einer arithmetisch-logischen Operation aus der Execute-Stufe und dem Load Forwarding eines geladenen Datums unterschieden.

Neben Forwarding bietet Code-Umordnung als Softwaretechnik ein Mittel zur Vermeidung von Stall Cycles.

e) Führen Sie den Code nun unter Berücksichtigung der Hardware-Techniken zur Vermeidung von Konflikten aus bei stets korrekter Sprungvorhersage. Die Berechnung der Sprungrichtung des Sprungziels während der Dekodierphase erfolge in einer separaten Adressberechnungseinheit nach Auswertung der Bedingung, die Sprungvorhersage erfolge in der Befehlsholstufe.

Einsatz von 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 Bedin-

gung. Da der Code noch nicht ausgeführt wurde, sind die Sprungzieladressen noch unbekannt und müssen in der Dekodierstufe berechnet werden.

Zeile	Befehl	Takt 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
		Zur Auflösung der RAW-Konflikte genügt 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									
6	BLTZ R4,L1												IF	ID	EX	MEM	WB								
		Sprungvorhersage sagt TAKEN vorher, Ziel jetzt 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.																							
		Bei spekulativ aus Zeile 13 geladenem Befehl würde Flushing nötig.																							
7	BGTZ R4,L2																		IF	ID	EX	MEM	WB		
		Ziel in Befehlswort enthalten, aber neuer Befehlszähler muss berechnet und geschrieben werden, daher 1 Leerzyklus.																							
8	J END																				IF	ID	EX	MEM	WB
16																							IF	ID	EX

f) Vergleichen Sie die Ausführungszeiten zwischen sequentieller, einfacher 5-stufiger, und 5-stufiger Pipeline mit Result Forwarding und Pipeline Interlocking. Wie groß ist die Geschwindigkeitszunahme? Wie groß ist jeweils der Durchsatz?

Führt man den Programmcode sequentiell aus, so kann ein Befehl in einem Taktzyklus abgearbeitet werden, wobei allerdings die Ausführungszeit eines solchen Zyklus sehr hoch ist und etwa der aller k Stufen einer Pipeline zusammen entspricht.

$T_{\text{seq}} = 17 * t_{\text{long\_cycle}} \approx 17 * 5 * t_{\text{short\_cycle}} = 85 * t_{\text{short\_cycle}}$

In einer einfachen 5stufigen Pipeline sind Leerzyklen nötig, die über den Compiler mittels Leeroperationen (NOPs) oder über die Hardware mittels Interlocking entstehen, um die Konflikte zu vermeiden. Zusätzlich muss die Zeit berücksichtigt werden, bis der letzte Befehl seine Ausführung vollständig beendet hat, also $(k-1)$ addiert werden.

$T_{\text{einfach}} = (17+2+2+2+3+3+3+2+3+3+3+2+3+3+3+5-1) * t_{\text{short\_cycle}} = 58 * t_{\text{short\_cycle}}$

Die Ausführungszeit der 17 Befehle in der Pipeline mit Forwarding, Sprungvorhersage und eigener Adressrechnungseinheit beträgt 24 Takte.

$T_{\text{komplex}} = (17 + 3 + 5 - 1) * t_{\text{short\_cycle}} = 24 * t_{\text{short\_cycle}}$

Die Ausführungsdauer verringert sich auf etwa ein Drittel.

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: Der optimale Speedup ist $S(17) = \frac{85}{21} = 4.05$ *, der optimale IPC-Wert ist 0,81.*

1.3 Sprungvorhersage

a) 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.

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	–

In Klammern steht der Wert von `R1` **vor** dem Sprung.

Hierbei müssen die letzten zwei Sprünge nicht berücksichtigt werden: Ihre Bedingung ist immer wahr und somit wird immer gesprungen.

Zwei-Bit-Prädiktor mit Hysteresezähler

*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

Achtung: je Sprung ein eigener, separater 2-Bit-Prädiktor!

b) 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.

Es werden lediglich zwei Fehlannahmen gemacht:

Sprungzeile	Richtung	Aktuelle Vorhersage			Neue Vorhersage	
		Historie	Prädiktor	Vorh.	Akt. Historie	Akt. 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)

1.4 Befehlsausführung mit Prädikaten

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

Die Initialisierung kann in eine Gruppe für die beiden MOV-Befehle gepackt werden.

Um die zwei Subtraktionsbefehle an den Labels L1 und L2 mit Prädikaten in eine Gruppe zu bringen, muss zunächst der Code umgestellt werden: erst werden die Operanden auf Gleichheit überprüft, dann zum Ende verzweigt oder der nächste Vergleich gemacht, der zur passenden Subtraktion führt, bevor die Schleife von vorne beginnt.

```
1 .implicit
2
3 INIT:  MOV    R2 = 24
4        MOV    R3 = 8          ;;
5
```

```
6  START: CMP.EQ P1, P2 = R2, R3 ;;
7     (P1) JMP    END
8     (P2) CMP.LT P3, P4 = R2, R3 ;;
9     (P3) SUB    R3, R3 = R2
10    (P4) SUB    R2, R2 = R3     ;;
11         JMP    START           ;;
12
13 END
```

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

2 Superskalartechnik

2.1 Algorithmus von Tomasulo

a) Gegeben sei ein superskalarer Prozessor mit folgenden Eigenschaften:

- Interne Parallelisierung nach Tomasulo
- Pipeline bestehend aus Fetch, Decode, Issue+Renaming, 1x-7x Execute, 0x oder 1x oder 3x Memory Access, ggf. Writeback. Die Dispatch- und Retirement-Phasen werden weggelassen.
  IF ID IS EX MEM WB
- Zwei Lade-Speichereinheiten (*Load/Store Unit*), eine Integer-Additionseinheit, eine Integer-Multiplikationseinheit, eine FP-Additionseinheit
- *Delayed branches*, kein Anhalten (*Stalling*) und keine dynamische Vorhersage, sondern fortwährendes Füllen der Pipeline; dazu sei ein Sprungzieladresscache vorhanden und die statische Vorhersage laute auf *Taken*
- Volles Bypassing
- FP-Register und normale Register können gleichzeitig in der WB-Stufe beschrieben werden
- Die Befehlszuordnungs- und Rückordnungsbandbreite betrage 4 Befehle; zwei Befehle werden pro Takt maximal geholt
- Die Auswertung der Sprungzieladresse erfolge in der Stufe *Execute*; das Schreiben des Befehlszählers in der WB-Stufe
- Auf dem Ergebnisbus können Ressourcenkonflikte entstehen

Folgender Code werde darauf ausgeführt, wobei R0=0, R1 eine Speicheradresse, R2=R1+24 und F2 beliebig sei:

```
1 LOOP: LD.D   F0,0(R1)    ; loads Mem[i]
2       ADD.D  F4,F0,F2    ; adds to Mem[i]
3       S.D    0(R1),F4    ; stores into Mem[i]
4       ADD    R1,R1,#8    ;
5       SUB    R3,R1,R2    ; R3 = R1–R2
```

```
6        BLTZ   R3,LOOP      ; delayed branch if R1 < R2
```

Für die Ausführungseinheiten gelten folgende Zeiten:

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

Alle Einheiten bis auf die Divisionseinheit seien intern mit Pipelines implementiert.

Zeichnen Sie den Verlauf der Pipeline ab Beginn der Schleife unter der Annahme, dass alle vorhergehenden Befehle bereits vollständig durchgeführt und gültig gemacht worden seien, und führen Sie Buch über die Befehlswarteschlange (*Instruction Queue*), die Reservation Stations, die Registerstatustabelle und den Rückordnungspuffer (*Reorder Buffer*).

Bemerkung: *Beachten Sie die Annahmen und Erläuterungen in den Übungsfolien bezüglich der hier dargestellten Belegung der Reservation Stations*

Takt 1 & 2

Befehlsfenster

Nummer	Befehl	Station
1	`LD.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

Reservation Stations

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						

Achtung: *Aktiv* bedeutet hier, dass nicht nur die Reservation Station belegt ist, sondern auch die Ausführungseinheit aktiv ist.

Registerstatustabelle

Feld	PhR0	PhR1	PhR2	PhF0	PhF2	PhF4	...
Architektur-Register	R1*	R2*		F0	F2*	F4	...
Qi							

Gültige Registerabbildungen sind mit einem Stern (*) gekennzeichnet.

Der Ladebefehl wird hier erst mit erfolgter Zuweisung zu einer Reservation Station eingetragen, häufig aber auch schon früher in mehrzeilige Reservation Stations während der

Issue-Phase, um dann auf die Verfügbarkeit der Operanden und das Angestoßenwerden (Dispatch) zu warten.

Rückordnungspuffer

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

Die Eintragungen in den Rückordnungspuffer erfolgen, wenn die Befehle die *In-Order*-Phase verlassen, um die spätere Gültigmachung in Programmreihenfolge zu gewährleisten; daher also noch in unserer Dekodierphase, also vor der *Issue*-Phase.

Pipeline

Befehl		Takt		
		1	2	3
`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

Takt 3 & 4

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`	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

Reservation Stations

Einheit	Aktiv	Befehl	Vj	Vk	Qj	Qk	A
L/S 1	j	LD.D					0 + [PhR0]
L/S 2	n						
Int-Add	n	ADD	[PhR0]	#8			
Int-Mul	n						
FP-Add	n						

Registerstatustabelle

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

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

Pipeline

Befehl		**Takt**				
		1	2	3	4	5
`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	IS	EX
`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

Takt 5 & 6

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

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]			
Int-Mul	n						
FP-Add	n						

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		

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

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

Takt 7 & 8

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

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	PhF4	PhF6	PhF8	PhF10	PhF12
Architektur-Register	R1	R2*	R1*	R3*	R1	F0*	F2*	F4	F0	F4	F0	F4
Qi								FP-Add	L/S2			

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.

Pipeline

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

Takt 9 & 10

Befehlsfenster

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

Reservation Stations

Einheit	Aktiv	Befehl	Vj	Vk	Qj	Qk	A
L/S 1	n	S.D	[PhF4]		FP-Add		0 + [PhR0]
L/S 2	j	LD.D		0			0 + [PhR2]
Int-Add	n	ADD	[PhR2]	8			
Int-Mul	n						
FP-Add	n						

Registerstatustabelle

Feld	PhR0	PhR1	PhR2	PhR3	PhR4	PhF0	PhF2	PhF4	PhF6	PhF8	PhF10	PhF12
Architektur-Register	R1	R2*	R1	R3*	R1	F0	F2*	F4*	F0	F4	F0	F4
Qi	Int-Add								L/S2			

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	L/S 1
1	PhF4	FP-Add

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

Pipeline

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

Pipeline

Befehl		**Takt**																				
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
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	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						
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

[1] Der CDB wird beim Speicherzugriff von `S.D` nicht benutzt.

[2] Der Befehlszähler wird nicht über den CDB geschrieben, daher entsteht hier kein Ressourcenkonflikt.

18 Befehle in 22 Takten bei einer minimalen Pipeline-Länge von 4 Stufen (Memory-Write) ergeben einen IPC-Wert von $\frac{18}{22-(4-1)} = \frac{18}{19} = 0,947$. Im Vergleich mit normalem Pipelining ist der Wert jedoch sehr gut, auch wenn es noch kein superskalarer Speedup ist.