Übungsblatt 7

7.1 Pipelining: Exceptions

Angenommen, die MIPS-Instruktion `sub $8, $9, $8` werde von der Pipeline lt. VO-Folie 3-118 ausgeführt und löse eine „arithmetic overflow"-Exception aus.

(a) Bestimmen Sie die Werte der für die Exception-Behandlung relevanten Steuersignale für jede Pipeline-Stufe, die diese Instruktion durchläuft. Erklären Sie zudem, warum manche Steuersignale im ID/EX-Pipeline-Register gespeichert werden, während andere direkt in die EX-Pipeline-Stufe geführt werden.

(b) Die Latenz der EX-Pipeline-Stufe könnte verringert werden, wenn die Behandlung der Exception erst in der nachfolgenden Pipeline-Stufe erfolgt. Erklären Sie anhand der gegebenen Instruktion die Nachteile dieses Ansatzes.

7.2 Statische „Dual Issue" Prozessoren

Gegeben sei ein statischer „Dual-Issue"-Prozessor (vgl. VO-Folie 3-133) mit fünf Pipeline-Stufen, auf dem ein Programm mit folgenden Befehlshäufigkeiten ausgeführt wird:

Befehlsklasse	Häufigkeit
ALU-Operationen	40%
beq (Sprung richtig vorhergesagt)	10%
beq (Sprung falsch vorhergesagt)	5%
lw	30%
sw	15%

Nehmen Sie an, dass der Prozessor stets *zwei beliebige Instruktionen* im gleichen Takt ausführen kann (mit Ausnahme von Branch-Befehlen), die Sprungvorhersage in der ersten Pipeline-Stufe erfolgt und Sprünge für „branch"-Befehle in der zweiten Pipeline-Stufe ausgeführt werden. Im Programm sind Branch-Befehle nicht unmittelbar hintereinander angeordnet, die Häufigkeit der Branch-Befehle an geraden bzw. ungeraden Wortadressen ist gleich, Leertakte aufgrund von Datenabhängigkeiten treten nicht auf, und „delay slots" werden nicht verwendet.

(a) Bestimmen Sie den CPI-Wert für die Ausführung dieses Programms.

(b) Welcher Speedup im Vergleich zu (a) würde erreicht, wenn die Sprungvorhersage perfekt wäre?

(c) Angenommen, der Prozessor habe nur ein Write-Port in der Registereinheit, d.h. es können nicht zwei Befehle parallel in die Registereinheit schreiben (vgl. VO-Folie 3-133). Welcher Speedup wird erreicht, wenn ein zweiter Write-Port hinzugefügt wird?

7.3 Schleifenabrollen, Superskalare Prozessoren

Gegeben sei das untenstehende Code Fragment. Ordnen Sie den Code der zweimal abgerollten Schleife so an, dass er optimal auf einem superskalaren Prozessor mit „Delayed Branching“ entsprechend VO-Folie 3-154 ausgeführt werden kann. Es gelten die Latenzen zwischen abhängigen Befehlen, wie in der am Ende des Übungsblattes stehenden Tabelle angegeben. Wie viele Takte werden pro Ergebniselement benötigt?

```
#           $f0 (v) and $f8 (u) contain constants
 loop:      l.d   $f2,  0($t0)      # load x[i]
            add.d $f2,  $f2,  $f8   # x[i] + u
            l.d   $f12, 0($t1)      # load y[j]
            add.d $f12, $f12, $f0   # y[j] + v
            div.d $f10, $f2,  $f12  # x[i] / y[j]
            add.d $f2,  $f10, $f12  # y[j] + x[i] / y[j]
            mul.d $f4,  $f2,  $f10  # (y[j]+x[i]/y[j])*((x[i] + v)/ y[j])
            s.d   $f4,  0($t2)      # store z[i]
            addi  $t0,  $t0,  8
            addi  $t1,  $t1,  8
            addi  $t2,  $t2,  8
            bne   $t0,  $t3,  loop
            nop
```

7.4 Dynamisches Pipeline-Scheduling

Stellen Sie die Ausführung des (nicht abgerollten und nicht umgeordneten) Codes aus Ü 6.5 auf dem superskalaren Beispielprozessor mit „Dynamic Scheduling“ und „Branch Prediction“ wie auf VO-Folie 3-156 dar. Wie viele Takte werden pro Ergebniselement benötigt?

Eigenschaften des Beispielprozessors:

- Es gibt zwei „Issue“-Pipelines, sodass in jedem Takt mit der Verarbeitung von zwei Befehlen begonnen werden kann: ein FP-Befehl und ein anderer Befehlstyp (INT, Branch, Load/Store).
- Es sind genügend Reservierungsstationen und Funktionseinheiten vorhanden, um alle Befehle der Schleife aufzunehmen.
- Zwischen abhängigen Befehlen in derselben Pipeline findet „Forwarding“ statt, sodass die Fertigstellung (Commit) eines Befehls und der Ausführungsbeginn eines abhängigen Befehls im selben Takt erfolgen können.
- Die „Commit“-Reihenfolge muss nicht mit der „Issue“-Reihenfolge übereinstimmen („out-of-order completion“).
- Die Dauer der „Execute“-Phase für verschiedene Befehlsgruppen sei durch folgende Tabelle gegeben. Die Dauer von „Branch“- und „Store“-Befehlen ist hier nicht relevant, weil sie jedenfalls kleiner als die Ausführungsdauer einer Iteration des gegebenen Codes ist und die Sprungvorhersage für die gegebene Schleife fast immer richtig ist.

Befehlsgruppe	FP Load	FP Arithmetik	Übrige Befehle
Execute/Takte	2	3	1

7.5 VLIW

Ordnen Sie den Code der sechsmal abgerollten Schleife aus Ü 6.5 so an, dass er optimal auf einem VLIW-Prozessor entsprechend VO-Folien 3-158 ausgeführt werden kann. Nehmen Sie an, dass der Prozessor 64 FP-Register besitzt und Branch Prediction implementiert. Die Latenzen zwischen abhängigen Befehlen entnehmen Sie bitte untenstehender Tabelle.

(a) Wie viele Takte werden pro Ergebniselement benötigt?

(b) Bestimmen Sie für die Ausführung dieses Codes die Effizienz der Prozessorauslastung und den IPC-Wert.

Latenzen für Befehlsabhängigkeiten (für Ü 7.3 und Ü 7.5):

Erzeugender Befehl (schreibt Register $x)	**Benutzender Befehl (liest Register $x)**	**Latenz / Zwischentakte (um Leerzyklen zu vermeiden)**
FP ALU operation	FP ALU operation	3
FP ALU operation	Store FP double	2
Load FP double	FP ALU operation	1
Load FP double	Store FP double	0
Load integer	Integer operation	1
Load integer	Branch	2
Integer operation	Integer operation	0
Integer operation	Branch	1