
Universität Karlsruhe (TH)
Forschungsuniversität • gegründet 1825

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

Zentralübung Rechnerstrukturen

Lösungsblatt 4: Parallelismus auf Befehlsebene

1 Moderne Mikroprozessoren

Heutige Prozessoren bringen sequentielle Befehlsfolgen intern parallel zur Ausführung.

1.1 Pipelining

In der Vorlesung haben Sie Pipelining kennengelernt.

a) Erläutern Sie, welche Konflikte in Pipelines auftreten können, und wie sie entstehen.

Ressourcen-/Strukturkonflikt, Datenkonflikt (Abhängigkeit: RAW, Gegenabhängigkeit: WAR, Ausgabeabhängigkeit: WAW), Steuerflußkonflikt

b) Welche Methoden gibt es, Ressourcenkonflikte in Pipelines zu vermeiden?

Stalling, Pipelining, zusätzliche Ressourcen hinzufügen

1.2 Sprungvorhersage

Ein wesentliches Problem bei Pipelines sind die bedingten Steuerflußbefehle, die im einfachsten Fall mit unmittelbar folgenden `nop`-Befehlen (Software, compilerbasiert) oder mittels Anhalten (*Stalling*) seitens der Pipeline-Hardware behandelt werden. Eine Verbesserung bringt die Technik der Sprungvorhersage, bei welcher der wahrscheinliche Ausführungspfad in die Pipeline geladen wird. Wurde der falsche Pfad vorhergesagt, müssen die darauffolgenden, bereits teilweise berechneten Instruktionen aus der Pipeline entfernt werden (*Flushing*).

a) Wo liegt der Unterschied zwischen Ein-und Zwei-Bit-Prädiktoren und (m,n)-Korrelationsprädiktoren?

Obwohl es sich bei beiden Prädiktoren bereits um dynamische Vorhersagetechniken handelt, werden Abhängigkeiten der Sprünge untereinander nicht berücksichtigt. Dies ist jedoch bei Korrelationsprädiktoren möglich, indem über ein m *Bit breits Schieberegister aus* 2^m n*-Bit-Prädiktoren ausgewählt wird. [**?**]*

b) Welche Stufe eines Prozessors ist für die Sprungvorhersage verantwortlich?

Die Befehlsholeinheit holt in Abhängigkeit der Vorhersage im Voraus geholter Sprungbefehle die nächsten Befehle und führt daher auch die Sprungvorhersage aus.

c) Führen Sie folgenden Codeabschnitt auf einem Zwei-Bit-Prädiktor mit Sättigungszähler (hier stand fälschlicherweise die ganze Zeit über Hysteresezähler!) und einem (2,2)-Korrelationsprädiktor aus (R0 ist „Zero“-Register). Der Zwei-Bit-Prädiktor sei mit *Weakly Not Taken* (WNT) initialisiert; beim Korrelationsprädiktor sei das Schieberegister mit (NT, NT) vorbelegt, die vier Prädiktoren seien ebenfalls mit WNT gefüllt. Hierbei müssen die letzten zwei Sprünge nicht berücksichtigt werden: Ihre Bedingung ist immer wahr und somit wird immer gesprungen.

```
1  INIT:   LOAD R1,#0   ; R1=0
2          LOAD R2,#2   ; R2=2
3  START:  CMP  R1,R0   ; R1==0?
4          BRNZ L1      ; if (R1!=R0) goto L1
5          LOAD R1,#1   ; R1=1
6  L1:     CMP  R1,R2   ; R1==R2?
7          BRNZ L2      ; if (R1!=R2) goto L2
8          LOAD R1,#0   ; R1=0
9          BRA start    ; goto START
10 L2:     LOAD R1,#2   ; R1=2
11         BRA start    ; goto START
```

Sprungverläufe

Sprung 1 BRNZ L1	Sprung 2 BRNZ L2	Sprung 3 BRA start	Sprung 4 BRA start
NT (R1=1)	T (R1=2)	–	T
T (R1=2)	NT (R1=0)	T	–
NT (R1=1)	T (R1=2)	–	T
T (R1=2)	NT (R1=0)	T	–

Zwei-Bit-Prädiktor

*Es werden **5** Fehlannahmen gemacht:*

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

Korrelationsprädiktor

*Beachten Sie, daß wir nur 4 Prädiktoren über das BHR adressieren, also beide Sprünge dieselben Register verwenden. Es werden dann **3** Fehlannahmen gemacht (Vorhersage im Format BHR/PHT):*

Sprung 1			Sprung 2		
Prädiktion	Sprung	Neue Vorh.	Prädiktion	Sprung	Neue Vorh.
(NT,NT)/WNT	NT	(NT,NT)/NT	(NT,NT)/NT	**T**	(NT,NT)/WNT & (NT,T)/WNT
(NT,T)/WNT	**T**	(NT,T)/WT & (T,T)/WNT	(T,T)/WNT	NT	(T,T)/NT & (T,NT)/WNT
(T,NT)/WNT	NT	(T,NT)/NT & (NT,NT)/WNT	(NT,NT)/WNT	**T**	(NT,NT)/WT & (NT,T)/WT
(NT,T)/WT	T	(NT,T)/T & (T,T)/NT	(T,T)/NT	NT	(T,T)/NT & (T,NT)/NT

1.3 Spekulative Befehlsausführung

a) Wie funktionieren Trace Caches?

Bei einem Trace Cache wird eine feste Anzahl an ausgeführten Befehlen nach jedem Sprung in einem assoziativen Speicher gesichert, so daß beim nächsten Durchlauf das Laden der dem Sprung nachfolgenden Befehle schneller vonstatten geht. Dies ist besonders hilfreich, wenn mehrere Pfade gleichzeitig ausgeführt werden sollen, und wenn die Speicherbandbreite nicht ausreichend für mehrere simultane Speicherzugriffe ist, oder auch der Prozessor eine potentiell hohe Zuordnungsbandbeite besitzt, die anders nicht genutzt werden kann. Dabei können mehrere bedingte Sprungbefehle in einem Trace vorkommen.

b) Welche Techniken ermöglichen Prädikation? Wo ist sie sinnvoll?

Zur Prädiktion sind in der Pipeline Statusbits nötig, welche den spekulativen Befehlen anzeigen, daß sie von einer Spekulation abhängen und daher noch nicht vor der endgültigen Auflösung gültig gemacht werden dürfen.

Prädikation ist besonders bei kurzen Verzweigungen sinnvoll, bei denen im Voraus nicht gut abgeschätzt werden kann, ob der Sprung genommen wird oder nicht. Ferner wird die Berechnung eines Sprungziels verhindert und der Code kann sequentiell geladen werden und so Caches besser nutzen.

c) Was passiert bei der Mehrpfadausführung?

Liegen längere Codestücke vor, bei denen zwei verschiedene Pfade ähnlich wahrscheinlich sind, können beide Pfade ausgeführt werden. Mittels Registerumbenennung werden mögliche Konflikte bereits im Voraus ausgelöst; bei Entscheidung ob der tatsächlichen Sprungrichtung werden die unnötig ausgeführten Befehle verworfen und die Befehle des korrekten Pfades gültig gemacht.

2 Superskalartechnik

Um einen IPC[1]-Wert > 1 zu erhalten, muß mehr als ein Befehl pro Takt vollendet werden. Dies ermöglichen zum Beispiel die Superskalartechnik und VLIW-Architekturen.

a) Geben Sie den grundsätzlichen Unterschied zwischen superskalaren und VLIW-Prozessoren an, was die Befehlsparallelisierung anbelangt.

[1] Instructions per Cycle = $\frac{1}{\text{Cycles per Instruction}}$

Bei VLIW ist der Name Programm: Im recht langen Befehlswort plaziert der Compiler so viele parallel ausführbare Befehle wie möglich. Der Prozessor ist von der Aufgabe der Parallelitätsfindung entbunden, dafür muß die Befehlsholeinheit ein großes Datenpaket holen. Die Befehlsaufführung geschieht vollkommen parallel.

Beim Superskalarprinzip gibt der Compiler keine Angaben bezüglich der parallelen Ausführbarkeit vor; der Prozessor entscheidet dynamisch unter Berücksichtigung der Ressourcennutzung und Auflösung möglicher Konflikte, welche Befehle zu welchem Zeitpunkt ausgeführt werden können. Der Vorteil ist hierbei, daß der Prozessor nach außen hin unverändert mit sequentieller Befehlsverarbeitung scheint.

b) Bei der Befehlsausführung außerhalb der Befehlsreihenfolge (*out-of-order*) muß auf die Einhaltung der ursprünglichen Semantik geachtet werden. Was passiert bei der präzisen Unterbrechungsbehandlung (*Precise Interrupts, Precise Exception Handling*)?

*Bei externen Unterbechungen (*Interrupts*) sowie synchronen Unterbrechungen (*exceptions*) muß sichergestellt werden, daß alle Befehle, die vor dem Eintreten der Unterbrechung in einem sequentiellen Modell ausgeführt würden, auch tatsächlich beendet werden, und zusätzlich alle weiteren in die Pipeline geladenen Befehle verworfen werden, so daß die „Sicht" auf den Prozessor für die Behandlungsroutine exakt die gleiche ist wie im sequentiellen Befehlsmodell.*

c) Welche Einheit ist für die korrekte Behandlung dieser Unterbrechungen verantwortlich?

Precise Interrupts werden in der Befehlszuordnungsstufe behandelt: Dort werden keine Befehle mehr eingeladen; bereits zugewiesene Befehle werden mittels der Rückordnungseinheit gültig gemacht, bis alle Umbenennungsregister wieder frei sind. Der theoretisch nächste Befehl im Befehlsfenster muß nach der Behandlungsroutine ausgeführt werden. Für die Behandlungsroutine selbst müssen vorab noch das Befehlsfenster und alle Puffer geleeert werden; anschließend muß der auszuführende Befehl neu geladen werden.

2.1 Ausführungseinheiten

a) Wie definieren sich Latenz und Durchsatz?

*Als **Latenz** definiert man die **zusätzliche** Zeit (in Takten), die vergeht, bis das Ergebnis einer Aktion weiteren Einheiten zur Verfügung steht. Latenz versteht sich also als die Wartezeit aus der Sicht eines darauffolgenden Befehls und ist damit eigentlich die Bearbeitungsdauer - 1.*

*Unter **Durchsatz** versteht man, wieviele Befehle pro Takt beendet werden können. Eine Einheit mit einer Bearbeitungsdauer von drei und nur einem Befehl pro Durchgang hat einen Durchsatz von $1/3$; bei 5 Takten Latenz und drei Befehlen in der Einheit beträgt der Durchsatz $\frac{3}{5+1} = 1/2$.*

b) Welches Konzept steht hinter Vektoreinheiten?

Vektoreinheiten sind in der Lage, dieselbe Operation auf mehreren Daten gleichzeitig auszuführen (Single Instruction – Multiple Data, SIMD). Dazu werden meist etwas größere

Datenwörter verwendet, die dann aber wiederum in kleineren Blöcken verarbeitet werden, z.B. auf einem 32-Bit-Prozessor 64-Bit-Datenwörter für die Vektoreinheiten, die dann 4 oder acht Pakete zu 16 oder 8 Bit gleichzeitig verarbeiten. Dazu sind zusätzliche Register nötig sowie eine Befehlssatzerweiterung um Registerladebefehle und zum Anstoßen der Ausführung.

2.2 Algorithmus von Tomasulo und Scoreboarding

a) Welche grundsätzlichen Phasen durchlaufen Befehle in superskalaren Pipelines?

Vorab muß die Befehlsholstufe durchlaufen werden, in der das Befehlsfenster gefüllt und die Sprungvorhersage durchgeführt wird.

*Gemäß Brinkschulte/Ungerer werden dann fünf Phasen durchlaufen: Dekodierung, Registerumbenennung, Befehlszuordnung, Ausführung, Rückordnung (*Retirement*). Im weiteren werden wir mit diesen Annahmen arbeiten. Die Registerumbennennung kann auch in der Dekodierungsstufe enthalten sein; dadurch wird selbige aber sehr groß und kann den langsamsten Pfad enthalten, nach welchem sich die maximal erzielbare Taktrate richtet. Dennoch fassen wir der Einfachheit und Übersicht halber die Stufen zusammen.*

Nach Hennessey & Patterson werden nur vier weitere Phasen benötigt: Issue, Read Operands, Execute, Write Results.

b) Worin liegt bei der Befehlsausführung außerhalb der Befehlsreihenfolge der konzeptionelle Unterschied zwischen dem Tomasulo-Algorithmus und dem Scoreboarding?

*Bei Tomasulo erfolgt das Anstoßen der Befehle denzentral aus den Reservation Stations einer Einheit oder einer Einheitenfamilie, und auch das Einlesen der Operanden beziehungsweise die Ergebnisweiterleitung (*Result Forwarding*) geschieht dezentral für jede einzelne Reservation Station.*

Beim Scoreboarding hingegen werden zentral alle noch auszuführenden Befehle bereitgehalten und bei Vorhandensein ihrer Operanden sowie einer passenden Ausführungseinheit einer solchen zugewiesen.

Welchen Vorteil hat Scoreboarding im Vergleich zum Tomasulo-Algorithmus?

*Die Entkopplung der Umbenennungs- und Zuweisungseinheit von den einzelnen Ausführungseinheiten sowie die dadurch entstehende Entlastung des Ergebnisbusses (*Common Data Bus, CDB*) stellen die großen Vorteile von Scoreboarding. Dennoch wird heutzutage noch immer vermehrt der Algorithmus von Tomasulo eingesetzt, wohl wegen der feineren Prozessorstufen.*

c) Gegeben sei ein superskalarer Prozessor mit folgenden Eigenschaften:

- Tomasulo-Pipeline
- zwei Lade-Speichereinheiten (*Load/Store Unit*), eine Integer-Additionseinheit, eine Integer-Multiplikationseinheit, eine FP-Additionseinheit, zwei FP-Multiplikationseinheiten und eine FP-Divisionseinheit

- Verzögerung bei bedingten Sprungbefehlen, 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
- 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 (*Instruction Counter, Program Counter*) in der WB-Stufe

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

```
1 LOOP: LD     F0,0(R1)    ; loads Mem[i]
2       ADDD   F4,F0,F2    ; adds to Mem[i]
3       SD     0(R1),F4    ; stores into Mem[i]
4       ADD    R1,R1,#8    ; executed whether branch is taken or not
5       BLT    R1,R2,LOOP  ; delayed branch if  R1 < R2
```

Für die Ausführungseinheiten gelten folgende Zeiten:

Einheit	**L/S**	**Integer-Add**	**Integer-Mul**	**FP-Add**	**FP-Mul**	**FP-Div**
Anzahl	2	1	1	1	2	1
Bearbeitungsdauer (in Takten)	3	1	3	2	7	25

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`	`F0,16(R1)`	ID
2	`ADDD`	`F4,F0,F2`	ID
3	`SD`	`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						
FP-Mul 1	n						
FP-Mul 2	n						
FP-Div	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 erst mit erfolgter Zuweisung zu einer Reservation Station eingetragen.

Rückordnungspuffer

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

Takt 3 & 4

Befehlsfenster

Nummer	Befehl		Station
1	`LD`	`F0,16(R1)`	M
2	`ADDD`	`F4,F0,F2`	ID
3	`SD`	`0(R1),F4`	ID
4	`ADD`	`R1,R1,#8`	IS
5	`BLT`	`R1,R2,LOOP`	ID
6	`LD`	`F0,16(R1)`	ID
7	`ADDD`	`F4,F0,F2`	IF
8	`SD`	`0(R1),F4`	IF

Reservation Stations

Einheit	Aktiv	Befehl	Vj	Vk	Qj	Qk	A
L/S 1	j	LD					16 + [PhR0]
L/S 2	n						
Int-Add	n	ADD	[PhR0]	#8			
Int-Mul	n						
FP-Add	n						
FP-Mul 1	n						
FP-Mul 2	n						
FP-Div	n						

Registerstatustabelle

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

Rückordnungspuffer

Befehlsnr.	Ziel	Quelle
6	PhF6	L/S 2
5	PC	Int-Add
4	PhR2	Int-Add
3	null	any L/S
2	PhF4	FP-Add
1	PhF0	L/S 1

Takt 5 & 6

Befehlsfenster

Nummer	Befehl		Station
1	`LD`	`F0,16(R1)`	M
2	`ADDD`	`F4,F0,F2`	ID
3	`SD`	`0(R1),F4`	ID
4	`ADD`	`R1,R1,#8`	WB
5	`BLT`	`R1,R2,LOOP`	IS
6	`LD`	`F0,16(R1)`	IS
7	`ADDD`	`F4,F0,F2`	ID
8	`SD`	`0(R1),F4`	ID
9	`ADD`	`R1,R1,#8`	ID
10	`BLT`	`R1,R2,LOOP`	ID
11	`LD`	`F0,16(R1)`	IF
12	`ADDD`	`F4,F0,F2`	IF

Reservation Stations

Einheit	Aktiv	Befehl	Vj	Vk	Qj	Qk	A
L/S 1	j	LD		16			16 + [PhR0]
L/S 2	n	LD	[PhR2]	16			
Int-Add	j	BLT	[PhR2]	[PhR1]			
Int-Mul	n						
FP-Add	n						
FP-Mul 1	n						
FP-Mul 2	n						
FP-Div	n						

Registerstatustabelle

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

Rückordnungspuffer

Befehlsnr.	**Ziel**	**Quelle**
10	PC	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	PhR2	Int-Add
3	null	any L/S Unit
2	PhF4	FP-Add
1	PhF0	L/S 1

Takt 7 & 8

Befehlsfenster

Nummer	Befehl		Station
1	`ADDD`	`F4,F0,F2`	EX
2	`SD`	`0(R1),F4`	ID
3	`BLT`	`R1,R2,LOOP`	WB
4	`LD`	`F0,16(R1)`	M
5	`ADDD`	`F4,F0,F2`	ID
6	`SD`	`0(R1),F4`	ID
7	`ADD`	`R1,R1,#8`	EX
8	`BLT`	`R1,R2,LOOP`	ID
9	`LD`	`F0,16(R1)`	ID
10	`ADDD`	`F4,F0,F2`	ID
11	`SD`	`0(R1),F4`	ID
12	`ADD`	`R1,R1,#8`	ID
13	`BLT`	`R1,R2,LOOP`	IF

Reservation Stations

Einheit	Aktiv	Befehl	Vj	Vk	Qj	Qk	A
L/S 1	n						
L/S 2	j	LD		16			16 + [PhR2]
Int-Add	n	ADD	[PhR2]	#8			
Int-Mul	n						
FP-Add	j	ADDD	[PhF0]	[PhF2]			
FP-Mul 1	n						
FP-Mul 2	n						
FP-Div	n						

Registerstatustabelle

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

Rückordnungspuffer

Befehlsnr.	**Ziel**	**Quelle**
12	PhR1	Int-Add
11	null	any L/S Unit
10	PhF12	FP-Add
9	PhF10	any L/S Unit
8	PC	Int-Add
7	PhR0	Int-Add
6	null	any L/S Unit
5	PhF8	FP-Add
4	PhF6	L/S 2
3	PC	Int-Add
2	null	any L/S Unit
1	PhF4	FP-Add

Pipeline

Befehl		**Stufe** 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
LD	F0,16(R1)	IF	ID	IS	M	M	M	WB													
ADDD	F4,F0,F2	IF	ID					IS	EX	EX	WB										
SD	0(R1),F4		IF	ID							IS	M/WB									
ADD	R1,R1,#8		IF	ID	IS	EX	WB														
BLT	R1,R2,LOOP			IF	ID		IS	EX	WB												
LD	F0,16(R1)			IF	ID		IS	M	M	M		WB[1]									
ADDD	F4,F0,F2				IF	ID						IS	EX	EX	WB						
SD	0(R1),F4				IF	ID									IS	M/WB					
ADD	R1,R1,#8					IF	ID		IS	EX			WB								
BLT	R1,R2,LOOP					IF	ID						IS	EX	WB[2]						
LD	F0,16(R1)						IF	ID					IS	M	M	M	WB				
ADDD	F4,F0,F2						IF	ID									IS	EX	EX	WB	
SD	0(R1),F4							IF	ID											IS	M/WB
ADD	R1,R1,#8							IF	ID							EX		WB			
BLT	R1,R2,LOOP								IF	ID								IS	EX		WB[3]

[1] Der CDB wird beim Speicherzugriff nicht benutzt.

[2] Der Befehlszähler wird nicht über den CDB geschrieben.

[3] Hier findet tatsächlich überhaupt kein Zugriff auf den CDB statt.

15 Befehle in 20 Takten bei einer minimalen Pipeline-Länge von 4 Stufen (Memory-Write) ergeben $IPC = \frac{15}{20-(4-1)} = \frac{15}{17} = 0.88$. Im Vergleich mit normalem Pipelining ist der Wert jedoch sehr gut, auch wenn es noch kein superskalarer Speedup ist.