Karlsruher Institut für Technologie

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

Zentralübung Rechnerstrukturen

Pipelining und Superskalartechniken

Besprechung: 26. Mai 2011

1 Pipelining

Die Befehlsabarbeitung eines Prozessors lässt sich in folgende Phasen einteilen:

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

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 20ps hinzu.

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

b) Unter der Annahme, dass der Prozessor ohne Pipeline ein CPI von 1.0 und die Version mit Pipeline ein CPI von 1.2 hat: welcher Speed-Up ergibt sich?

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 Algorithmus von Tomasulo I

a) 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 (Excute) und wann ihr entsprechendes Ergebnis auf den Ergebnisbus gelegt wird (Write Result).
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	Excutes	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`			

3 Algorithmus von Tomasulo II

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*).

4 SimpleScalar - sim-outorder

Der Simulator `sim-outorder` aus dem SimpleScalar Toolset ermöglicht die zyklengenaue Simulation von komplexen, superskalaren Prozessoren. Der Prozessor selbst kann über eine Vielzahl von Optionen parametrisiert werden, so ist z.B. möglich die Anzahl der Ausführungseinheiten oder die Zuordnungsbreite zu verändern.

Gegeben seien nun die Benchmarks `basicmath`, `qsort` und `susan` aus der MiBench Benchmark-Suite. Simulieren Sie nun diese drei Benchmarks mit Hilfe des `sim-outorder` Simulators. Variieren Sie die Anzahl der Integer- und Floating Point-ALU-Einheiten zwischen 1 und 3. Die Anzahl der Integer-ALUs können über den Parameter `-res:ialu`, die Anzahl der FP-ALUs über den Parameter `-res:fpalu` verändert werden.

Was können Sie bei diesen Benchmarks beobachten?

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 Ausführugnseinheiten arithmetisch-logische Befehle (add, sub, mul) ausführen können und die dritte für Load-/Store Operationen zuständig ist.

Slot 1	Slot 2	Slot 3