Karlsruher Institut für Technologie

Institut für Technische Informatik
Lehrstuhl für Rechnerarchitektur und Parallelverarbeitung
Prof. Dr. rer. nat. Wolfgang Karl

Zentralübung Rechnerstrukturen: Pipelining und Superskalartechniken

4. Aufgabenblatt

Besprechung: 7. Juni 2016

1 Pipelining

1. Pipeline und Zykluszeit

   Die Befehlsabarbeitung eines Prozessors lässt sich in folgende Phasen einteilen:

IF	ID	EX	MA	WB
250 ps	100 ps	130 ps	220 ps	50 ps

   Wenn der Prozessor mit einer 5-stufigen Pipeline (für jede Phase eine Pipelinestufe) ausgestattet wird, kommt bedingt durch das nötige Pipelineregister eine weitere Latenz von 20 ps hinzu.

   a) Wie groß ist die Zykluszeit des Prozessors ohne Pipeline, wie groß mit der 5-stufigen Pipeline?

   b) Welche Beschleunigung (Speed-Up) ergibt sich unter der Annahme, dass der Prozessor ohne Pipeline einen CPI von 1,0 und die Version mit Pipeline einen CPI von 1,2 hat?

   c) Wenn der Prozessor mit einer 3-stufigen Pipeline implementiert werden soll, müssen die existierenden 5 Pipelinestufen kombiniert werden. Wie würden Sie die 5 Phasen auf die drei Pipelinestufen aufteilen, um eine möglichst kurze Zykluszeit zu erhalten?

   d) Wenn der Prozessor auch mit einer 6-stufigen Pipeline realisiert werden könnte, welche Pipelinestufe würden Sie aufteilen, um die Zykluszeit des Prozessors zu senken?

2. Pipeline und Kontextwechsel

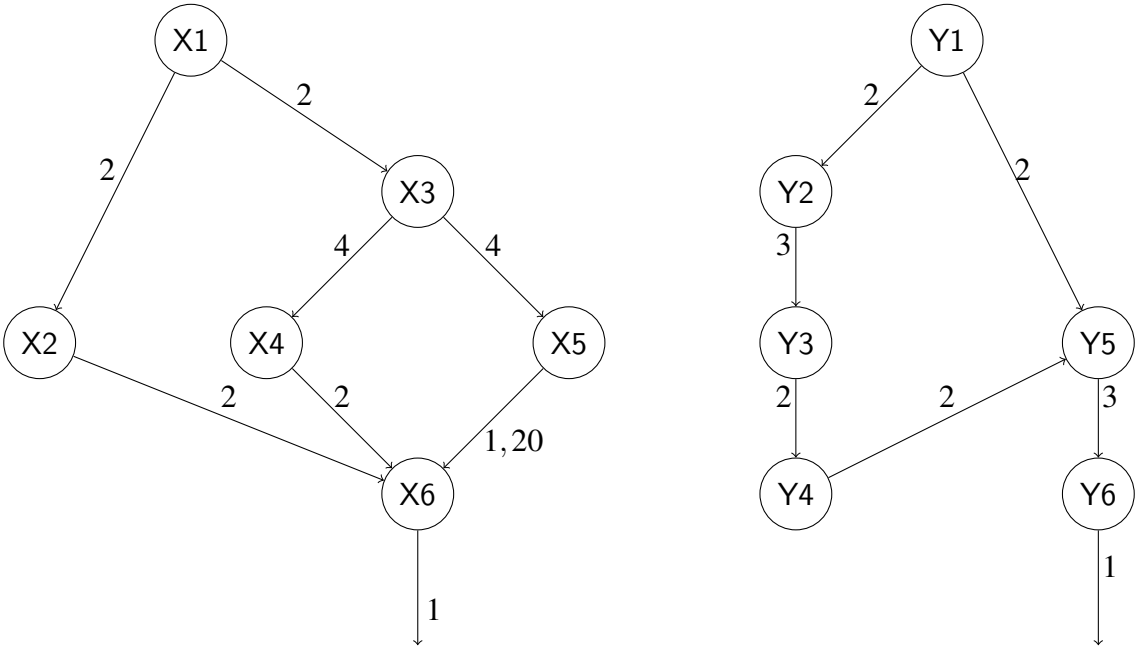
   Gegeben sei eine 5-stufige Pipeline, bei der ein Cache-Miss wie ein Konflikt behandelt wird, es also zu einem Pipeline Stall kommt. Nehmen Sie an, dass bei der Ausführung eines Bechnmarks alle 100 Zyklen eine L1 Cache-Miss auftritt und eine Behandlung dieses Misses 10 Zyklen, falls der Block im L2 Cache gefunden wird, oder 50 Zyklen benötigt, falls sich der Block ebenfalls nicht im L2 Cache befindet. Ein L2 Cache-Miss tritt alle 200 Zyklen auf. Nehmen Sie an, dass der CPI-Wert ohne Cache-Misses 1 betrage.

   a) Berechnen Sie den CPI-Wert unter Berücksichtigung der Cache-Misses.

   b) Gehen Sie nun davon aus, dass die Hardware mit zwei Threads arbeitet und es keinen Overhead für einen Kontextwechsel (Wechsel zwischen den Threads gibt). Beide Threads besitzen erneut das zuvor beschriebene Cache-Miss Verhalten. Berechnen Sie jeweils den CPI-Wert für beide Bechnmarkausführungen (Threads). Ergib sich eine Beschleunig? Falls ja wie und falls nein, erklären Sie warum ein Einsatz trotzdem sinnvoll sein könnte.

2 Multi-Threading

Betrachten Sie die Ausführung zweier Thread-Fragmente *X* und *Y* mit jeweils 6 Instruktionen. Gegeben ist der Abhängigkeitsgraph beider Threads, welcher die Abhängigkeiten zwischen den Instruktionen und die Latenzen angibt. Bei Instruktion *X*5 handelt sich um einen Cache-Zugriff, der im Fall eines Misses 20 Zyklen und im Falle eines Hits 1 Zyklus zur Ausführung benötigt. Gehen Sie davon aus, dass bei der Ausführung von X5 zunächst ein Cache-Miss auftritt.
Die Threads sollen auf einem System mit 2 HW-Threads, „Out-of-Order"-Ausführung und spekulativem Scheduling ausgeführt werden. Jeder HW-Thread hat seine eigene Instruction Fetch Queue (IFQ). Es können jeweils genau 2 Befehle geholt, dekodiert und zur Ausführung angestoßen werden. Außerdem können in jedem Zyklus bis zu 2 Befehle gültig gemacht (Retire) werden, der gemeinsame Datenbus (CDB) kann in jedem Zyklus 2 Befehle weiterreichen (Forwarding) und es können jeweils bis zu 2 Befehle aus einer einzigen, gemeinsamen Issue Queue mit 4 Einträgen zur Ausführung gebracht werden.



a) Geben Sie einen Schedule für Block Interleaving Multi-Threading an und gehen Sie dabei davon aus, dass mit der Ausführung von Thread *X* begonnen wird. Begonnen wird dabei mit Zyklus 1, in dem die ersten beiden Befehle in die Issue Queue eingetragen werden (Dispatch). Vervollständigen Sie die gegebene Tabelle.

Befehl	Dispatch (Issue Q Anzahl Befehle)	Issue	Register Fetch	Start Exec	Ende Exec	CDB	Retire (TX)	Retire (TY)
X1	1 (1)	2	3	4	5	6	7	
X2	1 (2)	4	5	6	7	8	9	

3 Algorithmus von Tomasulo I

Unter Verwendung des Algorithmus von Tomasulo, bestimmen Sie für jede Instruktion in der unten stehenden Sequenz, in welcher sie zugeteilt (Issue), sie zur Ausführung angestoßen (Execute) und wann ihr entsprechendes Ergebnis auf den Ergebnisbus gelegt wird (Write Result). Beginnen Sie mit Takt 1.

Gehen Sie von den folgenden Annahmen aus:

- Das Ergebnis kann erst im Takt nach dem Ende der Berechnung auf den Ergebnisbus gelegt werden.
- Befehle, die von anderen abhängen, können mit ihrer Berechnung erst im Takt nachdem der letzte Operand auf den Ergebnisbus gelegt wurde beginnen.
- Die Multiplikation benötigt vier Takte, die Division acht Takte, alle anderen Operationen zwei Takte zur Ausführung.
- Der Prozessor verfüge über eine Mul-/Div-Einheit und eine Add-/Sub-Einheit.
- Die Mul-/Div-Einheit hat eine Reservierungstabelle mit zwei Einträgen, die der Add-/Sub-Einheit vier.
- Der Prozessor kann nur eine Instruktion pro Takt auf die Reservierungstabellen zuteilen.
- Wenn mehrere Instruktionen gleichzeitig zur Ausführung angestoßen werden können, hat die ältere Instruktion Vorrang.

- Die Instruktionen werden in-order zugeteilt, d.h. wenn ein Befehl nicht sofort einer Reservierungstabelle zugeteilt werden kann, dann wird die Pipeline solange angehalten, bis diese Instruktion einer Reservierungstabelle zugeteilt werden kann.
- Der Prozessor verfüge über nur einen Ergebnisbus.
- Wenn mehrere Operationen gleichzeitig fertig werden, hat die Add-/Sub-Einheit Vorrang gegenüber der Mul-/Div-Einheit.

#	Instruktion	Issue	Executes	Writes Result
1	`mul r2, r1, r1`			
2	`div r4, r4, r2`			
3	`add r1, r4, r4`			
4	`add r2, r4, r3`			
5	`div r1, r2, r3`			
6	`sub r4, r4, r2`			
7	`add r3, r1, r2`			
8	`mul r1, r2, r3`			
9	`add r3, r3, r3`			
10	`sub r4, r4, r1`			

4 Algorithmus von Tomasulo II

Gegeben sei ein superskalarer Prozessor mit folgenden Eigenschaften:

- Interne Parallelisierung nach Tomasulo
- Pipeline bestehend aus Fetch, Decode, Issue+Renaming, Execute oder Memory Access, ggf. Writeback. Die Dispatch-und Retirement-Phasen werden weggelassen.
  IF ID IS EX/MA (WB)
- Zwei Lade-Speichereinheiten (*Load/Store Unit*), eine Integer-Additions/Subtraktionseinheit, eine Integer-Multiplikationseinheit, eine FP-Additionseinheit
- Statische Sprungvorhersage mit fortwährendem Füllen der Pipeline vom Sprungziel; dafür sei ein Sprungzieladresscache vorhanden und die Sprungvorhersage 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     ; branch if R1 < R2
```

Für die Ausführungseinheiten gelten folgende Zeiten:

Einheit	**L/S**	**Integer-Add/Sub**	**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*).

5 Very Long Instruction Word - VLIW

Der folgende Assembler-Code soll auf einem VLIW-Prozessor mit drei parallelen Ausführungseinheiten ausgeführt werden. Geben Sie hierfür eine möglichst effiziente Befehlsverteilung an. Die Befehle können beliebig umsortiert werden, so lange die Korrektheit der Anwendung gewährleistet ist.
Nehmen sie vereinfachend an, dass alle Befehle innerhalb eines Taktes abgearbeitet werden können.

```
1  add     r1, r2, r3              ; r1 = r2 + r3
2  sub     r5, r3, r5              ; r5 = r3 - r5
3  ld      r3, [r1]                ; Load r3 with [r1]
4  mul     r3, r3, r3              ; r3 = r3 * r3
5  st      [r5], r3                ; Store r3 in [r5]
6  ld      r9, [r7]                ; Load r9 with [r7]
7  ld      r11, [r12]              ; Load r11 with [r12]
8  add     r11, r11, r12           ; r11 = r11 + r12
9  mul     r11, r11, r9            ; r11 = r11 * r9
10 st      [r12], r11              ; Store r11 in [r12]
```

a) Nehmen Sie an, dass der Prozessor über drei Ausführungseinheiten verfügt, die jeweils alle Befehle ausführen können.

Slot 1	Slot 2	Slot 3

b) Nehmen Sie nun an, dass die ersten beiden Ausführungseinheiten arithmetisch-logische Befehle (add, sub, mul) ausführen können und die dritte für Load-/Store Operationen zuständig ist.

Slot 1	Slot 2	Slot 3

6 Literatur

Für Aufgaben 1.2 und 2 vergleiche M. Dubois, M. Annavaram, P. Stenström: Parallel Computer Organization and Design, Kap. 8